PHPUnit Manual

Sebastian Bergmann

PHPUnit Manual

Sebastian Bergmann

data de publicação Edition for PHPUnit 3.7. Updated on 2013-06-19.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Exemplos

Capítulo 1. Automatizando Testes

Mesmo bons programadores cometem erros. A diferença entre um bom programador e um mau programador é que o bom programador usa testes para detectar seus erros o mais cedo possível. Quanto antes você testar por um erro, maiores são suas chances de encontrá-lo e consertá-lo. Isso explica porque deixar os testes para um momento logo antes do lançamento do programa é tão problemático. A maioria dos erros nem sequer chega a ser encontrado, e o custo de consertar aqueles que você encontra é tão alto que você tem que fazer uma triagem com os erros porque você simplesmente não consegue consertar todos.

Testar com o PHPUnit não é uma atividade totalmente diferente do que você já costuma fazer. É apenas uma forma diferente de fazê-lo. A diferença está entre *testar*, isto é, verificar que seu programa se comporta como o esperado, e *fazer uma bateria de testes*, fragmentos de código executáveis que automaticamente testam se as partes (unidades) do programa estão corretas. Esses fragmentos de código executáveis são chamados de testes unitários.

Neste capítulo vamos de um simples código de teste baseado em `print`até um teste totalmente automatizado. Imagine que nos peçam para testar um `vetor`embutido do PHP. Uma pequena funcionalidade a se testar é a função `count()`. Para um vetor criado recentemente esperamos que a função `count()` retorne `0`. Após adicionarmos um elemento, `count()` deverá retornar `1`. Exemplo 1.1, “Testando operações de vetores” mostra o que queremos testar.

Exemplo 1.1. Testando operações de vetores

```
<?php
$componente = array();
// espera-se que $componente esteja vazio.

$componente[] = 'elemento';
// espera-se que $componente contenha um elemento.
?>
```

Um jeito bem simples de verificar se estamos obtendo os resultados que esperamos é imprimir o resultado de `count()` antes e depois de adicionar o elemento (veja Exemplo 1.2, “Usando print para testar operações de um vetor”). Se obtivermos `0` e depois `1`, `array` e `count()` se comportaram como esperado.

Exemplo 1.2. Usando print para testar operações de um vetor

```
<?php
$componente = array();
print count($componente) . "\n";

$componente[] = 'elemento';
print count($componente) . "\n";
?>
```

```
0
1
```

Agora gostaríamos de mudar de testes que exigem interpretação manual para testes que podem executar automaticamente. Em Exemplo 1.3, “Comparando os valores esperado e real para testar operações de vetores”, escrevemos a comparação do valor esperado e do real em nosso código de teste e imprimimos `ok` se os valores forem iguais. Se alguma vez virmos uma mensagem `não ok` saberemos que algo está errado.

Exemplo 1.3. Comparando os valores esperado e real para testar operações de vetores

```
<?php
$componente = array();
print count($componente) == 0 ? "ok\n" : "não ok\n";

$componente[] = 'elemento';
print count($componente) == 1 ? "ok\n" : "não ok\n";
?>
```

```
ok
ok
```

Agora fatoramos a saída de comparação dos valores esperado e real em uma função que gera uma Exceção (Exception) onde há uma discrepância (Exemplo 1.4, “Usando uma função de asserção para testar operações de vetores”). Isso nos traz dois benefícios: a escrita dos testes se torna mais fácil e só obteremos saída quando algo estiver errado.

Exemplo 1.4. Usando uma função de asserção para testar operações de vetores

```
<?php
$componente = array();
assertTrue(count($componente) == 0);

$componente[] = 'elemento';
assertTrue(count($componente) == 1);

function assertTrue($condicao)
{
if (!$condicao) {
throw new Exception('Asserção falhou.');
}
}
?>
```

O teste agora está totalmente automatizado. Em vez de apenas *testar* como fizemos em nossa primeira versão, com esta versão temos um *teste automatizado*.

O objetivo de usar testes automatizados é cometer menos erros. Ainda que seu código não fique perfeito, mesmo com testes excelentes, você perceberá uma redução dramática nos defeitos assim que começar a automatizar os testes. Testes automatizados darão a você uma justificada confiança em seu código. Você poderá usar essa confiança para atravessar barreiras mais difíceis de design (Refatoração), melhorar as relações com sua equipe (Testes Inter-Equipes) e clientes, além de voltar para casa toda noite com a prova de que seu sistema está melhor agora do que estava de manhã, graças aos seus esforços.

Capítulo 2. Objetivos do PHPUnit

Até agora só tivemos dois testes para o `vetor` embutido e a função `count()`. Quando começarmos a testar as numerosas funções `array_*()` que o PHP oferece, precisaremos escrever um teste para cada um deles. Podemos escrever a infraestrutura para todos esses testes do zero, porém é muito melhor escrever uma infraestrutura de testes uma vez e então escrever apenas as partes únicas de cada teste. O PHPUnit é essa infraestrutura.

Um framework como o PHPUnit tem que resolver uma série de restrições, sendo que algumas delas parecem sempre conflitar umas com as outras. Ao mesmo tempo, os testes devem ser:

Fáceis de aprender a escrever. Se for difícil aprender a escrever testes, os desenvolvedores não aprenderão a escrevê-los.

Fáceis de escrever. Se os testes não forem fáceis de escrever, os desenvolvedores não vão escrevê-los.

Fáceis de ler. Os códigos de teste não devem conter cabeçalhos estranhos, de forma que o próprio teste não se perca no meio do "ruído" que o envolve.

Fáceis de executar. Os testes devem executar ao toque de um botão e apresentar seus resultados de uma forma clara e inequívoca.

Rápidos de executar. Testes devem executar tão rápido que possam ser executados centenas ou milhares de vezes por dia.

Isolados. Os testes não devem afetar uns aos outros. Se a ordem na qual os testes estão for alterada, os resultados dos testes não devem mudar.

Combináveis. Nós devemos ser capazes de executar qualquer quantidade ou combinação de testes juntos. Isso é uma consequência do isolamento.

Existem dois principais conflitos nessas restrições:

Fácil de aprender a escrever versus fácil de escrever. Testes geralmente não precisam de toda a flexibilidade de uma linguagem de programação. Muitas ferramentas de teste fornecem suas próprias linguagens que só incluem o mínimo necessário de funções para escrever testes. Os testes resultantes são fáceis de ler e escrever porque eles não têm o "ruído" para distrair você do conteúdo dos testes. Porém aprender mais uma linguagem de programação e ajustar as ferramentas é um inconveniente que tumultua a mente.

Isolado versus rápido de executar. Se você quer que os resultados de um teste não afetem os resultados de outro teste, cada teste deve criar o ambiente completo antes de começar a executar e retornar o ambiente ao seu estado original quando terminar. Porém ajustar o ambiente pode levar um longo tempo: por exemplo, conectar a um banco de dados e inicializá-lo para um estado conhecido usando dados realistas.

O PHPUnit tenta resolver esses conflitos usando o próprio PHP como linguagem de testes. Algumas vezes o poder total do PHP é demais para escrever pequenos testes de uma só linha, mas ao usar PHP aproveitamos toda a experiência e ferramentas que os programadores já possuem. Já que estamos tentando convencer testadores desconfiados, quebrar a barreira de escrever esses testes iniciais é particularmente importante.

O PHPUnit peca no lado do isolamento sobre execução rápida. Testes isolados são valiosos porque eles fornecem uma resposta de alta qualidade. Você não precisa pegar um relatório com uma porção de falhas, que foram de fato causadas por um teste da suíte que falhou lá no começo e deixou o resto do ambiente bagunçado pelo resto dos testes. Esta orientação através de testes isolados encoraja designs com um grande número de objetos simples. Cada objeto pode ser testado rapidamente em isolamento. O resultado é melhores designs e resultados mais rápidos.

O PHPUnit assume que a maioria dos testes passam e não vale a pena informar os detalhes dos testes bem sucedidos. Quando um teste falha, é importante relatar esse fato. A grande maioria dos testes deve passar e não vale a pena comentá-los, mas contar o número de testes que executaram, vale. Esta é uma suposição realmente baseada nas classes de relatório, e não no núcleo do PHPUnit. Quando os resultados de um teste são relatados, você vê quantos deles foram executados, mas você só vê detalhes daqueles que falharam.

Espera-se que os testes sejam refinados, testando cada aspecto de um objeto. Por isso, na primeira vez que um teste falha, a execução de testes é interrompida e o PHPUnit informa a falha. É uma arte testar executando vários testes pequenos. Testes refinados melhoram o design geral do sistema.

Quando você testa um objeto com o PHPUnit, você só o faz através da interface pública dos objetos. Testar baseado apenas no comportamento público visível encoraja você a confrontar e resolver mais cedo problemas difíceis de design, antes que os resultados de um design pobre possam infectar grandes partes do sistema.

Capítulo 3. Instalando o PHPUnit

Existem três formas suportadas de instalar o PHPUnit. Você pode usar o Instalador PEAR Installer [http://pear.php.net/manual/en/guide.users.commandline.cli.php] ou Composer [http://getcomposer.org/] para baixar e instalar o PHPUnit assim como suas dependências. Você também pode baixar um PHP Archive (PHAR) [http://php.net/phar] do PHPUnit que tem todas as dependências exigidas (assim como algumas opcionais) do PHPUnit em um único arquivo.

Nota

O suporte ao Composer e PHP Archive (PHAR) foi adicionado no PHPUnit 3.7 (tido como estável desde o PHPUnit 3.7.5). Versões anteriores do PHPUnit não estão disponíveis através desses canais de distribuição.

Nota

O PHPUnit 3.7 exige PHP 5.3.3 (ou superior) mas PHP 5.4.7 (ou superior) é altamente recomendável.

A biblioteca PHP_CodeCoverage que é usada pelo PHPUnit para coletar e processar a informação de cobertura de código, depende do Xdebug 2.0.5 (ou superior) mas o Xdebug 2.2.0 (ou superior) é altamente recomendável.

PEAR

Os dois comandos seguintes (que talvez você tenha que executar como `root`) são todo o necessário para instalar o PHPUnit usando o Instalador PEAR:

```
pear install pear.phpunit.de/PHPUnitpear config-set auto_discover 1
pear install pear.phpunit.de/PHPUnit
```

Cuidado

Dependendo da distribuição do seu Sistema Operacional e/ou seu ambiente PHP, você pode ter que instalar o PEAR ou atualizar sua instalação PEAR já existente antes que você possa proceder com as instruções desta seção.

`sudo pear upgrade PEAR` geralmente é o suficiente para atualizar uma instalação PEAR existente. O Manual do PEAR [http://pear.php.net/manual/en/installation.getting.php] explica como fazer uma instalação nova do PEAR.

Composer

Para adicionar o PHPUnit como uma dependência local por-projeto ao seu projeto, simplesmente adicione a dependência que está em `phpunit/phpunit` ao arquivo `composer.json` do seu projeto. Aqui está um exemplo mínimo de um arquivo `composer.json` que apenas define uma dependência em tempo de desenvolvimento do PHPUnit 3.7:

```
{
"require-dev": {
"phpunit/phpunit": "3.7.*"
}
}
```

Para uma instalação autônoma para um sistema inteiro via Composer, um `composer.json` similar ao mostrado abaixo pode ser usado para um diretório arbitrário.

```
{
"require": {
"phpunit/phpunit": "3.7.*"
},
"config": {
"bin-dir": "/usr/local/bin/"
}
}
```

PHP Archive (PHAR)

Você também pode baixar um PHP Archive (PHAR) do PHPUnit que tem todas as dependências exigidas (assim como algumas opcionais) do PHPUnit em apenas um arquivo

```
chmod +x phpunit.pharwget http://pear.phpunit.de/get/phpunit.phar
chmod +x phpunit.phar
```

Pacotes opcionais

Os seguintes pacotes opcionais estão disponíveis:

`DbUnit`

porta DbUnit para PHP/PHPUnit para suportar interação com o banco de dados de teste.

Este pacote pode ser instalado via PEAR usando o seguinte comando:

```
pear install phpunit/DbUnit
```

Este pacote pode ser instalado via Composer adicionando a seguinte dependência `"require-dev"`:

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

`PHP_Invoker`

Uma classe utilitária para invocações com limite de tempo. Este pacote é exigido para forçar limites de tempo dos testes de um modo específico.

Este pacote pode ser instalado usando o seguinte comando:

```
pear install phpunit/PHP_Invoker
```

Este pacote pode ser instalado via Composer adicionando a seguinte dependência `"require-dev"`:

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

`PHPUnit_Selenium`

Integração do Selenium RC para PHPUnit.

Este pacote pode ser instalado via PEAR usando o seguinte comando:

```
pear install phpunit/PHPUnit_Selenium
```

Este pacote pode ser instalado via Composer adicionando a seguinte dependência `"require-dev"`:

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

`PHPUnit_Story` Executor de testes baseados em histórico para Desenvolvimento Guiado por Comportamentos com PHPUnit.

Este pacote pode ser instalado via PEAR usando o seguinte comando:

```
pear install phpunit/PHPUnit_Story
```

Este pacote pode ser instalado via Composer adicionando a seguinte dependência `"require-dev"`:

```
"phpunit/phpunit-story": "*"
```

`PHPUnit_SkeletonGenerator` Ferramenta que gera classes de esqueleto de teste a partir das classes dos códigos de produção e vice-versa.

Este pacote pode ser instalado usando o seguinte comando:

```
pear install phpunit/PHPUnit_SkeletonGenerator
```

`PHPUnit_TestListener_DBUS` Um ouvinte de testes que envia eventos para DBUS.

Este pacote pode ser instalado usando o seguinte comando:

```
pear install phpunit/PHPUnit_TestListener_DBUS
```

`PHPUnit_TestListener_XHProf` Um ouvinte de testes que usa XHProf para perfilar automaticamente o código de teste.

Este pacote pode ser instalado usando o seguinte comando:

```
pear install phpunit/PHPUnit_TestListener_XHProf
```

`PHPUnit_TicketListener_Fogbugz` Um ouvinte de tickets que interage com a API de problemas Fogbugz.

Este pacote pode ser instalado usando o seguinte comando:

```
pear install phpunit/PHPUnit_TicketListener_Fogbugz
```

`PHPUnit_TicketListener_GitHub` Um ouvinte de tickets que interage com a API de problemas do GitHub.

Este pacote pode ser instalado usando o seguinte comando:

```
pear install phpunit/PHPUnit_TicketListener_GitHub
```

`PHPUnit_TicketListener_GoogleCode` Um ouvinte de tickets que interage com a API de problemas do Google Code.

Este pacote pode ser instalado usando o seguinte comando:

```
pear install phpunit/PHPUnit_TicketListener_GoogleCode
```

`PHPUnit_TicketListener_Trac` Um ouvinte de tickets que interage com a API de problemas do Track.

Este pacote pode ser instalado usando o seguinte comando:

```
pear install phpunit/PHPUnit_TicketListener_Trac
```

Atualizando

Esta seção serve como uma coleção de problemas menores de BC pelos quais alguém poderia passar ao atualizar do PHPUnit 3.6 para o PHPUnit 3.7.

A atualização deve tanto ser fácil quanto trabalhar sem qualquer problema, já que foi testada em todos os principais frameworks OpenSource e não houve qualquer problema com eles. Ainda assim, cada projeto é diferente e se você ainda não experimentou uma das versões candidatas a lançamento e enfrentou um problema, este documento pode fornecer alguma ajuda.

Remoção do obsoleto OutputTestCase

A classe `PHPUnit_Extensions_OutputTestCase` foi removida. O PHPUnit 3.6 emitia uma notificação de obsolescência quando era usada. Para ver como a saída pode ser testada agora, veja “Testando Saídas”.

O diretório de trabalho atual será restaurado após cada caso de teste

Se um teste mudasse o diretório de trabalho atual (`cwd`) o PHPUnit incorria em erros quando gerava a cobertura da saída de código. Agora que o cwd é restaurado após cada caso de teste, você pode descobrir se um dos seus testes depende de outro teste alterando o cwd. Algo que não é desejável de qualquer forma, e deveria ser fácil de resolver.

Ouvintes de Teste disparam uma chamada de auto-carregamento

Ao usar ouvintes de testes como descrito em “Ouvintes de Teste”, o PHPUnit ignorava silenciosamente os ouvintes de teste perdidos e era bem difícil para o usuário resolver esses problemas. Agora uma chamada de auto-carregamento será disparada tentando localizar a classe. Se seu auto-carregador produzir um erro quando ele não encontrar um ouvinte de teste, você poderá incorrer em um problema aqui. Remover o ouvinte ou confirmar que ele está sendo carregado em seu `bootstrap.php` vai resolver isso.

Parâmetros para objetos falsos não são mais clonáveis

Anteriormente todos os parâmetros de objetos eram clonados quando falsificados. Isso causava problemas quando testes tentavam verificar se o mesmo objeto foi passado ou não a um método e outro problema com objetos não-clonáveis. Como uma longa e constante requisição de função, este comportamento foi mudado por muitos. Exemplo 10.14, “Testando se um método é chamado uma vez e com o objeto idêntico ao que foi passado” mostra onde a nova implementação pode ser útil. Exemplo 10.15, “Criando um objeto falso com clonagem de parâmetros ativada” mostra como voltar para o comportamento anterior.

`addUncoveredFilesFromWhitelist` foi substituído por `processUncoveredFilesFromWhitelist`

Ao gerar uma cobertura de código e usar `<whitelist addUncoveredFilesFromWhitelist="true">` todos os arquivos da lista-branca eram incluídos pelo PHPUnit. Esse era um problema para pessoas com código executável nesses arquivos. O PHPUnit agora vai escanear o arquivo e descobrir qual código é executável e qual não é, sem incluí-lo. Isto pode levar a diferentes relatórios de cobertura.

Para voltar ao antigo comportamento a configuração `<whitelist`

`processUncoveredFilesFromWhitelist=="true">` pode ser usada. Se você quer o comportamento com PHPUnit 3.6 e 3.7 é possível usar ambas as configurações por um tempo.

Valor padrão de `cacheTokens` mudou para `false`

Desde o PHPUnit `3.7.2` desligamos o cache de arquivos tokenizados por padrão. Ao processar coberturas de código para projetos grandes esse cache consumia muita memória e devido à mudança no comportamento da lista-branca, era problemático para pessoas com bases de código com mais de alguns milhares de classes.

Se seu projeto é menor ou você tem memória suficiente você vai ganhar um benefício em tempo de execução por adicionar `cacheTokens="true"` no seu arquivo `phpunit.xml`. Veja “PHPUnit”.

Capítulo 4. Escrevendo Testes para o PHPUnit

Exemplo 4.1, “Testando operações de vetores com o PHPUnit” 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 `Classe` vão dentro de uma classe `ClasseTest`.

2. `ClasseTest` herda (na maioria das vezes) de `PHPUnit_Framework_TestCase`.

3.

4. Dentro dos métodos de teste, métodos de confirmação como `assertEquals()` (veja “Asserções”) são usados para confirmar que um valor real equivale a um valor esperado.

Exemplo 4.1. Testando operações de vetores com o PHPUnit

```
<?php
class PilhaTest extends PHPUnit_Framework_TestCase
{
public function testPushEPop()
{
$pilha = array();
$this->assertEquals(0, count($pilha));

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

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

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

Dependências de Testes

> Testes Unitários são primeiramente escritos como uma boa prática para ajudar desenvolvedores a identificar e corrigir defeitos, a 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. Às vezes existem dependências implícitas entre métodos de teste, escondidas no cenário de implementação de um teste.
>
> —Adrian Kuhn et. al.

O PHPUnit suporta a declaração explícita de dependências 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 dá como resultado sua unidade sob teste como um valor retornado.
- Um consumidor é um método de teste que depende de um ou mais produtores e seus valores retornados.

Exemplo 4.2, “Usando a anotação `@depends` para expressar dependências” mostra como usar a anotação `@depends` para expressar dependências entre métodos de teste.

Exemplo 4.2. Usando a anotação `@depends` para expressar dependências

```
<?php
class PilhaTest extends PHPUnit_Framework_TestCase
{
public function testVazio()
{
$pilha = array();
$this->assertEmpty($pilha);

return $pilha;
}

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

return $pilha;
}

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

No exemplo acima o primeiro teste, `testVazio()`, 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 `testVazio()` e lhe é passado o resultado do qual ele depende como um argumento. Finalmente, `testPop()` depdende de `testPush()`.

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 Exemplo 4.3, “Explorando as dependências entre os testes”.

Exemplo 4.3. Explorando as dependências entre os testes

```
<?php
class FalhaDependenciaTest extends PHPUnit_Framework_TestCase
{
public function testUm()
{
$this->assertTrue(FALSE);
```

```
}

/**
* @depends testUm
*/
public function testDois()
{
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

FS

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) FalhaDependenciaTest::testUm
Failed asserting that false is true.

/home/sb/FalhaDependenciaTest.php:6

There was 1 skipped test:

1) FalhaDependenciaTest::testDois
This test depends on "FalhaDependenciaTest::testUm" to pass.


FAILURES!
Tests: 1, Assertions: 1, Failures: 1, Skipped: 1.phpunit --verbose FalhaDependenciaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

FS

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) FalhaDependenciaTest::testUm
Failed asserting that false is true.

/home/sb/FalhaDependenciaTest.php:6

There was 1 skipped test:

1) FalhaDependenciaTest::testDois
This test depends on "FalhaDependenciaTest::testUm" 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.

Provedores de Dados

Um método de teste pode aceitar argumentos arbitrários. Esses argumentos devem ser fornecidos por um método provedor de dados (`provedor()` em Exemplo 4.4, “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`.

Um método provedor de dados deve ser `public` e ou retornar um vetor de vetores ou um objeto que implementa a interface `Iterator` e produz 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.

Exemplo 4.4. Usando um provedor de dados que retorna um vetor de vetores

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

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DadosTest.php:9

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.phpunit DadosTest
PHPUnit 3.7.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DadosTest.php:9

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

Exemplo 4.5. Usando um provedor de dados que retorna um objeto Iterador

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

class DadosTest extends PHPUnit_Framework_TestCase
{
/**
* @dataProvider provedor
*/
public function testSoma($a, $b, $c)
{
$this->assertEquals($c, $a + $b);
}

public function provedor()
{
return new ArquivoCsvIterador('dados.csv');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DadosTest.php:11

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.phpunit DadosTest
PHPUnit 3.7.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DadosTest.php:11

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

Exemplo 4.6. A classe ArquivoCsvIterador

```
<?php
class ArquivoCsvIterador implements Iterator {
protected $arquivo;
protected $chave = 0;
protected $atual;
```

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

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

public function rebobinar() {
rebobinar($this->arquivo);
$this->atual = fgetcsv($this->arquivo);
$this->chave = 0;
}

public function valido() {
return !feof($this->arquivo);
}

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

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

public function proximo() {
$this->atual = fgetcsv($this->arquivo);
$this->chave++;
}
}
?>
```

Nota

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.

Nota

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 uma seção de dados. O resultado de um teste que usa provedores de dados não pode ser injetado dentro de um teste dependente.

Nota

Todos os provedores de dados são executados antes da primeira chamada à função `setUp`. Por isso você não pode acessar quaisquer variáveis que criar ali de dentro de um provedor de dados.

Testando Exceções

Exemplo 4.7, “Usando a anotação @expectedException” mostra como usar a anotação `@expectedException` para testar se uma exceção foi lançada dentro do código de teste.

Exemplo 4.7. Usando a anotação @expectedException

```
<?php
```

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExcecaoTest::testExcecao
Expected exception InvalidArgumentException


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ExcecaoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExcecaoTest::testExcecao
Expected exception InvalidArgumentException

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

Adicionalmente, você pode usar `@expectedExceptionMessage` e `@expectedExceptionCode` em combinação com `@expectedException` para testar a mensagem de exceção e o código de exceção como mostrado em Exemplo 4.8, “Usando as anotações @expectedExceptionMessage e @expectedExceptionCode”.

Exemplo 4.8. Usando as anotações @expectedExceptionMessage e @expectedExceptionCode

```
<?php
class ExcecaoTest extends PHPUnit_Framework_TestCase
{
/**
* @expectedException InvalidArgumentException
* @expectedExceptionMessage Mensagem Certa
*/
public function testExcecaoTemMensagemCerta()
{
throw new InvalidArgumentException('Alguma Mensagem', 10);
}

/**
* @expectedException InvalidArgumentException
```

```
* @expectedExceptionCode 20
*/
public function testExcecaoTemCodigoCerto()
{
throw new InvalidArgumentException('Alguma Mensagem', 10);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

FF

Time: 0 seconds, Memory: 3.00Mb

There were 2 failures:

1) ExcecaoTest::testExcecaoTemMensagemCerta
Failed asserting that exception message 'Alguma Mensagem' contains 'Mensagem Certa'.


2) ExcecaoTest::testExcecaoTemCodigoCerto
Failed asserting that expected exception code 20 is equal to 10.


FAILURES!
Tests: 2, Assertions: 4, Failures: 2.phpunit ExcecaoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

FF

Time: 0 seconds, Memory: 3.00Mb

There were 2 failures:

1) ExcecaoTest::testExcecaoTemMensagemCerta
Failed asserting that exception message 'Alguma Mensagem' contains 'Mensagem Certa'.

2) ExcecaoTest::testExcecaoTemCodigoCerto
Failed asserting that expected exception code 20 is equal to 10.

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

Mais exemplos de `@expectedExceptionMessage` e `@expectedExceptionCode` são mostrados em "`@expectedExceptionMessage`" e "`@expectedExceptionCode`" respectivamente.

Alternativamente, você pode usar o método `setExpectedException()` para definir a exceção esperada como mostrado em Exemplo 4.9, "Esperando uma exceção surgir do código de teste".

Exemplo 4.9. Esperando uma exceção surgir do código de teste

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

```
public function testExcecaoTemMensagemCerta()
{
$this->setExpectedException(
'InvalidArgumentException', 'Mensagem Certa'
);
throw new InvalidArgumentException('Alguma Mensagem', 10);
}

public function testExcecaoTemCodigoCerto()
{
$this->setExpectedException(
'InvalidArgumentException', 'Mensagem Certa', 20
);
throw new InvalidArgumentException('A Mensagem Certa', 10);
}
}?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

FFF

Time: 0 seconds, Memory: 3.00Mb

There were 3 failures:

1) ExcecaoTest::testExcecao
Expected exception InvalidArgumentException


2) ExcecaoTest::testExcecaoTemMensagemCerta
Failed asserting that exception message 'Alguma Mensagem' contains 'Mensagem Certa'.


3) ExcecaoTest::testExcecaoTemCodigoCerto
Failed asserting that expected exception code 20 is equal to 10.


FAILURES!
Tests: 3, Assertions: 6, Failures: 3.phpunit ExcecaoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

FFF

Time: 0 seconds, Memory: 3.00Mb

There were 3 failures:

1) ExcecaoTest::testExcecao
Expected exception InvalidArgumentException

2) ExcecaoTest::testExcecaoTemMensagemCerta
Failed asserting that exception message 'Alguma Mensagem' contains 'Mensagem Certa'.

3) ExcecaoTest::testExcecaoTemCodigoCerto
Failed asserting that expected exception code 20 is equal to 10.

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

Tabela 4.1, “Métodos para testar exceções” mostra os métodos fornecidos para testar exceções.

Tabela 4.1. Métodos para testar exceções

Método	Significado
`void setExpectedException(string $nomeExcecao[, string $mensagemExcecao = '', inteiro $codigoExcecao = NULL])`	Define os `$nomeExcecao`, `$mensagemExcecao`, e `$codigoExcecao.` esperados.
`String getExpectedException()`	Retorna o nome da exceção esperada.

Você também pode usar a abordagem mostrada em Exemplo 4.10, “Abordagem alternativa para testar exceções” para testar exceções

Exemplo 4.10. Abordagem alternativa para testar exceções

```
<?php
class ExcecaoTest extends PHPUnit_Framework_TestCase {
public function testExcecao() {
try {
// ... Código que se espera que lance uma exceção ...
}

catch (InvalidArgumentException $esperado) {
return;
}

$this->fail('Uma exceção esperada não foi criada.');
}
}
?>
```

Se o código que se espera que crie uma exceção em Exemplo 4.10, “Abordagem alternativa para testar exceções” não criá-la, a chamada subsequente ao `fail()` vai parar o teste e sinalizar um problema com o teste. Se a exceção esperada aparecer, o bloco `catch` será executado, e o teste terminará com sucesso.

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 Exemplo 4.11, “Esperando um erro PHP usando @expectedException”.

Exemplo 4.11. Esperando um erro PHP usando @expectedException

```
<?php
class ErroEsperadoTest extends PHPUnit_Framework_TestCase
{
/**
* @expectedException PHPUnit_Framework_Error
*/
public function testFalhaInclusao()
{
include 'arquivo_nao_existente.php';
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.
```

```
.

Time: 0 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)phpunit ErroEsperadoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

.

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.

Nota

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 com funções que dependem 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 um `PHPUnit_Framework_Error_Notice`.

Exemplo 4.12. Testando valores retornados do código que utiliza PHP Errors

```
<?php
class SupressaoErroTest extends PHPUnit_Framework_TestCase
{
public function testEscrevendoArquivo() {
$escritor = new EscritorArquivo;
$this->assertFalse(@$escritor->escrever('/nao-pode-escrever/arquivo', 'coisas'));
}
}
class EscritorArquivo
{
public function escrever($arquivo, $conteudo) {
$arquivo = fopen($arquivo, 'w');
if($arquivo == false) {
return false;
}
// ...
}
}

?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

.

Time: 1 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)phpunit SupressaoErroTest
PHPUnit 3.7.0 by Sebastian Bergmann.

.

Time: 1 seconds, Memory: 5.25Mb
```

```
OK (1 test, 1 assertion)
```

Sem a supressão de erros o teste teria relatado uma falha `fopen(/nao-pode-escrever/arquivo): failed to open stream: No such file or directory`.

Testando Saídas

Às vezes você quer assegurar que a execução de um método, por exemplo, gera uma saída esperada (via `echo` ou `print`, por exemplo). A classe `PHPUnit_Framework_TestCase` usa a função Output Buffering [http://www.php.net/manual/en/ref.outcontrol.php] para prover a funcionalidade necessária para isso.

Exemplo 4.13, "Testando a saída de uma função ou método" 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.

Exemplo 4.13. Testando a saída de uma função ou método

```
<?php
class SaidaTest extends PHPUnit_Framework_TestCase
{
public function testEsperadoFooRealFoo()
{
$this->expectOutputString('foo');
print 'foo';
}

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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


FAILURES!
Tests: 2, Assertions: 2, Failures: 1.phpunit SaidaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

.F

Time: 0 seconds, Memory: 5.75Mb
```

```
There was 1 failure:

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

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

Tabela 4.2, “Métodos para testar a saída” mostra os métodos fornecidos para testar saídas.

Tabela 4.2. Métodos para testar a saída

Método	Significado
`void expectOutputRegex(string $expressaoRegular)`	Define a saída que se espera combinar com a `$expressaoRegular`.
`void expectOutputString(string $stringEsperada)`	Define a saída que se espera ser igual a uma `$stringEsperada`.
`booleano setOutputCallback(callable $callback)`	Define um retorno que é usado, por exemplo, para normalizar a saída real.

Nota

Por favor, note que o PHPUnit engole todas as saídas que são emitidas durante a execução de um teste. Para ser mais exato, um teste que emite uma saída vai falhar.

Asserções

Esta seção lista os vários métodos de asserção (assertion) disponíveis.

`assertArrayHasKey()`

```
assertArrayHasKey(misto $chave, vetor $vetor[, string $mensagem =
'']);
```

Relata um erro identificado por `$mensagem` se `$vetor` não tiver a `$chave`.

`assertArrayNotHasKey()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.14. Uso de assertArrayHasKey()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.
```

```
F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) VetorTemChaveTest::testFalha
Failed asserting that an array has the key 'foo'.

/home/sb/VetorTemChaveTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit VetorTemChaveTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) VetorTemChaveTest::testFalha
Failed asserting that an array has the key 'foo'.

/home/sb/VetorTemChaveTest.php:6

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

assertClassHasAttribute()

`assertClassHasAttribute(string $nomeAtributo, string $nomeClasse[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$nomeClasse::nomeAtributo` não existir.

`assertClassNotHasAttribute()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.15. Uso de assertClassHasAttribute()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ClasseTemAtributoTest::testFalha
Failed asserting that class "stdClass" has attribute "foo".

/home/sb/ClasseTemAtributoTest.php:6
```

```
FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ClasseTemAtributoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ClasseTemAtributoTest::testFalha
Failed asserting that class "stdClass" has attribute "foo".

/home/sb/ClasseTemAtributoTest.php:6

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

assertClassHasStaticAttribute()

`assertClassHasStaticAttribute(string $nomeAtributo, string $nomeClasse[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$nomeClasse::nomeAtributo` não existir.

`assertClassNotHasStaticAttribute()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.16. Uso de assertClassHasStaticAttribute()

```php
<?php
class ClasseTemAtributoEstaticoTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertClassHasStaticAttribute('foo', 'stdClass');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ClasseTemAtributoEstaticoTest::testFalha
Failed asserting that class "stdClass" has static attribute "foo".

/home/sb/ClasseTemAtributoEstaticoTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ClasseTemAtributoEstaticoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb
```

```
There was 1 failure:

1) ClasseTemAtributoEstaticoTest::testFalha
Failed asserting that class "stdClass" has static attribute "foo".

/home/sb/ClasseTemAtributoEstaticoTest.php:6

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

assertContains()

`assertContains(misto $agulha, Iterador|vetor $bateria[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$agulha` não for um elemento de `$bateria`.

`assertNotContains()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeContains()` e `assertAttributeNotContains()` são empacotadores de conveniência que usam um atributo `public`, `protected`, ou `private` de uma classe ou objeto como a bateria (vetor que é um conjunto de pilhas).

Exemplo 4.17. Uso de assertContains()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContemTest::testFalha
Failed asserting that an array contains 4.

/home/sb/ContemTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ContemTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContemTest::testFalha
Failed asserting that an array contains 4.

/home/sb/ContemTest.php:6
```

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

`assertContains(string $agulha, string $bateria[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$agulha` não for uma substring de `$bateria`.

Exemplo 4.18. Uso de assertContains()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContemTest::testFalha
Failed asserting that 'foobar' contains "baz".

/home/sb/ContemTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ContemTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContemTest::testFalha
Failed asserting that 'foobar' contains "baz".

/home/sb/ContemTest.php:6

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

assertContainsOnly()

`assertContainsOnly(string $tipo, Iterador|vetor $bateria[, booleano $ehTipoNativo = NULL, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$bateria` não contiver apenas variáveis do tipo `$tipo`.

`$ehTipoNativo` é uma bandeira usada para indicar se `$tipo` é um tipo nativo do PHP ou não.

`assertNotContainsOnly()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeContainsOnly()` e `assertAttributeNotContainsOnly()` são empacotadores de conveniência que usam um atributo `public`, `protected`, ou `private` de uma classe ou objeto como o valor real.

Exemplo 4.19. Uso de assertContainsOnly()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ContemApenasTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ContemApenasTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ContemApenasTest.php:6

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

assertContainsOnlyInstancesOf()

```
assertContainsOnlyInstancesOf(string $nomeclasse, Traversable|vetor
$bateria[, string $mensagem = ''])
```

Relata um erro identificado por `$mensagem` se `$bateria` não contiver apenas instâncias da classe `$nomeclasse`.

Exemplo 4.20. Uso de assertContainsOnlyInstancesOf()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ContemApenasInstanciasDeTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ContemApenasInstanciasDeTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ContemApenasInstanciasDeTest.php:6

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

assertCount()

`assertCount($contaEsperada, $bateria[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o número de elementos em `$bateria` não for `$contaEsperada`.

`assertNotCount()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.21. Uso de assertCount()

```
<?php
class ContaTest extends PHPUnit_Framework_TestCase
{
```

```
public function testFalha()
{
$this->assertCount(0, array('foo'));
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ContaTest::testFalha
Failed asserting that actual size 1 matches expected size 0.

/home/sb/ContaTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ContaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ContaTest::testFalha
Failed asserting that actual size 1 matches expected size 0.

/home/sb/ContaTest.php:6

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

assertEmpty()

`assertEmpty(misto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$real` não estiver vazio.

`assertNotEmpty()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeEmpty()` e `assertAttributeNotEmpty()` são empacotadores de conveniência que podem ser aplicados a um atributo `public`, `protected`, ou `private` de uma classe ou objeto.

Exemplo 4.22. Uso de assertEmpty()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) VazioTest::testFalha
Failed asserting that an array is empty.

/home/sb/VazioTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit VazioTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) VazioTest::testFalha
Failed asserting that an array is empty.

/home/sb/VazioTest.php:6

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

assertEqualXMLStructure()

`assertEqualXMLStructure(DOMElement $elementoEsperado, DOMElement $elementoReal[, booleano $verificarAtributos = FALSE, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se a Estrutura XML do DOMElement em `$elementoReal` não é igual à estrutura XML do DOMElement em `$elementoEsperado`.

Exemplo 4.23. Uso de assertEqualXMLStructure()

```php
<?php
class IgualaEstruturaXMLTest extends PHPUnit_Framework_TestCase
{
public function testFalhaComNomesDeNosDiferentes()
{
$esperado = new DOMElement('foo');
$real = new DOMElement('bar');

$this->assertEqualXMLStructure($esperado, $real);
}

public function testFalhaComAtributosDeNosDiferentes()
{
$esperado = new DOMDocument;
$esperado->loadXML('<foo bar="true" />');

$real = new DOMDocument;
$real->loadXML('<foo/>');
```

```
$this->assertEqualXMLStructure(
$esperado->firstChild, $real->firstChild, TRUE
);
}

public function testFalhaComContagemDeFilhosDiferente()
{
$esperado = new DOMDocument;
$esperado->loadXML('<foo><bar/><bar/><bar/></foo>');

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

$this->assertEqualXMLStructure(
$esperado->firstChild, $real->firstChild
);
}

public function testFalhaComFilhosDiferentes()
{
$esperado = new DOMDocument;
$esperado->loadXML('<foo><bar/><bar/><bar/></foo>');

$real = new DOMDocument;
$real->loadXML('<foo><baz/><baz/><baz/></foo>');

$this->assertEqualXMLStructure(
$esperado->firstChild, $real->firstChild
);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.75Mb

There were 4 failures:

1) IgualaEstruturaXMLTest::testFalhaComNomesDeNosDiferentes
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'foo'
+'bar'

/home/sb/IgualaEstruturaXMLTest.php:9

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

/home/sb/IgualaEstruturaXMLTest.php:22

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

/home/sb/IgualaEstruturaXMLTest.php:35
```

```
4) IgualaEstruturaXMLTest::testFalhaComFilhosDiferentes
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'bar'
+'baz'

/home/sb/IgualaEstruturaXMLTest.php:48

FAILURES!
Tests: 4, Assertions: 8, Failures: 4.phpunit IgualaEstruturaXMLTest
PHPUnit 3.7.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.75Mb

There were 4 failures:

1) IgualaEstruturaXMLTest::testFalhaComNomesDeNosDiferentes
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'foo'
+'bar'

/home/sb/IgualaEstruturaXMLTest.php:9

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

/home/sb/IgualaEstruturaXMLTest.php:22

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

/home/sb/IgualaEstruturaXMLTest.php:35

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

/home/sb/IgualaEstruturaXMLTest.php:48

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

assertEquals()

`assertEquals(misto $esperado, misto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se as duas variáveis `$esperado` e `$real` não forem iguais.

`assertNotEquals()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeEquals()` e `assertAttributeNotEquals()` são empacotadores de conveniência que usam um atributo `public`, `protected`, ou `private` de uma classe ou objeto como valor real.

Exemplo 4.24. Uso de assertEquals()

```
<?php
class IgualaTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertEquals(1, 0);
}

public function testFalha2()
{
$this->assertEquals('bar', 'baz');
}

public function testFalha3()
{
$this->assertEquals("foo\nbar\nbaz\n", "foo\nbah\nbaz\n");
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

FFF

Time: 0 seconds, Memory: 5.25Mb

There were 3 failures:

1) IgualaTest::testFalha
Failed asserting that 0 matches expected 1.

/home/sb/IgualaTest.php:6

2) IgualaTest::testFalha2
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'bar'
+'baz'

/home/sb/IgualaTest.php:11

3) IgualaTest::testFalha3
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
 'foo
-bar
+bah
 baz
 '

/home/sb/IgualaTest.php:16
```

```
FAILURES!
Tests: 3, Assertions: 3, Failures: 3.phpunit IgualaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

FFF

Time: 0 seconds, Memory: 5.25Mb

There were 3 failures:

1) IgualaTest::testFalha
Failed asserting that 0 matches expected 1.

/home/sb/IgualaTest.php:6

2) IgualaTest::testFalha2
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'bar'
+'baz'

/home/sb/IgualaTest.php:11

3) IgualaTest::testFalha3
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
 'foo
-bar
+bah
 baz
 '

/home/sb/IgualaTest.php:16

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

Veja abaixo comparações mais especializadas são usadas para tipos específicos de argumentos para `$esperado` e `$real`.

`assertEquals(float $esperado, float $real[, string $mensagem = '', float $delta = 0])`

Relata um erro identificado por `$mensagem` se os dois ponto-flutuantes `$esperado` e `$real` não estiverem contidos no `$delta` de cada um.

Por favor, leia "O que cada cientista da computação deveria saber sobre aritmética de ponto-flutuante [http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html]" para entender porque `$delta` é necessário.

Exemplo 4.25. Uso de assertEquals() com ponto-flutuantes

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

```
}

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) IgualaTest::testFalha
Failed asserting that 1.1 matches expected 1.0.

/home/sb/IgualaTest.php:11

FAILURES!
Tests: 2, Assertions: 2, Failures: 1.phpunit IgualaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) IgualaTest::testFalha
Failed asserting that 1.1 matches expected 1.0.

/home/sb/IgualaTest.php:11

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

`assertEquals(DOMDocument $esperado, DOMDocument $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o canônico não-comentado dos documentos XML representados pelos dois objetos DOMDocument `$esperado` e `$real` não forem iguais.

Exemplo 4.26. Uso de assertEquals() com objetos DOMDocument

```
<?php
class IgualaTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$esperado = new DOMDocument;
$esperado->loadXML('<foo><bar/></foo>');

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

$this->assertEquals($esperado, $real);
}
```

```
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) IgualaTest::testFalha
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
<?xml version="1.0"?>
-<foo>
- <bar/>
-</foo>
+<bar>
+ <foo/>
+</bar>

/home/sb/IgualaTest.php:12

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit IgualaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) IgualaTest::testFalha
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
<?xml version="1.0"?>
-<foo>
- <bar/>
-</foo>
+<bar>
+ <foo/>
+</bar>

/home/sb/IgualaTest.php:12

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

`assertEquals(objeto $esperado, objeto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se os dois objetos `$esperado` e `$real` não tiverem os mesmos valores de atributos.

Exemplo 4.27. Uso de assertEquals() com objetos

```
<?php
```

```
class IgualaTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$esperado = new stdClass;
$esperado->foo = 'foo';
$esperado->bar = 'bar';

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

$this->assertEquals($esperado, $real);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) IgualaTest::testFalha
Failed asserting that two objects are equal.
--- Expected
+++ Actual
@@ @@
stdClass Object (
- 'foo' => 'foo'
- 'bar' => 'bar'
+ 'foo' => 'bar'
+ 'baz' => 'bar'
)

/home/sb/IgualaTest.php:14

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit IgualaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) IgualaTest::testFalha
Failed asserting that two objects are equal.
--- Expected
+++ Actual
@@ @@
stdClass Object (
- 'foo' => 'foo'
- 'bar' => 'bar'
+ 'foo' => 'bar'
+ 'baz' => 'bar'
)

/home/sb/IgualaTest.php:14
```

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

`assertEquals(vetor $esperado, vetor $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se os dois vetores `$esperado` e `$real` não forem iguais.

Exemplo 4.28. Uso de assertEquals() com vetores

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

/home/sb/IgualaTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit IgualaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

```
+ 1 => 'c'
+ 2 => 'd'
)

/home/sb/IgualaTest.php:6

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

assertFalse()

```
assertFalse(booleano $condicao[, string $mensagem = ''])
```

Relata um erro identificado por `$mensagem` se `$condicao` for `TRUE`.

Exemplo 4.29. Uso de assertFalse()

```
<?php
class FalsoTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertFalse(TRUE);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) FalsoTest::testFalha
Failed asserting that true is false.

/home/sb/FalsoTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit FalsoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) FalsoTest::testFalha
Failed asserting that true is false.

/home/sb/FalsoTest.php:6

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

assertFileEquals()

`assertFileEquals(string $esperado, string $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o arquivo especificado por `$esperado` não tiver o mesmo conteúdo que o arquivo especificado por `$real`.

`assertFileNotEquals()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.30. Uso de assertFileEquals()

```
<?php
class ArquivoIgualaTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertFileEquals('/home/sb/esperado', '/home/sb/real');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArquivoIgualaTest::testFalha
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'esperado
+'real
 '

/home/sb/ArquivoIgualaTest.php:6

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.phpunit ArquivoIgualaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArquivoIgualaTest::testFalha
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'esperado
+'real
 '

/home/sb/ArquivoIgualaTest.php:6

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

assertFileExists()

`assertFileExists(string $nomearquivo[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o arquivo especificado por `$nomearquivo` não existir.

`assertFileNotExists()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.31. Uso de assertFileExists()

```
<?php
class ArquivoExisteTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertFileExists('/caminho/para/arquivo');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ArquivoExisteTest::testFalha
Failed asserting that file "/caminho/para/arquivo" exists.

/home/sb/ArquivoExisteTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ArquivoExisteTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ArquivoExisteTest::testFalha
Failed asserting that file "/caminho/para/arquivo" exists.

/home/sb/ArquivoExisteTest.php:6

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

assertGreaterThan()

`assertGreaterThan(misto $esperado, misto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o valor de `$real` não for maior que o valor de `$esperado`.

`assertAttributeGreaterThan()` é um empacotador de conveniência que usa um atributo `public`, `protected`, ou `private` de uma classe ou objeto como valor real.

Exemplo 4.32. Uso de assertGreaterThan()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) MaiorQueTest::testFalha
Failed asserting that 1 is greater than 2.

/home/sb/MaiorQueTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit MaiorQueTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) MaiorQueTest::testFalha
Failed asserting that 1 is greater than 2.

/home/sb/MaiorQueTest.php:6

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

assertGreaterThanOrEqual()

`assertGreaterThanOrEqual(misto $esperado, misto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o valor de `$real` não for maior ou igual ao valor de `$esperado`.

`assertAttributeGreaterThanOrEqual()` é um empacotador de conveniência que usa `public`, `protected`, ou `private` de uma classe ou objeto como o valor real.

Exemplo 4.33. Uso de assertGreaterThanOrEqual()

```
<?php
```

```
class MaiorOuIgualTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertGreaterThanOrEqual(2, 1);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) MaiorOuIgualTest::testFalha
Failed asserting that 1 is equal to 2 or is greater than 2.

/home/sb/MaiorOuIgualTest.php:6

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.phpunit MaiorOuIgualTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) MaiorOuIgualTest::testFalha
Failed asserting that 1 is equal to 2 or is greater than 2.

/home/sb/MaiorOuIgualTest.php:6

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

assertInstanceOf()

`assertInstanceOf($esperado, $real[, $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$real` não for uma instância de `$esperado`.

`assertNotInstanceOf()` é o inverso desta asserção e recebe os mesmos argumentos..

`assertAttributeInstanceOf()` e `assertAttributeNotInstanceOf()` são empacotadores de conveniência que podem ser aplicados a atributos `public`, `protected`, ou `private` de uma classe ou objeto.

Exemplo 4.34. Uso de assertInstanceOf()

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

```
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/InstanciaDeTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit InstanciaDeTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/InstanciaDeTest.php:6

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

assertInternalType()

`assertInternalType($esperado, $real[, $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$real` não for do tipo `$esperado`.

`assertNotInternalType()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeInternalType()` e `assertAttributeNotInternalType()` são empacotadores de conveniência que podem ser aplicados a atributos `public`, `protected`, ou `private` de uma classe ou objeto.

Exemplo 4.35. Uso de assertInternalType()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.
```

```
F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) TipoInternoTest::testFalha
Failed asserting that 42 is of type "string".

/home/sb/TipoInternoTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit TipoInternoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) TipoInternoTest::testFalha
Failed asserting that 42 is of type "string".

/home/sb/TipoInternoTest.php:6

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

assertJsonFileEqualsJsonFile()

`assertJsonFileEqualsJsonFile(misto $arquivoEsperado, misto $arquivoReal[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o valor de `$arquivoReal` equivaler ao valor de `$arquivoEsperado`.

Exemplo 4.36. Uso de assertJsonFileEqualsJsonFile()

```
<?php
class ArquivoJsonIgualaArquivoJsonTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertJsonFileEqualsJsonFile(
'caminho/para/arquivo/esperado', 'caminho/para/arquivo/real');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

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

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.phpunit ArquivoJsonIgualaArquivoJsonTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ArquivoJsonIgualaArquivoJsonTest.php:5

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

assertJsonStringEqualsJsonFile()

`assertJsonStringEqualsJsonFile(misto $arquivoEsperado, misto $jsonReal[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o valor de `$jsonReal` equivaler ao valor de `$arquivoEsperado`.

Exemplo 4.37. Uso de assertJsonStringEqualsJsonFile()

```
<?php
class StringJsonIgualaArquivoJsonTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertJsonStringEqualsJsonFile(
'caminho/para/arquivo/ambiente', json_encode(array("Mascott" => "ux"));
}
}
?>
```

```


PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonStringEqualsJsonFile::testFalha
Failed asserting that '{"Mascott":"ux"}' matches JSON string "{"Mascott":"Tux"}".

/home/sb/StringJsonIgualaArquivoJsonTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.phpunit StringJsonIgualaArquivoJsonTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb
```

```
There was 1 failure:

1) JsonStringEqualsJsonFile::testFalha
Failed asserting that '{"Mascott":"ux"}' matches JSON string "{"Mascott":"Tux"}".

/home/sb/StringJsonIgualaArquivoJsonTest.php:5

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

assertJsonStringEqualsJsonString()

`assertJsonStringEqualsJsonString(misto $jsonEsperado, misto $jsonReal[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o valor de `$jsonReal` equivaler ao valor de `$jsonEsperado`.

Exemplo 4.38. Uso de assertJsonStringEqualsJsonString()

```php
<?php
class StringJsonIgualaStringJsonTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertJsonStringEqualsJsonString(
json_encode(array("Mascott" => "Tux"), json_encode(array("Mascott" => "ux"));
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringJsonIgualaStringJsonTest::testFalha
Failed asserting that two objects are equal.
--- Expected
+++ Actual
@@ @@
 stdClass Object (
 - 'Mascott' => 'Tux'
 + 'Mascott' => 'ux'
)

/home/sb/StringJsonIgualaStringJsonTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.phpunit StringJsonIgualaStringJsonTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:
```

```
1) StringJsonIgualaStringJsonTest::testFalha
Failed asserting that two objects are equal.
--- Expected
+++ Actual
@@ @@
 stdClass Object (
 - 'Mascott' => 'Tux'
 + 'Mascott' => 'ux'
)

/home/sb/StringJsonIgualaStringJsonTest.php:5

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

assertLessThan()

`assertLessThan(misto $esperado, misto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o valor de `$real` não for menor que o valor de `$esperado`.

`assertAttributeLessThan()` é um empacotador de conveniência que usa um atributo `public`, `protected`, ou `private` de uma classe ou objeto como valor real.

Exemplo 4.39. Uso de assertLessThan()

```php
<?php
class MenorQueTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertLessThan(1, 2);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) MenorQueTest::testFalha
Failed asserting that 2 is less than 1.

/home/sb/MenorQueTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit MenorQueTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:
```

```
1) MenorQueTest::testFalha
Failed asserting that 2 is less than 1.

/home/sb/MenorQueTest.php:6

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

assertLessThanOrEqual()

`assertLessThanOrEqual(misto $esperado, misto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o valor de `$real` não for menor ou igual ao valor de `$esperado`.

`assertAttributeLessThanOrEqual()` é um empacotador de conveniência que usa um atributo `public`, `protected`, ou `private` de uma classe ou objeto como valor real.

Exemplo 4.40. Uso de assertLessThanOrEqual()

```
<?php
class MenorOuIgualTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertLessThanOrEqual(1, 2);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) MenorOuIgualTest::testFalha
Failed asserting that 2 is equal to 1 or is less than 1.

/home/sb/LessThanOrEqualTest.php:6

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.phpunit MenorOuIgualTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) MenorOuIgualTest::testFalha
Failed asserting that 2 is equal to 1 or is less than 1.

/home/sb/LessThanOrEqualTest.php:6

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

assertNull()

`assertNull(misto $variavel[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$variavel` não for `NULL`.

`assertNotNull()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.41. Uso de assertNull()

```
<?php
class NuloTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertNull('foo');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) NuloTest::testFalha
Failed asserting that 'foo' is null.

/home/sb/NuloTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit NaoNuloTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) NuloTest::testFalha
Failed asserting that 'foo' is null.

/home/sb/NuloTest.php:6

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

assertObjectHasAttribute()

`assertObjectHasAttribute(string $nomeAtributo, objeto $objeto[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$objeto->nomeAtributo` não existir.

`assertObjectNotHasAttribute()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.42. Uso de assertObjectHasAttribute()

```
<?php
class ObjetoTemAtributoTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertObjectHasAttribute('foo', new stdClass);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ObjetoTemAtributoTest::testFalha
Failed asserting that object of class "stdClass" has attribute "foo".

/home/sb/ObjetoTemAtributoTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ObjetoTemAtributoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ObjetoTemAtributoTest::testFalha
Failed asserting that object of class "stdClass" has attribute "foo".

/home/sb/ObjetoTemAtributoTest.php:6

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

assertRegExp()

`assertRegExp(string $padrao, string $string[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se `$string` não combinar com a expressão regular `$padrao`.

`assertNotRegExp()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.43. Uso de assertRegExp()

```
<?php
class ExpressaoRegularTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertRegExp('/foo/', 'bar');
}
}
```

```
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ExpressaoRegularTest::testFalha
Failed asserting that 'bar' matches PCRE pattern "/foo/".

/home/sb/ExpressaoRegularTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ExpressaoRegularTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ExpressaoRegularTest::testFalha
Failed asserting that 'bar' matches PCRE pattern "/foo/".

/home/sb/ExpressaoRegularTest.php:6

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

assertStringMatchesFormat()

`assertStringMatchesFormat(string $formato, string $string[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se a `$string` não combinar com a string `$formato`.

`assertStringNotMatchesFormat()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.44. Uso de assertStringMatchesFormat()

```
<?php
class StringEquivaleFormatoTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertStringMatchesFormat('%i', 'foo');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb
```

```
There was 1 failure:

1) StringEquivaleFormatoTest::testFalha
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+$/s".

/home/sb/StringEquivaleFormatoTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit StringEquivaleFormatoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringEquivaleFormatoTest::testFalha
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+$/s".

/home/sb/StringEquivaleFormatoTest.php:6

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

A string de formato pode conter os seguintes espaços reservados:

- `%e`: Representa um separador de diretório, por exemplo `/` no Linux.
- `%s`: Um ou mais de qualquer coisa (caractere ou espaço em branco) exceto no último caractere da linha.
- `%S`: Zero ou mais de qualquer coisa (caractere ou espaço em branco) exceto no último caractere da linha.
- `%a`: Um ou mais de qualquer coisa (caractere ou espaço em branco) inclusive no último caractere da linha.
- `%A`: Zero ou mais de qualquer coisa (caractere ou espaço em branco) inclusive no último caractere da linha.
- `%w`: Zero ou mais caracteres de espaço em branco.
- `%i`: Um valor inteiro sinalizado, por exemplo `+3142`, `-3142`.
- `%d`: Um valor inteiro não-sinalizado, por exemplo `123456`.
- `%x`: Um ou mais caracteres hexadecimais. Isto é, caracteres dentro de `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 caractere único de qualquer tipo.

assertStringMatchesFormatFile()

`assertStringMatchesFormatFile(string $formatoArquivo, string $string[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se a `$string` não combinar com os conteúdos de `$formatoArquivo`.

`assertStringNotMatchesFormatFile()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.45. Uso de assertStringMatchesFormatFile()

```
<?php
class StringEquivaleFormatoArquivoTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertStringMatchesFormatFile('/caminho/para/esperado.txt', 'foo');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringEquivaleFormatoArquivoTest::testFalha
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+
$/s".

/home/sb/StringEquivaleFormatoArquivoTest.php:6

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.phpunit StringEquivaleFormatoArquivoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringEquivaleFormatoArquivoTest::testFalha
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+
$/s".

/home/sb/StringEquivaleFormatoArquivoTest.php:6

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

assertSame()

`assertSame(misto $esperado, misto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se as duas variáveis `$esperado` e `$real` não tiverem o mesmo tipo e valor.

`assertNotSame()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeSame()` e `assertAttributeNotSame()` são empacotadores de conveniência que usam um atributo `public`, `protected`, ou `private` de uma classe ou objeto como valor real.

Exemplo 4.46. Uso de assertSame()

```
<?php
```

```
class IdenticoTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertSame('2204', 2204);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) IdenticoTest::testFalha
Failed asserting that 2204 is identical to '2204'.

/home/sb/IdenticoTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit IdenticoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) IdenticoTest::testFalha
Failed asserting that 2204 is identical to '2204'.

/home/sb/IdenticoTest.php:6

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

`assertSame(objeto $esperado, objeto $real[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se as duas variáveis `$esperado` e `$real` não referenciarem ao mesmo objeto.

Exemplo 4.47. Uso de assertSame() with objects

```
<?php
class IdenticoTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertSame(new stdClass, new stdClass);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F
```

```
Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) IdenticoTest::testFalha
Failed asserting that two variables reference the same object.

/home/sb/IdenticoTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit IdenticoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) IdenticoTest::testFalha
Failed asserting that two variables reference the same object.

/home/sb/IdenticoTest.php:6

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

assertSelectCount()

`assertSelectCount(vetor $seletor, inteiro $conta, misto $real[, string $mensagem = '', booleano $ehHtml = TRUE])`

Relata um erro identificado por `$mensagem` se o seletor CSS `$seletor` não combinar com `$conta` elementos no DOMNode `$real`.

`$conta` pode ser um dos seguintes tipos:

- `booleano`: assegura para a presença de elementos que batam com seletor (TRUE) ou falta de elementos (FALSE).
- `integer`: assegura a contagem de elementos.
- `array`: assegura que a contagem está dentro do âmbito especificada ao usar <, >, <=, e >= como chaves.

Exemplo 4.48. Uso de assertSelectCount()

```
<?php
class SelecionaContaTest extends PHPUnit_Framework_TestCase
{
protected function setUp()
{
$this->xml = new DomDocument;
$this->xml->loadXML('<foo><bar/><bar/><bar/></foo>');
}

public function testAusenciaFalha()
{
$this->assertSelectCount('foo bar', FALSE, $this->xml);
}

public function testPresencaFalha()
{
```

```
$this->assertSelectCount('foo baz', TRUE, $this->xml);
}

public function testContaExataFalha()
{
$this->assertSelectCount('foo bar', 5, $this->xml);
}

public function testAmbitoFalha()
{
$this->assertSelectCount('foo bar', array('>'=>6, '<'=>8), $this->xml);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelecionaContaTest::testAusenciaFalha
Failed asserting that true is false.

/home/sb/SelecionaContaTest.php:12

2) SelecionaContaTest::testPresencaFalha
Failed asserting that false is true.

/home/sb/SelecionaContaTest.php:17

3) SelecionaContaTest::testContaExataFalha
Failed asserting that 3 matches expected 5.

/home/sb/SelecionaContaTest.php:22

4) SelecionaContaTest::testAmbitoFalha
Failed asserting that false is true.

/home/sb/SelecionaContaTest.php:27

FAILURES!
Tests: 4, Assertions: 4, Failures: 4.phpunit SelecionaContaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelecionaContaTest::testAusenciaFalha
Failed asserting that true is false.

/home/sb/SelecionaContaTest.php:12

2) SelecionaContaTest::testPresencaFalha
Failed asserting that false is true.

/home/sb/SelecionaContaTest.php:17
```

```
3) SelecionaContaTest::testContaExataFalha
Failed asserting that 3 matches expected 5.

/home/sb/SelecionaContaTest.php:22

4) SelecionaContaTest::testAmbitoFalha
Failed asserting that false is true.

/home/sb/SelecionaContaTest.php:27

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

assertSelectEquals()

`assertSelectEquals(vetor $seletor, string $conteudo, inteiro $conta, misto $real[, string $mensagem = '', booleano $ehHtml = TRUE])`

Relata um erro identificado por `$mensagem` se o seletor CSS `$seletor` não combinar com `$conta` elementos in the DOMNode `$real` com o valor `$conteudo`.

`$conta` pode ser de um dos seguintes tipos:

- `boolean`: assegura para a presença de elementos que combinam com o seletor (`TRUE`) ou falta de elementos (`FALSE`).
- `integer`: assegura a contagem de elementos.
- `array`: assegura que a contagem está dentro do âmbito especificado ao usar <, >, <=, e >= como chaves.

Exemplo 4.49. Uso de assertSelectEquals()

```php
<?php
class SelecionaIgualaTest extends PHPUnit_Framework_TestCase
{
protected function setUp()
{
$this->xml = new DomDocument;
$this->xml->loadXML('<foo><bar>Baz</bar><bar>Baz</bar></foo>');
}

public function testAusenciaFalha()
{
$this->assertSelectEquals('foo bar', 'Baz', FALSE, $this->xml);
}

public function testPresencaFalha()
{
$this->assertSelectEquals('foo bar', 'Bat', TRUE, $this->xml);
}

public function testContaExataFalha()
{
$this->assertSelectEquals('foo bar', 'Baz', 5, $this->xml);
}

public function testAmbitoFalha()
{
$this->assertSelectEquals('foo bar', 'Baz', array('>'=>6, '<'=>8), $this->xml);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelecionaIgualaTest::testAusenciaFalha
Failed asserting that true is false.

/home/sb/SelecionaIgualaTest.php:12

2) SelecionaIgualaTest::testPresencaFalha
Failed asserting that false is true.

/home/sb/SelecionaIgualaTest.php:17

3) SelecionaIgualaTest::testContaExataFalha
Failed asserting that 2 matches expected 5.

/home/sb/SelecionaIgualaTest.php:22

4) SelecionaIgualaTest::testAmbitoFalha
Failed asserting that false is true.

/home/sb/SelecionaIgualaTest.php:27

FAILURES!
Tests: 4, Assertions: 4, Failures: 4.phpunit SelecionaIgualaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelecionaIgualaTest::testAusenciaFalha
Failed asserting that true is false.

/home/sb/SelecionaIgualaTest.php:12

2) SelecionaIgualaTest::testPresencaFalha
Failed asserting that false is true.

/home/sb/SelecionaIgualaTest.php:17

3) SelecionaIgualaTest::testContaExataFalha
Failed asserting that 2 matches expected 5.

/home/sb/SelecionaIgualaTest.php:22

4) SelecionaIgualaTest::testAmbitoFalha
Failed asserting that false is true.

/home/sb/SelecionaIgualaTest.php:27

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

assertSelectRegExp()

```
assertSelectRegExp(vetor $seletor, string $padrao, inteiro $conta,
misto $real[, string $mensagem = '', booleano $ehHtml = TRUE])
```

Relata um erro identificado por `$mensagem` se o seletor CSS `$seletor` não combinar com `$conta` elementos no DOMNode `$real` com um valor que combine com `$padrao`.

`$conta` pode ser de um dos seguintes tipos:

- `boolean`: assegura para a presença de elementos que combinem com o seletor (`TRUE`) ou falta de elementos (`FALSE`).
- `integer`: assegura a contagem de elementos.
- `array`: assegura que a contagem está dentro do âmbito especificada ao usar `<`, `>`, `<=`, e `>=` como chaves.

Exemplo 4.50. Uso de assertSelectRegExp()

```
<?php
class SelecionaExpressaoRegularTest extends PHPUnit_Framework_TestCase
{
protected function setUp()
{
$this->xml = new DomDocument;
$this->xml->loadXML('<foo><bar>Baz</bar><bar>Baz</bar></foo>');
}

public function testAusenciaFalha()
{
$this->assertSelectRegExp('foo bar', '/Ba.*/', FALSE, $this->xml);
}

public function testPresencaFalha()
{
$this->assertSelectRegExp('foo bar', '/B[oe]z]/', TRUE, $this->xml);
}

public function testContaExataFalha()
{
$this->assertSelectRegExp('foo bar', '/Ba.*/', 5, $this->xml);
}

public function testAmbitoFalha()
{
$this->assertSelectRegExp('foo bar', '/Ba.*/', array('>'=>6, '<'=>8), $this->xml);
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelecionaExpressaoRegularTest::testAusenciaFalha
Failed asserting that true is false.

/home/sb/SelecionaExpressaoRegularTest.php:12
```

```
2) SelecionaExpressaoRegularTest::testPresencaFalha
Failed asserting that false is true.

/home/sb/SelecionaExpressaoRegularTest.php:17

3) SelecionaExpressaoRegularTest::testContaExataFalha
Failed asserting that 2 matches expected 5.

/home/sb/SelecionaExpressaoRegularTest.php:22

4) SelecionaExpressaoRegularTest::testAmbitoFalha
Failed asserting that false is true.

/home/sb/SelecionaExpressaoRegularTest.php:27

FAILURES!
Tests: 4, Assertions: 4, Failures: 4.phpunit SelecionaExpressaoRegularTest
PHPUnit 3.7.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelecionaExpressaoRegularTest::testAusenciaFalha
Failed asserting that true is false.

/home/sb/SelecionaExpressaoRegularTest.php:12

2) SelecionaExpressaoRegularTest::testPresencaFalha
Failed asserting that false is true.

/home/sb/SelecionaExpressaoRegularTest.php:17

3) SelecionaExpressaoRegularTest::testContaExataFalha
Failed asserting that 2 matches expected 5.

/home/sb/SelecionaExpressaoRegularTest.php:22

4) SelecionaExpressaoRegularTest::testAmbitoFalha
Failed asserting that false is true.

/home/sb/SelecionaExpressaoRegularTest.php:27

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

assertStringEndsWith()

`assertStringEndsWith(string $sufixo, string $string[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se a `$string` não terminar com `$sufixo`.

`assertStringEndsNotWith()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.51. Uso de assertStringEndsWith()

```
<?php
class StringTerminaComTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
```

```
{
$this->assertStringEndsWith('sufixo', 'foo');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 1 second, Memory: 5.00Mb

There was 1 failure:

1) StringTerminaComTest::testFalha
Failed asserting that 'foo' ends with "sufixo".

/home/sb/StringTerminaComTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit StringTerminaComTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 1 second, Memory: 5.00Mb

There was 1 failure:

1) StringTerminaComTest::testFalha
Failed asserting that 'foo' ends with "sufixo".

/home/sb/StringTerminaComTest.php:6

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

assertStringEqualsFile()

`assertStringEqualsFile(string $arquivoEsperado, string $stringReal[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o arquivo especificado por `$arquivoEsperado` não tiver `$stringReal` como seu conteúdo.

`assertStringNotEqualsFile()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.52. Uso de assertStringEqualsFile()

```
<?php
class StringIgualaArquivoTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertStringEqualsFile('/home/sb/esperado', 'real');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.
```

```
F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) StringIgualaArquivoTest::testFalha
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'esperado
-'
+'real'

/home/sb/StringIgualaArquivoTest.php:6

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.phpunit StringIgualaArquivoTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) StringIgualaArquivoTest::testFalha
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'esperado
-'
+'real'

/home/sb/StringIgualaArquivoTest.php:6

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

assertStringStartsWith()

`assertStringStartsWith(string $prefixo, string $string[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se a `$string` não começar com `$prefixo`.

`assertStringStartsNotWith()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.53. Uso de assertStringStartsWith()

```php
<?php
class StringComecaComTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertStringStartsWith('prefixo', 'foo');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringComecaComTest::testFalha
Failed asserting that 'foo' starts with "prefixo".

/home/sb/StringComecaComTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit StringComecaComTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringComecaComTest::testFalha
Failed asserting that 'foo' starts with "prefixo".

/home/sb/StringComecaComTest.php:6

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

assertTag()

`assertTag(vetor $equiparador, string $real[, string $mensagem = '', booleano $ehHtml = TRUE])`

Relata um erro identificado por `$mensagem` se `$real` não combinar com `$equiparador`.

`$equiparador` é um vetor associativo que especifica o critério de combinação para a asserção:

- `id`: O nó com o atributo `id` fornecido que deve combinar com o valor correspondente.
- `tag`: O tipo de nó deve combinar com o valor correspondente.
- `attributes`: O atributo do nó deve combinar com o valor correspondente no vetor associativo `$atributos`.
- `content`: O conteúdo do texto deve combinar com o valor fornecido.
- `parent`: O pai do nó deve combinar com o vetor associativo `$pai`.
- `child`: Pelo menos um dos filhos imediatos do nó deve combinar com o critério descrito no vetor associativo `$filho`.
- `ancestor`: Pelo menos um dos ancestrais do nó deve combinar com o critério descrito pelo vetor associativo `$ancestral`.
- `descendant`: Pelo menos um dos ancestrais do nó deve combinar com o critério descrito pelo vetor associativo `$descendente`.
- `children`: vetor associativo para contagem de filhos de um nó.

- `count`: O número de filhos que combinam deve ser igual a este número.
- `less_than`: O número de filhos que combinam deve ser menor que este número.
- `greater_than`: O número de filhos que combinam deve ser maior que este número.
- `only`: Outro vetor associativo que consiste de chaves a serem usadas para combinar em um filho, e apenas filhos que combinem serão contados.

`assertNotTag()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.54. Uso de assertTag()

```
<?php
// Equiparador que assegura que há um elemento com uma id="minha_id".
$equiparador = array('id' => 'minha_id');

// Equiparador que assegura que há uma tag "span".
$equiparador = array('tag' => 'span');

// Equiparador que assegura que há uma tag "span" com o conteúdo
// "Olá Mundo".
$equiparador = array('tag' => 'span', 'content' => 'Hello World');

// Equiparador que assegura que há uma tag "span" com o conteúdo
// correspondendo ao padrão da expressão regular
$equiparador = array('tag' => 'span', 'content' => 'regexp:/Try P(HP|ython)/');

// Equiparador que assegura que há um "span" com um atributo de
// classe "list".
$equiparador = array(
'tag' => 'span',
'attributes' => array('class' => 'list')
);

// Equiparador que assegura que há um "span" dentro de uma "div".
$equiparador = array(
'tag' => 'span',
'parent' => array('tag' => 'div')
);

// Equiparador que assegura que há um "span" em algum lugar dentro
// de uma "table".
$equiparador = array(
'tag' => 'span',
'ancestor' => array('tag' => 'table')
);

// Equiparador que assegura que há um "span" com pelo menos um
// filho "em".
$equiparador = array(
'tag' => 'span',
'child' => array('tag' => 'em')
);

// Equiparador ue assegura que há um "span" contendo uma
// (possivelmente aninhada) tag "strong".
$equiparador = array(
'tag' => 'span',
'descendant' => array('tag' => 'strong')
);

// Equiparador que assegura que há um "span" contendo 5-10 tags
```

```
// "em" como filhos imediatos.
$equiparador = array(
'tag' => 'span',
'children' => array(
'less_than' => 11,
'greater_than' => 4,
'only' => array('tag' => 'em')
)
);

// Equiparador que assegura que há uma "div", com um ancestral "ul"
// e um pai "li" (com class="enum"), e contendo um descendente "span"
// que contém um elemento com id="meu_teste" e o texto "Olá Mundo".
$equiparador = array(
'tag' => 'div',
'ancestor' => array('tag' => 'ul'),
'parent' => array(
'tag' => 'li',
'attributes' => array('class' => 'enum')
),
'descendant' => array(
'tag' => 'span',
'child' => array(
'id' => 'my_test',
'content' => 'Hello World'
)
)
);

// Use assertTag() para aplicar um $equiparador a um pedaço do $html.
$this->assertTag($equiparador, $html);

// Use assertTag() para aplicar um $equiparador a um pedaço do $xml.
$this->assertTag($equiparador, $xml, '', FALSE);
?>
```

`assertThat()`

Asserções mais complexas podem ser formuladas usando a classe `PHPUnit_Framework_Constraint`. Elas podem ser avaliados usando o método `assertThat()`. Exemplo 4.55, “Uso de assertThat()” mostra como as restrições `logicalNot()` e `equalTo()` podem ser usados para expressar a mesma asserção que `assertNotEquals()`.

```
assertThat(misto $valor, PHPUnit_Framework_Constraint $restricao[,
$mensagem = ''])
```

Relata um erro identificado por `$mensagem` se o `$valor` não combinar com `$restricao`.

Exemplo 4.55. Uso de assertThat()

```
<?php
class BiscoitoTest extends PHPUnit_Framework_TestCase
{
public function testEquals()
{
$oBiscoito = new Biscoito('Gengibre');
$meuBiscoito = new Biscoito('Gengibre');

$this->assertThat(
$oBiscoito,
$this->logicalNot(
$this->equalTo($meuBiscoito)
)
```

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

Tabela 4.3, “Restrições” mostra as classes `PHPUnit_Framework_Constraint` disponíveis.

Tabela 4.3. Restrições

Restrição	Significado
`PHPUnit_Framework_Constraint_Attribute attribute(PHPUnit_Framework_Constraint $restricao, $nomeAtributo)`	Restrição que aplica outro restritor a um atributo de uma classe ou objeto.
`PHPUnit_Framework_Constraint_IsAnything anything()`	Restrição que aceita a inserção de qualquer valor.
`PHPUnit_Framework_Constraint_ArrayHasKey arrayHasKey(misto $chave)`	Restrição que assegura que o vetor avaliado possui a chave fornecida.
`PHPUnit_Framework_Constraint_TraversableContains contains(misto $valor)`	Restrição que assegura que o `vetor` ou objeto que implementa a interface `Iterator` é avaliado por conter apenas um valor fornecido.
`PHPUnit_Framework_Constraint_TraversableContainsOnly containsOnly(string $tipo)`	Restrição que assegura que o `vetor` ou objeto que implementa a interface `Iterator` é avaliado por conter apenas os valores de um tipo fornecido.
`PHPUnit_Framework_Constraint_TraversableContainsOnly containsOnlyInstancesOf(string $nomeclasse)`	Restrição que assegura que o `array` ou objeto que implementa a interface `Iterator` é avaliado por conter apenas instâncias de um nome de classe fornecido.
`PHPUnit_Framework_Constraint_IsEqual equalTo($valor, $delta = 0, $profundidadeMaxima = 10)`	Restrição que verifica se um valor é igual a outro.
`PHPUnit_Framework_Constraint_Attribute attributeEqualTo($nomeAtributo, $valor, $delta = 0, $profundidadeMaxima = 10)`	Restrição que verifica se um valor é igual a um atributo de uma classe ou de um objeto.
`PHPUnit_Framework_Constraint_FileExists fileExists()`	Restrição que verifica se existe o arquivo(nome) que é avaliado.
`PHPUnit_Framework_Constraint_GreaterThan greaterThan(misto $valor)`	Restrição que assegura que o valor avaliado é maior que um valor fornecido.
`PHPUnit_Framework_Constraint_OrgreaterThanOrEqual(misto $valor)`	Restrição que assegura que o valor avaliado é maior ou igual a um valor fornecido.
`PHPUnit_Framework_Constraint_ClassHasAttribute classHasAttribute(string $nomeAtributo)`	Restrição que assegura que a classe que é avaliada possui um atributo fornecido.
`PHPUnit_Framework_Constraint_ClassHasStaticAttribute classHasStaticAttribute(string $nomeAtributo)`	Restrição que assegura que a classe avaliada possui um atributo estático fornecido.

Restrição	Significado
`PHPUnit_Framework_Constraint_ObjectHasAttribute hasAttribute(string $nomeAtributo)`	Restrição que assegura que o objeto avaliado possui um atributo fornecido.
`PHPUnit_Framework_Constraint_IsIdentical identicalTo(misto $valor)`	Restrição que assegura que um valor é idêntico a outro.
`PHPUnit_Framework_Constraint_IsFalse isFalse()`	Restrição que assegura que o valor avaliado é `FALSE`.
`PHPUnit_Framework_Constraint_IsInstanceOf isInstanceOf(string $nomeClasse)`	Restrição que assegura que o objeto avaliado é uma instância de uma classe fornecida.
`PHPUnit_Framework_Constraint_IsNull isNull()`	Restrição que assegura que o valor avaliado é `NULL`.
`PHPUnit_Framework_Constraint_IsTrue isTrue()`	Restrição que assegura que o valor avaliado é `TRUE`.
`PHPUnit_Framework_Constraint_IsType isType(string $tipo)`	Restrição que assegura que o valor avaliado é de um tipo especificado.
`PHPUnit_Framework_Constraint_LessThan lessThan(misto $valor)`	Restrição que assegura que o valor avaliado é menor que um valor fornecido.
`PHPUnit_Framework_Constraint_Or lessThanOrEqual(misto $valor)`	Restrição que assegura que o valor avaliado é menor ou igual a um valor fornecido.
`logicalAnd()`	AND lógico.
`logicalNot(PHPUnit_Framework_Constraint $restricao)`	NOT lógico.
`logicalOr()`	OR lógico.
`logicalXor()`	XOR lógico.
`PHPUnit_Framework_Constraint_PCREMatch matchesRegularExpression(string $padrao)`	Restrição que assegura que a string avaliada combina com uma expressão regular.
`PHPUnit_Framework_Constraint_StringContains stringContains(string $string, booleano $case)`	Restrição que assegura que a string avaliada contém uma string fornecida.
`PHPUnit_Framework_Constraint_StringEndsWith stringEndsWith(string $sufixo)`	Restrição que assegura que a string avaliada termina com um sufixo fornecido.
`PHPUnit_Framework_Constraint_StringStartsWith stringStartsWith(string $prefixo)`	Restrição que assegura que a string avaliada começa com um prefixo fornecido.

assertTrue()

```
assertTrue(booleano $condicao[, string $mensagem = ''])
```

Relata um erro identificado por `$mensagem` se `$condicao` is `FALSE`.

Exemplo 4.56. Uso de assertTrue()

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) VerdadeiroTest::testFalha
Failed asserting that false is true.

/home/sb/VerdadeiroTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit VerdadeiroTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) VerdadeiroTest::testFalha
Failed asserting that false is true.

/home/sb/VerdadeiroTest.php:6

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

assertXmlFileEqualsXmlFile()

`assertXmlFileEqualsXmlFile(string $arquivoEsperado, string $arquivoReal[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o documento XML em `$arquivoReal` não for igual ao documento XML em `$arquivoEsperado`.

`assertXmlFileNotEqualsXmlFile()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.57. Uso de assertXmlFileEqualsXmlFile()

```
<?php
class ArquivoXmlIgualaArquivoXmlTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
```

```
{
$this->assertXmlFileEqualsXmlFile(
'/home/sb/esperado.xml', '/home/sb/real.xml');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArquivoXmlIgualaArquivoXmlTest::testFalha
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
<?xml version="1.0"?>
<foo>
- <bar/>
+ <baz/>
</foo>

/home/sb/ArquivoXmlIgualaArquivoXmlTest.php:7

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.phpunit ArquivoXmlIgualaArquivoXmlTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArquivoXmlIgualaArquivoXmlTest::testFalha
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
<?xml version="1.0"?>
<foo>
- <bar/>
+ <baz/>
</foo>

/home/sb/ArquivoXmlIgualaArquivoXmlTest.php:7

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

assertXmlStringEqualsXmlFile()

`assertXmlStringEqualsXmlFile(string $arquivoEsperado, string $xmlReal[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o documento XML em `$xmlReal` não for igual ao documento XML em `$arquivoEsperado`.

`assertXmlStringNotEqualsXmlFile()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.58. Uso de assertXmlStringEqualsXmlFile()

```
<?php
class StringXmlIgualaArquivoXmlTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertXmlStringEqualsXmlFile(
'/home/sb/esperado.xml', '<foo><baz/></foo>');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) StringXmlIgualaArquivoXmlTest::testFalha
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
<?xml version="1.0"?>
<foo>
- <bar/>
+ <baz/>
</foo>

/home/sb/StringXmlIgualaArquivoXmlTest.php:7

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.phpunit StringXmlIgualaArquivoXmlTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) StringXmlIgualaArquivoXmlTest::testFalha
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
<?xml version="1.0"?>
<foo>
- <bar/>
+ <baz/>
</foo>

/home/sb/StringXmlIgualaArquivoXmlTest.php:7

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

assertXmlStringEqualsXmlString()

`assertXmlStringEqualsXmlString(string $xmlEsperado, string $xmlReal[, string $mensagem = ''])`

Relata um erro identificado por `$mensagem` se o documento XML em `$xmlReal` não for igual ao documento XML em `$xmlEsperado`.

`assertXmlStringNotEqualsXmlString()` é o inverso desta asserção e recebe os mesmos argumentos.

Exemplo 4.59. Uso de assertXmlStringEqualsXmlString()

```
<?php
class StringXmlIgualaStringXmlTest extends PHPUnit_Framework_TestCase
{
public function testFalha()
{
$this->assertXmlStringEqualsXmlString(
'<foo><bar/></foo>', '<foo><baz/></foo>');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringXmlIgualaStringXmlTest::testFalha
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
<?xml version="1.0"?>
<foo>
- <bar/>
+ <baz/>
</foo>

/home/sb/StringXmlIgualaStringXmlTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit StringXmlIgualaStringXmlTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringXmlIgualaStringXmlTest::testFalha
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
<?xml version="1.0"?>
```

```
<foo>
- <bar/>
+ <baz/>
</foo>

/home/sb/StringXmlIgualaStringXmlTest.php:7

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

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.

Exemplo 4.60. Saída de erro gerada quando uma comparação de vetores falha

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

```
PHPUnit 3.6.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

/home/sb/VetorDifereTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit VetorDifereTest
PHPUnit 3.6.0 by Sebastian Bergmann.

F

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

```
There was 1 failure:

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

/home/sb/VetorDifereTest.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 em que 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.

Exemplo 4.61. Saída de erro quando uma comparação de um vetor longo falha

```
<?php
class VetorLongoDifereTest extends PHPUnit_Framework_TestCase
{
public function testIgualdade() {
$this->assertEquals(
array(0,0,0,0,0,0,0,0,0,0,0,0,1,2,3 ,4,5,6),
array(0,0,0,0,0,0,0,0,0,0,0,0,1,2,33,4,5,6)
);
}
}
?>
```

```
PHPUnit 3.6.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

```
)


/home/sb/VetorLongoDifereTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit VetorLongoDifereTest
PHPUnit 3.6.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

/home/sb/VetorLongoDifereTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 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 'fraca' de comparação em vetores ou objetos.

Exemplo 4.62. Caso extremo na geração de uma diferenciação quando se usa uma comparação fraca

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

```
PHPUnit 3.6.0 by Sebastian Bergmann.

F
```

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

There was 1 failure:

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


/home/sb/ComparacaoFracaVetorTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ComparacaoFracaVetorTest
PHPUnit 3.6.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

/home/sb/ComparacaoFracaVetorTest.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 5. 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 3.7.0 by Sebastian Bergmann.

..

Time: 0 seconds


OK (2 tests, 2 assertions)phpunit VetorTest
PHPUnit 3.7.0 by Sebastian Bergmann.

..

Time: 0 seconds

OK (2 tests, 2 assertions)
```

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

`S` Impresso quando o teste é pulado (veja Capítulo 9, *Testes Incompletos e Pulados*).

`I` Impresso quando o teste é marcado como incompleto ou ainda não implementado (veja Capítulo 9, *Testes Incompletos e Pulados*).

O PHPUnit distingue entre *falhas* e *erros*. Uma falha é uma asserção violada do PHPUnit 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.

Comutadores de linha-de-comando

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

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

--log-junit <file> Log test execution in JUnit XML format to file.
--log-tap <file> Log test execution in TAP format to file.
--log-json <file> Log test execution in JSON format.

--coverage-clover <file> Generate code coverage report in Clover XML format.
```

```
--coverage-html <dir> Generate code coverage report in HTML format.
--coverage-php <file> Serialize PHP_CodeCoverage object to file.
--coverage-text=<file> Generate code coverage report in text format.
Default to writing to the standard output.

--testdox-html <file> Write agile documentation in HTML format to file.
--testdox-text <file> Write agile documentation in Text format to file.

--filter <pattern> Filter which tests 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.

--loader <loader> TestSuiteLoader implementation to use.
--printer <printer> TestSuiteListener implementation to use.
--repeat <times> Runs the test(s) repeatedly.

--tap Report test execution progress in TAP format.
--testdox Report test execution progress in TestDox format.

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

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

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

-h|--help Prints this usage information.
--version Prints the version and exits.phpunit --help
PHPUnit 3.7.0 by Sebastian Bergmann.

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

--log-junit <file> Log test execution in JUnit XML format to file.
--log-tap <file> Log test execution in TAP format to file.
--log-json <file> Log test execution in JSON format.

--coverage-clover <file> Generate code coverage report in Clover XML format.
--coverage-html <dir> Generate code coverage report in HTML format.
--coverage-php <file> Serialize PHP_CodeCoverage object to file.
--coverage-text=<file> Generate code coverage report in text format.
Default to writing to the standard output.

--testdox-html <file> Write agile documentation in HTML format to file.
--testdox-text <file> Write agile documentation in Text format to file.

--filter <pattern> Filter which tests 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.
```

```
--loader <loader> TestSuiteLoader implementation to use.
--printer <printer> TestSuiteListener implementation to use.
--repeat <times> Runs the test(s) repeatedly.

--tap Report test execution progress in TAP format.
--testdox Report test execution progress in TestDox format.

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

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

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

-h|--help Prints this usage information.
--version Prints the version 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 fornecidos pela classe `UnitTest`. Espera-se que esta classe seja declarada no arquivo-fonte especificado.

`--log-junit`

Gera um arquivo de registro no formato Junit XML para a execução dos testes. Veja Capítulo 18, *Registrando* for more details.

`--log-tap`

Gera um arquivo de registro usando o formato Test Anything Protocol (TAP) [http://testanything.org/]para a execução dos testes. Veja Capítulo 18, *Registrando* para mais detalhes.

`--log-json`

Gera um arquivo de registro usando o formato JSON [http://www.json.org/]. Veja Capítulo 18, *Registrando* para mais detalhes.

`--coverage-html`

Gera um relatório de cobertura de código no formato HTML. Veja Capítulo 14, *Análise de Cobertura de Código* para mais detalhes.

Por favor note que esta funcionalidade só está disponível quando as extensões tokenizer e Xdebug estão instaladas.

`--coverage-clover`	Gera um arquivo de registro no formato XML com a informação sobre a cobertura de código da execução dos testes. Veja Capítulo 18, *Registrando* para mais detalhes. Por favor note que essa funcionalidade só está disponível quando as extensões tokenizer e Xdebug estão instaladas.
`--coverage-php`	Gera um objeto serializado PHP_CodeCoverage com a informação da cobertura de código. Por favor note que essa funcionalidade só está disponível quando as extensões tokenizer e Xdebug estão instaladas.
`--coverage-text`	Gera um arquivo de registro ou saída em linha-de-comando em um formato humanamente legível com a informação de cobertura de código da execução dos testes. Veja Capítulo 18, *Registrando* para mais detalhes. Por favor note que essa funcionalidade só está disponível quando as extensões tokenizer e Xdebug estão instaladas.
`--testdox-html` e `--testdox-text`	Gera documentação ágil em formato HTML ou texto plano para os testes que são executados. Veja Capítulo 15, *Outros Usos para Testes* para mais detalhes.
`--filter`	Apenas executa os testes cujos nomes combinam com o padrão fornecido. O padrão pode tanto ser o nome de um único teste quanto uma expressão regular [http://www.php.net/pcre] que combina múltiplos nomes de testes.
`--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 disponíveis.
`--loader`	Especifica a implementação `PHPUnit_Runner_TestSuiteLoader` a ser usada. A suíte de carregadores de teste padrão irá procurar pelo arquivo-fonte no diretório de trabalho atual e em cada diretório que está especificado na configuração de diretiva `include_path` do PHP. Seguindo as Convenções de Nomenclatura do PEAR, um nome de classe como `Project_Package_Class` é mapeado para o nome de arquivo-fonte `Project/Package/Class.php`.
`--printer`	Especifica o impressor de resultados a ser usado. A classe impressora deve estender `PHPUnit_Util_Printer` e implementa a interface `PHPUnit_Framework_TestListener`.
`--repeat`	Executa repetidamente o(s) teste(s) um determinado número de vezes.

`--tap`	Relata o progresso do teste usando o Test Anything Protocol (TAP) [http://testanything.org/]. Veja Capítulo 18, *Registrando* para mais detalhes.
`--testdox`	Relata o progresso do teste como uma documentação ágil. Veja Capítulo 15, *Outros Usos para Testes* para mais detalhes.
`--colors`	Usa cores na saída.
`--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-skipped`	Para a execução no primeiro teste pulado.
`--stop-on-incomplete`	Para a execução no primeiro teste incompleto.
`--strict`	Executa os testes em modo estrito.
`--verbose`	Saída mais verbosa de informações, por exemplo os nomes dos testes que ficaram incompletos ou foram pulados.
`--process-isolation`	Executa cada teste em um processo PHP separado.
`--no-globals-backup`	Não faz backup e restaura $GLOBALS. Veja “Estado Global” para mais detalhes.
`--static-backup`	Faz backup e restaura atributos estáticos das classes definidas pelo usuário. Veja “Estado Global” para mais detalhes.
`--bootstrap`	Um arquivo PHP "bootstrap" que é executado antes dos testes.
`--configuration`, `-c`	Lê a configuração de um arquivo XML. Veja Apêndice C, *O arquivo de configuração XML* para mais detalhes. Se `phpunit.xml` ou `phpunit.xml.dist` (nessa ordem) existirem 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.
`--debug`	Informação da saída de depuração como o nome de um teste quando a execução inicia.

Capítulo 6. 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 ???, o ambiente era simplesmente o vetor que está guardado na variável `$ambiente` Na maior parte do tempo, porém, o ambiente será mais complexo 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 de teste 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 Exemplo 4.2, “Usando a anotação `@depends` para expressar dependências” usamos a relação produtor-consumidor entre testes para compartilhar ambientes. Isso nem sempre é desejável, ou mesmo possível. Exemplo 6.1, “Usando setUp() para criar a pilha de ambientes” mostra como podemos escrever os testes do `PilhaTest` de forma que o próprio ambiente não é reutilizado, mas o código que o cria. Primeiro declaramos a variável de instância `$pilha`, que usaremos no lugar de uma variável do método local. Então colocamos a criação do ambiente `vetor` dentro do método `setUp()` Finalmente, removemos o código redundante dos métodos de teste e usamos a nova variável de instância, `$this->pilha`, em vez da variável de método local `$pilha` com o método de asserção `assertEquals()`.

Exemplo 6.1. Usando setUp() para criar a pilha de ambientes

```
<?php
class PilhaTest extends PHPUnit_Framework_TestCase
{
protected $pilha;

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

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

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

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

```
?>
```

Os métodos-modelo `setUp()` e `tearDown()` ão 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 de cada primeiro e depois do último teste da classe de casos de teste, respectivamente, serem executados.

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

Exemplo 6.2. Exemplo mostrando todos os métodos-modelo disponíveis

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

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

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

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

public function testDois()
{
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 3.7.0 by Sebastian Bergmann.

MetodosModeloTest::setUpBeforeClass
MetodosModeloTest::setUp
MetodosModeloTest::assertPreConditions
MetodosModeloTest::testOne
MetodosModeloTest::assertPostConditions
MetodosModeloTest::tearDown
MetodosModeloTest::setUp
MetodosModeloTest::assertPreConditions
MetodosModeloTest::testTwo
MetodosModeloTest::tearDown
MetodosModeloTest::onNotSuccessfulTest
MetodosModeloTest::tearDownAfterClass


Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) MetodosModeloTest::testDois
Failed asserting that <boolean:false> is true.
/home/sb/MetodosModeloTest.php:30

FAILURES!
Tests: 2, Assertions: 2, Failures: 1.phpunit MetodosModeloTest
PHPUnit 3.7.0 by Sebastian Bergmann.

MetodosModeloTest::setUpBeforeClass
MetodosModeloTest::setUp
MetodosModeloTest::assertPreConditions
MetodosModeloTest::testOne
MetodosModeloTest::assertPostConditions
MetodosModeloTest::tearDown
MetodosModeloTest::setUp
MetodosModeloTest::assertPreConditions
MetodosModeloTest::testTwo
MetodosModeloTest::tearDown
MetodosModeloTest::onNotSuccessfulTest
MetodosModeloTest::tearDownAfterClass

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) MetodosModeloTest::testDois
Failed asserting that <boolean:false> is true.
/home/sb/MetodosModeloTest.php:30

FAILURES!
Tests: 2, Assertions: 2, Failures: 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 ignorar o

`tearDown()`. de modo geral. Porém, se você criar muitos objetos em seu `setUp()`, você pode querer remover a configuração `unset()` você pode querer remover a configuração `tearDown()` para que eles possam ser coletados como lixo. A coleta de lixo dos objetos dos casos de teste não é previsível.

Variantes

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.

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.

Exemplo 6.3, “Compartilhando ambientes entre os testes de uma suíte de testes” usa os métodos-modelo `setUpBeforeClass()` e `tearDownAfterClass()` para conectar ao banco de dados antes do primeiro teste da classe de casos de teste e a desconectar do banco de dados após o último teste dos casos de teste, respectivamente.

Exemplo 6.3. Compartilhando ambientes entre os testes de uma suíte de testes

```
<?php
class BancoDeDadosTest extends PHPUnit_Framework_TestCase
{
protected static $dbh;

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

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

Não dá pra enfatizar o suficiente o quanto 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 Capítulo 10, *Dublês de Testes*), do que criando dependências entre os testes em tempo de execução e ignorando a oportunidade de melhorar seu design.

Estado Global

É difícil testar um código que usa singletons (instâncias únicas de objetos). [http://googletesting.blogspot.com/2008/05/tott-using-dependancy-injection-to.html] 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 mudança de um teste para uma variável global 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.

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. Opcionalmente, este isolamento pode ser estendido a atributos estáticos de classes.

Nota

A implementação das operações de backup e restauração para atributos estáticos de classes exige PHP 5.3 (ou superior).

A implementação das operações de backup e restauração para variáveis globais e atributos estáticos de classes utiliza `serialize()` e `unserialize()`.

Objetos de algumas classes fornecidas pelo próprio PHP, como `PDO` por exemplo, não podem ser serializadas 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 seção chamada "`@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 MeuTest extends PHPUnit_Framework_TestCase
{
    protected $backupListaNegraGlobais = array('variavelGlobal');

    // ...
}
```

Nota

Por favor, note que definir o atributo `$backupListaNegraGlobais` dentro do método `setUp()`, por exemplo, não tem efeito.

A anotação `@backupStaticAttributes` que é discutida na seção chamada "`@backupStaticAttributes`"pode ser usada para controlar as operações de backup e restauração para atributos estáticos. Alternativamente, você pode fornecer uma lista-negra de atributos estáticos que deverão ser excluídos das operações de backup e restauração como esta:

```
class MeuTest extends PHPUnit_Framework_TestCase
{
    protected $backupListaNegraAtributosEstaticos = array(
```

```
      'nomeClasse' => array('nomeAtributo')
    );

    // ...
}
```

Nota

Por favor, note que definir o atributo `$backupStaticAttributesBlacklist` dentro do método `setUp()` , por exemplo, não tem efeito.

Capítulo 7. Organizando Testes

Um dos objetivos do PHPUnit (veja Capítulo 2, *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 só classe.

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

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 Object_Freezer [http://github.com/sebastianbergmann/php-object-freezer/] Observando a estrutura de diretórios desse projeto, podemos ver que as classes dos casos de teste no diretório `Testes` espelha o pacote e estrutura de classes do Sistema Sob Teste (SST – ou SST: System Under Teste) no diretório `Object`:

```
Object                              Tests
|-- Freezer                         |-- Freezer
|   |-- HashGenerator               |   |-- HashGenerator
|   |   `-- NonRecursiveSHA1.php    |   |   `-- NonRecursiveSHA1Test.php
|   |-- HashGenerator.php           |   |
|   |-- IdGenerator                 |   |-- IdGenerator
|   |   `-- UUID.php                |   |   `-- UUIDTest.php
|   |-- IdGenerator.php             |   |
|   |-- LazyProxy.php               |   |
|   |-- Storage                     |   |-- Storage
|   |   `-- CouchDB.php             |   |   `-- CouchDB
|   |                               |   |       |-- WithLazyLoadTest.php
|   |                               |   |       `-- WithoutLazyLoadTest.php
|   |-- Storage.php                 |   |-- StorageTest.php
|   `-- Util.php                    |   `-- UtilTest.php
`-- Freezer.php                     `-- FreezerTest.php
```

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 3.7.0 by Sebastian Bergmann.

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

Time: 0 seconds

OK (75 tests, 164 assertions)phpunit Tests
PHPUnit 3.7.0 by Sebastian Bergmann.

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

Time: 0 seconds
```

```
OK (75 tests, 164 assertions)
```

Nota

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 `Object_FreezerTest` em `Tests/FreezerTest.php`, podemos usar o seguinte comando:

```
PHPUnit 3.7.0 by Sebastian Bergmann.

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

Time: 0 seconds

OK (28 tests, 60 assertions)phpunit Tests/FreezerTest
PHPUnit 3.7.0 by Sebastian Bergmann.

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

Time: 0 seconds

OK (28 tests, 60 assertions)
```

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

```
PHPUnit 3.7.0 by Sebastian Bergmann.

.

Time: 0 seconds

OK (1 test, 2 assertions)phpunit --filter testFreezingAnObjectWorks Tests
PHPUnit 3.7.0 by Sebastian Bergmann.

.

Time: 0 seconds

OK (1 test, 2 assertions)
```

Nota

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.

Compondo uma Suíte de Testes Usando uma Configuração XML

O arquivo de configuração XML do PHPUnit (Apêndice C, *O arquivo de configuração XML*) também pode ser usado para compor uma suíte de testes. Exemplo 7.1, “Compondo uma Suíte de Testes Usando uma Configuração XML” mostra um exemplo mínimo que adicionará todas as classes `*Test` que forem encontradas em arquivos `*Test.php` quando os `Tests` forem atravessados recursivamente.

Exemplo 7.1. Compondo uma Suíte de Testes Usando uma Configuração XML

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

A ordem de execução dos testes pode ser explicitada:

Exemplo 7.2. Compondo uma Suíte de Testes Usando uma Configuração XML

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

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.

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 (provavelmente muito) 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. Pode ser instalada da seguinte forma:

```
pear install phpunit/DbUnit
```

Fornecedores Suportados para Testes de Banco de Dados

O DbUnit atualmente suporta MySQL, PostgreSQL, Oracle e SQLite. Através das integrações Zend Framework [http://framework.zend.com] ou Doctrine 2 [http://www.doctrine-project.org] ele tem acesso a outros sistemas como IBMDB2 ou Microsoft SQL Server.

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: esse tipo de testes é complexo tanto em configuração quanto em manutenção. Enquanto testar contra seu banco de dados você precisará ter cuidado com as seguintes variáveis:

- 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 da maior importância para resolver o problema do “banco de dados ser uma entrada global”.

O DbUnit ajuda a simplificar todos esses problemas com testes de bancos de dados de 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 [http://www.doctrine-project.org]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.

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 a saída
4. Teardown

> **O que é um ambiente (fixture)?**
>
> Descreve o estado inicial em que sua aplicação e seu banco de dados estão ao executar um teste.

Testar um banco de dados exige que você utilize pelo menos o setup e 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:

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.

2. Configurar o ambiente

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

3–5. Executar Teste, Verificar saída e 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 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”.

Configuração de 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:

```
require_once "PHPUnit/Framework/TestCase.php";

class MeuTest extends PHPUnit_Framework_TestCase
{
    public function testCalculo()
    {
        $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()`:

```
require_once "PHPUnit/Extensions/Database/TestCase.php";

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

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

Implementando `getConnection()`

Para permitir que a limpeza e o carregamento de funcionalidades do ambiente funcionem, a Extensão do Banco de Dados do PHPUnit exige acesso a uma conexão abstrata do banco de dados através dos fornecedores 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 usada apenas para limpeza e configuração do 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 do Banco de Dados” explica a API desta interface e como você pode usá-la da melhor forma possível.

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 Conjunto de Dados e 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($nomearquivo)` que representa um conjunto de dados através de uma representação XML.

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 atingir 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 [http://www.doctrine-project.org] ou Propel [http://www.propelorm.org/] 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 [http://www.phpunit.de/manual/current/en/textui.html] para executar esse código sempre que seus testes forem executados.

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 a você especificar um ambiente de dados diferente para cada caso de teste:

```
require_once "PHPUnit/Extensions/Database/TestCase.php";

abstract class MinhaApp_Testes_CasosDeTesteDeBancoDeDadosTest extends PHPUnit_Extensions
{
    // instancie o pdo apenas uma vez por limpeza de teste/carregamento de ambiente
    static private $pdo = null;
```

```
    // instancie PHPUnit_Extensions_Database_DB_IDatabaseConnection apenas uma vez por t
    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;
    }
}
```

Entretanto, 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 [http://www.phpunit.de/manual/current/en/appendixes.configuration.html#appendixes.configuration.php-ini-constants-variables] você pode tornar a conexão com o banco de dados configurável por execução de teste. Primeiro vamos criar um arquivo “phpunit.xml” em seu diretório testes da aplicação, de forma semelhante a isto:

```
<?xml version="1.0" encoding="UTF-8" ?>
<phpunit>
    <php>
        <var name="BD_DSN" value="mysql:dbname=meulivrodevisitas;host=localhost" />
        <var name="BD_USUARIO" value="usuario" />
        <var name="BD_SENHA" value="senha" />
        <var name="BD_NOMEBD" value="meulivrodevisitas" />
    </php>
</phpunit>
```

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

```
abstract class Generic_Tests_DatabaseTestCase extends PHPUnit_Extensions_Database_TestCa
{
    // instancie o pdo apenas uma vez por limpeza de teste/carregamento de ambiente
    static private $pdo = null;

    // instancie PHPUnit_Extensions_Database_DB_IDatabaseConnection apenas uma vez por t
    private $conn = null;

    final public function getConnection()
    {
        if ($this->conn === null) {
            if (self::$pdo == null) {
                self::$pdo = new PDO( $GLOBALS['BD_DSN'], $GLOBALS['BD_USUARIO'], $GLOBA
            }
            $this->conn = $this->createDefaultDBConnection(self::$pdo, $GLOBALS['BD_NOME
        }

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

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.

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 Conjunto de Dados e Tabela de Dados permitem a comparação dessas fontes conceitualmente diferentes, emulando a armazenagem relacional de banco de dados em uma abordagem semanticamente similar.

Um fluxo de trabalho para asserções em banco de dados nos seus testes então consiste de três passos 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, ...);
- Assertar 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.

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 usadas 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 chamada XML Plano. É um formato xml 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>
    <livrodevisitas id="1" conteudo="Olá amigo!" usuario="joao" criado="2010-04-24 17:15
    <livrodevisitas id="2" conteudo="Eu gostei!" usuario="nanda" criado="2010-04-26 12:1
</dataset>
```

Isso é, obviamente, fácil de escrever. Aqui `<livrodevisitas>` é o nome da tabela onde duas linhas são inseridas dentro de cada com quatro colunas “id”, “conteudo”, “usuario” e “criado” 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>
    <livrodevisitas />
</dataset>
```

O manejamento 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 seu livro de visitas vai permitir entradas anônimas representadas por um valor NULL na coluna usuario, um estado hipotético para a tabela do livrodevisitas seria parecido com:

```
<?xml version="1.0" ?>
<dataset>
    <livrodevisitas id="1" conteudo="Olá amigo!" usuario="joao" criado="2010-04-24 17:15
    <livrodevisitas id="2" conteudo="Eu gostei!" criado="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 pode 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”, “conteudo”, “usuario” e “criado” são colunas da tabela livrodevisitas. Para a segunda linha, onde “usuario” não está definido, um NULL seria inserido no banco de dados.

Quando a primeira entrada do livrodevisitas for apagada do conjunto de dados, apenas “id”, “conteudo” e “criado” seriam colunas da tabela livrodevisitas, já que “usuario” 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 as 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($nomearquivo)`:

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

Conjunto de Dados XML

Existe uma 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 definida anteriormente no Livrodevisitas em XML Plano seria como:

```
<?xml version="1.0" ?>
<dataset>
    <table name="livrodevisitas">
        <column>id</column>
        <column>conteudo</column>
        <column>usuario</column>
        <column>criado</column>
        <row>
            <value>1</value>
            <value>Olá amigo!</value>
            <value>joao</value>
            <value>2010-04-24 17:15:23</value>
        </row>
        <row>
            <value>2</value>
            <value>Eu gostei!</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 aninhados `<row>`. Não definir nenhum elemento `<row>` significa que a tabela está vazia. As tags `<value>` e `<null />` têm que ser especificadas na ordem especificada nos 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($nomearquivo)`:

```
class MeuCasoDeTest extends PHPUnit_Extensions_Database_TestCase
{
    public function getDataSet()
    {
        return $this->createXMLDataSet('meuAmbienteXml.xml');
    }
}
```

Conjunto de Dados XML MySQL

Este novo formato de arquivo XML é específico para o servidor de banco de dados MySQL [http://www.mysql.com]. 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 [nomeusuario] --password=[senha] [bancodedados] > /caminho/para/ar
```

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

```
class MeuCasoDeTest extends PHPUnit_Extensions_Database_TestCase
{
    public function getDataSet()
    {
        return $this->createMySQLXMLDataSet('/caminho/para/arquivo.xml');
    }
}
```

Conjunto de Dados YAML

Novidade do PHPUnit 3.4, é a capacidade de especificar um Conjunto de Dados no popular formato YAML. Para funcionar, você deve instalar o PHPUnit 3.4 através do PEAR com a dependência opcional do Symfony YAML. Então você poderá escrever um Conjunto de Dados YAML para o exemplo do livrodevisitas:

```
livrodevisitas:
  -
    id: 1
    conteudo: "Olá amigo!"
    usuario: "joao"
    criado: 2010-04-24 17:15:23
  -
    id: 2
    conteudo: "Eu gostei!"
    usuario:
    criado: 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 `coluna1: ""`.

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

```
require_once "PHPUnit/Extensions/Database/DataSet/YamlDataSet.php";

class LivroDeVisitasYamlTest extends PHPUnit_Extensions_Database_TestCase
{
    protected function getDataSet()
    {
        return new PHPUnit_Extensions_Database_DataSet_YamlDataSet(
            dirname(__FILE__)."/_files/livrodevisitas.yml"
        );
    }
}
```

Conjunto de Dados CSV

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

```
id;conteudo;usuario;criado
```

```
1;"Olá amigo!";"joao";"2010-04-24 17:15:23"
2;"Eu gostei!""nanda";"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 ser inserido na coluna.

Você pode criar um Conjunto de Dados CSV chamando:

```
require_once 'PHPUnit/Extensions/Database/DataSet/CsvDataSet.php';

class LivroDeVisitasCsvTest extends PHPUnit_Extensions_Database_TestCase
{
    protected function getDataSet()
    {
        $conjuntoDados = new PHPUnit_Extensions_Database_DataSet_CsvDataSet();
        $conjuntoDados->addTable('livrodevisitas', dirname(__FILE__)."/_files/livrodevis
        return $conjuntoDados;
    }
}
```

Vetor de Conjunto de Dados

Não existe (ainda) Conjunto de Dados baseado em vetor na Extensão para Banco de Dados do PHPUnit, mas podemos implementar o nosso próprio facilmente. Nosso exemplo do livrodevisitas deveria ficar parecido com:

```
class VetorLivroDeVisitasTest extends PHPUnit_Extensions_Database_TestCase
{
    protected function getDataSet()
    {
        return new MinhaApp_DbUnit_VetorConjuntoDeDados(array(
            'livrodevisitas' => array(
                array('id' => 1, 'conteudo' => 'Olá amigo!', 'usuario' => 'joao', 'criad
                array('id' => 2, 'conteudo' => 'Eu gostei!', 'usuario' => null, 'criado'
            ),
        ));
    }
}
```

Um Conjunto de Dados do PHP em algumas vantagens óbvias sobre todas as outras conjuntos de dados baseadas em arquivos:

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

Para este Conjunto de Dados, como nos Conjunto 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”, “conteudo”, “usuario” e “criado”.

A implementação para este vetor Conjunto de Dados é simples e direta:

```
require_once 'PHPUnit/Util/Filter.php';

require_once 'PHPUnit/Extensions/Database/DataSet/AbstractDataSet.php';
require_once 'PHPUnit/Extensions/Database/DataSet/DefaultTableIterator.php';
require_once 'PHPUnit/Extensions/Database/DataSet/DefaultTable.php';
require_once 'PHPUnit/Extensions/Database/DataSet/DefaultTableMetaData.php';

PHPUnit_Util_Filter::addFileToFilter(__FILE__, 'PHPUNIT');
```

```
class MinhaApp_DbUnit_VetorConjuntoDeDadosTest extends PHPUnit_Extensions_Database_DataS
{
    /**
     * @var array
     */
    protected $tabelas = array();

    /**
     * @param array $dados
     */
    public function __construct(array $dados)
    {
        foreach ($dados AS $nomeTabela => $linhas) {
            $colunas = array();
            if (isset($linhas[0])) {
                $colunas = array_keys($linhas[0]);
            }

            $metaDados = new PHPUnit_Extensions_Database_DataSet_DefaultTableMetaData($n
            $tabela = new PHPUnit_Extensions_Database_DataSet_DefaultTable($metaDados);

            foreach ($linhas AS $linha) {
                $tabela->addRow($linha);
            }
            $this->tabelas[$nomeTabela] = $tabela;
        }
    }

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

    public function getTable($nomeTabela)
    {
        if (!isset($this->tabelas[$nomeTabela])) {
            throw new InvalidArgumentException("$nomeTabela não é uma tabela do banco de
        }

        return $this->tabelas[$nomeTabela];
    }
}
```

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:

```
$ds = new PHPUnit_Extensions_Database_DataSet_QueryDataSet($this->getConnection());
$ds->addTable('livrodevisitas');
```

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

```
$ds = new PHPUnit_Extensions_Database_DataSet_QueryDataSet($this->getConnection());
$ds->addTable('livrodevisitas', 'SELECT * FROM livrodevisitas');
```

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

```
$ds = new PHPUnit_Extensions_Database_DataSet_QueryDataSet($this->getConnection());
```

```
$ds->addTable('livrodevisitas', 'SELECT id, conteudo FROM livrodevisitas ORDER BY criado
```

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 Conexões de Fábrica.

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

```
class MeuTesteLivrodevisitasMySqlTest extends PHPUnit_Extensions_Database_TestCase
{
    /**
     * @return PHPUnit_Extensions_Database_DB_IDatabaseConnection
     */
    public function getConnection()
    {
        $bancodedados = 'meu_bancodedados';
        $pdo = new PDO('mysql:...', $usuario, $senha);
        return $this->createDefaultDBConnection($pdo, $bancodedados);
    }

    public function testLivrodevisitas()
    {
        $conjuntoDados = $this->getConnection()->criarDataSet();
        // ...
    }

    public function testLivrodevisitasFiltrado()
    {
        $nomesTabelas = array('livrodevisitas');
        $conjuntoDados = $this->getConnection()->criarConjuntoDeDados($nomesTabelas);
        // ...
    }
}
```

Conjunto de Dados de Reposição

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 Reposição 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 reposição. Para fazer nosso exemplo do livro de visitas funcionar com valores NULL devemos especificar o arquivo como:

```
<?xml version="1.0" ?>
<dataset>
    <livrodevisitas id="1" conteudo="Olá amigo!" usuario="joe" criado="2010-04-24 17:15:
    <livrodevisitas id="2" conteudo="Eu gostei!" usuario="##NULL##" criado="2010-04-26 1
</dataset>
```

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

```
require_once 'PHPUnit/Extensions/Database/DataSet/ReplacementDataSet.php';
```

```
class ReposicaoTest extends PHPUnit_Extensions_Database_TestCase
{
    public function getDataSet()
    {
        $ds = $this->createFlatXmlDataSet('meuAmbienteXmlPlano.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 BD DataSet para filtrar as colunas dos conjuntos de dados.

```
class FiltroConjuntoDeDadosTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testIncluirLivrodevisitasFiltrado()
    {
        $nomesTabelas = array('livrodevisitas');
        $conjuntoDados = $this->getConnection()->createDataSet();

        $filtroConjuntoDeDados = new PHPUnit_Extensions_Database_DataSet_DataSetFilter($
        $filtroConjuntoDeDados->addIncludeTables(array('livrodevisitas'));
        $filtroConjuntoDeDados->setIncludeColumnsForTable('livrodevisitas', array('id',
        // ..
    }

    public function testExcluirLivrodevisitasFiltrado()
    {
        $nomesTabelas = array('livrodevisitas');
        $conjuntoDeDados = $this->getConnection()->createDataSet();

        $filtroConjuntoDeDados = new PHPUnit_Extensions_Database_DataSet_DataSetFilter($
        $filtroConjuntoDeDados->addExcludeTables(array('foo', 'bar', 'baz')); // mantém
        $filtroConjuntoDeDados->setExcludeColumnsForTable('livrodevisitas', array('usuar
        // ..
    }
}
```

> **NOTA** Você não pode usar ambos os filtros de coluna excluir e incluir 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 *ambiente1.xml*:

```
<?xml version="1.0" ?>
<dataset>
    <livrodevisitas id="1" conteudo="Olá amigo!" usuario="joao" criado="2010-04-24 17:15
</dataset>
```

e *ambiente2.xml*:

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

```
    <livrodevisitas id="2" conteudo="Eu gostei!" usuario="##NULL##" criado="2010-04-26 1
</dataset>
```

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

```
class CompostoTest extends PHPUnit_Extensions_Database_TestCase
{
    public function getDataSet()
    {
        $ds1 = $this->criarConjuntoDeDadosXmlPlano('ambiente1.xml');
        $ds2 = $this->criarConjuntoDeDadosXmlPlano('ambiente2.xml');

        $dsComposta = new PHPUnit_Extensions_Database_DataSet_CompositeDataSet();
        $dsComposta->adicionarConjuntoDeDados($ds1);
        $dsComposta->adicionarConjuntoDeDados($ds2);

        return $dsComposta;
    }
}
```

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.

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 sua próprio Conjunto de Dados ou Tabela de Dados.

```
interface PHPUnit_Extensions_Database_DataSet_IDataSet extends IteratorAggregate
{
    public function getNomesTabelas();
    public function getTabelasMetaData($nomeTabela);
    public function getTabela($nomeTabela);
    public function assertEquals(PHPUnit_Extensions_Database_DataSet_IDataSet $outra);

    public function getIteradorReverso();
}
```

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 método adicional do iterador reverso é requerido para truncar corretamente as tabelas em ordem reversa à especificada.

> Para entender a necessidade de um iterador reverso pense em duas tabelas (*TabelaA* e *TabelaB*) onde *TabelaB* possui uma chave estrangeira em uma coluna da *TabelaA*. Se para a configuração do ambiente uma linha é inserida na *TabelaA* e então um registro dependente na *TabelaB*, então é óbvio que para deletar todos os conteúdos das tabelas a execução em ordem reversa vai causar problemas com 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 todas as conjuntos de dados baseadas em arquivo como `YamlDataSet`, `XmlDataSet` ou `FlatXmlDataSet`.

Uma tabela também é representada pela seguinte interface:

```
interface PHPUnit_Extensions_Database_DataSet_ITable
{
    public function getTabelaMetaDados();
    public function getContagemLinhas();
    public function getValor($linha, $coluna);
    public function getLinha($linha);
    public function assertEquals(PHPUnit_Extensions_Database_DataSet_ITable $outro);
}
```

Com exceção do método `getTabelaMetaDados()` 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 `getTabelaMetaDados()` 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 coluna da tabela, ordenado por suas aparições nos resultados
- Um vetor de colunas de chaves-primárias.

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

A API de Conexão

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

```
interface PHPUnit_Extensions_Database_DB_IDatabaseConnection
{
    public function criarConjuntoDeDados(Array $nomesTabelas = NULL);
    public function criarTabelaQuery($nomeResultado, $sql);
    public function getContagemLinhas($nomeTabela, $clausulaWhere = NULL);

    // ...
}
```

1. O método `criarConjuntoDeDados()` cria um Conjunto de Dados de Banco de Dados (BD) como descrito na seção de implementações de Conjunto de Dados.

   ```
   class ConexaoTest extends PHPUnit_Extensions_Database_TestCase
   {
       public function testCriarConjuntoDeDados()
       {
           $nomesTabelas = array('livrodevisitas');
           $conjuntoDados = $this->getConnection()->criarConjuntoDeDados();
       }
   }
   ```

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

```
class ConexaoTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testCriarTabelaQuery()
    {
        $nomesTabela = array('livrodevisitas');
        $tabelaQuery = $this->getConnection()->criarTabelaQuery('livrodevisitas', 'SEL
    }
}
```

3. O método `getContagemLinhas()` 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:

```
class ConexaoTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testGetContagemLinhas()
    {
        $this->assertEquals(2, $this->getConnection()->getContagemLinhas('livrodevisit
    }
}
```

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:

Assertando 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 livro de visitas não apenas temos as duas entradas iniciais que nos acompanharam em todos os exemplos anteriores, mas uma terceira:

```
class LivroDeVisitasTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testAdicionaEntrada()
    {
        $this->assertEquals(2, $this->getConnection()->getContagemLinhas('livrodevisitas

        $guestbook = new Guestbook();
        $guestbook->addEntry("suzy", "Olá mundo!");

        $this->assertEquals(3, $this->getConnection()->getContagemLinhas('livrodevisitas
    }
}
```

Assertando 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 vetor/Arquivo:

```
class LivroDeVisitasTest extends PHPUnit_Extensions_Database_TestCase
{
```

```
    public function testAdicionaEntrada()
    {
        $livrodevisitas = new LivrodeVisitas();
        $livrodevisitas->addEntry("suzy", "Olá mundo!");

        $tabelaQuery = $this->getConnection()->criaTabelaQuery(
            'livrodevisitas', 'SELECT * FROM livrodevisitas'
        );
        $tabelaEsperada = $this->criaDatasetXmlPlano("livroEsperado.xml")
                               ->getTabela("livrodevisitas");
        $this->assertTablesEqual($tabelaEsperada, $tabelaQuery);
    }
}
```

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

```
<?xml version="1.0" ?>
<dataset>
    <livrodevisitas id="1" conteudo="Olá amigo!" usuario="joao" criado="2010-04-24 17:15
    <livrodevisitas id="2" conteudo="Eu gostei!" usuario="nanda" criado="2010-04-26 12:1
    <livrodevisitas id="3" conteudo="Olá mundo!" usuario="suzy" criado="2010-05-01 21:47
</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 “criado” da asserção.

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

```
<?xml version="1.0" ?>
<dataset>
    <livrodevisitas id="1" conteudo="Olá amigo!" usuario="joao" />
    <livrodevisitas id="2" conteudo="Eu gostei!" usuario="nanda" />
    <livrodevisitas id="3" conteudo="Olá mundo!" usuario="suzy" />
</dataset>
```

Nós temos que consertar a chamada da Tabela Query:

```
$tabelaQuery = $this->getConnection()->criarTabelaQuery(
    'livrodevisitas', 'SELECT id, conteudo, usuario FROM livrodevisitas'
);
```

Assertando o Resultado de uma Query

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

```
class QueryComplexaTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testQueryComplexa()
    {
        $tabelaQuery = $this->getConnection()->criarTabelaQuery(
            'minhaQueryComplexa', 'SELECT queryComplexa...'
        );
        $tabelaEsperada = $this->criarConjuntoDeDadosXmlPlano("assercaoQueryComplexa.xml
                               ->getTabela("minhaQueryComplexa");
        $this->assertTablesEqual($tabelaEsperada, $tabelaQuery);
    }
}
```

Assertando o Estado de Múltiplas Tabelas

Certamente você pode assertar 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 de asserções de Conjuntos de Dados.

1. Você pode usar o Conjunto de Dados de Banco de Dados (BD) e compará-lo com um Conjunto de Dados Baseado em Arquivo.

```
class AssercaoConjuntoDeDadosTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testCriarAssercaoDeConjuntoDeDados()
    {
        $conjuntoDeDados = $this->getConnection()->criaConjuntoDeDados(array('livrodev
        $conjuntoDeDadosEsperado = $this->createFlatXmlDataSet('livrodevisitas.xml');
        $this->assertDataSetsEqual($conjuntoDeDadosEsperado, $conjuntoDeDados);
    }
}
```

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

```
class AssercoesConjuntoDeDadosTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testAssercaoManualDeConjuntoDeDados()
    {
        $conjuntoDeDados = new PHPUnit_Extensions_Database_DataSet_QueryDataSet();
        $conjuntoDeDados->addTable('livrodevisitas', 'SELECT id, conteudo, usuario FRO
        $conjuntoDeDadosEsperado = $this->createFlatXmlDataSet('livrodevisitas.xml');

        $this->assertDataSetsEqual($conjuntoDeDadosEsperado, $conjuntoDeDados);
    }
}
```

Perguntas Mais Frequentes

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 [http://www.doctrine-project.org] ou eZ Components [http://www.ezcomponents.org] possuem ferramentas poderosas que permitem a você criar o esquema de banco de dados através de estruturas de dados pré-definidas, porém 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.

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.

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.

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

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

Capítulo 9. Testes Incompletos e Pulados

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 testAlgumaCoisa()
{
}
```

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 teste não implementado não ajuda em nada, já que o teste será interpretado como uma falha. Isso seria tão errado quando 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 o 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.

Exemplo 9.1, “Marcando um teste como incompleto” mostra uma classe de caso de teste, `ExemploTest`, que contém um método de teste, `testAlgumaCoisa()`. Chamando o método de conveniência `markTestIncomplete()` (que automaticamente traz uma exceção `PHPUnit_Framework_IncompleteTestError`) no método de teste, marcamos o teste como sendo incompleto.

Exemplo 9.1. Marcando um teste como incompleto

```
<?php
class ExemploTest extends PHPUnit_Framework_TestCase
{
public function testAlgumaCoisa()
{
// Opcional: Teste alguma coisa aqui, se quiser
$this->assertTrue(TRUE, 'Já deveria funcionar.');

// Pare aqui e marque este teste como incompleto.
$this->markTestIncomplete(
'Este teste ainda não foi implementado.'
);
}
}
?>
```

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 3.7.0 by Sebastian Bergmann.

I
```

```
Time: 0 seconds, Memory: 3.75Mb

There was 1 incomplete test:

1) ExemploTest::testAlgumaCoisa
Este teste ainda não foi implementado.

/home/sb/SampleTest.php:12
OK, but incomplete or skipped tests!
Tests: 1, Assertions: 1, Incomplete: 1.phpunit --verbose ExemploTest
PHPUnit 3.7.0 by Sebastian Bergmann.

I

Time: 0 seconds, Memory: 3.75Mb

There was 1 incomplete test:

1) ExemploTest::testAlgumaCoisa
Este teste ainda não foi implementado.

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

Tabela 9.1, “API para Testes Incompletos” mostra a API para marcar testes como incompletos.

Tabela 9.1. API para Testes Incompletos

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

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 vários 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.

Exemplo 9.2, “Pulando um teste” mostra uma classe de caso de teste, `BancoDeDadosTest`, que contém um método de teste, `testConnection()`. No método modelo da classe do caso de teste `setUp()`, verificamos se uma extensão MySQLi está disponível e usamos o método `markTestSkipped()`para pular o teste caso contrário.

Exemplo 9.2. Pulando um teste

```
<?php
class BancoDeDadosTest extends PHPUnit_Framework_TestCase
{
protected function setUp()
{
if (!extension_loaded('mysqli')) {
$this->markTestSkipped(
'A extensão MySQLi não está disponível.'
);
}
}
```

```
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 3.7.0 by Sebastian Bergmann.

S

Time: 0 seconds, Memory: 3.75Mb

There was 1 skipped test:

1) BancoDeDadosTest::testConnection
A extensão MySQLi não está disponível.

/home/sb/BancoDeDadosTest.php:9
OK, but incomplete or skipped tests!
Tests: 1, Assertions: 0, Skipped: 1.phpunit --verbose BancoDeDadosTest
PHPUnit 3.7.0 by Sebastian Bergmann.

S

Time: 0 seconds, Memory: 3.75Mb

There was 1 skipped test:

1) BancoDeDadosTest::testConnection
A extensão MySQLi não está disponível.

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

Tabela 9.2, “API para Pular Testes” mostra a API para pular testes.

Tabela 9.2. API para Pular Testes

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

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.

Tabela 9.3. Possíveis usos para @requires

Tipo	Valores Possíveis	Exemplos	Outro exemplo
`PHP`	Qualquer identificador de versão do PHP	@requires PHP 5.3.3	@requires PHP 5.4-dev

Tipo	Valores Possíveis	Exemplos	Outro exemplo
`PHPUnit`	Qualquer identificador de versão do PHPUnit	@requires PHPUnit 3.6.3	@requires PHPUnit 3.7
`função`	Qualquer parâmetro válido para function_exists [http://php.net/function_exists]	@requires function imap_open	@requires function ReflectionMethod::setAccessible
`extensão`	Qualquer nome de extensão	@requires extension mysqli	@requires extension curl

Exemplo 9.3. Pulando casos de teste usando @requires

```
<?php
/**
* @requires extension mysqli
*/
class BancoDeDadosTest extends PHPUnit_Framework_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 includes dependentes de versão na seção chamada “Suítes de Teste”

Capítulo 10. Dublês de Testes

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

> Às vezes é muito difícil testar o sistema sob teste (SST) 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 ambiente realmente dependente (DOC), podemos substitui-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!
>
> —Gerard Meszaros

O método `getMock($nomeClasse)` fornecido pelo PHPUnit pode ser usado 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.

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

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.

Aviso

Por favor atente para o fato de que a gestão de parâmetros foi mudada. A implementação anterior clona todos os parâmetros de objetos. Isso não permite verificar se o mesmo objeto foi passado para um método ou não. Exemplo 10.14, “Testando se um método é chamado uma vez e com o objeto idêntico ao que foi passado” mostra onde a nova implementação pode ser útil. Exemplo 10.15, “Criando um objeto falso com clonagem de parâmetros ativada” mostra como voltar para o comportamento anterior.

Esboços (stubs)

A prática de substituir um objeto por um dublê de teste que (opcionalmente) retorna valores de retorno configurados é chamada de *esboçamento*. Você pode usar um *esboço* para "substituir um ambiente 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."

Exemplo 10.2, “Esboçando uma chamada de método para retornar um valor fixo” mostra como esboçar chamadas de método e configurar valores de retorno. Primeiro usamos o método `getMock()` que é fornecido pela classe `PHPUnit_Framework_TestCase` para configurar um esboço de objeto que parece com um objeto de `SomeClass` (Exemplo 10.1, “A classe que queremos esboçar”). Então usamos a Interface Fluente [http://martinfowler.com/bliki/FluentInterface.html] 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".

Exemplo 10.1. A classe que queremos esboçar

```
<?php
class AlgumaClasse
{
public function fazAlgumaCoisa()
{
// Faça algo.
}
}
?>
```

Exemplo 10.2. Esboçando uma chamada de método para retornar um valor fixo

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

class EsbocoTest extends PHPUnit_Framework_TestCase
{
public function testEsboco()
{
// Cria um esboço para a classe AlgumaClasse.
$esboco = $this->getMock('AlgumaClasse');

// Configura o esboço.
$esboco->expects($this->any())
->method('fazAlgumaCoisa')
->will($this->returnValue('foo'));

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

"Atrás dos bastidores" o PHPUnit automaticamente gera uma nova classe PHP que implementa o comportamento desejado quando um método `getMock()` é usado. A classe de dublê de teste gerada pode ser configurada através dos argumentos opcionais do método `getMock()`.

- Por padrão, todos os métodos das classes fornecidas são substituídos com um dublê de teste que apenas retorna `NULL` a menos que um valor de retorno seja configurado usando `will($this->returnValue())`, por exemplo.
- Quando o segundo parâmetro (opcional) é fornecido, apenas os métodos cujos nomes estão no vetor são substituídos com um dublê de teste configurável. O comportamento dos outros métodos não é alterado.
- O terceiro parâmetro (opcional) pode conter um vetor de parâmetros que é passado para o construtor da classe original (que por padrão não é substituído com a implementação falsa).
- O quarto parâmetro (opcional) pode ser usado para especificar um nome de classe para a classe de dublê de teste gerada.
- O quinto parâmetro (opcional) pode ser usado para desabilitar a chamada para o construtor da classe original.

- O sexto parâmetro (opcional) pode ser usado para desabilitar a chamada para o construtor do clone da classe original.

- O sétimo parâmetro (opcional) pode ser usado para desabilitar o `__autoload()` durante a geração da classe de dublê de teste.

Alternativamente, a Mock Builder API pode ser usada para configurar a classe de dublê de teste gerada. Exemplo 10.3, “Usando a Mock Builder API para configurar a classe de dublê de teste gerada” mostra um exemplo. Aqui temos uma lista dos métodos que podem ser usados na interface fluente do Mock Builder:

- `setMethods(vetor $metodos)` 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.

- `setConstructorArgs(vetor $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($nome)` 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 do clone da classe original.

- `disableAutoload()` pode ser usado para desabilitar o `__autoload()` durante a geração da classe de dublê de teste.

Exemplo 10.3. Usando a Mock Builder API para configurar a classe de dublê de teste gerada

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

class EsbocoTest extends PHPUnit_Framework_TestCase
{
public function testEsboco()
{
// Cria um esboço para a classe AlgumaClasse.
$esboco = $this->getMockBuilder('AlgumaClasse')
->disableOriginalConstructor()
->getMock();

// Configura o esboço.
$esboco->expects($this->any())
->method('fazAlgumaCoisa')
->will($this->returnValue('foo'));

// Chamar $esboco->fazAlgumaCoisa() agora vai retornar 'foo'.
$this->assertEquals('foo', $esboco->fazAlgumaCoisa());
}
}
?>
```

Às vezes você quer retornar um dos argumentos de uma chamada de método (inalterada) como o resultado de uma chamada ao método esboçado. Exemplo 10.4, “Esboçando uma chamada

de método para retornar um dos argumentos" mostra como você pode conseguir isso usando `returnArgument()` em vez de `returnValue()`.

Exemplo 10.4. Esboçando uma chamada de método para retornar um dos argumentos

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

class EsbocoTest extends PHPUnit_Framework_TestCase
{
public function testRetornaArgumentoEsboco()
{
// Cria um esboço para a classe AlgumaClasse.
$esboco = $this->getMock('AlgumaClasse');

// Configura o esboço.
$esboco->expects($this->any())
->method('fazAlgumaCoisa')
->will($this->returnArgument(0));

// $esboco->fazAlgumaCoisa('foo') retorna 'foo'.
$this->assertEquals('foo', $esboco->fazAlgumaCoisa('foo'));

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

Ao testar uma interface fluente, às vezes é útil fazer um método esboçado retornar uma referência ao objeto esboçado. Exemplo 10.5, "Esboçando uma chamada de método para retornar uma referência ao objeto esboçado" mostra como você pode usar `returnSelf()` para conseguir isso.

Exemplo 10.5. Esboçando uma chamada de método para retornar uma referência ao objeto esboçado

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

class EsbocoTest extends PHPUnit_Framework_TestCase
{
public function testRetornaEleMesmo()
{
// Cria um esboço para a classe AlgumaClasse.
$esboco = $this->getMock('AlgumaClasse');

// Configura o esboço.
$esboco->expects($this->any())
->method('fazAlgumaCoisa')
->will($this->returnSelf());

// $esboco->fazAlgumaCoisa() retorna $esboco.
$this->assertSame($esboco, $esboco->fazAlgumaCoisa());
}
}
?>
```

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 Exemplo 10.6, “Esboçando uma chamada de método para retornar o valor de um mapa” para ter um exemplo.

Exemplo 10.6. Esboçando uma chamada de método para retornar o valor de um mapa

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

class EsbocoTest extends PHPUnit_Framework_TestCase
{
public function testRetornaEsbocoMapaValores()
{
// Cria um esboço para a classe AlgumaClasse.
$esboco = $this->getMock('AlgumaClasse');

// Cria um mapa de argumentos para valores retornados.
$mapa = array(
array('a', 'b', 'c', 'd'),
array('e', 'f', 'g', 'h')
);

// Configura o esboço.
$esboco->expects($this->any())
->method('fazAlgumaCoisa')
->will($this->returnValueMap($mapa));

// $esboco->fazAlgumaCoisa() retorna valores diferentes dependendo
// dos argumentos fornecidos.
$this->assertEquals('d', $esboco->fazAlgumaCoisa('a', 'b', 'c'));
$this->assertEquals('h', $esboco->fazAlgumaCoisa('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 Exemplo 10.7, “Esboçando uma chamada de método para retornar um valor de um callback” para ter um exemplo.

Exemplo 10.7. Esboçando uma chamada de método para retornar um valor de um callback

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

class EsbocoTest extends PHPUnit_Framework_TestCase
{
public function testRetornaEsbocoCallback()
{
// Cria um esboço para a classe AlgumaClasse.
$esboco = $this->getMock('AlgumaClasse');

// Configura o esboço.
$esboco->expects($this->any())
->method('fazAlgumaCoisa')
->will($this->returnCallback('str_rot13'));

// $esboco->fazAlgumaCoisa($argumento) retorna str_rot13($argumento).
$this->assertEquals('fbzrguvat', $esboco->fazAlgumaCoisa('algo'));
```

```
}
}
?>
```

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 Exemplo 10.8, “Esboçando uma chamada de método para retornar uma lista de valores na ordem especificada” para ter um exemplo.

Exemplo 10.8. Esboçando uma chamada de método para retornar uma lista de valores na ordem especificada

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

class EsbocoTest extends PHPUnit_Framework_TestCase
{
public function testEsbocoChamadasConsecutivas()
{
// Cria um esboço para a classe AlgumaClasse.
$esboco = $this->getMock('AlgumaClasse');

// Configura o esboço.
$esboco->expects($this->any())
->method('fazAlgumaCoisa')
->will($this->onConsecutiveCalls(2, 3, 5, 7));

// $esboco->fazAlgumaCoisa() retorna um valor diferente de cada vez
$this->assertEquals(2, $esboco->fazAlgumaCoisa());
$this->assertEquals(3, $esboco->fazAlgumaCoisa());
$this->assertEquals(5, $esboco->fazAlgumaCoisa());
}
}
?>
```

Em vez de retornar um valor, um método esboçado também pode causar uma exceção. Exemplo 10.9, “Esboçando uma chamada de método para lançar uma exceção” mostra como usar `throwException()` para fazer isso.

Exemplo 10.9. Esboçando uma chamada de método para lançar uma exceção

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

class EsbocoTest extends PHPUnit_Framework_TestCase
{
public function testEsbocoLancaExcecao()
{
// Cria um esboço para a classe AlgumaClasse.
$esboco = $this->getMock('AlgumaClasse');

// Configura o esboço.
$esboco->expects($this->any())
->method('fazAlgumaCoisa')
->will($this->throwException(new Exception));

// $esboco->fazAlgumaCoisa() lança uma exceção
$esboco->fazAlgumaCoisa();
}
}
?>
```

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 `BancoDeDados` que implementa a interface `IBancoDeDados` . Então, você pode criar um esboço de implementação da `IBancoDeDados` 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.

Objetos Falsos

A prática de substituir um objeto por um dublê de teste que verifica expectativas, por exemplo assertando um método 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".

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. Exemplo 10.10, “As classes Sujeito e Observador que são parte do Sistema Sob Teste (SST)” mostra o código para as classes `Sujeito` e `Observador` que são parte do Sistema Sob Teste (SST).

Exemplo 10.10. As classes Sujeito e Observador que são parte do Sistema Sob Teste (SST)

```
<?php
class Sujeito
{
protected $observadores = array();

public function anexar(Observador $observador)
{
$this->observadors[] = $observador;
}

public function fazAlgumaCoisa()
{
// Faça algo.
// ...

// Notifica aos observadores que fizemos algo.
$this->notify('algo');
}

public function fazAlgumaCoisaRuim()
{
foreach ($this->observadores as $observador) {
$observador->reportError(42, 'Alguma coisa ruim aconteceu.', $this);
}
}

protected function notify($argumento)
{
```

```
foreach ($this->observadores as $observador) {
$observador->update($argumento);
}
}

// Outros métodos.
}

class Observador
{
public function update($argumento)
{
// Faça algo.
}

public function realatarErro($codigoErro, $mensagemErro, Sujeito $sujeito)
{
// Faça algo.
}

// Outros métodos.
}
?>
```

Exemplo 10.11, “Testando se um método é chamado uma vez e com o argumento especificado” mostra como usar um objeto falso para testar a interação entre os objetos `Sujeito` e `Observador`.

Primeiro usamos o método `getMock()` 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 `Observador` é substituído por uma implementação falsificada.

Exemplo 10.11. Testando se um método é chamado uma vez e com o argumento especificado

```
<?php
class SujeitoTest extends PHPUnit_Framework_TestCase
{
public function testObservadoresEstaoAtualizados()
{
// Cria uma falsificação para a classe Observador,
// apenas falsificando o método atualizar().
$observador = $this->getMock('Observador', array('atualizar'));

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

// Cria um objeto Sujeito e anexa a ele o objeto
// Observador falso.
$sujeito = new Subject;
$sujeito->attach($observador);

// Chama o método fazAlgumaCoisa() no objeto $sujeito
// no qual esperamos chamar o método atualizar()
// do objeto falso Observador, com a string 'algo'.
$sujeito->fazAlgumaCoisa();
}
}
?>
```

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

Exemplo 10.12. Testando se um método é chamado com um número de argumentos restringidos de formas diferentes

```
<?php
class SujeitoTest extends PHPUnit_Framework_TestCase
{
public function testErroRelatado()
{
// Cria uma falsificação para a classe Observador,
// falsificando o método reportError()
$observador = $this->getMock('Observador', array('relatarErro'));

$observador->expects($this->once())
->method('relatarErro')
->with($this->greaterThan(0),
$this->stringContains('Algo'),
$this->anything());

$sujeito = new Sujeito;
$sujeito->attach($observador);

// O método fazAlgumaCoisaRuim() deveria relatar um erro
// ao observador via método reportError()
$sujeito->fazAlgumaCoisaRuim();
}
}
?>
```

Tabela 4.3, “Restrições” mostra as restrições que podem ser aplicadas aos argumentos do método e Tabela 10.1, “Equiparadores” mostra os equiparadores que estão disponíveis para especificar o número de invocações.

Tabela 10.1. Equiparadores

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

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

Exemplo 10.13. Testando os métodos concretos de uma classe abstrata

```
<?php
abstract class ClasseAbstrata
{
public function metodoConcreto()
{
return $this->metodoAbstrato();
}

public abstract function metodoAbstrato();
}

class ClasseAbstrataTest extends PHPUnit_Framework_TestCase
{
public function testMetodoConcreto()
{
$esboco = $this->getMockForAbstractClass('ClasseAbstrata');
$esboco->expects($this->any())
->method('metodoAbstrato')
->will($this->returnValue(TRUE));

$this->assertTrue($esboco->metodoConcreto());
}
}
?>
```

Exemplo 10.14. Testando se um método é chamado uma vez e com o objeto idêntico ao que foi passado

```
<?php
class FooTest extends PHPUnit_Framework_TestCase
{
public function testObjetoIdenticoPassado()
{
$objetoEsperado = new stdClass;

$falso = $this->getMock('stdClass', array('foo'));
$falso->expects($this->once())
->method('foo')
->with($this->identicalTo($objetoEsperado));

$falso->foo($objetoEsperado);
}
}
?>
```

Exemplo 10.15. Criando um objeto falso com clonagem de parâmetros ativada

```
<?php
class FooTest extends PHPUnit_Framework_TestCase
{
public function testObjetoIdenticoPassado()
{
$argumentosClonados = true;

$falso = $this->getMock(
'stdClass',
```

```
array(),
array(),
'',
FALSE,
TRUE,
TRUE,
$argumentosClonados
);

// ou usando o mock builder
$falso = $this->getMockBuilder('stdClass')->enableArgumentCloning()->getMock();

// agora você falsifica seus parâmetros de clones de modo que a restrição identicalTo vá
}
}
?>
```

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 fáceis o esboço e falsificação dos serviços web, o `getMockFromWsdl()` pode ser usado da mesma forma que o `getMock()` (vide 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.

Exemplo 10.16, “Esboçando um serviço web” mostra como `getMockFromWsdl()` pode ser usado para esboçar, por exemplo, o erviço web descrito em `GoogleSearch.wsdl`.

Exemplo 10.16. Esboçando um serviço web

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

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

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

$resultado = new StdClass;
$resultado->documentFiltering = FALSE;
$resultado->searchComments = '';
$resultado->estimatedTotalResultsCount = 378000;
$resultado->estimateIsExact = FALSE;
$resultado->resultElements = array($elemento);
$resultado->searchQuery = 'PHPUnit';
```

```
$resultado->startIndex = 1;
$resultado->endIndex = 1;
$resultado->searchTips = '';
$resultado->directoryCategories = array();
$resultado->searchTime = 0.248822;

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

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

Esboçando o Sistema de Arquivos

vfsStream [https://github.com/mikey179/vfsStream] é um stream wrapper [http://www.php.net/streams] para um sistema de arquivos virtual [http://en.wikipedia.org/wiki/Virtual_file_system] que pode ser útil em testes unitários para falsificar um sistema de arquivos real.

Para instalar o vfsStram, o canal PEAR (`pear.bovigo.org`) que é usado para esta distribuição precisa ser registrado com o ambiente PEAR local.

```
pear channel-discover pear.bovigo.org
```

Isso só precisa ser feito uma vez. Agora o Instalador PEAR pode ser usado para instalar o vfsStream:

```
pear install bovigo/vfsStream-beta
```

Exemplo 10.17, “Uma classe que interage com um sistema de arquivos” mostra a classe que interage com o sistema de arquivos.

Exemplo 10.17. Uma classe que interage com um sistema de arquivos

```
<?php
class Exemplo
{
protected $id;
protected $diretorio;

public function __construct($id)
{
```

```
$this->id = $id;
}

public function setDiretorio($diretorio)
{
$this->diretorio = $diretorio . SEPARADOR_DE_DIRETORIO . $this->id;

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

Sem um sistema de arquivos virtual como o vfsStream não poderíamos testar o método `setDirectory()` isolado de influências externas (veja Exemplo 10.18, “Testando uma classe que interage com o sistema de arquivos”).

Exemplo 10.18. Testando uma classe que interage com o sistema de arquivos

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

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

public function testDiretorioFoiCriado()
{
$exemplo = new Exemplo('id');
$this->assertFalse(file_exists(dirname(__FILE__) . '/id'));

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

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

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.

Exemplo 10.19, “Falsificando o sistema de arquivos em um teste para a classe que interage com o sistema de arquivos” 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.

Exemplo 10.19. Falsificando o sistema de arquivos em um teste para a classe que interage com o sistema de arquivos

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

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

public function testDiretorioFoiCriado()
{
$exemplo = new Exemplo('id');
$this->assertFalse(vfsStreamWrapper::getRoot()->hasChild('id'));

$exemplo->setDirectory(vfsStream::url('diretorioExemplo'));
$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 11. Práticas de Teste

> 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 pode querer escrever testes que te darão retorno com informação.
>
> —Erich Gamma

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 [http://martinfowler.com/bliki/DefinitionOfRefactoring.html]com segurança. 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 é discutida no próximo capítulo.

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; parece que o conserto de um 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 computando 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 consertar o defeito, você realmente tê-lo consertado, porque o novo teste reduz a probabilidade de desfazer o conserto com futuras modificações no código. Todos os testes que você escreveu antes reduzem a probabilidade de causar diferentes problemas inadvertidamente.

Testes unitários oferecem muitas vantagens:

- Testar dá aos autores e revisores dos códigos 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. Isso vai permitir ao projeto realizar [...] maiores mudanças arquitetônicas [...] rápida e confiavelmente.

—Benjamin Smedberg

Capítulo 12. Desenvolvimento Guiado por Testes

Testes Unitários são uma parte vital de várias práticas de desenvolvimento e processos como Programar Testes Primeiro, Programação Extrema [http://en.wikipedia.org/wiki/Extreme_Programming], e Desenvolvimento Guiado por Testes [http://en.wikipedia.org/wiki/Test-driven_development]. Eles também permitem um Projeto-por-Contrato [http://en.wikipedia.org/wiki/Design_by_Contract] em linguagens de programação que não suportam essa metodologia com construções de linguagem.

Você pode usar o PHPUnit para escrever testes uma vez que você tiver terminado de programar. Porém quanto antes um teste é escrito depois de um erro ser introduzido, mais valioso o teste se torna. Então em vez de escrever os testes meses depois do código estar "completo", podemos escrever os testes dias ou horas ou minutos depois da possível introdução de um defeito. Por que parar aqui? Por que não escrever os testes um pouco antes da possível introdução de um defeito?

Programar Testes Primeiro, que é parte da Programação Extrema e Desenvolvimento Guiado por Testes, constrói essa ideia e leva isso ao extremo. Com o poder computacional de hoje, temos a oportunidade de executar milhares de testes, milhares de vezes por dia. Nós podemos usar o retorno de todos esses testes para programar em pequenos passos, cada qual levando consigo a segurança de um novo teste automatizado além de todos os testes que vieram antes dele. Os testes são como ancoradores, assegurando que você, não importa o que aconteça, uma vez que tiver feito progresso só poderá cair até aqui.

Quando você escreve o teste primeiro ele não poderá ser executado, pois você estará chamando objetos e métodos que ainda não foram programados. Isso pode parecer estranho no começo, mas depois de um tempo você vai se acostumar. Pense em Programar Testes Primeiro como uma abordagem pragmática para seguir o princípio da programação orientada a objetos de programar para uma interface em vez de programar para uma implementação: enquanto você está escrevendo o teste você está pensando sobre a interface do objeto que está testando -- como esse objeto se parece do lado de fora. Quando você vai fazer o teste realmente funcionar, você está pensando sobre pura implementação. A interface é consertada a partir da falha desse teste.

> O ponto do Desenvolvimento Guiado por Testes [http://en.wikipedia.org/wiki/Test-driven_development] é dirigir a funcionalidade que o programa realmente precisa, em vez do que o programador imagina que provavelmente deverá ter. A forma que isso é feito pode parecer contra-intuitiva no começo, se não totalmente boba, mas não apenas faz sentido, como também rapidamente se torna uma forma natural e elegante de desenvolver programas.
>
> —Dan North

O que segue é necessariamente uma introdução abreviada ao Desenvolvimento Guiado por Testes. Você pode explorar o tópico posteriormente em outros livros, como *Test-Driven Development* [Beck2002] de Kent Beck ou *A Practical Guide to Test-Driven Development* de Dave Astels[Astels2003].

Exemplo da Conta Bancária

Nesta seção, vamos dar uma olhada no exemplo de uma classe que representa uma conta bancária. O contrato para a classe `ContaBancaria`não apenas exige métodos get e set para o saldo da conta, mas também métodos para depositar e sacar dinheiro. Também especifica as duas seguintes condições que devem ser garantidas:

- O saldo inicial da conta bancária deve ser zero.
- O saldo da conta bancária não pode se tornar negativo.

Nós escrevemos os testes para a classe `ContaBancaria` antes de escrevermos o código propriamente dito da classe. Nós usamos as condições do contrato como base para os testes e nomeamos os testes conforme o mesmo, como mostrado em Exemplo 12.1, “Testes para a classe ContaBancaria”.

Exemplo 12.1. Testes para a classe ContaBancaria

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

class ContaBancariaTest extends PHPUnit_Framework_TestCase
{
protected $cb;

protected function setUp()
{
$this->cb = new ContaBancaria;
}

public function testSaldoInicialZero()
{
$this->assertEquals(0, $this->cb->getSaldo());
}

public function testSaldoNaoPodeFicarNegativo()
{
try {
$this->cb->sacarDinheiro(1);
}

catch (ExcecaoContaBancaria $e) {
$this->assertEquals(0, $this->cb->getSaldo());

return;
}

$this->fail();
}

public function testSaldoNaoPodeFicarNegativo2()
{
try {
$this->cb->depositarDinheiro(-1);
}

catch (ExcecaoContaBancaria $e) {
$this->assertEquals(0, $this->cb->getSaldo());

return;
}

$this->fail();
}
}
?>
```

Agora escrevemos somente o mínimo de código necessário para o primeiro teste, `testSaldoInicialZero()`, passar. Em nosso exemplo essa quantidade equivale a implementar o método `getBalance()` da classe `ContaBancaria`, como mostrado em Exemplo 12.2, “Código necessário para que o testSaldoInicialZero() passe”.

Exemplo 12.2. Código necessário para que o testSaldoInicialZero() passe

```
<?php
class ContaBancaria
{
protected $saldo = 0;

public function getSaldo()
{
return $this->saldo;
}
}
?>
```

O teste para a primeira condição do contrato agora passa, mas os testes para a segunda condição do contrato falham, porque ainda temos que implementar os métodos que esses testes chamam.

```
PHPUnit 3.7.0 by Sebastian Bergmann.

.
Fatal error: Call to undefined method ContaBancaria::sacarDinheiro()phpunit ContaBancari
PHPUnit 3.7.0 by Sebastian Bergmann.

.
Fatal error: Call to undefined method ContaBancaria::sacarDinheiro()
```

Para que os testes que asseguram a segunda condição do contrato passem, agora precisamos implementar os métodos `sacarDinheiro()`, `depositarDinheiro()`, e `setSaldo()` como mostrado em Exemplo 12.3, “A classe ContaBancaria completa”. Esses métodos são escritos de forma a gerar uma `ExcecaoContaBancaria` quando forem chamados com valores ilegais que poderiam violar as condições do contrato.

Exemplo 12.3. A classe ContaBancaria completa

```
<?php
class ContaBancaria
{
protected $saldo = 0;

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

protected function setSaldo($saldo)
{
if ($saldo >= 0) {
$this->saldo = $saldo;
} else {
throw new ExcecaoContaBancaria;
}
}

public function depositarDinheiro($saldo)
{
$this->setSaldo($this->getSaldo() + $saldo);

return $this->getSaldo();
}

public function sacarDinheiro($saldo)
```

```
{
$this->setSaldo($this->getSaldo() - $saldo);

return $this->getSaldo();
}
}
?>
```

Agora os testes que asseguram a segunda condição do contrato também passam:

```
PHPUnit 3.7.0 by Sebastian Bergmann.

...

Time: 0 seconds


OK (3 tests, 3 assertions)phpunit ContaBancariaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

...

Time: 0 seconds

OK (3 tests, 3 assertions)
```

Alternativamente, você pode usar os métodos estáticos de asserção fornecidos pela classe `PHPUnit_Framework_Assert` para escrever as condições do contrato como asserções do tipo Projeto-por-Contrato em seu código, como mostrado em Exemplo 12.4, “A classe ContaBancaria com asserções projetadas-por-contrato”. Quando uma dessas asserções falha, surge uma exceção `PHPUnit_Framework_AssertionFailedError` Com essa abordagem, você escreve menos código para as verificações das condições do contrato e os testes se tornam mais legíveis. Porém, você adiciona uma dependência em tempo de execução ao PHPUnit no seu projeto.

Exemplo 12.4. A classe ContaBancaria com asserções projetadas-por-contrato

```
<?php
class ContaBancaria
{
private $saldo = 0;

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

protected function setSaldo($saldo)
{
PHPUnit_Framework_Assert::assertTrue($saldo >= 0);

$this->saldo = $saldo;
}

public function depositarDinheiro($quantia)
{
PHPUnit_Framework_Assert::assertTrue($quantia >= 0);

$this->setSaldo($this->getSaldo() + $quantia);

return $this->getSaldo();
}
```

```
public function sacarDinheiro($quantia)
{
PHPUnit_Framework_Assert::assertTrue($quantia >= 0);
PHPUnit_Framework_Assert::assertTrue($this->saldo >= $quantia);

$this->setSaldo($this->getSaldo() - $quantia);

return $this->getSaldo();
}
}
?>
```

Escrevendo as condições do contrato nos testes, temos usado o conceito Projeto-por-Contrato para programar a classe `ContaBancaria` Então escrevemos, seguindo a abordagem Programe Testes Primeiro, o código precisava fazer os testes passarem. Porém esquecemos de escrever testes que chamem `setSaldo()`, `depositarDinheiro()`, e `sacarDinheiro()` com valores legais que não violem as condições do contrato. Precisamos de um meio para testar nossos testes ou pelo menos medir sua qualidade. Tal meio é a análise da informação de cobertura-de-código que discutiremos a seguir.

Capítulo 13. Desenvolvimento Guiado por Comportamento

Em [Astels2006], Dave Astels aponta o seguinte:

- Programação Extrema [http://en.wikipedia.org/wiki/Extreme_Programming] tinha originalmente a regra de testar tudo que pudesse possivelmente quebrar.
- Porém agora a prática dos testes na Programação Extrema evoluiu para o Desenvolvimento Guiado por Testes [http://en.wikipedia.org/wiki/Test-driven_development] (veja Capítulo 12, *Desenvolvimento Guiado por Testes*).
- Mas as ferramentas ainda forçam os desenvolvedores a pesar em termos de testes e asserções em vez de especificações.

> ### Então se não é sobre testes, é sobre o quê?
>
> É sobre descobrir o quê você está tentando fazer antes de sair desesperado tentando fazê-lo. Você escreve uma especificação que resolve um pequeno aspecto do comportamento de uma forma concisa, inequívoca e executável. É simples assim. Isso significa que você escreve testes? Não. Isso significa que você escreve especificações sobre o quê o seu código terá que fazer. Significa que você especifica o comportamento do seu código antes da hora. Mas não muito antes. De fato, logo antes de você escrever o código é melhor, porque é quando você tem o máximo de informações à mão até chegar a esse ponto. Como no DGT bem feito, você trabalha com pequenos incrementos... especificando um pequeno aspecto do comportamento de cada vez, então implementando-o.
>
> Quando você perceber que é tudo uma questão de especificar os comportamentos e não de escrever testes, seu ponto de vista muda. Repentinamente a ideia de ter uma classe de teste para cada uma das suas classes de produção se torna ridiculamente limitadora. E a ideia de testar cada um de seus métodos com seu próprio método de testes (uma relação 1-1) se torna hilária.
>
> —Dave Astels

O foco do Desenvolvimento Guiado por Comportamento [http://en.wikipedia.org/wiki/Behavior_driven_development] a linguagem e as interações usadas no processo de desenvolvimento de programas. Desenvolvedores Guiados por Comportamento usam suas linguagens nativas em combinação com a linguagem ubíqua de Design Guiado por Domínio [http://en.wikipedia.org/wiki/Domain_driven_design] para descrever o propósito e o benefício de seus códigos. Isso permite aos desenvolvedores focarem-se no motivo do código ser criado, em vez dos detalhes técnicos, minimizando a tradução entre a linguagem técnica na qual o código é escrito e a linguagem de domínio falada pelos especialistas em domínios".

A Classe `PHPUnit_Extensions_Story_TestCase` adiciona um framework de história que facilita a definição de uma Linguagem de Domínio Específico [http://en.wikipedia.org/wiki/Domain-specific_programming_language] para Desenvolvimento Guiado por Comportamento. Pode ser instalada assim:

```
pear install phpunit/PHPUnit_Story
```

Dentro de um *cenário*, `given()`, `when()`, e `then()` representam um *passo* cada um. `and()` é do mesmo tipo do passo anterior. Os seguintes métodos são declarados `abstract` no `PHPUnit_Extensions_Story_TestCase` e precisam ser implementados:

- `runGiven(&$mundo, $acao, $argumentos)`

  ...

- `runWhen(&$mundo, $acao, $argumentos)`

  ...

- `runThen(&$mundo, $acao, $argumentos)`

  ...

Exemplo do Jogo de Boliche

Nesta seção vamos dar uma olhada no exemplo de uma classe que calcula a pontuação para um jogo de boliche. As regras são as seguintes:

- O jogo consiste de 10 quadros.
- Em cada quadro o jogador tem duas oportunidades de derrubar 10 pinos.
- A pontuação para um quadro é o número total de pinos derrubados, mais bônus para strikes e spares.
- Um spare é quando um jogador derruba todos os 10 pinos em duas tentativas.

  O bônus para esse quadro é o número de pinos derrubados na próxima rodada.

- Um strike é quando um jogador derruba todos os pinos na primeira tentativa.

  O bônus para esse quadro é o valor das duas próximas bolas jogadas.

Exemplo 13.1, “Especificação para a classe JogoDeBoliche” mostra como as regras acima podem ser escritas como cenários de especificação usando o `PHPUnit_Extensions_Story_TestCase`.

Exemplo 13.1. Especificação para a classe JogoDeBoliche

```
<?php
require_once 'PHPUnit/Extensions/Story/TestCase.php';
require_once 'JogoBoliche.php';

class JogoBolicheSpec extends PHPUnit_Extensions_Story_TestCase
{
/**
* @scenario
*/
public function pontuacaoPorJogarNaCanaletaEh0()
{
$this->given('Novo jogo')
->then('Pontuação deve ser', 0);
}

/**
* @scenario
*/
public function pontuacaoParaTodosUmEh20()
{
$this->given('Novo jogo')
->when('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
```

```
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->and('Jogador arremessa', 1)
->then('Pontuação deve ser', 20);
}

/**
* @scenario
*/
public function pontuacaoParaUmSpareE3Eh16()
{
$this->given('Novo jogo')
->when('Jogador arremessa', 5)
->and('Jogador arremessa', 5)
->and('Jogador arremessa', 3)
->then('Pontuação deve ser', 16);
}

/**
* @scenario
*/
public function pontuacaoParaUmStrikeE3E4Eh24()
{
$this->given('Novo jogo')
->when('Jogador arremessa', 10)
->and('Jogador arremessa', 3)
->and('Jogador arremessa', 4)
->then('Pontuação deve ser', 24);
}

/**
* @scenario
*/
public function pontuacaoParaJogoPerfeitoEh300()
{
$this->given('Novo jogo')
->when('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->and('Jogador arremessa', 10)
->then('Pontuação deve ser', 300);
}

public function runGiven(&$mundo, $acao, $argumentos)
{
switch($action) {
case 'New game': {
$world['game'] = new BowlingGame;
$world['rolls'] = 0;
}
```

```
break;

default: {
return $this->notImplemented($action);
}
}
}

public function runWhen(&$mundo, $acao, $argumentos)
{
switch($acao) {
case 'Jogador arremessa': {
$mundo['jogo']->arremesso($argumentos[0]);
$mundo['arremessos']++;
}
break;

default: {
return $this->notImplemented($acao);
}
}
}

public function runThen(&$mundo, $acao, $argumentos)
{
switch($acao) {
case 'Pontuação deve ser': {
for ($i = $mundo['rolls']; $i < 20; $i++) {
$mundo['jogo']->arremesso(0);
}

$this->assertEquals($argumentos[0], $mundo['jogo']->pontuacao());
}
break;

default: {
return $this->notImplemented($acao);
}
}
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

EspecJogoBoliche
 [x] Pontuação por jogar na canaleta eh 0

   Given Novo jogo
    Then Pontuação deve ser 0

 [x] Pontuação para todos um eh 20

   Given Novo jogo
    When Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
```

```
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
    Then Pontuação deve ser 20

 [x] Pontuação para um spare e 3 eh 16

   Given Novo jogo
    When Jogador arremessa 5
     and Jogador arremessa 5
     and Jogador arremessa 3
    Then Pontuação deve ser 16

 [x] Pontuação para um strike e 3 e 4 eh 24

   Given Novo jogo
    When Jogador arremessa 10
     and Jogador arremessa 3
     and Jogador arremessa 4
    Then Pontuação deve ser 24

 [x] Pontuação para jogo perfeito eh 300

   Given Novo jogo
    When Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
     and Jogador arremessa 10
    Then Pontuação deve ser 300

Scenarios: 5, Failed: 0, Skipped: 0, Incomplete: 0.phpunit --printer PHPUnit_Extensions_
PHPUnit 3.7.0 by Sebastian Bergmann.

EspecJogoBoliche
 [x] Pontuação por jogar na canaleta eh 0

   Given Novo jogo
    Then Pontuação deve ser 0

 [x] Pontuação para todos um eh 20

   Given Novo jogo
    When Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
     and Jogador arremessa 1
```

```
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
    and Jogador arremessa 1
   Then Pontuação deve ser 20

[x] Pontuação para um spare e 3 eh 16

  Given Novo jogo
   When Jogador arremessa 5
    and Jogador arremessa 5
    and Jogador arremessa 3
   Then Pontuação deve ser 16

[x] Pontuação para um strike e 3 e 4 eh 24

  Given Novo jogo
   When Jogador arremessa 10
    and Jogador arremessa 3
    and Jogador arremessa 4
   Then Pontuação deve ser 24

[x] Pontuação para jogo perfeito eh 300

  Given Novo jogo
   When Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
    and Jogador arremessa 10
   Then Pontuação deve ser 300

Scenarios: 5, Failed: 0, Skipped: 0, Incomplete: 0.
```

Capítulo 14. Análise de Cobertura de Código

> A beleza de testar não se encontra no esforço, mas na eficiência.
>
> Saber o que deveria ser testado é lindo, e saber o que está sendo testado é lindo.
>
> —Murali Nandigama

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. Isso ajuda a responder questões como:

- Como você descobre o código que ainda não foi testado -- ou, em outras palavras, ainda não foi *coberto* por um teste?
- Como você mede o quanto os testes estão completos?

Um exemplo sobre o quê podem significar as estatísticas de cobertura de código é que se existir um método com 100 linhas de código, e apenas 75 dessas linhas são realmente executadas quando os testes são executados, então considera-se que o método tem uma cobertura de código de 75 porcento.

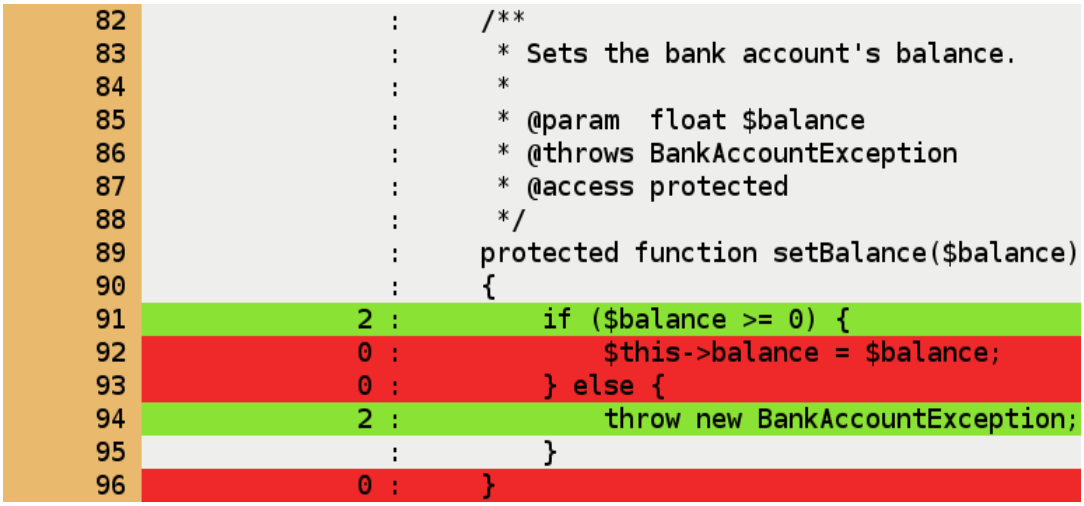
A funcionalidade de cobertura de código do PHPUnit faz uso do componente PHP_CodeCoverage [http://github.com/sebastianbergmann/php-code-coverage] que por sua vez aproveita a funcionalidade da cobertura de declarações fornecida pela extensão Xdebug [http://www.xdebug.org/] para PHP.

Deixe-nos gerar um relatório de cobertura de código para a classe `ContaBancaria` de Exemplo 12.3, “A classe ContaBancaria completa”.

```
PHPUnit 3.7.0 by Sebastian Bergmann.

...

Time: 0 seconds

OK (3 tests, 3 assertions)

Generating report, this may take a moment.phpunit --coverage-html ./report ContaBancaria
PHPUnit 3.7.0 by Sebastian Bergmann.

...

Time: 0 seconds

OK (3 tests, 3 assertions)

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

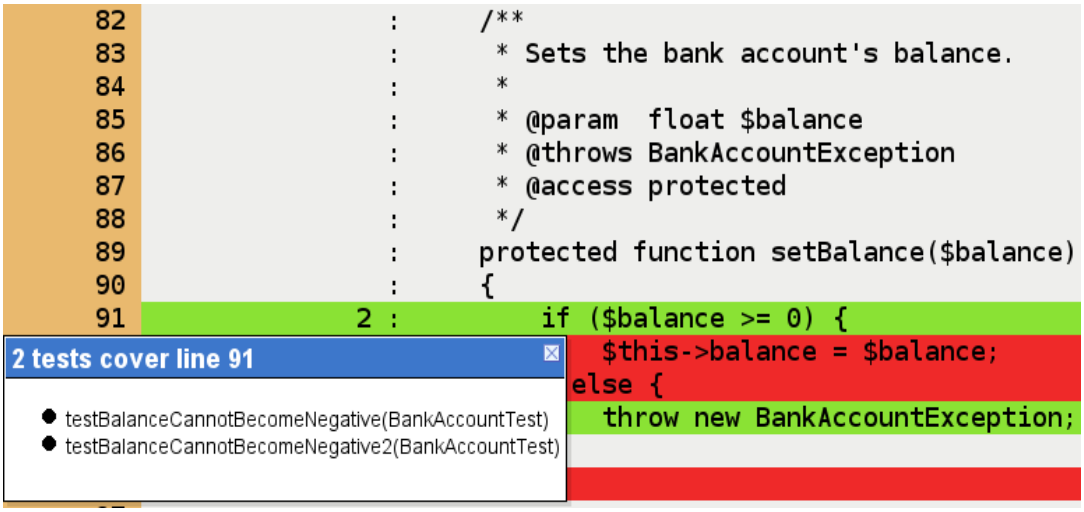
Figura 14.1, “Cobertura de Código para `setSaldo()`” mostra um resumo do relatório de Cobertura de Código. Linhas de código que foram executadas enquanto os testes executavam estão destacadas em verde, linhas de código que são executáveis mas não foram executadas estão destacadas em vermelho, e o "código morto" está destacado em cinza. O número à esquerda da linha de código atual indica quantos testes cobrem aquela linha.

Figura 14.1. Cobertura de Código para `setSaldo()`



Clicando no número da linha de uma linha coberta abrirá um painel (veja Figura 14.2, “Painel com informações sobre cobertura de testes”) que mostra os casos de testes que cobrem esta linha.

Figura 14.2. Painel com informações sobre cobertura de testes



O relatório de cobertura de código para nosso exemplo `ContaBancaria` mostra que não temos quaisquer testes ainda que chamem os métodos `setBalance()`, `depositMoney()`, e `withdrawMoney()` com valores legais. Exemplo 14.1, “Teste perdido para conseguir completa cobertura de código” mostra um teste que pode ser adicionado à classe de caso de teste `ContaBancariaTest` para cobrir completamente a classe `ContaBancaria`.

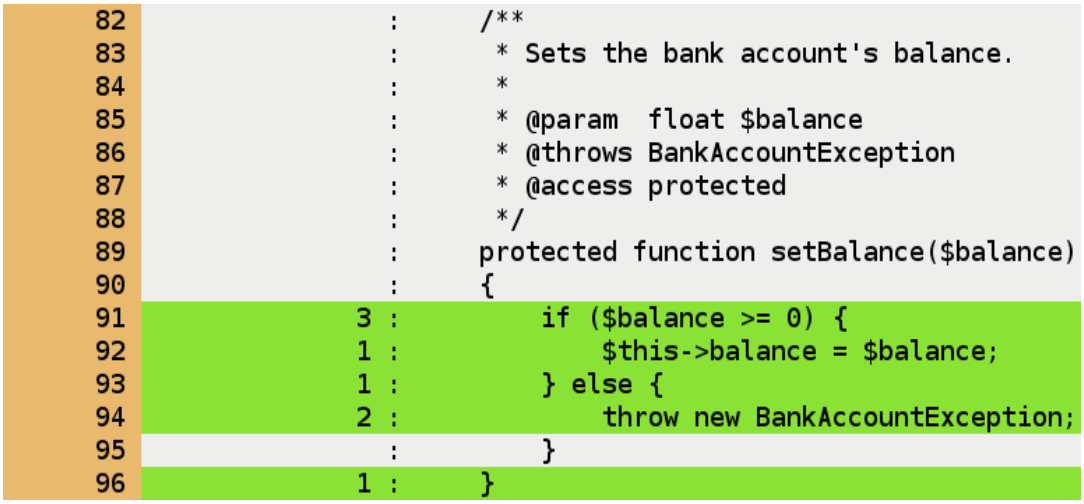
Exemplo 14.1. Teste perdido para conseguir completa cobertura de código

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

class ContaBancariaTest extends PHPUnit_Framework_TestCase
{
// ...

public function testDepositarSacarDinheiro()
{
$this->assertEquals(0, $this->cb->getSaldo());
$this->cb->depositarDinheiro(1);
$this->assertEquals(1, $this->cb->getSaldo());
```

```
$this->cb->sacarDinheiro(1);
$this->assertEquals(0, $this->cb->getSaldo());
}
}
?>
```

Figura 14.3, “Cobertura de Código para `setSaldo()` com teste adicional” mostra a cobertura de código para o método `setSaldo()` com o teste adicional.

Figura 14.3. Cobertura de Código para `setSaldo()` com teste adicional



Especificando métodos cobertos

A anotação `@covers` (veja Tabela B.1, “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. Exemplo 14.2, “Testes que especificam quais métodos querem cobrir” mostra um exemplo.

Exemplo 14.2. Testes que especificam quais métodos querem cobrir

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

class ContaBancariaTest extends PHPUnit_Framework_TestCase
{
protected $cb;

protected function setUp()
{
$this->cb = new ContaBancaria;
}

/**
* @covers ContaBancaria::getSaldo
*/
public function testSaldoInicialEhZero()
{
$this->assertEquals(0, $this->cb->getSaldo());
}

/**
* @covers ContaBancaria::sacarDinheiro
*/
```

```
public function testSaldoNaoPodeFicarNegativo()
{
try {
$this->cb->sacarDinheiro(1);
}

catch (ExcecaoContaBancaria $e) {
$this->assertEquals(0, $this->cb->getSaldo());

return;
}

$this->fail();
}

/**
* @covers ContaBancaria::depositarDinheiro
*/
public function testBalanceCannotBecomeNegative2()
{
try {
$this->cb->depositarDinheiro(-1);
}

catch (ExcecaoContaBancaria $e) {
$this->assertEquals(0, $this->cb->getSaldo());

return;
}

$this->fail();
}

/**
* @covers ContaBancaria::getSaldo
* @covers ContaBancaria::depositarDinheiro
* @covers ContaBancaria::sacarDinheiro
*/

public function testDepositarSacarDinheiro()
{
$this->assertEquals(0, $this->cb->getSaldo());
$this->cb->depositarDinheiro(1);
$this->assertEquals(1, $this->cb->getSaldo());
$this->cb->sacarDinheiro(1);
$this->assertEquals(0, $this->cb->getSaldo());
}
}
?>
```

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.

Exemplo 14.3. Um teste que especifica que nenhum método deve ser coberto

```
<?php
class IntegracaoLivroDeVisitasTest extends PHPUnit_Extensions_Database_TestCase
{
/**
* @coversNothing
*/
public function testAdicionaEntrada()
{
```

```
$livrodevisitas = new LivroDeVisitas();
$livrodevisitas->adicionaEntrada("suzy", "Olá mundo!");

$tabelaQuery = $this->getConnection()->criarTabelaQuery(
'livrodevisitas', 'SELECT * FROM livrodevisitas'
);
$expectedTable = $this->criarConjuntoDadosXmlPlano("livroEsperado.xml")
->getTabela("livrodevisitas");
$this->assertTablesEqual($tabelaEsperada, $tabelaQuery);
}
}
?>
```

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 permite que você o faça usando as anotações `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` e `@codeCoverageIgnoreEnd` como mostrado em Exemplo 14.4, “Usando as anotações `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` e `@codeCoverageIgnoreEnd`”.

Exemplo 14.4. Usando as anotações `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` e `@codeCoverageIgnoreEnd`

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

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

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

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

Incluindo e Excluindo Arquivos

Por padrão, todos os arquivos de código-fonte que contêm pelo menos uma linha de código que tenha sido executada (e apenas esses arquivos) são incluídos no relatório. Os arquivos de código-fonte que são incluídos no relatório podem ser filtrados usando uma abordagem de lista-negra ou lista-branca.

A lista-negra é pré-preenchida com todos os arquivos de código-fonte do próprio PHPUnit assim como os testes. Quando a lista-branca está vazia (padrão), a lista-negra é usada. Quando a lista-branca não está vazia, a lista-branca é usada. Cada arquivo na lista-branca é adicionado ao relatório de cobertura de código independentemente de ter sido executado ou não. Todas as linhas em tal arquivo, incluindo aquelas que não são executáveis, são contadas como não executadas.

Quando você define `processUncoveredFilesFromWhitelist="true"` na sua configuração do PHPUnit (veja “Incluindo e Excluindo Arquivos para Cobertura de Código”) então esses arquivos serão incluídos pelo PHP_CodeCoverage para calcular adequadamente o número de linhas executáveis.

Nota

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.

O arquivo de configuração XML do PHPUnit (veja “Incluindo e Excluindo Arquivos para Cobertura de Código”) pode ser usado para controlar as listas branca e negra. Usar uma lista-branca é a melhor prática recomendada para controlar a lista de arquivos incluídos no relatório de cobertura de código.

Casos Extremos

De modo geral é seguro dizer que o PHPUnit oferece a você uma informação de cobertura de código "baseada em linha", mas devido ao modo que a informação é coletada, existem alguns casos extremos dignos de atenção.

Exemplo 14.5.

```
<?php
// Por ser "baseado em linha" e não em declaração,
// uma linha sempre terá um estado de cobertura
if(false) esta_chamada_de_funcao_aparece_como_coberta();

// Devido ao modo que 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!
    tambem_vai_aparecer_como_coberta();

// Para evitar isso é necessário usar chaves
if(false){
    esta_chamada_nunca_aparecera_como_coberta();
}
?>
```

Capítulo 15. 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.

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 teste 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 "Saldo eh inicialmente 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 "Saldo nao pode ficar negativo" aparecerá apenas uma vez, assumindo-se que todos os testes foram bem-sucedidos.

Vamos dar uma olhada na documentação ágil gerada para a classe `ContaBancaria` class (from Exemplo 12.1, “Testes para a classe ContaBancaria”):

```
PHPUnit 3.7.0 by Sebastian Bergmann.

ContaBancaria
 [x] Saldo eh inicialmente zero
 [x] Saldo nao pode ficar negativophpunit --testdox ContaBancariaTest
PHPUnit 3.7.0 by Sebastian Bergmann.

ContaBancaria
 [x] Saldo eh inicialmente zero
 [x] Saldo nao pode ficar negativo
```

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 o 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.

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 16. Gerador de Esqueleto

O Gerador de Esqueleto do PHPUnit é uma ferramenta que pode gerar esqueletos de classes de teste a partir de classes códigos de produção e vice-versa. Pode ser instalado usando-se o seguinte comando:

```
pear install phpunit/PHPUnit_SkeletonGenerator
```

Gerando um Esqueleto de Classe de Caso de Teste

Quando você está escrevendo testes para um código existente, você tem que escrever os mesmos fragmentos de código como:

```
public function testMetodo()
{
}
```

de novo e de novo. O Gerador de Esqueleto do PHPUnit pode ajudá-lo analisando o código da classe existente e gerando um esqueleto de classe de caso de teste para ele.

Exemplo 16.1. A classe Calculadora

```
<?php
class Calculadora
{
public function soma($a, $b)
{
return $a + $b;
}
}
?>
```

O seguinte exemplo mostra como gerar um esqueleto de classe de caso de teste para uma classe chamada `Calculadora` (veja Exemplo 16.1, “A classe Calculadora”).

```

PHPUnit Skeleton Generator 1.0.0 by Sebastian Bergmann.

Wrote skeleton for "CalculadoraTest" to "/home/sb/CalculadoraTest.php".phpunit-skelgen -
PHPUnit Skeleton Generator 1.0.0 by Sebastian Bergmann.

Wrote skeleton for "CalculadoraTest" to "/home/sb/CalculadoraTest.php".
```

Para cada método na classe original, haverá um caso de teste incompleto (veja Capítulo 9, *Testes Incompletos e Pulados*) na classe de caso de teste gerada.

Classes separadas por nome e o Gerador de Esqueleto

Quando você está usando o gerador de esqueleto para gerar código baseado em uma classe que é declarada em um namespace [http://php.net/namespace] você tem que fornecer o nome qualificado da classe assim como o caminho para o arquivo-fonte em que está declarado.

Por exemplo, para a classe `Calculadora` que está declarada no namespace `project` você precisa invocar o gerador de esqueleto desta forma:

```

PHPUnit Skeleton Generator 1.0.0 by Sebastian Bergmann.
```

```
Wrote skeleton for "project\CalculadoraTest" to "/home/sb/CalculadoraTest.php".phpuni
PHPUnit Skeleton Generator 1.0.0 by Sebastian Bergmann.

Wrote skeleton for "project\CalculadoraTest" to "/home/sb/CalculadoraTest.php".
```

Abaixo está a saída da execução da classe de caso de teste gerada.

```
PHPUnit 3.7.0 by Sebastian Bergmann.

I

Time: 0 seconds, Memory: 3.50Mb

There was 1 incomplete test:

1) CalculadoraTest::testSoma
This test has not been implemented yet.

/home/sb/CalculadoraTest.php:38
OK, but incomplete or skipped tests!
Tests: 1, Assertions: 0, Incomplete: 1.phpunit --bootstrap Calculadora.php --verbose Cal
PHPUnit 3.7.0 by Sebastian Bergmann.

I

Time: 0 seconds, Memory: 3.50Mb

There was 1 incomplete test:

1) CalculadoraTest::testSoma
This test has not been implemented yet.

/home/sb/CalculadoraTest.php:38
OK, but incomplete or skipped tests!
Tests: 1, Assertions: 0, Incomplete: 1.
```

Você pode usar a anotação `@assert` no bloco de documentação de um método para gerar automaticamente testes simples, porém significativos, em vez de casos de testes incompletos. Exemplo 16.2, “A classe Calculadora com anotações @assert” mostra um exemplo.

Exemplo 16.2. A classe Calculadora com anotações @assert

```php
<?php
class Calculadora
{
/**
* @assert (0, 0) == 0
* @assert (0, 1) == 1
* @assert (1, 0) == 1
* @assert (1, 1) == 2
*/
public function soma($a, $b)
{
return $a + $b;
}
}
?>
```

Cada método na classe original é verificada por anotações `@assert`. Estas são transformadas em um código de teste como

```
/**
     * Generated from @assert (0, 0) == 0.
     */
    public function testSoma() {
        $o = new Calculadora;
        $this->assertEquals(0, $o->soma(0, 0));
    }
```

Abaixo está a saída da execução da classe de caso de teste gerada.

```
PHPUnit 3.7.0 by Sebastian Bergmann.

....

Time: 0 seconds, Memory: 3.50Mb

OK (4 tests, 4 assertions)phpunit --bootstrap Calculadora.php --verbose CalculadoraTest
PHPUnit 3.7.0 by Sebastian Bergmann.

....

Time: 0 seconds, Memory: 3.50Mb

OK (4 tests, 4 assertions)
```

Tabela 16.1, “Variantes suportadas da anotação @assert” mostra variantes suportadas para a anotação `@assert` e como elas podem ser transformadas em código de teste.

Tabela 16.1. Variantes suportadas da anotação @assert

Anotação	Transformada para
`@assert (...) == X`	`assertEquals(X, method(...))`
`@assert (...) != X`	`assertNotEquals(X, method(...))`
`@assert (...) === X`	`assertSame(X, method(...))`
`@assert (...) !== X`	`assertNotSame(X, method(...))`
`@assert (...) > X`	`assertGreaterThan(X, method(...))`
`@assert (...) >= X`	`assertGreaterThanOrEqual(X, method(...))`
`@assert (...) < X`	`assertLessThan(X, method(...))`
`@assert (...) <= X`	`assertLessThanOrEqual(X, method(...))`
`@assert (...) throws X`	`@expectedException X`

Gerando uma Classe Esqueleto de uma Classe de Caso de Teste

Quando você está fazendo Desenvolvimento Guiado por Teste (veja Capítulo 12, *Desenvolvimento Guiado por Testes*) e escreve seus testes antes do código que o teste exercita, o PHPUnit pode ajudá-lo a gerar esqueletos de classe das classes de casos de testes.

Seguindo a convenção de que os testes para uma classe `Unidade` são escritos em uma classe chamada `UnidadeTest`, a fonte da classe de caso de teste é pesquisada para encontrar variáveis que referenciem objetos da classe `Unidade` e analisa que métodos são chamados nesses objetos. Por

exemplo, dê uma olhada em Exemplo 16.4, “O esqueleto gerado da classe JogoBoliche” que foi gerado baseado na análise de Exemplo 16.3, “A classe JogoBolicheTest”.

Exemplo 16.3. A classe JogoBolicheTest

```
<?php
class JogoBolicheTest extends PHPUnit_Framework_TestCase
{
protected $jogo;

protected function setUp()
{
$this->jogo = new JogoBoliche;
}

protected function arremessarVarias($n, $pinos)
{
for ($i = 0; $i < $n; $i++) {
$this->jogo->arremessa($pinos);
}
}

public function testPontuacaoPorJogarNaCanaletaEh0()
{
$this->arremessarVarias(20, 0);
$this->assertEquals(0, $this->jogo->pontuacao());
}
}
?>
```

```
PHPUnit Skeleton Generator 1.0.0 by Sebastian Bergmann.

Wrote skeleton for "JogoBoliche" to "./JogoBoliche.php".phpunit-skelgen --class JogoBoli
PHPUnit Skeleton Generator 1.0.0 by Sebastian Bergmann.

Wrote skeleton for "JogoBoliche" to "./JogoBoliche.php".
```

Exemplo 16.4. O esqueleto gerado da classe JogoBoliche

```
<?php
/**
* Generated by PHPUnit_SkeletonGenerator on 2012-01-09 at 16:55:58.
*/
class JogoBoliche
{
/**
* @todo Implement arremessa().
*/
public function arremessa()
{
// Remove the following line when you implement this method.
throw new RuntimeException('Not yet implemented.');
}

/**
* @todo Implement pontuacao().
*/
public function pontuacao()
{
// Remove the following line when you implement this method.
throw new RuntimeException('Not yet implemented.');
```

```
}
}
?>
```

Abaixo está a saída da execução dos testes contra a classe gerada.

```
PHPUnit 3.7.0 by Sebastian Bergmann.

E

Time: 0 seconds, Memory: 3.50Mb

There was 1 error:

1) JogoBolicheTest::testPontuacaoPorJogarNaCanaletaEh0
RuntimeException: Not yet implemented.

/home/sb/JogoBoliche.php:13
/home/sb/JogoBolicheTest.php:14
/home/sb/JogoBolicheTest.php:20

FAILURES!
Tests: 1, Assertions: 0, Errors: 1.phpunit --bootstrap JogoBoliche.php JogoBolicheTest
PHPUnit 3.7.0 by Sebastian Bergmann.

E

Time: 0 seconds, Memory: 3.50Mb

There was 1 error:

1) JogoBolicheTest::testPontuacaoPorJogarNaCanaletaEh0
RuntimeException: Not yet implemented.

/home/sb/JogoBoliche.php:13
/home/sb/JogoBolicheTest.php:14
/home/sb/JogoBolicheTest.php:20

FAILURES!
Tests: 1, Assertions: 0, Errors: 1.
```

Capítulo 17. PHPUnit e Selenium

Servidor Selenium

Servidor Selenium [http://seleniumhq.org/] é uma ferramenta de testes que permite a você escrever testes automatizados de interface de usuário para aplicações web em qualquer linguagem de programação contra qualquer website HTTP usando um dos principais navegadores. Ele realiza tarefas automatizadas no navegador guiando seu processo através do sistema operacional. O Selenium executa os testes diretamente em um navegador, exatamente como os usuários reais fazem. Esses testes podem ser usados para ambos *testes de aceitação* (realizando testes de alto-nível no sistema integrado em vez de apenas testar cada unidade do sistema independentemente) e *testes de compatibilidade de navegador* (testando a aplicação web em diferentes sistemas operacionais e navegadores).

O único cenário suportado do PHPUnit_Selenium é o do servidor Selenium 2.x. O servidor pode ser acessado através da Api clássica Selenium RC, já presente no 1.x, ou com a API WebDriver (parcialmente implementada) do PHPUnit_Selenium 1.2.

A razão por trás dessa decisão é que o Selenium 2 é compatível e o Selenium RC não é mais mantido.

Instalação

Primeiro, instale o Servidor Selenium:

1. Baixe um arquivo de distribuição do Servidor Selenium [http://seleniumhq.org/download/].
2. Descompacte o arquivo de distribuição e copie o `selenium-server-standalone-2.9.0.jar` (verifique o sufixo da versão) para `/usr/local/bin`, por exemplo.
3. Inicie o Servidor Selenium executando **`java -jar /usr/local/bin/selenium-server-standalone-2.9.0.jar`**.

Segundo, instale o pacote PHPUnit_Selenium, necessário para acessar nativamente o Servidor Selenium do PHPUnit:

```
pear install phpunit/PHPUnit_Selenium
```

Agora podemos enviar comandos para o Servidor Selenium usando seu protocolo cliente/servidor.

PHPUnit_Extensions_Selenium2TestCase

O caso de teste `PHPUnit_Extensions_Selenium2TestCase` permite a você usar a API WebDriver (parcialmente implementada).

Exemplo 17.1, “Exemplo de uso para PHPUnit_Extensions_Selenium2TestCase” mostra como testar os conteúdos do elemento `<title>` do site `http://www.example.com/`.

Exemplo 17.1. Exemplo de uso para PHPUnit_Extensions_Selenium2TestCase

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

```
public function testTitle()
{
$this->url('http://www.exemplo.com/');
$this->assertEquals('Página WWW de Exemplo', $this->title());
}

}
?>
```

```
PHPUnit 3.6.10 by Sebastian Bergmann.

F

Time: 28 seconds, Memory: 3.00Mb

There was 1 failure:

1) WebTest::testTitle
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'Página WWW de Exemplo'
+'IANA — Domínio de Exemplo'

/home/giorgio/WebTest.php:13

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit WebTest
PHPUnit 3.6.10 by Sebastian Bergmann.

F

Time: 28 seconds, Memory: 3.00Mb

There was 1 failure:

1) WebTest::testTitle
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'Página WWW de Exemplo'
+'IANA — Domínio de Exemplo'

/home/giorgio/WebTest.php:13

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

Os comandos do Selenium2TestCase são implementados via __call(). Por favor, recorra ao teste de ponta a ponta para o PHPUnit_Extensions_Selenium2TestCase [https://github.com/sebastianbergmann/phpunit-selenium/blob/master/Tests/Selenium2TestCaseTest.php] para uma lista de cada característica suportada.

PHPUnit_Extensions_SeleniumTestCase

A extensão de caso de teste `PHPUnit_Extensions_SeleniumTestCase` implementa o protocolo cliente/servidor para conversar com o Servidor Selenium assim como métodos de asserção especializados para testes web.

Exemplo 17.2, “Exemplo de uso para PHPUnit_Extensions_SeleniumTestCase” mostra como testar os conteúdos do elemento `<title>` para o site `http://www.exemplo.com/`.

Exemplo 17.2. Exemplo de uso para PHPUnit_Extensions_SeleniumTestCase

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

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

public function testTitle()
{
$this->open('http://www.exemplo.com/');
$this->assertTitle('Página WWW de Exemplo');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 9 seconds, Memory: 6.00Mb

There was 1 failure:

1) WebTest::testTitle
Current URL: http://www.iana.org/dominios/exemplo/

Failed asserting that 'IANA — Domínios de Exemplo' matches PCRE pattern "/Página WWW de


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit WebTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 9 seconds, Memory: 6.00Mb

There was 1 failure:

1) WebTest::testTitle
Current URL: http://www.iana.org/dominios/exemplo/

Failed asserting that 'IANA — Domínios de Exemplo' matches PCRE pattern "/Página WWW de

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

Diferente da classe `PHPUnit_Framework_TestCase` classes de caso de teste que estendem o `PHPUnit_Extensions_SeleniumTestCase` têm que prover um método `setUp()`. Esse método é usado para configurar a sessão do Servidor Selenium. Veja Tabela 17.1, “API do Servidor Selenium: Setup” para uma lista de métodos que estão disponíveis para isso.

Tabela 17.1. API do Servidor Selenium: Setup

Método	Significado
`void setBrowser(string $browser)`	Define o navegador a ser usado pelo Servidor Selenium.
`void setBrowserUrl(string $browserUrl)`	Define a URL base para os testes.
`void setHost(string $host)`	Define o nome do host para a conexão do Servidor Selenium.
`void setPort(int $port)`	Define a porta de conexão para o Servidor Selenium
`void setTimeout(int $timeout)`	Define o timeout para a conexão do Servidor Selenium.
`void setSleep(int $seconds)`	Define o número de segundos que o cliente do Servidor Selenium deve esperar entre cada envio de comandos de ação para o Servidor Selenium.

O PHPUnit pode opcionalmente fazer uma captura de tela quando um teste do Selenium falha. Para habilitar isso, configure `$captureScreenshotOnFailure`, `$screenshotPath` e `$screenshotUrl` em sua classe de caso de teste como mostrado em Exemplo 17.3, “Capturando a tela quando um teste falha”.

Exemplo 17.3. Capturando a tela quando um teste falha

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

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

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

public function testTitle()
{
$this->open('http://www.exemplo.com/');
$this->assertTitle('Página WWW de Exemplo');
}
}
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 7 seconds, Memory: 6.00Mb

There was 1 failure:

1) WebTest::testTitle
Current URL: http://www.iana.org/dominios/exemplo/
```

```
Screenshot: http://localhost/screenshots/334b080f2364b5f11568ee1c7f6742c9.png

Failed asserting that 'IANA — Domínio de Exemplo' matches PCRE pattern "/Página WWW de E


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit WebTest
PHPUnit 3.7.0 by Sebastian Bergmann.

F

Time: 7 seconds, Memory: 6.00Mb

There was 1 failure:

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

Failed asserting that 'IANA — Domínio de Exemplo' matches PCRE pattern "/Página WWW de E

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

Você pode executar cada teste usando um conjunto de navegadores: Em vez de usar `setBrowser()` para configurar um navegador, você pode declarar um vetor `public static` chamado `$browsers` em sua classe de caso de testes. Cada item nesse vetor descreve uma configuração de navegador. Cada um desses navegadores pode ser hospedado em diferentes Servidores Selenium. Exemplo 17.4, “Definindo configurações de múltiplos navegadores” mostra um exemplo.

Exemplo 17.4. Definindo configurações de múltiplos navegadores

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

class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
public static $browsers = array(
array(
'name' => 'Firefox no Linux',
'browser' => '*firefox',
'host' => 'my.linux.box',
'port' => 4444,
'timeout' => 30000,
),
array(
'name' => 'Safari no MacOS X',
'browser' => '*safari',
'host' => 'my.macosx.box',
'port' => 4444,
'timeout' => 30000,
),
array(
'name' => 'Safari no Windows XP',
'browser' => '*custom C:\Program Files\Safari\Safari.exe -url',
'host' => 'my.windowsxp.box',
'port' => 4444,
'timeout' => 30000,
),
array(
'name' => 'Internet Explorer no Windows XP',
'browser' => '*iexplore',
'host' => 'my.windowsxp.box',
```

```
'port' => 4444,
'timeout' => 30000,
)
);

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

public function testTitle()
{
$this->open('http://www.exemplo.com/');
$this->assertTitle('Página Web de Exemplo');
}
}
?>
```

`PHPUnit_Extensions_SeleniumTestCase` pode coletar a informação de cobertura de código para execução de testes através do Selenium:

1. Copie `PHPUnit/Extensions/SeleniumTestCase/phpunit_coverage.php` 1. para dentro do diretório raiz de documentos do seu servidor web.

2. Em seu arquivo de configuração `php.ini` do servidor web, configure `PHPUnit/Extensions/SeleniumTestCase/prepend.php` e `PHPUnit/Extensions/SeleniumTestCase/append.php` como o `auto_prepend_file` e `auto_append_file`, respectivamente.

3. Na sua classe de caso de teste que estende `PHPUnit_Extensions_SeleniumTestCase`, use

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

   para configurar a URL para o script `phpunit_coverage.php`.

Tabela 17.2, “Asserções” lista os vários métodos de asserção que o `PHPUnit_Extensions_SeleniumTestCase` fornece.

Tabela 17.2. Asserções

Asserção	Significado
`void assertElementValueEquals(string $locator, string $text)`	Relata um erro se o valor do elemento identificado por `$locator` não é igual ao `$text` informado.
`void assertElementValueNotEquals(string $locator, string $text)`	Relata um erro se o valor do elemento identificado por `$locator` é igual ao `$text` informado.
`void assertElementValueContains(string $locator, string $text)`	Relata um erro se o valor do elemento identificado por `$locator` não contém o `$text` informado.
`void assertElementValueNotContains(string $locator, string $text)`	Relata um erro se o valor do elemento identificado por `$locator` contém o `$text` informado.
`void assertElementContainsText(string $locator, string $text)`	Relata um erro se o elemento identificado por `$locator` não contém o `$text` informado.
`void assertElementNotContainsText(string $locator, string $text)`	Relata um erro se o elemento identificado por `$locator` contém o `$text` informado.

Asserção	Significado
`void assertSelectHasOption(string $selectLocator, string $option)`	Relata um erro se a opção informada não estiver disponível.
`void assertSelectNotHasOption(string $selectLocator, string $option)`	Relata um erro se a opção informada estiver disponível.
`void assertSelected($selectLocator, $option)`	Relata um erro se o rótulo informado não estiver selecionado.
`void assertNotSelected($selectLocator, $option)`	Relata um erro se o rótulo informado estiver selecionado.
`void assertIsSelected(string $selectLocator, string $value)`	Relata um erro se o valor informado não estiver selecionado.
`void assertIsNotSelected(string $selectLocator, string $value)`	Relata um erro se o valor informado estiver selecionado.

Tabela 17.3, “Métodos Modelo” mostra o método modelo de `PHPUnit_Extensions_SeleniumTestCase`:

Tabela 17.3. Métodos Modelo

Método	Significado
`void defaultAssertions()`	Sobreposição para realizar asserções que são compartilhadas por todos os testes de um caso de teste. Este método é chamado após cada comando ser enviado ao Servidor Selenium.

Por favor, verifique a documentação dos comandos do Selenium [http://release.seleniumhq.org/selenium-core/1.0.1/reference.html] para uma referência de todos os comandos disponíveis e como eles são utilizados.

Os comandos do Selenium 1 são implementados dinamicamente via __call. Verifique também a documentação API para PHPUnit_Extensions_SeleniumTestCase_Driver::__call() [https://github.com/sebastianbergmann/phpunit-selenium/blob/master/PHPUnit/Extensions/SeleniumTestCase/Driver.php#L410] para uma lista de todos os métodos suportados do lado do PHP, juntamente com argumentos e tipos de retorno quando disponíveis.

Usando o método `runSelenese($nomearquivo)` você também pode executar um teste Selenium a partir de sua especificação Selenese/HTML. Além disso, usando o atributo estático `$seleneseDirectory`, você pode criar automaticamente objetos de teste a partir de um diretório que contenha arquivos Selenese/HTML. O diretório especificado é pesquisado recursivamente por arquivos `.htm` que possam conter Selenese/HTML. Exemplo 17.5, “Usando um diretório de arquivos Selenese/HTML como testes” mostra um exemplo.

Exemplo 17.5. Usando um diretório de arquivos Selenese/HTML como testes

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

class SeleneseTests extends PHPUnit_Extensions_SeleniumTestCase
{
public static $seleneseDirectory = '/caminho/para/arquivos';
}
?>
```

Desde o Selenium 1.1.1, um recurso experimental foi incluído para permitir ao usuário compartilhar a sessão entre testes. O único caso suportado é compartilhar a sessão entre todos os testes quando um único navegador é usado. Chame `PHPUnit_Extensions_SeleniumTestCase::shareSession(true)` em seu arquivo bootstrap para permitir o compartilhamento de sessão. A sessão será reiniciada no caso de testes mal-sucedidos (falhos ou incompletos); cabe ao usuário evitar interações entre testes seja resetando cookies ou deslogando da aplicação sob teste (com um método tearDown()).

Capítulo 18. Registrando

O PHPUnit pode produzir vários tipos de arquivos de registro (logfiles).

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 [http://ant.apache.org/manual/OptionalTasks/junit.html]. 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 arquivo de registro XML seguinte 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 integer:2 matches expected value integer:1.

/home/sb/FailureErrorTest.php:8
</failure>
</testcase>
<testcase name="testError"
```

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

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

Resultados de Teste (TAP)

O Test Anything Protocol (TAP) [http://testanything.org/] é uma interface simples baseada em texto do Perl entre módulos de teste. O seguinte exemplo mostra o arquivo de registro TAP gerado para os testes em `ArrayTest`:

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

O seguinte arquivo de registro TAP foi gerado para dois testes, `testFailure` e `testError`, de uma classe de caso de teste chamada `FailureErrorTest` e e mostra como falhas e erros são denotados.

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

Resultados de Teste (JSON)

O JavaScript Object Notation (JSON) [http://www.json.org/] é um formato leve de intercâmbio de dados. O seguinte exemplo mostra as mensagem JSON geradas para os testes em `ArrayTest`:

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

As mensagens JSON seguintes foram geradas para dois testes, `testFailure` e `testError`, de uma classe de caso de teste chamada `FailureErrorTest` e mostra como falhas e erros são denotados.

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

```
"test":"testFailure(FailureErrorTest)","status":"fail",
"time":0.0082459449768066,"trace":[],
"message":"Failed asserting that <integer:2> is equal to <integer:1>."}
{"event":"test","suite":"FailureErrorTest",
"test":"testError(FailureErrorTest)","status":"error",
"time":0.0083680152893066,"trace":[],"message":""}
```

Cobertura de Código (XML)

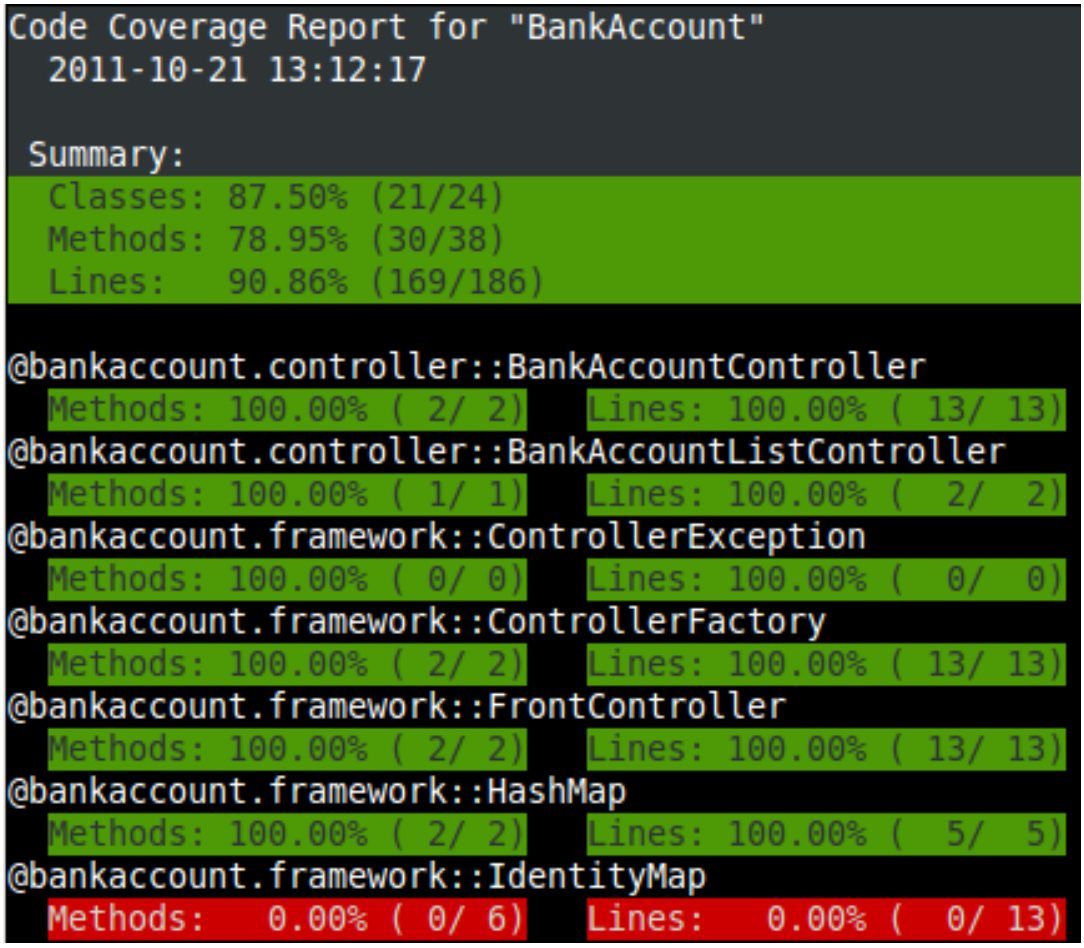
O arquivo de registro no formato XML para informação de cobertura de código produzido pelo PHPUnit é amplamente baseado naquele usado pelo Clover [http://www.atlassian.com/software/clover/]. O seguinte exemplo mostra o arquivo de registro XML gerado para os testes em `ContaBancariaTest`:

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

Cobertura de Código (TEXTO)

Saída de cobertura de código humanamente 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 maiores projetos 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 `php://stdout` isso vai honrar a configuração `--colors`. Escrever em 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”.

Figura 18.1. Saída de Cobertura de Código na linha-de-comando com cores



Capítulo 19. 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.

Subclasse PHPUnit_Framework_TestCase

Escreva asserções personalizadas e métodos utilitários em uma subclasse abstrata do `PHPUnit_Framework_TestCase` e derive suas classes de caso de teste dessa classe. Essa é uma das formas mais fáceis de estender o PHPUnit.

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 em Exemplo 19.1, “Os métodos assertTrue() e isTrue() da classe PHPUnitFramework_Assert”, o método `assertTrue()` é apenas um empacotador em torno dos métodos `isTrue()` e `assertThat()`: `isTrue()` cria um objeto comparador que é passado para `assertThat()` para avaliação.

Exemplo 19.1. Os métodos assertTrue() e isTrue() da classe PHPUnitFramework_Assert

```
<?php
abstract class PHPUnit_Framework_Assert
{
// ...

/**
* Asserta que uma condição é verdade.
*
* @param boolean $condicao
* @param string $menssagem
* @throws PHPUnit_Framework_AssertionFailedError
*/
public static function assertTrue($condicao, $mensagem = '')
{
self::assertThat($condicao, self::isTrue(), $mensagem);
}

// ...

/**
* Retorna um objeto equiparador 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;
}

// ...
}?>
```

Exemplo 19.2, “A classe PHPUnit_Framework_Constraint_IsTrue” mostra como `PHPUnit_Framework_Constraint_IsTrue` estende a classe base abstrata para objetos comparadores (ou restritores), `PHPUnit_Framework_Constraint`.

Exemplo 19.2. A classe PHPUnit_Framework_Constraint_IsTrue

```
<?php
class PHPUnit_Framework_Constraint_IsTrue extends PHPUnit_Framework_Constraint
{
/**
* Avalia a restrição para o parâmetro $outro. Retorna TRUE se a
* restrição é confirmada, FALSE caso contrário.
*
* @param misto $outro Valor ou objeto a avaliar.
* @return bool
*/
public function matches($outro)
{
return $outro === TRUE;
}

/**
* Retorna uma representação string da restrição.
*
* @return string
*/
public function toString()
{
return 'é verdade';
}
}?>
```

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 como contá-las para estatísticas. Além disso, o método `isTrue()` pode ser usado como um comparador ao configurar objetos falsos.

Implementando PHPUnit_Framework_TestListener

Exemplo 19.3, “Um simples ouvinte de teste” mostra uma implementação simples da interface `PHPUnit_Framework_TestListener`.

Exemplo 19.3. Um simples ouvinte de teste

```
<?php
class SimplesOuvinteDeTest implements PHPUnit_Framework_TestListener
{
public function addError(PHPUnit_Framework_Test $teste, Exception $e, $tempo)
{
printf("Erro ao executar o teste '%s'.\n", $teste->getName());
}

public function addFailure(PHPUnit_Framework_Test $teste, PHPUnit_Framework_AssertionFai
{
printf("O Teste '%s' falhou.\n", $teste->getName());
}

public function addIncompleteTest(PHPUnit_Framework_Test $teste, Exception $e, $tempo)
{
printf("O Teste '%s' está incompleto.\n", $teste->getName());
}

public function addSkippedTest(PHPUnit_Framework_Test $teste, Exception $e, $tempo)
```

```
{
printf("O Teste '%s' foi pulado.\n", $teste->getName());
}

public function startTest(PHPUnit_Framework_Test $teste)
{
printf("O Teste '%s' iniciou.\n", $teste->getName());
}

public function endTest(PHPUnit_Framework_Test $teste, $tempo)
{
printf("O Teste '%s' terminou.\n", $teste->getName());
}

public function startTestSuite(PHPUnit_Framework_TestSuite $suite)
{
printf("A Suíte de Testes '%s' iniciou.\n", $suite->getName());
}

public function endTestSuite(PHPUnit_Framework_TestSuite $suite)
{
printf("A Suíte de Testes '%s' terminou.\n", $suite->getName());
}
}
?>
```

Em “Ouvintes de Teste” você pode ver como configurar o PHPUnit para anexar seu ouvinte de teste para a execução do teste.

Subclasse PHPUnit_Extensions_TestDecorator

ocê pode envolver casos de teste ou suítes de teste em uma subclasse do `PHPUnit_Extensions_TestDecorator` e usar o padrão de design do Decorador para realizar algumas ações antes e depois da execução do teste.

O PHPUnit navega com dois decoradores de teste concretos: `PHPUnit_Extensions_RepeatedTest` e `PHPUnit_Extensions_TestSetup`. O formador é usado para executar um teste repetidamente e apenas o conta como bem-sucedido se todas as iterações forem bem-sucedidas. O último foi discutido em Capítulo 6, *Ambientes*.

Exemplo 19.4, “O Decorador RepeatedTest” mostra uma versão resumida do decorador de teste `PHPUnit_Extensions_RepeatedTest` que ilustra como escrever seus próprios decoradores de teste.

Exemplo 19.4. O Decorador RepeatedTest

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

class PHPUnit_Extensions_RepeatedTest extends PHPUnit_Extensions_TestDecorator
{
private $repetirVezes = 1;

public function __construct(PHPUnit_Framework_Test $teste, $repetirVezes = 1)
{
parent::__construct($teste);

if (is_integer($repetirVezes) &&
$repetirVezes >= 0) {
$this->repetirVezes = $repetirVezes;
```

```
}
}

public function count()
{
return $this->repetirVezes * $this->teste->count();
}

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

for ($i = 0; $i < $this->repetirVezes && !$resultado->shouldStop(); $i++) {
$this->teste->run($resultado);
}

return $resultado;
}
}
?>
```

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 *data-driven tests*, por exemplo.

Exemplo 19.5, “Um teste guiado por dados” mostra uma classe de caso de teste guiado por dados que compara valores de um arquivo com Valores Separados por Vírgulas (CSV). Cada linha de tal arquivo parece com `foo;bar`, onde o primeiro valor é o qual esperamos e o segundo é o real.

Exemplo 19.5. Um teste guiado por dados

```
<?php
class GuiadoPorDadosTest implements PHPUnit_Framework_Test
{
private $linhas;

public function __construct($arquivoDados)
{
$this->linhas = file($arquivoDados);
}

public function count()
{
return 1;
}

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

foreach ($this->linhas as $linha) {
$resultado->startTest($this);
PHP_Timer::start();
$stopTime = NULL;

list($esperado, $real) = explode(';', $linha);
```

```
try {
PHPUnit_Framework_Assert::assertEquals(
trim($esperado), trim($real)
);
}

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

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

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

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

return $resultado;
}
}

$teste = new GuiadoPorDadosTest('arquivo_dados.csv');
$resultado = PHPUnit_TextUI_TestRunner::run($teste);
?>
```

```
PHPUnit 3.7.0 by Sebastian Bergmann.

.F

Time: 0 seconds

There was 1 failure:

1) GuiadoPorDadosTest
Failed asserting that two strings are equal.
expected string <bar>
difference < x>
got string <baz>
/home/sb/GuiadoPorDadosTest.php:32
/home/sb/GuiadoPorDadosTest.php:53

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

Apêndice A. Assertions

Tabela A.1, “Assertions” mostra todas as variedades de asserções.

Tabela A.1. Assertions

Assertion
`assertArrayHasKey($chave, $vetor, $mensagem = '')`
`assertArrayNotHasKey($chave, $vetor, $mensagem = '')`
`assertAttributeContains($agulha, $nomeAtributoBateria, $classeOuObjetoBateria, $mensagem = '', $ignorarCaixa = FALSE, $verificarIdentidadeDoObjeto = TRUE)`
`assertAttributeContainsOnly($tipo, $nomeAtributoBateria, $classeOuObjetoBateria, $ehTipoNativo = NULL, $mensagem = '')`
`assertAttributeCount($contagemEsperada, $nomeAtributoBateria, $classeOuObjetoBateria, $mensagem = '')`
`assertAttributeEmpty($nomeAtributoBateria, $classeOuObjetoBateria, $mensagem = '')`
`assertAttributeEquals($esperado, $nomeRealAtributo, $classeOuObjetoReal, $mensagem = '', $delta = 0, $profMax = 10, $canonicalizar = FALSE, $ignorarCaixa = FALSE)`
`assertAttributeGreaterThan($esperado, $nomeRealAtributo, $classeOuObjetoReal, $mensagem = '')`
`assertAttributeGreaterThanOrEqual($esperado, $nomeRealAtributo, $classeOuObjetoReal, $mensagem = '')`
`assertAttributeInstanceOf($esperado, $nomeAtributo, $classeOuObjeto, $mensagem = '')`
`assertAttributeInternalType($esperado, $nomeAtributo, $classeOuObjeto, $mensagem = '')`
`assertAttributeLessThan($esperado, $nomeRealAtributo, $classeOuObjetoReal, $mensagem = '')`
`assertAttributeLessThanOrEqual($esperado, $nomeRealAtributo, $classeOuObjetoReal, $mensagem = '')`
`assertAttributeNotContains($agulha, $nomeAtributoBateria, $classeOuObjetoBateria, $mensagem = '', $ignorarCaixa = FALSE, $verificarIdentidadeDoObjeto = TRUE)`
`assertAttributeNotContainsOnly($tipo, $nomeAtributoBateria, $classeOuObjetoBateria, $ehTipoNativo = NULL, $mensagem = '')`
`assertAttributeNotCount($contagemEsperada, $nomeAtributoBateria, $classeOuObjetoBateria, $mensagem = '')`
`assertAttributeNotEmpty($nomeAtributoBateria, $classeOuObjetoBateria, $mensagem = '')`
`assertAttributeNotEquals($esperado, $nomeRealAtributo, $classeOuObjetoReal, $mensagem = '', $delta = 0, $profMax = 10, $canonicalizar = FALSE, $ignorarCaixa = FALSE)`
`assertAttributeNotInstanceOf($esperado, $nomeAtributo, $classeOuObjeto, $mensagem = '')`
`assertAttributeNotInternalType($esperado, $nomeAtributo, $classeOuObjeto, $mensagem = '')`

Assertion
`assertAttributeNotSame($esperado, $nomeRealAtributo, $classeOuObjetoReal, $mensagem = '')`
`assertAttributeSame($esperado, $nomeRealAtributo, $classeOuObjetoReal, $mensagem = '')`
`assertClassHasAttribute($nomeAtributo, $nomeClasse, $mensagem = '')`
`assertClassHasStaticAttribute($nomeAtributo, $nomeClasse, $mensagem = '')`
`assertClassNotHasAttribute($nomeAtributo, $nomeClasse, $mensagem = '')`
`assertClassNotHasStaticAttribute($nomeAtributo, $nomeClasse, $mensagem = '')`
`assertContains($agulha, $bateria, $mensagem = '', $ignorarCaixa = FALSE, $verificarIdentidadeDoObjeto = TRUE)`
`assertContainsOnly($tipo, $bateria, $ehTipoNativo = NULL, $mensagem = '')`
`assertContainsOnlyInstancesOf($nomeclasse, $bateria, $mensagem = '')`
`assertCount($contagemEsperada, $bateria, $mensagem = '')`
`assertEmpty($real, $mensagem = '')`
`assertEqualXMLStructure(DOMElement $elementoEsperado, DOMElement $elementoReal, $verificarAtributos = FALSE, $mensagem = '')`
`assertEquals($esperado, $real, $mensagem = '', $delta = 0, $profMax = 10, $canonicalizar = FALSE, $ignorarCaixa = FALSE)`
`assertFalse($condicao, $mensagem = '')`
`assertFileEquals($esperado, $real, $mensagem = '', $canonicalizar = FALSE, $ignorarCaixa = FALSE)`
`assertFileExists($nomearquivo, $mensagem = '')`
`assertFileNotEquals($esperado, $real, $mensagem = '', $canonicalizar = FALSE, $ignorarCaixa = FALSE)`
`assertFileNotExists($nomearquivo, $mensagem = '')`
`assertGreaterThan($esperado, $real, $mensagem = '')`
`assertGreaterThanOrEqual($esperado, $real, $mensagem = '')`
`assertInstanceOf($esperado, $real, $mensagem = '')`
`assertInternalType($esperado, $real, $mensagem = '')`
`assertJsonFileEqualsJsonFile($arquivoEsperado, $arquivoReal, $mensagem = '')`
`assertJsonFileNotEqualsJsonFile($arquivoEsperado, $arquivoReal, $mensagem = '')`
`assertJsonStringEqualsJsonFile($arquivoEsperado, $jsonReal, $mensagem = '')`
`assertJsonStringEqualsJsonString($jsonEsperado, $jsonReal, $mensagem = '')`
`assertJsonStringNotEqualsJsonFile($arquivoEsperado, $jsonReal, $mensagem = '')`

Assertion
`assertJsonStringNotEqualsJsonString($jsonEsperado, $jsonReal, $mensagem = '')`
`assertLessThan($esperado, $real, $mensagem = '')`
`assertLessThanOrEqual($esperado, $real, $mensagem = '')`
`assertNotContains($agulha, $bateria, $mensagem = '', $ignorarCaixa = FALSE, $verificarIdentidadeDoObjeto = TRUE)`
`assertNotContainsOnly($tipo, $bateria, $ehTipoNativo = NULL, $mensagem = '')`
`assertNotCount($contagemEsperada, $bateria, $mensagem = '')`
`assertNotEmpty($real, $mensagem = '')`
`assertNotEquals($esperado, $real, $mensagem = '', $delta = 0, $profMax = 10, $canonicalizar = FALSE, $ignorarCaixa = FALSE)`
`assertNotInstanceOf($esperado, $real, $mensagem = '')`
`assertNotInternalType($esperado, $real, $mensagem = '')`
`assertNotNull($real, $mensagem = '')`
`assertNotRegExp($padrao, $string, $mensagem = '')`
`assertNotSame($esperado, $real, $mensagem = '')`
`assertNotSameSize($esperado, $real, $mensagem = '')`
`assertNotTag($equiparador, $real, $mensagem = '', $ehHtml = TRUE)`
`assertNull($real, $mensagem = '')`
`assertObjectHasAttribute($nomeAtributo, $objeto, $mensagem = '')`
`assertObjectNotHasAttribute($nomeAtributo, $objeto, $mensagem = '')`
`assertRegExp($padrao, $string, $mensagem = '')`
`assertSame($esperado, $real, $mensagem = '')`
`assertSameSize($esperado, $real, $mensagem = '')`
`assertSelectCount($seletor, $conta, $real, $mensagem = '', $ehHtml = TRUE)`
`assertSelectEquals($seletor, $conteudo, $conta, $real, $mensagem = '', $ehHtml = TRUE)`
`assertSelectRegExp($seletor, $padrao, $conta, $real, $mensagem = '', $ehHtml = TRUE)`
`assertStringEndsNotWith($sufixo, $string, $mensagem = '')`
`assertStringEndsWith($sufixo, $string, $mensagem = '')`
`assertStringEqualsFile($arquivoEsperado, $stringReal, $mensagem = '', $canonicalizar = FALSE, $ignorarCaixa = FALSE)`
`assertStringMatchesFormat($formato, $string, $mensagem = '')`
`assertStringMatchesFormatFile($formatoArquivo, $string, $mensagem = '')`
`assertStringNotEqualsFile($arquivoEsperado, $stringReal, $mensagem = '', $canonicalizar = FALSE, $ignorarCaixa = FALSE)`
`assertStringNotMatchesFormat($formato, $string, $mensagem = '')`
`assertStringNotMatchesFormatFile($formatoArquivo, $string, $mensagem = '')`

Assertion
`assertStringStartsNotWith($prefixo, $string, $mensagem = '')`
`assertStringStartsWith($prefixo, $string, $mensagem = '')`
`assertTag($equiparador, $real, $mensagem = '', $ehHtml = TRUE)`
`assertThat($valor, PHPUnit_Framework_Constraint $restricao, $mensagem = '')`
`assertTrue($condicao, $mensagem = '')`
`assertXmlFileEqualsXmlFile($arquivoEsperado, $arquivoReal, $mensagem = '')`
`assertXmlFileNotEqualsXmlFile($arquivoEsperado, $arquivoReal, $mensagem = '')`
`assertXmlStringEqualsXmlFile($arquivoEsperado, $xmlReal, $mensagem = '')`
`assertXmlStringEqualsXmlString($xmlEsperado, $xmlReal, $mensagem = '')`
`assertXmlStringNotEqualsXmlFile($arquivoEsperado, $xmlReal, $mensagem = '')`
`assertXmlStringNotEqualsXmlString($xmlEsperado, $xmlReal, $mensagem = '')`

Apêndice B. Anotações

Uma anotação é uma forma especial de metadados sintáticos que podem ser adicionados ao código-fonte de algumas linguagens de programação. Enquanto o 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 do PHP para anotar o código-fonte. Os blocos de documentação PHP são reflexivos: eles podem ser acessados através do método da API de Reflexão `getDocComment()` a nível de função, classe, método e atributo. Aplicações como o PHPUnit usam essa informação em tempo de execução para configurar seu comportamento.

Este apêndice mostra todas as variedades de anotações suportadas pelo PHPUnit.

@author

A anotação `@author` é um apelido para a anotação `@group` (veja a seção chamada "`@group`") e permite filtrar os testes baseado em seus autores.

@backupGlobals

As operações de backup e restauração para variáveis globais podem ser completamente desabilitadas para todos os testes de uma classe de caso de teste como esta:

```
/**
 * @backupGlobals disabled
 */
class MeuTest extends PHPUnit_Framework_TestCase
{
    // ...
}
```

A anotação `@backupGlobals` também pode ser usada a nível de método de teste. Isso permite uma configuração refinada das operações de backup e restauração:

```
/**
 * @backupGlobals disabled
 */
class MeuTest extends PHPUnit_Framework_TestCase
{
    /**
     * @backupGlobals enabled
     */
    public function testQueInterageComVariaveisGlobais()
    {
        // ...
    }
}
```

@backupStaticAttributes

As operações de backup e restauração para atributos estáticos de classes podem ser completamente desabilitadas para todos os testes de uma classe de caso de teste como esta:

```
/**
 * @backupStaticAttributes disabled
 */
class MeuTest extends PHPUnit_Framework_TestCase
{
```

```
    // ...
}
```

A anotação `@backupStaticAttributes` também pode ser usada a nível de método de teste. Isso permite uma configuração refinada das operações de backup e restauração:

```
/**
 * @backupStaticAttributes disabled
 */
class MeuTest extends PHPUnit_Framework_TestCase
{
    /**
     * @backupStaticAttributes enabled
     */
    public function testQueInterageComAtributosEstaticos()
    {
        // ...
    }
}
```

@codeCoverageIgnore*

As anotações `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` e `@codeCoverageIgnoreEnd` podem ser usadas para excluir linhas de código da análise de cobertura.

Para uso, veja a seção chamada “Ignorando Blocos de Código”.

@covers

A anotação `@covers` pode ser usada no código de teste para especificar quais métodos um método de teste quer testar:

```
/**
 * @covers ContaBancaria::getSaldo
 */
public function testSaldoEhInicialmenteZero()
{
    $this->assertEquals(0, $this->cb->getSaldo());
}
```

Se fornecida, apenas a informação de cobertura de código para o(s) método(s) especificado(s) será considerada.

Tabela B.1, “Anotações para especificar quais métodos são cobertos por um teste” mostra a sintaxe da anotação `@covers`.

Tabela B.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

Anotação	Descrição
	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 o método de teste anotado cobre todos os métodos que não sejam privados de uma dada classe.

@coversNothing

A anotação `@coversNothing` pode ser usada no código de teste para especificar que nenhuma informação de cobertura de código será gravada para o caso de teste anotado.

Isso pode ser usado para testes de integração. Veja Exemplo 14.3, “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 sobrepujar quaisquer tags `@covers`.

@dataProvider

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

Veja a seção chamada “Provedores de Dados” para mais detalhes.

@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 componente de teste por um produtor e passá-la aos consumidores dependentes. Exemplo 4.2, “Usando a anotação `@depends` para expressar dependências” mostra como usar a anotação `@depends` para expressar dependências entre métodos de teste.

Veja a seção chamada “Dependências de Testes” para mais detalhes.

@expectedException

??? mostra como usar a anotação `@expectedException` para testar se uma exceção é lançada dentro do código testado.

Veja a seção chamada “Testando Exceções” para mais detalhes.

@expectedExceptionCode

A anotação `@expectedExceptionCode` em conjunção com a `@expectedException` permite fazer asserções no código de erro de uma exceção lançada, permitindo diminuir uma exceção específica.

```
class MeuTest extends PHPUnit_Framework_TestCase
{
    /**
     * @expectedException MinhaExcecao
     * @expectedExceptionCode 20
     */
    public function testExcecaoTemCodigoErro20()
    {
        throw new MinhaExcecao('Alguma Mensagem', 20);
    }
}
```

Para facilitar o teste e reduzir a duplicação, um atalho pode ser usado para especificar uma constante de classe como uma `@expectedExceptionCode` usando a sintaxe "`@expectedExceptionCode ClassName::CONST`".

```
class MeuTest extends PHPUnit_Framework_TestCase
{
    /**
      * @expectedException MinhaExcecao
      * @expectedExceptionCode MyClass::ERRORCODE
      */
    public function testExcecaoTemCodigoErro20()
    {
      throw new MyException('Alguma Mensagem', 20);
    }
}
class MinhaClasse
{
    const ERRORCODE = 20;
}
```

@expectedExceptionMessage

A anotação `@expectedExceptionMessage` trabalha de modo similar a `@expectedExceptionCode` já que lhe permite fazer uma asserção na mensagem de erro de uma exceção.

```
class MeuTest extends PHPUnit_Framework_TestCase
{
    /**
     * @expectedException MinhaExcecao
     * @expectedExceptionMessage Alguma Mensagem
```

```
     */
    public function testExcecaoTemMensagemCerta()
    {
        throw new MinhaExcecao('Alguma Mensagem', 20);
    }
}
```

A mensagem esperada pode ser uma substring de uma Mensagem de exceção. Isso pode ser útil para assertar apenas que um certo nome ou parâmetro que foi passado é mostrado na exceção e não fixar toda a mensagem de exceção no teste.

```
class MeuTest extends PHPUnit_Framework_TestCase
{
     /**
      * @expectedException MinhaExcecao
      * @expectedExceptionMessage quebrado
      */
     public function testExcecaoTemMensagemCerta()
     {
         $param = "quebrado";
         throw new MinhaExcecao('Parâmetro Inválido "'.$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`”.

@group

Um teste pode ser marcado como pertencente a um ou mais grupos usando a anotação `@group` desta forma:

```
class MeuTest extends PHPUnit_Framework_TestCase
{
    /**
     * @group especificacao
     */
    public function testAlgumaCoisa()
    {
    }

    /**
     * @group regressao
     * @group bug2204
     */
    public function testOutraCoisa()
    {
    }
}
```

Testes podem ser selecionados para execução baseada em grupos usando os comutadores `--group` e `--exclude-group` do executor de teste em linha-de-comando ou usando as respectivas diretivas do arquivo de configuração XML.

@outputBuffering

A anotação `@outputBuffering` pode ser usada para controlar a memória temporária (buffering) de saída http://www.php.net/manual/en/intro.outcontrol.php desta forma:

```
/**
 * @outputBuffering enabled
 */
class MeuTest extends PHPUnit_Framework_TestCase
{
    // ...
}
```

A anotação `@outputBuffering` também pode ser usada a nível de método de teste. Isso permite um controle refinado da memória temporária de saída:

```
/**
 * @outputBuffering disabled
 */
class MeuTest extends PHPUnit_Framework_TestCase
{
    /**
     * @outputBuffering enabled
     */
    public function testQueImprimeAlgumaCoisa()
    {
        // ...
    }
}
```

@requires

A anotação `@requires` pode ser usada para pular testes quando pré-condição, como a Versão do PHP ou extensões instaladas, não batem.

Uma lista completa de possibilidades e exemplos pode ser encontrada em Tabela 9.3, “Possíveis usos para @requires”

@runTestsInSeparateProcesses

@runInSeparateProcess

@test

Como uma alternativa para prefixar seus nomes de métodos de teste com `test`, você pode usar a anotação `@test` no Bloco de Documentação de um método para marcá-lo como um método de teste.

```
/**
 * @test
 */
public function saldoInicialDeveSer0()
{
    $this->assertEquals(0, $this->cb->getSaldo());
}
```

@testdox

@ticket

Apêndice C. O arquivo de configuração XML

PHPUnit

Os atributos do elemento `<phpunit>` podem ser usados para configurar a funcionalidade do núcleo do PHPUnit.

```
<phpunit backupGlobals="true"
backupStaticAttributes="false"
<!--bootstrap="/caminho/para/bootstrap.php"-->
cacheTokens="false"
colors="false"
convertErrorsToExceptions="true"
convertNoticesToExceptions="true"
convertWarningsToExceptions="true"
forceCoversAnnotation="false"
mapTestClassNameToCoveredClassName="false"
printerClass="PHPUnit_TextUI_ResultPrinter"
<!--printerFile="/caminho/para/ResultPrinter.php"-->
processIsolation="false"
stopOnError="false"
stopOnFailure="false"
stopOnIncomplete="false"
stopOnSkipped="false"
testSuiteLoaderClass="PHPUnit_Runner_StandardTestSuiteLoader"
<!--testSuiteLoaderFile="/caminho/para/StandardTestSuiteLoader.php"-->
strict="false"
verbose="false">
<!-- ... -->
</phpunit>
```

A configuração XML acima corresponde ao comportamento padrão do executor de teste TextUI documentado na seção chamada “Comutadores de linha-de-comando”.

Opções adicionais que não estão disponíveis como comutadores em linha-de-comando são:

`convertNoticesToExceptions`, `convertWarningsToExceptions`, `convertErrorsToExceptions`: Pode ser usado para desligar a conversão automática de cada notificação/aviso/erro do PHP em uma exceção.

`forceCoversAnnotation`: A Cobertura de Código só será gravada para testes que usem a anotação `@covers` documentada na seção chamada “`@covers`”.

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="Minha Suíte de Testes">
<directory>/caminho/para/arquivos/*Test.php</directory>
<file>/caminho/para/MeuTest.php</file>
<exclude>/caminho/para/excluidos</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 `/caminho/para/*Test.php` e `/caminho/para/MeuTest.php` se a versão do PHP for no mínio 5.3.0.

```
<testsuites>
<testsuite name="Minha Suíte de Testes">
<directory suffix="Test.php" phpVersion="5.3.0" phpVersionOperator=">=">/caminho/para/ar
<file phpVersion="5.3.0" phpVersionOperator=">=">/caminho/para/MeuTest.php</file>
</testsuite>
</testsuites>
```

O atributo `phpVersionOperator` é opcional e padronizado para `>=`.

Grupos

O elemento `<groups>` e seus filhos `<include>`, `<exclude>`, e `<group>` podem ser usados para selecionar grupos de testes de uma suíte de testes que (não) deveriam ser executadas.

```
<groups>
<include>
<group>nome</group>
</include>
<exclude>
<group>nome</group>
</exclude>
</groups>
```

A configuração XML acima corresponde a invocar o executor de testes TextUI com os seguintes comutadores:

- `--group nome`
- `--exclude-group nome`

Incluindo e Excluindo Arquivos para Cobertura de Código

O elemento `<filter>` e seus filhos podem ser usados para configurar a lista-negra e lista-branca para o relatório de cobertura de código.

```
<filter>
<blacklist>
<directory suffix=".php">/caminho/para/arquivos</directory>
<file>/path/to/file</file>
<exclude>
<directory suffix=".php">/caminho/para/arquivos</directory>
<file>/caminho/para/arquivo</file>
</exclude>
</blacklist>
<whitelist processUncoveredFilesFromWhitelist="true">
<directory suffix=".php">/caminho/para/arquivos</directory>
<file>/caminho/para/arquivo</file>
<exclude>
<directory suffix=".php">/caminho/para/arquivos</directory>
<file>/caminho/para/arquivo</file>
</exclude>
</whitelist>
</filter>
```

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" charset="UTF-8"
highlight="false" lowUpperBound="35" highLowerBound="70"/>
<log type="coverage-clover" target="/tmp/coverage.xml"/>
<log type="coverage-php" target="/tmp/coverage.serialized"/>
<log type="coverage-text" target="php://stdout" showUncoveredFiles="false"/>
<log type="json" target="/tmp/logfile.json"/>
<log type="tap" target="/tmp/logfile.tap"/>
<log type="junit" target="/tmp/logfile.xml" logIncompleteSkipped="false"/>
<log type="testdox-html" target="/tmp/testdox.html"/>
<log type="testdox-text" target="/tmp/testdox.txt"/>
</logging>
```

A configuração XML acima corresponde a invocar o executor de teste TextUI com os seguintes comutadores:

- `--coverage-html /tmp/report`
- `--coverage-clover /tmp/coverage.xml`
- `--coverage-php /tmp/coverage.serialized`
- `--coverage-text`
- `--log-json /tmp/logfile.json`
- `> /tmp/logfile.txt`
- `--log-tap /tmp/logfile.tap`
- `--log-junit /tmp/logfile.xml`
- `--testdox-html /tmp/testdox.html`
- `--testdox-text /tmp/testdox.txt`

Os atributos `charset`, `highlight`, `lowUpperBound`, `highLowerBound`, `logIncompleteSkipped` e `showUncoveredFiles` não possuem comutador TextUI equivalente.

- `charset`: Conjunto de caracteres a ser usado para as páginas HTML geradas.
- `highlight`: Quando definidos como `true`, o código em seus relatórios de cobertura terá a sintaxe destacada.
- `lowUpperBound`: Máxima porcentagem de cobertura para ser considerado de "baixamente" coberto.
- `highLowerBound`: Mínima porcentagem de cobertura para ser considerado "altamente" coberto.

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="MeuOuvinte" file="/caminho/opcional/para/MeuOuvinte.php">
<arguments>
<array>
<element key="0">
<string>Sebastian</string>
</element>
</array>
<integer>22</integer>
<string>Abril</string>
<double>19.78</double>
<null/>
<object class="stdClass"/>
</arguments>
</listener>
</listeners>
```

A configuração XML acima corresponde a anexar o objeto `$ouvinte` (veja abaixo) à execução de teste:

```
$ouvinte = new MeuOuvinte(
array('Sebastian'),
22,
'Abril',
19.78,
NULL,
new stdClass
);
```

Setting PHP INI settings, Constants and Global Variables

O elemento `<php>` e seus filhos podem ser usados para definir configurações, constantes e variáveis globais do PHP. 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';
```

Configurando Navegadores para Selenium RC

O elemento `<selenium>` e seus filhos `<browser>` podem ser usados para configurar uma lista de servidores Selenium RC.

```
<selenium>
<browser name="Firefox no Linux"
browser="*firefox /usr/lib/firefox/firefox-bin"
host="my.linux.box"
port="4444"
timeout="30000"/>
</selenium>
```

O arquivo de configuração XML acima corresponde ao seguinte código PHP:

```
class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
public static $navegadores = array(
array(
'name' => 'Firefox no Linux',
'browser' => '*firefox /usr/lib/firefox/firefox-bin',
'host' => 'my.linux.box',
'port' => 4444,
'timeout' => 30000
)
);

// ...
}
```

Apêndice D. Índice

Índice Remissivo

Símbolos

A

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

V

W

X

Apêndice E. Bibliografia

[Astels2003] *Test Driven Development*. David Astels. Copyright © 2003. Prentice Hall. ISBN 0131016490.

[Astels2006] *A New Look at Test-Driven Development*. David Astels. Copyright © 2006. http://blog.daveastels.com/files/BDD_Intro.pdf.

[Beck2002] *Test Driven Development by Example*. Kent Beck. Copyright © 2002. Addison-Wesley. ISBN 0-321-14653-0.

[Meszaros2007] *xUnit Test Patterns: Refactoring Test Code*. Gerard Meszaros. Copyright © 2007. Addison-Wesley. ISBN 978-0131495050.

Apêndice F. Copyright

```
Copyright (c) 2005-2012 Sebastian Bergmann.

This work is licensed under the Creative Commons Attribution 3.0
Unported License.

A summary of the license is given below, followed by the full legal
text.

---------------------------------------------------------------------

You are free:

    * to Share - to copy, distribute and transmit the work
    * to Remix - to adapt the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by
the author or licensor (but not in any way that suggests that they
endorse you or your use of the work).

    * For any reuse or distribution, you must make clear to others
      the license terms of this work. The best way to do this is with
      a link to this web page.

    * Any of the above conditions can be waived if you get
      permission from the copyright holder.

    * Nothing in this license impairs or restricts the author's moral
      rights.

Your fair dealing and other rights are in no way affected by the
above.

This is a human-readable summary of the Legal Code (the full
license) below.

=====================================================================

Creative Commons Legal Code
Attribution 3.0 Unported

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE
LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN
ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS
INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO
WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS
LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS
CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS
PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE
WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW
IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND
AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS
LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU
```

```
THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF
SUCH TERMS AND CONDITIONS.

1. Definitions

   a. "Adaptation" means a work based upon the Work, or upon the
      Work and other pre-existing works, such as a translation,
      adaptation, derivative work, arrangement of music or other
      alterations of a literary or artistic work, or phonogram or
      performance and includes cinematographic adaptations or any
      other form in which the Work may be recast, transformed, or
      adapted including in any form recognizably derived from the
      original, except that a work that constitutes a Collection
      will not be considered an Adaptation for the purpose of this
      License. For the avoidance of doubt, where the Work is a
      musical work, performance or phonogram, the synchronization of
      the Work in timed-relation with a moving image ("synching")
      will be considered an Adaptation for the purpose of this
      License.

   b. "Collection" means a collection of literary or artistic works,
      such as encyclopedias and anthologies, or performances,
      phonograms or broadcasts, or other works or subject matter
      other than works listed in Section 1(f) below, which, by
      reason of the selection and arrangement of their contents,
      constitute intellectual creations, in which the Work is
      included in its entirety in unmodified form along with one or
      more other contributions, each constituting separate and
      independent works in themselves, which together are assembled
      into a collective whole. A work that constitutes a Collection
      will not be considered an Adaptation (as defined above) for
      the purposes of this License.

   c. "Distribute" means to make available to the public the
      original and copies of the Work or Adaptation, as appropriate,
      through sale or other transfer of ownership.

   d. "Licensor" means the individual, individuals, entity or
      entities that offer(s) the Work under the terms of this License.

   e. "Original Author" means, in the case of a literary or artistic
      work, the individual, individuals, entity or entities who
      created the Work or if no individual or entity can be
      identified, the publisher; and in addition (i) in the case of
      a performance the actors, singers, musicians, dancers, and
      other persons who act, sing, deliver, declaim, play in,
      interpret or otherwise perform literary or artistic works or
      expressions of folklore; (ii) in the case of a phonogram the
      producer being the person or legal entity who first fixes the
      sounds of a performance or other sounds; and, (iii) in the
      case of broadcasts, the organization that transmits the
      broadcast.

   f. "Work" means the literary and/or artistic work offered under
      the terms of this License including without limitation any
      production in the literary, scientific and artistic domain,
      whatever may be the mode or form of its expression including
      digital form, such as a book, pamphlet and other writing; a
      lecture, address, sermon or other work of the same nature; a
      dramatic or dramatico-musical work; a choreographic work or
      entertainment in dumb show; a musical composition with or
      without words; a cinematographic work to which are assimilated
      works expressed by a process analogous to cinematography; a
      work of drawing, painting, architecture, sculpture, engraving
```

or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

d. to Distribute and Publicly Perform Adaptations.

e. For the avoidance of doubt:

```
      i. Non-waivable Compulsory License Schemes. In those
         jurisdictions in which the right to collect royalties
         through any statutory or compulsory licensing scheme cannot
         be waived, the Licensor reserves the exclusive right to
         collect such royalties for any exercise by You of the
         rights granted under this License;

      ii. Waivable Compulsory License Schemes. In those
          jurisdictions in which the right to collect royalties
          through any statutory or compulsory licensing scheme can
          be waived, the Licensor waives the exclusive right to
          collect such royalties for any exercise by You of the
          rights granted under this License; and,

      iii. Voluntary License Schemes. The Licensor waives the right
           to collect royalties, whether individually or, in the
           event that the Licensor is a member of a collecting
           society that administers voluntary licensing schemes, via
           that society, from any exercise by You of the rights
           granted under this License.

The above rights may be exercised in all media and formats whether
now known or hereafter devised. The above rights include the right
to make such modifications as are technically necessary to exercise
the rights in other media and formats. Subject to Section 8(f), all
rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly
   made subject to and limited by the following restrictions:

   a. You may Distribute or Publicly Perform the Work only under the
      terms of this License. You must include a copy of, or the
      Uniform Resource Identifier (URI) for, this License with every
      copy of the Work You Distribute or Publicly Perform. You may
      not offer or impose any terms on the Work that restrict the
      terms of this License or the ability of the recipient of the
      Work to exercise the rights granted to that recipient under
      the terms of the License. You may not sublicense the Work. You
      must keep intact all notices that refer to this License and to
      the disclaimer of warranties with every copy of the Work You
      Distribute or Publicly Perform. When You Distribute or
      Publicly Perform the Work, You may not impose any effective
      technological measures on the Work that restrict the ability
      of a recipient of the Work from You to exercise the rights
      granted to that recipient under the terms of the License. This
      Section 4(a) applies to the Work as incorporated in a
      Collection, but this does not require the Collection apart
      from the Work itself to be made subject to the terms of this
      License. If You create a Collection, upon notice from any
      Licensor You must, to the extent practicable, remove from the
      Collection any credit as required by Section 4(b), as
      requested. If You create an Adaptation, upon notice from any
      Licensor You must, to the extent practicable, remove from the
      Adaptation any credit as required by Section 4(b), as requested.

   b. If You Distribute, or Publicly Perform the Work or any
      Adaptations or Collections, You must, unless a request has
      been made pursuant to Section 4(a), keep intact all copyright
      notices for the Work and provide, reasonable to the medium or
      means You are utilizing: (i) the name of the Original Author
      (or pseudonym, if applicable) if supplied, and/or if the
      Original Author and/or Licensor designate another party or
      parties (e.g., a sponsor institute, publishing entity,
```

journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv), consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4 (b) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

c. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

   f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be

```
      enforced according to the corresponding provisions of the
      implementation of those treaty provisions in the applicable
      national law. If the standard suite of rights granted under
      applicable copyright law includes additional rights not
      granted under this License, such additional rights are deemed
      to be included in the License; this License is not intended to
      restrict the license of any rights under applicable law.

Creative Commons is not a party to this License, and makes no
warranty whatsoever in connection with the Work. Creative Commons
will not be liable to You or any party on any legal theory for any
damages whatsoever, including without limitation any general,
special, incidental or consequential damages arising in connection
to this license. Notwithstanding the foregoing two (2) sentences,
if Creative Commons has expressly identified itself as the Licensor
hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the
Work is licensed under the CCPL, Creative Commons does not authorize
the use by either party of the trademark "Creative Commons" or any
related trademark or logo of Creative Commons without the prior
written consent of Creative Commons. Any permitted use will be in
compliance with Creative Commons' then-current trademark usage
guidelines, as may be published on its website or otherwise made
available upon request from time to time. For the avoidance of
doubt, this trademark restriction does not form part of this
License.

Creative Commons may be contacted at http://creativecommons.org/.

======================================================================
```