PHPUnit 袖珍指南

速查，参考

Sebastian Bergmann

适用于 PHPUnit 3.2 版，2008-03-15 更新。

前言

“你打算何时写 PHPUnit 文档？”对于这个问题，长久以来，我的回答是“你不需要 PHPUnit 文档。只需阅读 JUnit 的文档或者买本关于 JUnit 的书籍，将用于 JUnit 的 Java 代码示例改编为用于 PHPUnit 的 PHP 代码即可。” 当我向 O'Reilly 德国办事处的 Barbara

Weiss 和 Alexandra Follenius 谈及这些，他们鼓励我考虑是不是写本可作为 PHPUnit 文档的书。

必备条件

本书讨论的是 PHPUnit，一个用于采用 PHP 程序设计语言进行测试驱动开发的开源框架。本版次适用于 3.2 版的 PHPUnit。当然，大多数示例应该也可用于 2.0-3.1 版的 PHPUnit。本书后面的"用于 PHP 4 的 PHPUnit"（附录 B - 译注）部分涉及了适用于 PHP 4 的旧版 PHPUnit，它们已不再积极开发。

读者需要很好的理解使用 PHP 5 进行面向对象程序设计。对于德国读者，我可以推荐我写的书，[Bergmann2005]，作为 PHP 5 OOP 的入门。一本关于此主题的英文书是 Andi Gutmans、Stig Bakken 和 Derick Rethans 的[GuBaRe2005]。

自由，免费

你自己的私有版本，我更希望你将反馈和补丁发送至 <sb@sebastian-bergmann.de>。

本书约定

下面是本书排版方面的一些约定：

斜体字

指示新的术语、URL、电邮地址、文件名、文件扩展名、路径名、目录和 Unix 实用程序。

等宽字体

指示命令、选项、开关、变量、函数、命名空间、方法、模块、参数、值、对象、文件内容或者命令输出。

等宽粗体

显示应由用户输入的命令或其他文本。

等宽斜体

显示应被用户提供的值替换的文本。

你应该特别留意采用下列样式从（一般）文本中分离出来的注意点：

注意

这是个提示、建议或者常规注意。它含有关于在谈话题的有用的辅助信息。

警告

这是个警告或提醒。

鸣谢

我要感谢 Kent Beck 和 Erich Gamma 开发了 JUnit，给了我编写 PHPUnit 的灵感。还要感谢 Kent Beck 写了“JUnit Pocket Guide”[Beck2004]，激起了我写本书的念头。感谢本书的发起者，O'Reilly 的 Allison Randal、Alexandra Follenius 和 Barbara Weiss。

感谢 Andi Gutmans、Zeev Suraski 和 Marcus Börger 在 PHP 5 的核心，Zend 引擎 2 方面的工作。感谢 Derick Rethans 开发了 Xdebug，这个 PHP 扩展让 PHPUnit 拥有了（分析）代码覆盖率功能。最后，感谢模拟对象系统的初始开发者 Jan Borsodi、协助制作代码覆盖率报表生成程序的 Michael Lively

Jr.和 Jan Kneschke，还有为 Phing 编写 PHPUnit 任务的 Michiel Rook。

第 1 章 自动化测试

再好的程序员也会犯错．程序员的好坏区别在于，好的程序员尽可能早地利用测试检测错误所在。越早测试，发现错误的机会越大，发现和修正的代价就越小。这解释了为什么在软件发布前才进行测试会有那么多问题。大多数错误都未被发现，而且修正已发现错误的代价如此之大，以至于你必须有选择地进行处理，因为根本不可能全部修复。

使用 PHPUnit 进行测试与你曾经的做法并非完全不同。它只是在操作方式上有所不同。区别在于*测试*和*执行成套测试*，前者检查程序行为是否符合预期，后者是可运行的代码片断，它们自动测试软件各部分（单元）的正确性。这些可运行的代码片断称为单元测试。

本章我们从简单的基于 `print` 的测试代码开始，到完整的自动化测试（结束）。假设要我们测试 PHP 内建的 `array`。一个要测试的功能是函数 `sizeof()`。对于新建的数组，我们期望 `sizeof()` 函数返回 `0`。假如一个元素后，`sizeof()` 应该返回 `1`。范例 1.1 是我们要测试的（代码）。

范例 1.1: 测试数组和 sizeof()

```
<?php
$fixture = array();

// $fixture 应该为空。

$fixture[] = 'element';

// $fixture 应该含有一个元素。

?>
```

检查我们是否得到预期结果的简单方法是在增加元素前后显示 `sizeof()` 的结果（见 Example 1.2）。如果先后得到 `0` 和 `1`，`array` 和 `sizeof()` 行为符合预期。

范例 1.2: 使用 print 测试数组和 sizeof()

```
<?php
$fixture = array();
print sizeof($fixture) . "\n";

$fixture[] = 'element';
print sizeof($fixture) . "\n";
?>
```

```
0

1
```

现在，我们要把需要人工解释（结果）的测试变为自动运行的测试。在范例 1.3 中，我们把预期值和实际值的比较写入测试代码，如果相同就输出 ok。一旦看到 not ok 消息，我们就知道有错误。

范例 1.3: 比较预期值和实际值以测试数组和 sizeof()

```
<?php
$fixture = array();
print sizeof($fixture) == 0 ? "ok\n" : "not ok\n";

$fixture[] = 'element';
print sizeof($fixture) == 1 ? "ok\n" : "not ok\n";
?>
```

```
ok

ok
```

现在我们把预期值和实际值得比较提取出来，放入一个函数，它会在存在差异时扔出一个异常。（范例 1.4）。这有 2 个好处：编写测试变得更容易，而且只在出错时产生输出。

范例 1.4: 使用断言函数测试数组和 sizeof()

```
<?php
$fixture = array();
assertTrue(sizeof($fixture) == 0);
```

```
$fixture[] = 'element';
assertTrue(sizeof($fixture) == 1);

function assertTrue($condition)
{
    if (!$condition) {
        throw new Exception('Assertion failed.');
    }
}
?>
```

现在测试完全自动化了。相对于我们的第一版*测试*，该版已是*自动化测试*。

使用自动化测试的目的是少犯错。即使有了极好的测试，你的代码仍不理想，一旦开始自动化测试，你将会发现缺陷明显减少。自动化测试确保你的代码的可信性。这种可信性使你可以在设计上做出更大胆的跨越（解构），同队友合作得更好（跨团队测试），改善同客户的关系，并且每晚安心回家，因为（事实）证明你的努力使系统比以前更好了。

第 2 章 PHPUnit 的目标

目前为止，我们只有 2 个用于内建的 `array` 和 `sizeof()` 函数的测试。当我们开始测试 PHP 提供的众多 `array_*()` 函数时，我们将需要为它们中的每一个都编写一个测

试。我们可以从头开始为所有的测试编写基础设施。然而，一次性地编写一个测试基础设施并且只编写每个测试的特有部分会更好。PHPUnit 就是这样的一个基础设施。

类似 PHPUnit 这样的框架必须解决一系列的限制，其中的一些似乎相互冲突。同时，测试应该：

（编写）易学。

> 如果编写测试很难学，开发者就不会去学。

易写。

> 如果测试不易编写，开发者就不会去写。

易读。

> 测试代码不应引入额外的开销（复杂性？ - 译注），以便测试本身不被旁杂干扰。

容易执行。

> 测试应该点下按钮就能运行，并且以清晰明确的格式呈现结果。

快速执行。

测试应该快速运行，以便能够每天运行成百上千次。

相互独立。

测试不应该相互影响。如果测试的运行顺序改变，测试的结果应该不变。

组合。

我们应能将任意数量的测试以任意的组合运行。这是相互独立的必然结果。

这些约束中有 2 个主要的冲突：

（编写）易学 VS 易写。

通常测试不需要程序语言的全部灵活性。许多测试工具提供它们特有的脚本语言，这些语言只含有编写测试所需特性的最小集。由此测试易于读写，因为没有干扰是你从测试内容分心。然而，学习其他程序语言和程序工具集并不方便，也会干扰思维。

相互独立 VS 快速执行。

如要一个测试的结果不影响其它测试，每个测试都应在运行前创建完整的运行场景（world）状态，并在结束时将它恢复为原始状态。可是，建立场景需时很久：例如连接数据库并用真实数据初始化为已知状态。

PHPUnit 使用 PHP 作为测试语言来解决这些冲突。有时全功能的 PHP 对于编写短小直接的测试显得小题大做，但是另一方面 PHP 让我们可以利用程序员掌握的全部经验和工具。由于我们要说服心怀抵触的测试员，降低编写那些初始测试的门槛是非常重要的。

PHPUnit errs on the side of isolation over quick execution. 独立测试的价值在于它们提供高质量的反馈。你不会看到带有一连串测试失败的报告，这实际上是由测试集开头的一个测试失败弄乱了其他测试的场景引起的。这种趋向于分离的测试鼓励带有大量简单对象的设计。每个对象都能单独被快速测试。由此得到的结果是更好的设计*和*更快的测试。

PHPUnit 假设多数测试会成功，并且成功测试的细节不值得报告。失败的测试才值得注意和报告。除了统计运行的测试数量，大多数成功的测试无须关注。这个假设被内建于报告类而不是 PHPUnit 内核中。在测

试报告中，你能看到运行了多少测试，但是只有失败的测试才有明细。

测试应该有良好的粒度（划分）-只测试某个对象的某个方面。因此，首次测试失败就会中断测试运行并且 PHPUnit 会报告该错误。运行众多小测试很巧妙。测试具有良好的粒度会改善系统的总体设计。

使用 PHPUnit 测试一个对象，你只能利用对象公用接口。让测试仅仅基于公共可见性行为，使你在差劲的设计在系统中广为蔓延之前就能面对和解决困难的设计问题。

第 3 章 安装 PHPUnit

PHPUnit 应该使用 PEAR 安装程序安装。它为 PHP 软件包提供分发系统，是 PEAR 的支柱，并且从 4.3.0 版开始随 PHP 一同发行。

用于分发 PHPUnit 的 PEAR 通道（`pear.phpunit.de`）需要在本地 PEAR 环境登记：

```
pear channel-discover pear.phpunit.de
```

这种做法只需进行一次。现在 PEAR 安装程序可从 PHPUnit 通道安装软件包了：

```
pear install phpunit/PHPUnit
```

之后，你能在你的本地 PEAR 目录中见到 PHPUnit 源代码文件；该路经通常是 */usr/lib/php/PHPUnit*.

尽管安装 PHPUnit 只支持通过 PEAR 安装，你也能手动安装 PHPUnit。对于手动安装，做法如下：

1. 从 `http://pear.phpunit.de/get/` 下载一个发行存档并解压至你的 *php.ini* 配置文件的 `include_path` 中列举的一个目录。
2. 准备 *phpunit* 脚本：
   a. 将 *pear-phpunit* 脚本改名为 *phpunit*。
   b. 用你得 PHP 命令行解释器的路径（通常是 */usr/bin/php*）替换文件中的`@php_bin@`。
   c. 将它拷贝到你的 *PATH* 中的一个目录并使之可运行（`chmod +x phpunit`）。
3. 准备 *PHPUnit/Util/Fileloader.php* 脚本：
   a. 用你得 PHP 命令行解释器的路径（通常是 */usr/bin/php*）替换文件中的`@php_bin@`。

第 4 章 编写 PHPUnit 测试

范例 4.1 显示我们必须如何修改范例 1.4 中的 2 各测试以使它们可用于 PHPUnit。

范例 4.1: 使用 PHPUnit 测试数组和 sizeof()

```
<?php
require_once 'PHPUnit/Framework.php';

class ArrayTest extends PHPUnit_Framework_TestCase
{
    public function testNewArrayIsEmpty()
    {
        // 创建数组 fixture。
        $fixture = array();

        // 断言数组 fixture 的尺寸是 0。
        $this->assertEquals(0, sizeof($fixture));
    }

    public function testArrayContainsAnElement()
    {
        // 创建数组 fixture。
        $fixture = array();

        // 向数组 fixture 增加一个元素。
        $fixture[] = 'Element';

        //断言数组 fixture 的尺寸是 1。
        $this->assertEquals(1, sizeof($fixture));
    }
}
?>
```

> 只要你想到输入一些东西到 print 语句或调试表达式中，就用测试代替它。

> --Martin Fowler

范例 4.1 显示使用 PHPUnit 编写测试的基本步骤：

1. 将类 `Class` 的测试写入类 `ClassTest`。
2. `ClassTest` 继承（通常）自 `PHPUnit_Framework_TestCase`。
3. 测试都是公用方法，命名为 `test*`。

   另外，你可以在方法的文档注释块（docblock）中使用 `@test` 注解来把它标记为测试方法。

4. 在测试方法中，类似 `assertEquals()`（见表 22.1）的断言方法用来断言一个实际值（应该）匹配一个期望值。

数据提供者

测试方法能接受人意的参数。这些参数将由一个数据提供者方法提供（范例 4.2 中的 `provider()`）。 要用到的数据提供着方法是用 `@dataProvider` 注解指定的。

数据提供者方法必须是 `public` 和 `static` 的，而且返回值必须是数组的数组或者实现了 `Iterator` 接口并且每次迭代产生一个数组的对象。对于每个数组（即集合的部分），都将以其内容为参数调用测试方法。

范例 4.2: 使用数据提供者

```
<?php
class DataTest extends PHPUnit_Framework_TestCase
{
    public static function provider()
    {
        return array(
          array(0, 0, 0),
          array(0, 1, 1),
          array(1, 0, 1),
          array(1, 1, 3)
        );
    }

    /**
     * @dataProvider provider
     */
    public function testAdd($a, $b, $c)
    {
        $this->assertEquals($c, $a + $b);
    }
}
?>
```

```
phpunit DataTest

PHPUnit 3.2.10 by Sebastian Bergmann.



...F



Time: 0 seconds



There was 1 failure:
```

```
1) testAdd(DataTest) with data (1, 1, 3)

Failed asserting that <integer:2> matches expected value
<integer:3>.

/home/sb/DataTest.php:21


FAILURES!

Tests: 4, Failures: 1.
```

测试异常

范例 4.3 显示如何使用`@expectedException`注解测试被测试的代码中是否跑出异常。

范例 4.3: 使用@expectedException 注解

```
<?php
require_once 'PHPUnit/Framework.php';

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

```
phpunit ExceptionTest

PHPUnit 3.2.10 by Sebastian Bergmann.
```

```
F

Time: 0 seconds

There was 1 failure:

1) testException(ExceptionTest)
Expected exception InvalidArgumentException

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

另外，可用 `setExpectedException()` 方法设定期望的异常，如范例 4.4 所示。

范例 4.4: 期望被测试代码引起异常

```
<?php
require_once 'PHPUnit/Framework.php';

class ExceptionTest extends PHPUnit_Framework_TestCase
{
    public function testException()
    {
        $this->setExpectedException('InvalidArgumentExcep
tion');
    }
```

```
}
?>
```

```
phpunit ExceptionTest

PHPUnit 3.2.10 by Sebastian Bergmann.


F


Time: 0 seconds


There was 1 failure:


1) testException(ExceptionTest)

Expected exception InvalidArgumentException


FAILURES!

Tests: 1, Failures: 1.
```

表 4.1 现是可用于测试异常的方法。

表 4.1. 用于测试异常的方法

方法	含义
`void setExpectedException(string $exceptionName)`	设置期望的异常名称为`$exceptionName`。

方法	含义
`String getExpectedException()`	返回期望的异常名称。

你也可以使用范例 4.5 中显示的方法测试异常。

范例 4.5: 测试异常的另一方法

```
<?php
require_once 'PHPUnit/Framework.php';

class ExceptionTest extends PHPUnit_Framework_TestCase {
    public function testException() {
        try {
            // ... 期望引发异常的代码 ...
        }

        catch (InvalidArgumentException $expected) {
            return;
        }

        $this->fail('未引发期望的异常。');
    }
}
?>
```

如果范例 4.5 中期望引发异常的代码没有引发期望的异常，那么后面对 `fail()` （见表 22.3）的调用将

终端测试并发信号报告测试问题。如果引发了期望的异常，将执行 `catch` 块，测试将成功结束。

第 5 章 命令行测试启动器

PHPUnit 命令行测试启动器可通过 *phpunit* 命令调用。下面的代码显示如何借助 PHPUnit 命令行测试启动器运行测试：

```
phpunit ArrayTest

PHPUnit 3.2.10 by Sebastian Bergmann.



..



Time: 0 seconds




OK (2 tests)
```

对于每个测试的运行，PHPUnit 命令行工具输出一个字符指示进展：

.

当测试成功时输出。

F

当运行测试方法过程中一个断言失败时输出。

E

当运行测试方法过程中产生一个错误时输出。

S

当测试被跳过时输出（见第 10 章）。

I

当测试被标记为不完整或未实现时输出（见第 10 章）。

PHPUnit 区分*失败*和*错误*。失败是指违反 PHPUnit 断言，例如 `assertEquals()` 调用失败。错误是指意外的异常或 PHP 错误。有时这种区分非常有用，因为错误通常比失败容易修复。如果你得到一大串问题，最好先解决所有错误，然后再看还有没有失败。

我们看看下面代码中的命令行测试启动器的开关：

```
phpunit --help

PHPUnit 3.2.10 by Sebastian Bergmann.


Usage: phpunit [switches] UnitTest [UnitTest.php]

```

```
  --log-graphviz <file>  Log test execution in GraphViz
markup.

  --log-json <file>     Log test execution in JSON format.

  --log-tap <file>      Log test execution in TAP format to
file.

  --log-xml <file>      Log test execution in XML format to
file.

  --log-metrics <file>   Write metrics report in XML format.

  --log-pmd <file>      Write violations report in PMD XML
format.



  --coverage-html <dir>  Generate code coverage report in
HTML format.

  --coverage-xml <file>  Write code coverage information in
XML format.



  --test-db-dsn <dsn>    DSN for the test database.

  --test-db-log-rev <r>  Revision information for database
logging.

  --test-db-prefix ...   Prefix that should be stripped from
filenames.

  --test-db-log-info ... Additional information for database
logging.



  --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).


  --loader <loader>     TestSuiteLoader implementation to
use.

  --repeat <times>      Runs the test(s) repeatedly.


  --tap                Report test execution progress in TAP
format.

  --testdox            Report test execution progress in
TestDox format.


  --no-syntax-check     Disable syntax check of test source
files.

  --stop-on-failure     Stop execution upon first error or
failure.

  --verbose            Output more verbose information.

  --wait               Waits for a keystroke after each test.


  --skeleton           Generate skeleton UnitTest class for
Unit in Unit.php.


  --help               Prints this usage information.
```

```
  --version            Prints the version and exits.


  --configuration <file> Read configuration from XML file.
  -d key[=value]        Sets a php.ini value.
```

`phpunit UnitTest`

运行类 `UnitTest` 提供的测试。该类应该在源文件 *`UnitTest.php`* 中声明。

`UnitTest` 必须是继承自 `PHPUnit_Framework_TestCase` 的类或者提供 `public static suite()` 方法的类，该方法返回一个 `PHPUnit_Framework_Test` 对象，例如一个 `PHPUnit_Framework_TestSuite` 类的实例。

`phpunit UnitTest UnitTest.php`

运行类 `UnitTest` 提供的测试。着各类应该在指定的源文件中声明。

`--log-graphviz`

使用 GraphViz 标记为运行的测试生成日志文件。该文件可用比如工具 `dot` 呈现。可见第 16 章中更详细的资料。

请注意，该选项只有当 PEAR 软件包 `Image_GraphViz` 已被安装时才可用。

`--log-json`

产生 JSON 格式的日志文件。更多信息见第 16 章。

`--log-tap`

为运行的测试产生 Test Anything Protocol (TAP)各式的日志文件。更多详细信息见第 16 章。

`--log-xml`

为运行的测试产生 XML 格式的日志文件。更多详细信息见第 16 章。

`--log-metrics`

为测试中的系统产生附带软件规格的 XML 格式的日志文件。更多详细信息见第 17 章 。

请注意，该选项只在已安装 tokenizer 和 Xdebug 扩展时才可用。

`--log-pmd`

产生 XML 格式的日志文件，并且附带比如基于软件规格的违例。更多详细信息见第 17 章。

请注意，该选项只在已安装 tokenizer 和 Xdebug 扩展时才可用。

`--coverage-xml`

为运行的测试产生附带代码覆盖率信息的 XML 格式日志文件。更多详细信息见第 16 章。

请注意，该选项只在已安装 tokenizer 和 Xdebug 扩展时才可用。

`--coverage-html`

产生 HTML 格式的代码覆盖率报告。更多详细信息见第 14 章。

请注意，该选项只在已安装 tokenizer 和 Xdebug 扩展时才可用。

`--charset`

指定将用于`--report` 的字符集。

`--test-db-*`

将测试结果和代码覆盖率数据写入数据库。更多详细信息见第 16 章。

请注意，该选项只在已安装 PDO 扩展时才可用。

`--testdox-html` 和`--testdox-text`

位运行的测试产生 HTML 或纯文本格式的 agile 文档。更多详细信息见第 15 章。

`--filter`

只运行名称匹配给定模式的测试。该模式可为单个测试的名称或匹配多个测试名的正则表达式。

`--group`

只运行来自指定组（一个或多个）的测试。可用 `@group` 注解将测试标记为属于某个组。

`--exclude-group`

排除来自指定组(一个或多个)的测试。可用`@group`注解将测试标记为属于某个组。

`--loader`

指定要用的 `PHPUnit_Runner_TestSuiteLoader` 实现。

标准的测试套件加载器将在当前工作目录和 PHP 的 `include_path` 配置指令指定的每个目录中查找源文件。依照 PEAR 的命名约定，例如名为 `Project_Package_Class` 的类被映射为源文件 *`Project/Package/Class.php`*。

`--repeat`

重复运行测试指定的次数。

`--stop-on-failure`

首次错误或失败即停止运行。

`--no-syntax-check`

禁用对测试源文件的语法检查。

`--tap`

使用 Test Anything Protocol (TAP)报告测试进程。更多信息见第 16 章。

`--testdox`

将测试进程作为 agile 文档进行报告。更多信息见第 15 章。

`--verbose`

输出更多详细信息，例如未完成或被跳过的测试的名字。

`--wait`

每个测试之后等待按键。如果你在一个只在活动时（active）才能保持打开状态的窗口中运行测试，这会很有用。

`--skeleton`

为类 `Unit`（在 *`Unit.php`* 中）产生一个测试用例类 `UnitTest`（在 *`UnitTest.php`* 中）。更多信息见第 18 章。

`--configuration`

从 XML 文件中读取配置。更多信息见附录 A。

`-d`

设置给定的 PHP 配置选项的值。

注意

当被测试代码含有 PHP 语法错误时，TextUI 测试启动器可能不打印错误信息就退出。标准的测试套件加载器会检查测试套件源文件的 PHP 语法错误，但是不包括被测试套件源文件包含的源文件。

第 6 章 Fixtures

编写测试的一个最耗时部分是编写代码设置场景为已知状态以及在测试完成时将其还原为初始状态。这个已知状态称为测试的 *fixture*。

在范例 4.1 中，fixture 只是存储在变量`$fixture`中的简单数组。然而，大多时候 fixture 比简单数组更复杂，因此相关装配代码的数量也随之增长。测试的实际内容迷失在这些装配 fixture 的干扰中了。当你编写多个带有相似 fixture 的测试时，这个问题变得更糟。如果没有测试框架的帮助，我们就得为我们编写的每个测试复制那些装配 fixture 的代码。

PHPUnit 支持共享装配代码。在某个测试方法运行前，会调用一个名为`setUp()`的模板方法。`setUp()`是你创建测试所需对象的地方。一旦测试方法已经结束，不论成功还是失败，就会调用另一个称为`tearDown()`的模板方法。`tearDown()`是你清理测试用到的对象的地方。

我们现在可以解构范例 4.1，使用`setUp()`消除之前的代码复制。首先声明实例变量，`$fixture`，它将取代方法的局部变量。然后将数组 fixture 的创建过程放入方法`setUp()`中。最后，从测试方法中移除冗余代码，使用新引入的实例变量，`$this->fixture`，取代用在断言方法`assertEquals()`处的局部变量`$fixture`。

范例 6.1: 使用 setUp()创建数组 fixture

```php
<?php
require_once 'PHPUnit/Framework.php';

class ArrayTest extends PHPUnit_Framework_TestCase
{
    protected $fixture;

    protected function setUp()
    {
        // 创建数组 fixture。
        $this->fixture = array();
    }

    public function testNewArrayIsEmpty()
    {
        // 断言数组 fixture 的尺寸是 0。
        $this->assertEquals(0, sizeof($this->fixture));
    }

    public function testArrayContainsAnElement()
    {
        // 向数组 fixture 增加一个元素。
        $this->fixture[] = 'Element';

        // 断言数组 fixture 的尺寸是 1。
        $this->assertEquals(1, sizeof($this->fixture));
    }
}
?>
```

运行每个测试方法都会分别调用 `setUp()`和 `tearDown()` 一次。然而，对同一测试用例类中的所有测试方法只调用二者一次似乎比较解决，（但是）这么做会导致编写完全相互独立的测试异常困难。

不只是`setUp()`和`tearDown()`对每个测试方法都只运行一次，甚至所有测试方法都是运行于新的测试用例类上的（见第 21 章）。

setUp()多 tearDown()少

理论上 `setUp()`和 `tearDown()`是精确（一一）对应的，但事实上并非如此。实际上，只有你在 `setUp()`中分配了如文件或套接字等外部资源时，才需要实现 `tearDown()`。如果你的 `setUp()`只创建了普通得 PHP 对象，通常可以忽略 `tearDown()`。可是，如果在 `setUp()` 中创建了很多对象，你可能想在 `tearDown()`中 `unset()` 指向那些对象的变量以使它们可被垃圾回收。测试用例相关对象的垃圾回收并不是课预知的。

变异

当两个测试的装配器（setup）只有些许的不同时会怎么样？有两种可能：

- 如果 `setUp()`的差异非常微小，把其中有区别的代码移到测试方法中。
- 如果某个 `setUp()`确实不同，你需要一个不同的测试用例类。根据装配器中的差异命名它。

共享 Fixture

很少需要在测试之间共享 fixture，这其中的多数情形源于一个未解决的设计问题。

多个测试之间共享 fixture 的一个不错且有意义的例子是数据库连接：登陆一次，重用数据库连接而不是每个测试都创建新连接。这使得测试运行加快。

范例 6.2 分别使用类 `PHPUnit_Framework_TestSuite`（见"套件级装配器"一章）的模板方法 `setUp()`和 `tearDown()`在测试套件的首个测试之前连接数据库以及套件的最后测试之后断开数据库连接。

`PHPUnit_Framework_TestSuite`对象的`$sharedFixture`属性在`PHPUnit_Framework_TestSuite`对象集合及`PHPUnit_Framework_TestCase`对象中都可用。

范例 6.2: 在测试套件内的各测试间共享 fixture

```
<?php
require_once 'PHPUnit/Framework.php';
```

```
class DatabaseTestSuite extends PHPUnit_Framework_TestSui
te
{
    protected function setUp()
    {
        $this->sharedFixture = new PDO(
          'mysql:host=wopr;dbname=test',
          'root',
          ''
        );
    }

    protected function tearDown()
    {
        $this->sharedFixture = NULL;
    }
}
?>
```

测试间共享 fixture 带来的测试价值的缩减怎么强调都不过分。潜在的设计问题是对象间并不是松散耦合的。利用存根（见第 11 章）解决潜在的设计问题后再编写测试，比在测试间建立运行时依赖关系并错过改善设计的机会能得到更好的结果。(翻译是否有误？ - 译注）

第 7 章 组织测试套件

PHPUnit 的目标（见第 2 章）之一是测试应是可组合的：我们应能同时运行任意数量或任意组合的测试，

例如整个工程的所有测试，或者工程局部组件的所有类的测试，或者只是单个类的测试。

PHPUnit 框架的 `PHPUnit_Framework_TestSuite` 类允许我们将一些测试组织在若干测试套件构成的一个层次结构中。让我们通过一个例子看看 PHPUnit 特有的测试套件。

范例 7.1 显示一个删节版本的 *Tests/AllTests.php*，范例 7.2 显示一个删节版本的 *Tests/Framework/AllTests.php*。

范例 7.1: 类 AllTests

```
<?php
if (!defined('PHPUnit_MAIN_METHOD')) {
    define('PHPUnit_MAIN_METHOD', 'AllTests::main');
}

require_once 'PHPUnit/Framework.php';
require_once 'PHPUnit/TextUI/TestRunner.php';

require_once 'Framework/AllTests.php';
// ...

class AllTests
{
    public static function main()
    {
        PHPUnit_TextUI_TestRunner::run(self::suite());
    }

    public static function suite()
```

```
    {
        $suite = new PHPUnit_Framework_TestSuite('PHPUnit
');

        $suite->addTest(Framework_AllTests::suite());
        // ...

        return $suite;
    }
}

if (PHPUnit_MAIN_METHOD == 'AllTests::main') {
    AllTests::main();
}
?>
```

范例 7.2: 类 Framework_AllTests

```
<?php
if (!defined('PHPUnit_MAIN_METHOD')) {
    define('PHPUnit_MAIN_METHOD', 'Framework_AllTests::ma
in');
}

require_once 'PHPUnit/Framework.php';
require_once 'PHPUnit/TextUI/TestRunner.php';

require_once 'Framework/AssertTest.php';
// ...

class Framework_AllTests
{
    public static function main()
    {
        PHPUnit_TextUI_TestRunner::run(self::suite());
    }
```

```
    public static function suite()
    {
        $suite = new PHPUnit_Framework_TestSuite('PHPUnit
 Framework');

        $suite->addTestSuite('Framework_AssertTest');
        // ...

        return $suite;
    }
}

if (PHPUnit_MAIN_METHOD == 'Framework_AllTests::main') {
    Framework_AllTests::main();
}
?>
```

类 `Framework_AssertTest` 是个扩展了 `PHPUnit_Framework_TestCase` 的标准测试用例。

运行 *`Tests/AllTests.php`* 则使用 TextUI 测试启动器运行全部测试，然而运行 *`Tests/Framework/AllTests.php`* 则只运行类 `PHPUnit_Framework_*` 的测试。

只有像这样运行测试才会调用类 `AllTests` 中声明的 `main()` 方法：

```
php AllTests.php

PHPUnit 3.2.10 by Sebastian Bergmann.
```

```
.............................................................
..  60 / 227

.............................................................
.. 120 / 227

.............................................................
.. 180 / 227

........................SSSSSS.............


Time: 4 seconds




OK, but incomplete or skipped tests!

Tests: 227, Skipped: 6.
```

运行 `phpunit AllTests` 来启动测试不会调用 `main()` 方法。

套件级装配器

类 `PHPUnit_Framework_TestSuite` 提供两个模板方法，`setUp()` 和 `tearDown()`，它们分别在测试套件的首个测试前和最后测试后被调用。

范例 7.3: 类 MySuite

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

class MySuite extends PHPUnit_Framework_TestSuite
{
    public static function suite()
```

```
    {
        return new MySuite('MyTest');
    }

    protected function setUp()
    {
        print "\nMySuite::setUp()";
    }

    protected function tearDown()
    {
        print "\nMySuite::tearDown()";
    }
}
?>
```

范例 7.3 中，被加入测试套件 `MySuite` 中的测试用例类 `MyTest` 有两个测试方法，`testOne()`和 `testTwo()`，以及方法 `setUp()`和 `tearDown()`。输出显示了这八个方法的调用顺序：

```
MySuite::setUp()

MyTest::setUp()

MyTest::testOne()

MyTest::tearDown()

MyTest::setUp()

MyTest::testTwo()

MyTest::tearDown()

MySuite::tearDown()
```

通过类 PHPUnit_Framework_TestSuite 的 setUp() 方法存储在 $this->sharedFixture 中的变量，能作为 $this->sharedFixture 在测试套件对象聚集的所有的测试中使用（见“共享 Fixture”一章）。

第 8 章 测试用例扩展

PHPUnit 提供了用于测试类的标准基类 PHPUnit_Framework_TestCase 的扩展，这有助于编写用于输出和性能衰退的测试。

测试输出

有时候会想要断定方法的执行情况，比如期望产生怎样的输出（例如通过 echo 或 print）。类 PHPUnit_Extensions_OutputTestCase 使用 PHP 的输出缓冲特性提供此处所需的功能。

范例 8.1 显示如何继承 PHPUnit_Extensions_OutputTestCase 并用它的 expectOutputString() 方法设置期望的输出。如果未产生期望的输出，测试将被定为失败。

范例 8.1: 使用 PHPUnit_Extensions_OutputTestCase

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

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

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

```
phpunit OutputTest

PHPUnit 3.2.10 by Sebastian Bergmann.


.F


Time: 0 seconds


There was 1 failure:


1) testExpectBarActualBaz(OutputTest)

Failed asserting that two strings are equal.

expected string <bar>
```

```
difference      <  x>
got string      <baz>


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

表 8.1 PHPUnit_Extensions_OutputTestCase 提供的方法。

表 8.1. OutputTestCase

方法	含义
void expectOutputRegex(string $regularExpression)	设定期望值为输出匹配 $regularExpression。
void expectOutputString(string $expectedString)	设定期望值为输出同 $expectedString 一样。
bool setOutputCallback(callable $callback)	设定回调方法，例如用于规格化实际输出。

测试性能

你可以让测试类扩展自`PHPUnit_Extensions_PerformanceTestCase`，以便可以测试（例如）函数或方法的运行是否超过指定的时间限制。

范例 8.2 显示如何继承`PHPUnit_Extensions_PerformanceTestCase`并用它的`setMaxRunningTime()`方法设定测试的最大运行时间。如果测试运行时间不在该时间限制以内，则被定为失败。

范例 8.2: 使用 PHPUnit_Extensions_PerformanceTestCase

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


class PerformanceTest extends PHPUnit_Extensions_Performa
nceTestCase
{
    public function testPerformance()
    {
        $this->setMaxRunningTime(2);
        sleep(1);
    }
}
?>
```

表 8.2 显示 PHPUnit_Extensions_PerformanceTestCase 提供的方法。

表 8.2. PerformanceTestCase

方法	含义
void setMaxRunningTime(int $maxRunningTime)	设定测试的最大运行时间为$maxRunningTime（单位：秒）。
integer getMaxRunningTime()	返回测试允许的最大运行时间。

还有两个 PHPUnit_Framework_TestCase 的扩展，PHPUnit_Extensions_Database_TestCase 和 PHPUnit_Extensions_SeleniumTestCase，它们分别在第 9 章和第 19 章中有所涉及。

第 9 章 数据库测试

当给你的软件创建测试时，你可能碰到需要单元测试的数据库代码。已经创建的数据库扩展提供了简便的方法将数据库置于某个已知状态，执行你的数据库相关的代码，并且确保在数据库中找到期望的数据。

创建一个新的数据库单元测试的最快方法时扩展类 `PHPUnit_Extensions_Database_TestCase`。该类提供了一些功能，可用来创建数据库连接，向数据库中植入数据，以及之后执行一个对数据库中的内容和可构成多种格式的数据集的比较的测试。你能在范例 9.1 中看到 `getConnection()`和 `getDataSet()`的实现例子。

范例 9.1: 构造一个数据库测试用例

```
<?php
require_once 'PHPUnit/Extensions/Database/TestCase.php';

class DatabaseTest extends PHPUnit_Extensions_Database_Te
stCase
{
    protected function getConnection()
    {
        $pdo = new PDO('mysql:host=localhost;dbname=testd
b', 'root', '');
        return $this->createDefaultDBConnection($pdo, 'te
stdb');
    }

    protected function getDataSet()
    {
        return $this->createFlatXMLDataSet(dirname(__FILE
__).'/_files/bank-account-seed.xml');
    }
}
?>
```

方法 `getConnection()`必须返回一个接口 `PHPUnit_Extensions_Database_DB_IDatabaseConnection` 的实现。方法 `createDefaultDBConnection()`能用来返回一个数据库连接。它接受一个 `PDO` 对象作为第一个参数，以及要测试的模式（schema）的名字作为第二个参数。

方法 `getDataSet()`必须返回一个接口 `PHPUnit_Extensions_Database_DataSet_IDataSet` 的实现。当前 PHPUnit 中有三种数据集可用。这些数据集在"数据集"一章中论述。

表 9.1. 数据库测试用例方法

方法	含义
`PHPUnit_Extensions_Database_DB_IDatabaseConnection getConnection()`	需要实现为返回数据库连接，该连接将被用于检查期望的数据集和表。
`PHPUnit_Extensions_Database_DataSet_IDataSet getDataSet()`	需要实现

方法	含义
	为返回数据集，该数据集将被用于数据库和装配和拆卸操作。
`PHPUnit_Extensions_Database_Operation_DatabaseOperation getSetUpOperation()`	需要重写为返回一个特定操作，该操作将在每个测试的开端在测试数据库上执行。“操作”一节中有各种操作的详细信

方法	含义
	息。
`PHPUnit_Extensions_Database_Operation_DatabaseOperation getTearDownOperation()`	需要重写为返回一个特定操作，该操作将在每个测试的末端在测试数据库上执行。“操作”一节中有各种操作的详细信息。

方法	含义
`PHPUnit_Extensions_Database_DB_DefaultDatabaseConnection createDefaultDBConnection(PDO $connection, string $schema)`	返回一个封装了 `PDO` 对象 `$connection` 的数据库连接。`$schema` 指定了将被测试的数据库模式。该方法的结果可从 `getConnection()` 返回。
`PHPUnit_Extensions_Database_DataSet_FlatXmlDataSet createFlatXMLDataSet(string $xmlFile)`	返回一个平整的 xml 数据集，它是从位于

方法	含义
	`$xmlFile` 指定的绝对路径处的 xml 文件创建的。关于平整的 xml 文件的更多详细信息可在“平整的 XML 数据集”一节找到。该方法的结果可从 `getDataSet()` 返回。
`PHPUnit_Extensions_Database_DataSet_XmlDataSet createXMLDataSet(string $xmlFile)`	返回一个 xml 数据集，它是

方法	含义
	从位于 `$xmlFile` 指定的绝对路径处的 xml 文件创建的。关于 xml 文件的更多详细信息可在“XML 数据集”一节找到。该方法的结果可从 `getDataSet()` 返回。
`void assertTablesEqual(PHPUnit_Extensions_Database_DataSet_ITable $expected, PHPUnit_Extensions_Database_DataSet_ITable $actual)`	如果表 `$expected` 的内容不匹配表

方法	含义
	$actual 中的内容则报错。
void assertDataSetsEqual(PHPUnit_Extensions_Database_DataSet_IDataSet $expected, PHPUnit_Extensions_Database_DataSet_IDataSet $actual)	如果数据集 $expected 的内容不匹配数据集$actual 中的内容则报错。

数据集

...

平整的 XML 数据集

...

XML 数据集

...

操作

...

数据库操作最佳实践

...

第 10 章 未完成和跳过的测试

未完成的测试

当你在处理一个新的测试用例类时，可能想由编写空测试方法开始，像这样：

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

以明了要编写哪些测试。空测试方法的问题是PHPUnit 框架把它们解释为成功。这个误解导致测试报告实效-你没法判断测试是实际上成功了还是仍未完成。在未实现的测试方法中调用`$this->fail()`也没有帮助，因为此后测试将被解释为失败。这将同解释未实现的测试为成功一样都是错误的。

如果我们分别将成功的测试和失败的必做绿灯和红灯，我们还需要黄灯标记未完成或未实现的测试。

PHPUnit_Framework_IncompleteTest 是个标记接口，用于标记当测试结果为未完成或当前未实现时引发的异常。PHPUnit_Framework_IncompleteTestError 是该接口的标准实现。

范例 10.1 先是一个测试用例类，SampleTest，它含有一个测试方法 testSomething()。通过在该方法中调用方便的 markTestIncomplete() 方法（自动引发 PHPUnit_Framework_IncompleteTestError 异常），我们把这个测试标记为未完成。

范例 10.1: 标记测试为未完成

```
<?php
require_once 'PHPUnit/Framework.php';

class SampleTest extends PHPUnit_Framework_TestCase
{
    public function testSomething()
    {
        //可选：随便测试什么都可以。
        $this->assertTrue(TRUE, 'This should already work.
');

        // 在这儿停住并将测试标记为未完成。
        $this->markTestIncomplete(
          'This test has not been implemented yet.'
        );
    }
}
?>
```

在 PHPUnit 命令行测试启动器的输出中，未完成的测试由 I 表示，如下面的例子所示：

```
phpunit --verbose SampleTest

PHPUnit 3.2.10 by Sebastian Bergmann.


SampleTest

I



Time: 0 seconds


There was 1 incomplete test:


1) testSomething(SampleTest)

This test has not been implemented yet.

/home/sb/SampleTest.php:14


OK, but incomplete or skipped tests!

Tests: 1, Incomplete: 1.
```

表 10.1 显示用于将测试标记为未完成的 API。

表 10.1. 未完成测试的 API

方法	含义
`void markTestIncomplete()`	标记当前测试为未完成。
`void markTestIncomplete(string $message)`	标记当前测试为未完成，同时使用`$message`作为说明性信息。

跳过的测试

特定的环境中并非所有的测试都能运行。考虑个例子，一个具有多个驱动以支持不同数据库系统的数据库提取层。MySQL 驱动的测试当然只能在 MySQL 服务器上运行。

范例 10.2 显示一个测试用例，`DatabaseTest`，它有个测试方法，`testConnection()`。在测试用例类的模板方法`setUp()`中，我们检查 MySQLi 扩展是否可用，否则使用`markTestSkipped()`方法跳过测试。

范例 10.2: 跳过一个测试

```
<?php
require_once 'PHPUnit/Framework.php';

class DatabaseTest extends PHPUnit_Framework_TestCase
{
    protected function setUp()
```

```
    {
        if (!extension_loaded('mysqli')) {
            $this->markTestSkipped(
              'The MySQLi extension is not available.'
            );
        }
    }

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

在 PHPUnit 命令行测试启动器的输出中，未完成的测试由 S 表示，如下面的例子所示：

```
phpunit --verbose DatabaseTest

PHPUnit 3.2.10 by Sebastian Bergmann.


DatabaseTest

S




Time: 0 seconds


There was 1 skipped test:
```

```
1) testConnection(DatabaseTest)

The MySQLi extension is not available.

/home/sb/DatabaseTest.php:11


OK, but incomplete or skipped tests!

Tests: 1, Skipped: 1.
```

表 10.2 显示用于跳过测试的 API。

表 10.2. 用于跳过测试的 API

方法	含义
`void markTestSkipped()`	标记当前测试为要跳过的。
`void markTestSkipped(string $message)`	标记当前测试为要跳过的，同时用`$message`作为说明性信息。

第 11 章 模拟对象

有时你需要检查某对象是否被正确调用了。看个例子：假如我们要测试是否正确调用了某观测另一对象的对象的方法，本例中是`update()`。

在范例 11.1 中，我们先用类`PHPUnit_Framework_TestCase`（见表 22.7）提供的`getMock()`方法为`Observer`装配一

个模拟对象。我们给出一个数组作为 `getMock()` 方法的第二个参数（可选的），因此只有类 `Observer` 的 `update()` 方法被替换为模拟实现。

然后我们使用 PHPUnit 提供的 Fluent 接口指定仿制品的行为和期望值。大体上，这意味着你不需要创建若干临时对象——例如，一个用于指定你期望 `update()` 方法被调用，另一个用于期望的参数——然后把它们绑在一起配置期望值。 如例中所示，改为链接方法调用。这使得代码更易读和"流畅"。

范例 11.1: 测试某方法被附以指定的参数调用一次

```
<?php
require_once 'PHPUnit/Framework.php';

class ObserverTest extends PHPUnit_Framework_TestCase
{
    public function testUpdateIsCalledOnce()
    {
        // 为 Observer 类创建一个只模拟 update() 方法的模拟对象。
        $observer = $this->getMock('Observer', array('upd
ate'));

        // 设定 update() 方法的期望值为只被以字符串"something"为
参数调用一次。
        $observer->expects($this->once())
                 ->method('update')
                 ->with($this->equalTo('something'));
```

```
        // 创建一个 Subject 对象并附上模拟的 Observer 对象。
        $subject = new Subject;
        $subject->attach($observer);

        // 调用$subject 对象的 doSomething()方法，我们期望该方法

        // 以字符串“something”为参数调用模拟的 Observer 对象的
update()方法。
        $subject->doSomething();
    }
}
?>
```

表 11.1 列出可用的匹配器，它们可用于表示期望模拟的方法要被运行多少次。

表 11.1. 匹配器

匹配器	含义
PHPUnit_Framework_MockObject_Matcher_AnyInvokedCount any()	返回一个匹配器，当它评估

匹配器	含义
	的方法被执行 0 或多次时匹配。
`PHPUnit_Framework_MockObject_Matcher_InvokedCount never()`	返回一个匹配器，当它评估的方法被从未被执行时匹配。

匹配器	含义
`PHPUnit_Framework_MockObject_Matcher_InvokedAtLeastOnce atLeastOnce()`	返回一个匹配器，当它评估的方法被至少执行一次时匹配。
`PHPUnit_Framework_MockObject_Matcher_InvokedCount once()`	返回一个匹配器，当它评估的方

匹配器	含义
	法恰好被执行一次时匹配。
PHPUnit_Framework_MockObject_Matcher_InvokedCount exactly(int $count)	返回一个匹配器，当它评估的方法正好被执行 $count 次时匹配。

匹配器	含义
`PHPUnit_Framework_MockObject_Matcher_InvokedAtIndex at(int $index)`	返回一个匹配器，当它评估的方法在特定的 `$index` 上被调用时匹配。

在表 22.2 中可以找到一些约束条件，它们可被一起用于这些匹配器。

自分流

另外，你能应用*自分流模式*代替模拟对象来测试 Subject 实现。该方法将测试用例自身用作存根。术语自分流来自医学行业的实践，安装管子从动脉取血并放回静脉来为注射麻药提供合适的位置。

首先，我们让测试用例类实现 Observer，它是要观测 Subject 的对象要实现的接口：

```
class ObserverTest extends PHPUnit_Framework_TestCase
implements Observer
{
}
```

下一步，我们实现一个 Observer 方法，update()，来检查当被观测的 Subject 对象的状态改变时该方法被调用：

```
public $wasCalled = FALSE;

public function update(Subject $subject)
{
  $this->wasCalled = TRUE;
}
```

现在我们可以写测试了。我们创建一个新的 Subject 对象并附上测试对象作为观测者。当 Subject 的状态改变——例如，通过调用它的 doSomething()方法——Subject

对象必须调用注册为观测者的所有对象的 update() 方法。我们用我们的 update() 实现中设置的 $wasCalled 实例变量检查 Subject 对象是否做期望的事情：

```
public function testUpdate()
{
  $subject = new Subject;
  $subject->attach($this);
  $subject->doSomething();

  $this->assertTrue($this->wasCalled);
}
```

注意我们创建一个新的 Subject 对象而不是依赖一个全局实例。存根鼓励这种形式的设计。它降低对象间的耦合并改善重用性。

如果你不熟悉自分流模式，（会感觉）测试很难阅读。这是什么？为什么一个测试用例同时也是个观测者？一旦你习惯这种用语，测试就易于阅读了。要了解的是测试的所有东西在同一个类中。

存根

只测试一件事情的测试比可能在很多地方产生失败的测试更有益。如何排除外部因素对测试的影响？简单地说，就是把花费大、杂乱、不可靠、速度慢或复杂

的资源换成为你的测试起见自动产生的存根。例如，你可以用常量替换实际上非常复杂的计算，至少可以针对单个测试（这样做）。

存根解决了分配代价高昂的外部资源的问题。例如，在测试间共享类似数据库连接有用，但为测试起见从根本上避免使用数据库会更好。

范例 11.2 显示如何存根化方法调用以及装配返回值。

范例 11.2: 存根化一个方法调用

```
<?php
require_once 'PHPUnit/Framework.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testStub()
    {
        $stub = $this->getMock('SomeClass', array('doSomething'));
        $stub->expects($this->any())
             ->method('doSomething')
             ->will($this->returnValue('foo'));

        // 调用$stub->doSomething()会立刻返回“foo”。
    }
}
?>
```

表 11.2 列出可用来为存根方法调用配置返回值的方法。

表 11.2. 存根 API

方法	含义
`PHPUnit_Framework_MockObject_Stub_Exception throwException(Exception $exception)`	设定对方法的所有调用都会跑出的异常。
`PHPUnit_Framework_MockObject_Stub_Return returnValue(mixed $value)`	设定对方法的所有调用都返回 `$value`。
`PHPUnit_Framework_MockObject_Stub_ConsecutiveCalls onConsecutiveCalls(mixed $value, ...)`	为方法的连续调用装配不同

方法	含义
	的返回值。

此外，你能自己编写存根并据此改善你的设计。通过单个 faÃ§ade（façade 模式？）访问广泛使用的资源，这样就能方便地用存根替换资源。例如，你有个 `Database` 对象，接口 `IDatabase` 的实现，而不是让直接数据库访问散落在代码各处。然后就能创建一个实现了 `IDatabase` 的存根并用于你的测试。你甚至可以创建一个选项，用于（选择）带存根数据库或真实的数据库运行测试，这样你的测试就既能在开发期间用于本地测试，又能用于真实数据库的综合测试。

需要存根化的功能趋向于集成在同一个对象中，提高了内聚性。通过使用单个内聚的接口引入各种功能，你就降低了同系统其他部分的耦合。

第 12 章 测试实践

> 你总是可以编写更多的测试。可是，你会很快发现，只有小部分测试（你能想象）是确实有用的。你需要的是编写这样的测

> 试，即使你认为它们应该成功却失败了，
> 或者即使你认为它们应该失败却成功了。
> 换个角度，考虑术语花费/收益（所指的）
> 方面。你应该编写能回馈信息的测试。
>
> --Erich Gamma

开发期间

当你你需要改变正在处理的软件的内部结构，以便它更易于理解以及能更轻易地修改而不改变其显著的行为时，一个测试套件在应用这些名为安全解构（的变更）方面是无价的。否则，当你实施重构时，你可能不会注意到系统崩溃了。

利用单元测试校验解构的转换步幅确实保持行为（不变）且未引入错误时，下面的情形有助于改善项目的代码和设计：

1.所有单元测试运行正确。
2.代码表达了设计原则。
3.代码没有冗余。
4.代码包含最少的类和方法。

需要向系统增加新功能时，先编写测试。那么当测试运行（正常）时你将完成开发。下一章会详细讨论本实践。

调试期间

当得到一个故障报告，你可能冲动要尽快修复故障。经验表明，这种动力不会带来益处；很可能对故障的修复引起另外的故障。

你可以这样做来把握你的冲动：

1. 证实可以再现故障。
2. 在代码中找到故障的最小尺度示范。例如，如果一个数字在输出中出现错误，就找到计算它的对象。
3. 编写一个自动化测试，它此时失败但是将在故障修复后成功。
4. 修复故障。

找到故障的最小的可靠再现给你真正检查故障起因的机会。当修复故障时，你编写的测试将会提高真正的修复它的机会，因为新测试降低了将来的代码改变将修缮复原的可能性。你以前写的所有测试降低了不经意地引发一个不同问题的可能性。

> 单元测试提供很多优势：
>
> - 测试给代码作者和审阅者带来信心，即补丁产生正确的结果。
> - 创作测试用例可以有效地推动开发者发现边界案例。
> - 测试提供了快速捕捉退化的良好方法，并且确保退化不会再次出现。
> - 单元测试为如何使用 API 提供了可行的范例，并且对于文档方面的努力也非常地有助益。
>
> 总之，完整的单元测试降低了任何个体改变的代价和风险。它将允许项目快速、大胆地获得主要架构上的改进。
>
> --Benjamin Smedberg

第 13 章 测试优先程序设计

单元测试是一些软件开发实践和处理过程，例如测试优先程序设计、极限编程和测试驱动开发的至关重要的组成部分。它们也虑及那些并未在语言结构层面提供支持该方法的程序设计语言中的契约式设计。

你可以在完成编程后立刻应用 PHPUnit 编写测试。然而，在引入错误后越早编写测试，它就越有价值。所以我们可以在可能引入缺陷的数日、数小时或数分钟后编写测试，而不是在代码完成的数月之后。为什么停步于此？为什么不在稍早于可能引入的缺陷之前编写测试？

构成了极限编程和测试驱动开发的测试优先程序设计建立于这个观念之上并把它发挥到了极致。以今日的计算能力，我们可以每天运行数以千计的测试数千次。我们可把从所有测试得来的反馈用于小幅步进式的编程，除了之前的所有测试，每一步都有新的自动化测试提供保证。。测试就像（攀岩用的）岩钉，确保不论发生什么都不会退回去。

最初编写测试时，它可能无法运行，因为你正在访问还未编程的对象和方法。起初会感觉这很奇怪，但是不久你就会习惯。把测试优先程序设计视为遵循针对接口而非针对实现的面向对象原则的实用途径：编写测试时你思考的是正在测试的对象的接口——从外面看该对象是什么样子。当你转为让测试真正地工作，你思考的是纯粹的实现。接口会被失败的测试修正。

> 测试驱动开发的要点是驱动软件实际所需的功能，而非程序员认为它应该具有的功能。这种做法起初会让你感到不适应，如果不是完全糊涂的话，但它不只是有意义，也会很快成为一种自然而文雅的软件开发方式。
>
> --Dan North

下列所述是个必要的测试优先程序设计的简短入门。你可以在其他书中更深入地探究这个主题，例如 Kent Beck 写的 *Test-Driven Development*[Beck2002] 或 Dave Astels 的 *A Practical Guide to Test-Driven Development*[Astels2003]。

银行账户实例

本节我们考虑个例子，一个表现银行账户的类。类 `BankAccount` 约定不仅需要获取和设定银行账户余额的方法，也需要存款和提款的方法。它也规定了下面两个必须保证的情形：

- 银行账户的初始余额必须为 0。
- 银行账户的余额不能为负数。

我们依照测试优先程序设计的方式，在编写 BankAccount 类代码之前为其编写测试。我们使用约定情形作为测试的基础并依此命名测试方法，如范例 13.1 所示：

范例 13.1: 用于 BankAccount 类的测试

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'BankAccount.php';

class BankAccountTest extends PHPUnit_Framework_TestCase
{
    protected $ba;

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

    public function testBalanceIsInitiallyZero()
    {
        $this->assertEquals(0, $this->ba->getBalance());
    }

    public function testBalanceCannotBecomeNegative()
    {
        try {
            $this->ba->withdrawMoney(1);
        }

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

            return;
        }
```

```
        $this->fail();
    }

    public function testBalanceCannotBecomeNegative2()
    {
        try {
            $this->ba->depositMoney(-1);
        }

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

            return;
        }

        $this->fail();
    }
}
?>
```

我们现在编写让第一个测试，`testBalanceIsInitiallyZero()`通过所需要的最少数量的代码。在我们的范例中这相当于实现 `BankAccount` 类的 `getBalance()`方法，如范例 13.2 所示。

范例 13.2: 通过 testBalanceIsInitiallyZero()测试所需的代码

```
<?php
class BankAccount
{
    protected $balance = 0;
```

```
    public function getBalance()
    {
        return $this->balance;
    }
}
?>
```

现在用于第一种约定情形的测试通过了，但是用于第二种的测试失败了，原因是我们还需要实现它们调用的方法。

```
phpunit BankAccountTest

PHPUnit 3.2.10 by Sebastian Bergmann.



.

Fatal error: Call to undefined method
BankAccount::withdrawMoney()
```

要让确保第二种约定情形的测试通过，我们现在需要实现方法 `withdrawMoney()`、`depositMoney()`和 `setBalance()`，如范例 13.3 所示。这些方法是以这样的方式编写的：当它们被以违反约定情形的非法数值调用时会引发一个 `BankAccountException`。

范例 13.3: 完整的 BankAccount 类

```php
<?php
class BankAccount
{
    protected $balance = 0;

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

    protected function setBalance($balance)
    {
        if ($balance >= 0) {
            $this->balance = $balance;
        } else {
            throw new BankAccountException;
        }
    }

    public function depositMoney($balance)
    {
        $this->setBalance($this->getBalance() + $balance);


        return $this->getBalance();
    }

    public function withdrawMoney($balance)
    {
        $this->setBalance($this->getBalance() - $balance);


        return $this->getBalance();
    }
}
?>
```

现在保证第二种约定情形的测试也通过了：

```
phpunit BankAccountTest

PHPUnit 3.2.10 by Sebastian Bergmann.


...


Time: 0 seconds




OK (3 tests)
```

另外，你可用 `PHPUnit_Framework_Assert` 类提供的静态断言方法将编写约定条件，并把它们作为契约式设计风格的断言放入你的代码中，如范例 13.4 中所示。当这些断言中的某个失败时，会引发一个 `PHPUnit_Framework_AssertionFailedError` 异常。这种方式减少了约定情形检查的代码并使测试更易读。然而，你将一个运行时依赖加入了项目。

范例 13.4: 带有契约式设计断言的 BankAccount 类

```
<?php
require_once 'PHPUnit/Framework.php';

class BankAccount
```

```
{
    private $balance = 0;

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

    protected function setBalance($balance)
    {
        PHPUnit_Framework_Assert::assertTrue($balance >=
0);

        $this->balance = $balance;
    }

    public function depositMoney($amount)
    {
        PHPUnit_Framework_Assert::assertTrue($amount >= 0)
;

        $this->setBalance($this->getBalance() + $amount);

        return $this->getBalance();
    }

    public function withdrawMoney($amount)
    {
        PHPUnit_Framework_Assert::assertTrue($amount >= 0)
;
        PHPUnit_Framework_Assert::assertTrue($this->balan
ce >= $amount);

        $this->setBalance($this->getBalance() - $amount);

        return $this->getBalance();
    }
}
?>
```

通过把约定条件写入测试，我们已用契约式设计编制了 `BankAccount` 类。然后我们遵循测试优先程序设计编写了测试通过所需的代码。然而，我们忘记了编写以不违反约定情形的合法值调用 `setBalance()`、`depositMoney()`和 `withdrawMoney()`的测试。我们需要一种方法来测试我们的测试，至少是衡量它们的质量。这种方法是代码覆盖率信息分析，我们下一步将讨论它。

第 14 章 代码覆盖率分析

你以获悉如何应用单元测试来测试你的代码？但是如何测试你的测试？如何找到尚未被测试的代码——或者换句话说，测试尚未*覆盖*的代码？如何读两侧是完整性？所有这些问题都能通过一个称为代码覆盖率分析的实践来解答。当测试运行时代码覆盖率分析让你明了产品代码的哪些部分被执行了。

PHPUnit 的代码覆盖率分析利用 Xdebug 扩展提供的语句覆盖率功能。举个关于语句覆盖率表示什么的例子，如果某方法有 100 行代码，而且运行测试时实际只执行了其中的 75 行，那么认为该方法的代码覆盖率是 75%。

让我们为范例 13.3 中的类 BankAccount 生成一个代码覆盖率报告。

```
phpunit --coverage-html ./report BankAccountTest

PHPUnit 3.2.10 by Sebastian Bergmann.


...


Time: 0 seconds


OK (3 tests)


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

图形 14.1 显示一个来自代码覆盖率报告的节选。当运行测试时，执行过的代码行被绿色高亮显示，可执行但并为执行的代码行被红色高亮显示，而"无用代码"悲橙色高亮显示。实际代码行左侧的数字指示有多少涉及那一行。

图表 **14.1.setBalance()**的代码覆盖率

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

单击一个被覆盖到的行的行号会打开一个面板（见图表 14.2），它显示涉及该行的测试用例。

图表 **14.2.** 带涉及测试信息的面板

```
82                  :     /**
83                  :      * Sets the bank account's bala
84                  :      *
85                  :      * @param  float $balance
86                  :      * @throws BankAccountExceptio
87                  :      * @access protected
88                  :      */
89                  :     protected function setBalance($
90                  :     {
91                2 :         if ($balance >= 0) {
                                $this->balance = $balan
                            else {
                                throw new BankAccountEx
```

2 tests cover line 91

- testBalanceCannotBecomeNegative(BankAccountTest)
- testBalanceCannotBecomeNegative2(BankAccountTest)

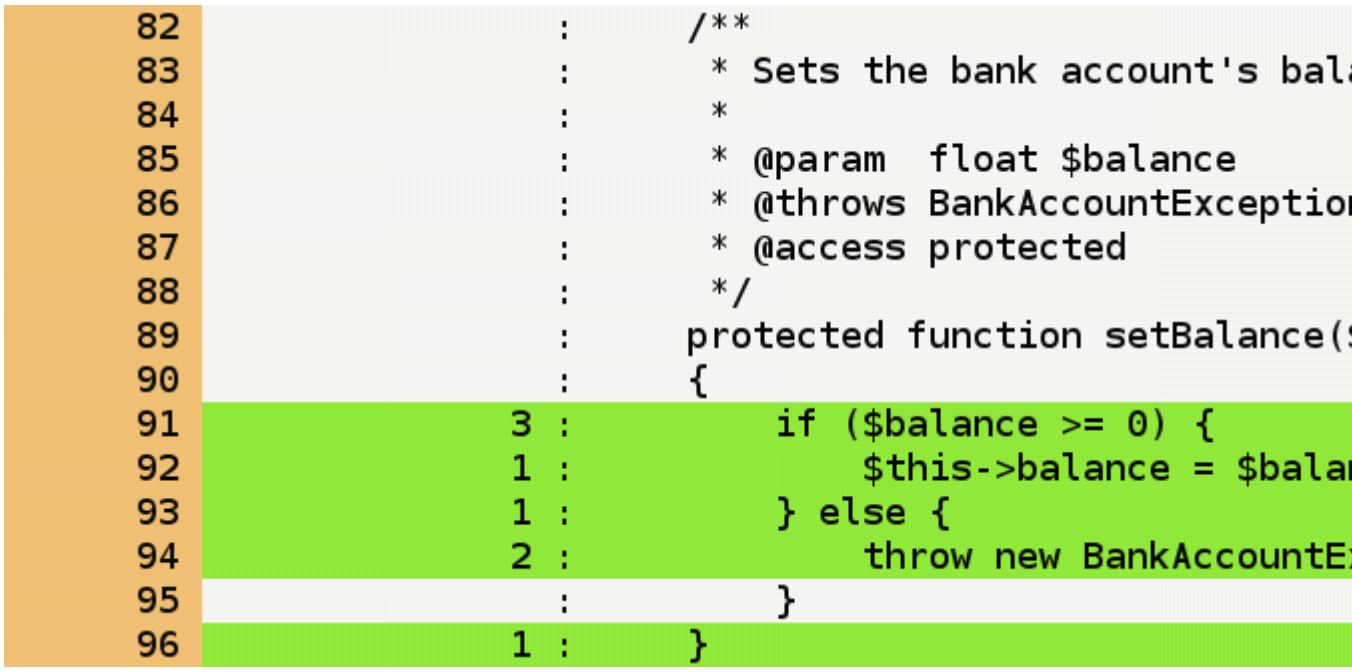
我们的 BankAccount 范例的代码覆盖率报告显示，我们尚无任何以合法值调用 setBalance()、depositMoney() 和 withdrawMoney() 等方法的测试。范例 14.1 显示一个可被加入 BankAccountTest 测试用例类以完全覆盖类 BankAccount 的测试。

范例 14.1: 达到完全代码覆盖率所缺失的测试

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'BankAccount.php';

class BankAccountTest extends PHPUnit_Framework_TestCase
{
    // ...

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

图表 14.3 显示带有补充测试的 setBalance() 方法的代码覆盖率。

图表 **14.3.** 带有补充测试的 `setBalance()`的代码覆盖率



指定覆盖的方法

注解`@covers`可被用于测试代码中以指明测试方法要测试哪些方法。如果规定了，则只考虑指定的方法的代码覆盖率信息。范例 14.2 显示一个例子。

范例 14.2: 一些指明了要覆盖哪些方法的测试

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'BankAccount.php';

class BankAccountTest extends PHPUnit_Framework_TestCase
{
    protected $ba;
```

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

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

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

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

            return;
        }

        $this->fail();
    }

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

```
        }

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

            return;
        }

        $this->fail();
    }

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

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

忽略代码块

有时候你会有些无法测试的代码块，而且你可能想在代码覆盖率分析时忽略它们。PHPUnit 允许你利用

`@codeCoverageIgnoreStart` 和`@codeCoverageIgnoreEnd` 注解达到这个目的，如范例 14.3 中所示。

范例 14.3: 使用`@codeCoverageIgnoreStart` 和`@codeCoverageIgnoreEnd` 注解

```
<?php
class Sample
{
    // ...

    public function doSomething()
    {
        if (0) {
            // @codeCoverageIgnoreStart
            $this->doSomethingElse();
            // @codeCoverageIgnoreEnd
        }
    }

    // ...
}
?>
```

注解`@codeCoverageIgnoreStart` 和`@codeCoverageIgnoreEnd` 之间的代码行被作为已执行（如果它们是可执行的）且不会被高亮显示。

包含和排除文件

缺省情况下，所有至少包含一行已执行的代码的源文件（而且只有这些文件）都会被包括在报表中。你可用 PHPUnit_Util_Filter API（见表 14.1）配置被包含在报表中的源文件。

表 14.1. PHPUnit_Util_Filter API

方法	含义
void addDirectoryToFilter(string $directory)	将某目录中的所有以.php为后缀的文件加入黑名单（递归的）。
void addDirectoryToFilter(string $directory, string $suffix)	将某目录中的所有以$suffix为后缀的文件加入黑名单（递归的）。
void addFileToFilter(string $filename)	将一个文件加入黑名单。
void removeDirectoryFromFilter(string $directory)	从黑名单中删除所有来自某

方法	含义
	目录的以.php为后缀的文件（递归的）。
void removeDirectoryFromFilter(string $directory, string $suffix)	从黑名单中删除所有来自某目录的以$suffix为后缀的文件（递归的）。
void removeFileFromFilter(string $filename)	从黑名单中删除一个文件。
void addDirectoryToWhitelist(string $directory)	将某目录中的所有以.php为后缀的文件加入白名单（递归的）。
void addDirectoryToWhitelist(string $directory, string $suffix)	将某目录中的所有以$suffix为后缀的文件加入白名单（递归的）。

方法	含义
`void addFileToWhitelist(string $filename)`	将一个文件加入白名单。
`void removeDirectoryFromWhitelist(string $directory)`	从白名单中删除所有来自某目录的以`.php`为后缀的文件（递归的）。
`void removeDirectoryFromWhitelist(string $directory, string $suffix)`	从白名单中删除所有来自某目录的以`$suffix`为后缀的文件（递归的）。
`void removeFileFromWhitelist(string $filename)`	从白名单中删除一个文件。

黑名单是用 PHPUnit 自己的和测试的所有源文件预填充的。当白名单为空时（缺省），黑名单被使用。当白名单不空时，白名单被使用。

第 15 章 测试的其他用途

一旦习惯了编写自动化测试，你或许会发现测试的更多用途。这儿有一些例子。

敏捷文档

典型地，在使用了敏捷方法开发，例如极限编程的项目中，文档跟不上频繁改变的项目的设计和代码。极限编程要求*代码的集体所有*，因此所有开发者都需要了解整个系统如何运转。如果你得到了足够的训练从而将“宣告名字”用于你的测试以说明类应该做什么，你可用 PHPUnit 的 TestDox 功能基于项目的测试来为项目生成自动化文档。该文档带给开发者一个关于项目中的每个类应该做什么的概述。

PHPUnit 的 TestDox 功能查看测试类及其所有测试方法名字，把它们从骆驼命名法变换成句子：`testBalanceIsInitiallyZero()`变成“Balance is initially zero”。如果一些测试方法的名字只在后缀一个或多个数字方面不同，例如 `testBalanceCannotBecomeNegative()` 和 `testBalanceCannotBecomeNegative2()`，假如这些测试都成功，句子“Balance cannot become negative”只会出现一次。

我们来看下为 BankAccount 类（来自范例 13.1）产生的敏捷文档：

```
phpunit --testdox BankAccountTest

PHPUnit 3.2.10 by Sebastian Bergmann.


BankAccount

 - Balance is initially zero

 - Balance cannot become negative
```

另外，利用--testdox-html 和--testdox-text 参数，生成的敏捷文档可为 HTML 或无格式文本格式，并且被写入文件。

敏捷文档可被用于证明你做的关于项目中用到的外部软件包的假设。当你用到一个外部的软件包，就要被置于某些风险之中——软件包行为不符预期，而且其未来版本会发生细微的变化，这些变化会破坏你的代码而不让你知道。你可通过每次做出假设时编写测试来处理这些风险。如果你的测试成功了，那么你的假设就是正确的。如果你用测试证明你的所有假设，就无需担心外部软件包的未来发布版本：如果测试成功，你的系统将继续运转。

跨团队测试

当你用测试证明假设，你就拥有了自己的测试。软件包提供者——你的假设就是关于它——对你的测试一无所知。如想同软件包提供者的关系更紧密，你们可用测试进行交流和协作。

当你同软件包提供者就协作达成一致，你们就能一起编写测试。可以这样做，让测试展现出尽可能的假设。隐藏的假设是协作的终结。通过这些测试，你确切地证明了你期望的来自提供的软件包的东西。当所有的测试运行时，提供者就知道软件包完成了。

通过使用存根（见本书前面关于"模拟对象"一章），你能进一步同提供者分离：提供者的职责是让测试同软件包的真实实现一起运行。你的职责是让测试适合运行你自己的代码。在拥有提供的软件包的真实实现之前，你使用存根对象。依据这种方式，两个团队可以独立地开发。

第 16 章 日志

PHPUnit 支持以多种格式记录测试结果。

XML 格式

PHPUnit 支持的 XML 格式基于 JUnit task for Apache Ant 用的那种，不太严格。下面的例子显示了为 `ArrayTest` 中的测试生成的 XML 日志文件：

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

<testsuites>

  <testsuite name="ArrayTest" file="/home/sb/ArrayTest.php"
tests="2"

           failures="0" errors="0"
time="0.0034592151641846">

    <testcase name="testNewArrayIsEmpty" class="ArrayTest"

            file="/home/sb/ArrayTest.php"
time="0.00038909912109375"/>

    <testcase name="testArrayContainsAnElement"
class="ArrayTest"

            file="/home/sb/ArrayTest.php"
time="0.0030701160430908"/>

  </testsuite>

</testsuites>
```

下面的 XML 日志文件是为名为 `FailureErrorTest` 的测试用例类的两个测试 `testFailure` 和 `testError` 生成的，并且显示如何表示失败和错误。

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

<testsuites>

  <testsuite name="FailureErrorTest"
file="/home/sb/FailureErrorTest.php"
```

```
          tests="2" failures="1" errors="1"
time="0.0008389949798584">

   <testcase name="testFailure" class="FailureErrorTest"

           file="/home/sb/FailureErrorTest.php"
time="0.00056290626525879">

     <failure
type="PHPUnit_Framework_ExpectationFailedException">

Failed asserting that two strings are equal.

--- Expected

+++ Actual

@@ -1 +1 @@

-foo

+bar



/home/sb/FailureErrorTest.php:8

</failure>

   </testcase>

   <testcase name="testError" class="FailureErrorTest"

           file="/home/sb/FailureErrorTest.php"
time="0.00027608871459961">

     <error type="Exception">Exception: An error occured.

/home/sb/FailureErrorTest.php:13

</error>

   </testcase>

  </testsuite>

</testsuites>
```

代码覆盖率（XML）

PHPUnit 支持的用于代码覆盖率信息日志的 XML 格式基于 Clover 用的那种，但并不严格。下面的例子显示为 `BankAccountTest` 中的测试生成的XML日志文件：

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

<coverage generated="1184835473" phpunit="3.2.10">

  <project name="BankAccountTest" timestamp="1184835473">

    <file name="/home/sb/BankAccount.php">

      <class name="BankAccountException">

        <metrics methods="0" coveredmethods="0"
statements="0"

                 coveredstatements="0" elements="0"
coveredelements="0"/>

      </class>

      <class name="BankAccount">

        <metrics methods="4" coveredmethods="4"
statements="13"

                 coveredstatements="5" elements="17"
coveredelements="9"/>

      </class>

      <line num="77" type="method" count="3"/>

      <line num="79" type="stmt" count="3"/>

      <line num="89" type="method" count="2"/>

      <line num="91" type="stmt" count="2"/>
```

```
    <line num="92" type="stmt" count="0"/>

    <line num="93" type="stmt" count="0"/>

    <line num="94" type="stmt" count="2"/>

    <line num="96" type="stmt" count="0"/>

    <line num="105" type="method" count="1"/>

    <line num="107" type="stmt" count="1"/>

    <line num="109" type="stmt" count="0"/>

    <line num="119" type="method" count="1"/>

    <line num="121" type="stmt" count="1"/>

    <line num="123" type="stmt" count="0"/>

    <metrics loc="126" ncloc="37" classes="2" methods="4"
coveredmethods="4"

             statements="13" coveredstatements="5"
elements="17"

             coveredelements="9"/>

   </file>

   <metrics files="1" loc="126" ncloc="37" classes="2"
methods="4"

            coveredmethods="4" statements="13"
coveredstatements="5"

            elements="17" coveredelements="9"/>

  </project>

</coverage>
```

JavaScript 对象表示法（JSON）

JavaScript 对象表示法（JSON）是个轻量级的数据互换格式。下面的例子显示为 `ArrayTest` 中的测试生成的 JSON 消息：

```
{"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":""}
```

下面的 JSON 消息是为名为 `FailureErrorTest` 的测试用例类的两个测试 `testFailure` 和 `testError` 生成的，并且显示如何表示失败和错误。

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

{"event":"test","suite":"FailureErrorTest",

 "test":"testFailure(FailureErrorTest)","status":"fail",

 "time":0.000483989716,"trace":[],"message":""}

{"event":"test","suite":"FailureErrorTest",

 "test":"testError(FailureErrorTest)","status":"error",

 "time":0.000466108322,"trace":[],"message":""}
```

Test Anything Protocol（TAP）

Test Anything Protocol（TAP）是 Perl 中基于文本的测试模块间的简单接口。下面的例子显示了为 `ArrayTest` 中的测试生成的 TAP 日志文件：

```
1..2

# TestSuite "ArrayTest" started.

ok 1 - testNewArrayIsEmpty(ArrayTest)

ok 2 - testArrayContainsAnElement(ArrayTest)

# TestSuite "ArrayTest" ended.
```

下面的 TAP 日志文件是为名为 `FailureErrorTest` 的测试用例类的两个测试 `testFailure` 和 `testError` 生成的，并且显示如何表示失败和错误。

```
1..2

# TestSuite "FailureErrorTest" started.

not ok 1 - Failure: testFailure(FailureErrorTest)

not ok 2 - Error: testError(FailureErrorTest)

# TestSuite "FailureErrorTest" ended.
```

GraphViz 标记

利用 GraphViz 工具，PHPUnit 可以生成关于测试结果的描述图示，它可以多种有用的格式，例如图像，呈现为图表。

```
phpunit --log-graphviz BankAccount.dot BankAccountTest
PHPUnit 3.2.10 by Sebastian Bergmann.

...

Time: 0 seconds

OK (3 tests)
```

下面的例子显示为 `BankAccountTest` 中的测试生成的 GraphViz 标记（并且存为当前目录中的 `BankAccount.dot` 文件）：

```
digraph G {
graph
[ overlap="scale",splines="true",sep=".1",fontsize="8" ];
"BankAccountTest" [ color="green" ];
subgraph "cluster_BankAccountTest" {
label="";
"testBalanceIsInitiallyZero" [ color="green" ];
"testBalanceCannotBecomeNegative" [ color="green" ];
"testBalanceCannotBecomeNegative2" [ color="green" ];
}
"BankAccountTest" -> "testBalanceIsInitiallyZero";
"BankAccountTest" -> "testBalanceCannotBecomeNegative";
"BankAccountTest" -> "testBalanceCannotBecomeNegative2";
```

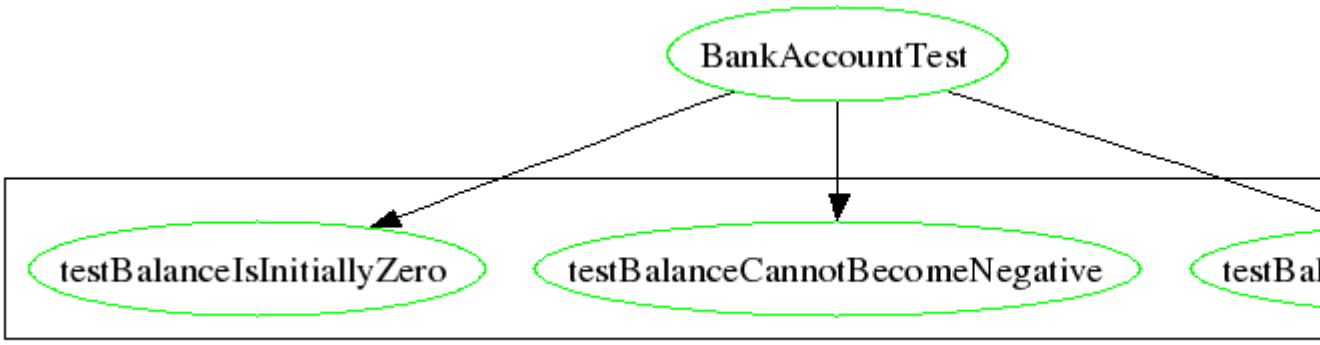
```
}
```

现在我们可用 GraphViz 套件中的命令行工具 `dot` 从该标记生成图形：

```
dot -T png -o BankAccount.png BankAccount.dot
```

图 16.1 显示从上面的 GraphViz 标记生成的测试结果的图示。

图 16.1. 测试结果的图示



成功的测试用绿色边框显示，失败和错误是红色边框，未完成和跳过的是黄色边框。如果有未成功的子节点（如测试），那么父节点（如测试套件）被显示为非绿色边框。

测试数据库

PHPUnit 可以把测试结果和代码覆盖率数据写入测试数据库。有几个关于未来特性的想法依赖该数据。

1. 测试套件的每次*运行*对应 *run* 表中的一行。
2. 构成一次测试运行的每个*测试*对应 *test* 表中的一行。
3. 构成一个修订版的每个*文件*对应 *code_file* 表中的一行。
4. 构成一个修订版的文件中声明的每个*类*对应 *code_class* 表中的一行。
5. 构成一个修订版的文件中声明的每个*方法*对应 *code_method* 表中的一行。
6. 属于构成修订版的一个文件的每个*代码行*对应 *code_line* 表中的一行。
7. *code_coverage* 表将每个测试连接到它覆盖的代码行。

在可以将测试结果和代码覆盖率数据写入数据库前，我们需要利用一个提供的模式定义创建数据库：

```
sqlite3 BankAccount.db <
PHPUnit/Util/Log/Database/SQLite3.sql
```

现在我们可以执行测试套件并将测试结果和代码覆盖率数据写入数据库：

```
phpunit --test-db-dsn sqlite:///home/sb/BankAccount.db
--test-db-log-rev 1 BankAccountTest

PHPUnit 3.2.10 by Sebastian Bergmann.



...



Time: 0 seconds




OK (3 tests)



Storing code coverage data in database, this may take a moment.
```

表 16.1 显示 TextUI 测试启动器（见第 5 章）理解的用于测试数据库的参数。

表 16.1. 用于测试数据库的 TextUI 参数

参数	含义
`--test-db-dsn <dsn>`	用于数据库连接的 PDO 数据源名（DSN）。DSN 通常由 PDO 驱动名后跟冒号和 PDO 驱动规范连接语法组成。
`--test-db-log-rev <r>`	一个唯一标识代码库的当前修订版的数字，例如 Subversion 全

参数	含义
	局修订号。
`--test-db-log-info ...`	例如关于测试环境的附加信息。

第 17 章 软件度量（暂无内容）

项目混乱检测（暂无内容）

第 18 章 框架（Skeleton）生成器

当你为现存代码编写测试时，你不得再三地编写类似

```
public function testMethod()
{
}
```

一样的相同的代码片断。PHPUnit 能通过分析现存的类代码并为它生成一个测试用例类的框架来提供帮助。

范例 18.1: 计算器类

```
<?php
class Calculator
{
    public function add($a, $b)
    {
        return $a + $b;
    }
```

```
}
?>
```

下面的例子显示如何为名为 Calculator（见范例 18.1）的类生成测试类框架。

```
phpunit --skeleton Calculator

PHPUnit 3.2.10 by Sebastian Bergmann.



Wrote test class skeleton for Calculator to
CalculatorTest.php.
```

对于原始类中的每个方法，在生成的测试用例类中会对应地存在一个未实现的测试用例（见第 10 章）。

下面是运行生成的测试用例类的输出。

```
phpunit --verbose CalculatorTest

PHPUnit 3.2.10 by Sebastian Bergmann.



CalculatorTest

I




Time: 0 seconds
```

```
There was 1 incomplete test:


1) testAdd(CalculatorTest)

This test has not been implemented yet.

/home/sb/CalculatorTest.php:54


OK, but incomplete or skipped tests!

Tests: 1, Incomplete: 1.
```

注解

你可在方法的文档块中使用`@assert`注解自动地生成简单却有意义的测试来代替未完成的测试用例。范例 18.2 显示一个例子。

范例 18.2: 带@assert 注解的计算器类

```
<?php
class Calculator
{
    /**
     * @assert (0, 0) == 0
     * @assert (0, 1) == 1
     * @assert (1, 0) == 1
     * @assert (1, 1) == 2
     */
    public function add($a, $b)
    {
        return $a + $b;
    }
}
?>
```

原始类中的每个方法都进行`@assert`注解的检测。这些被转变为测试代码，像这样

```
    /**
     * Generated from @assert (0, 0) == 0.
     */
    public function testAdd() {
        $o = new Calculator;
        $this->assertEquals(0, $o->add(0, 0));
    }
```

下面是运行生成的测试用例类的输出。

```
phpunit CalculatorTest

PHPUnit 3.2.10 by Sebastian Bergmann.


....


Time: 0 seconds



OK (4 tests)
```

表 18.1 显示受支持的`@assert`注解的各种变体以及它们被如何转变为测试代码。

表 18.1 受支持的@assert 注解的各种变体

注解	转变为
`@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`

第 19 章 PHPUnit 和 Selenium

Selenium RC

Selenium RC 是一款测试工具，允许你为 web 应用编写自动化的用户接口测试，支持任何程序设计语言、任何 HTTP 站点以及任何主流浏览器。它使用 Selenium Core，这是个利用 JavaScript 执行自动浏览器任务的库。Selenium 测试直接运行于浏览器

中，就像真实做的那样。这些测试既可用于*验收测试*（通过在集成系统上执行更高级的测试而不是只独立地测试系统的各个单元），又可用于*浏览器兼容性测试*（通过测试不同操作系统和浏览器上的 web 应用）。

我看看下 Selenium RC 是如何安装的：

1. 下载一个 Selenium RC 分发包存档。
2. 分发包存档并拷贝 *server/selenium-server.jar* 至 */usr/local/bin*（例如）。
3. 通过运行 **java -jar /usr/local/bin/selenium-server.jar** 启动 Selenium RC 服务器。

现在我们可以利用它的客户/服务器协议向 Selenium RC 服务器发送命令了。

PHPUnit_Extensions_SeleniumTestCase

`PHPUnit_Extensions_SeleniumTestCase` 测试用例扩展将同 Selenium RC 通话的客户/服务器协议实现为专门用于 web 测试的断言方法。

范例 19.1 显示如何测试 http://www.example.com/站点的<title>元素的内容。

范例 19.1: PHPUnit_Extensions_SeleniumTestCase 使用举例

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

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

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

不同于 PHPUnit_Framework_TestCase 类，扩展了 PHPUnit_Extensions_SeleniumTestCase 的测试用例类必须提供 setUp() 方法。该方法用于配置 Selenium RC 会话。可用于此的方法列表见表 19.1。

表 **19.1. Selenium RC API**：装配

方法	含义
`void setBrowser(string $browser)`	Selenium RC 服务器使用的浏览器。
`void setBrowserUrl(string $browserUrl)`	设置用于测试的基址 URL。
`void setHost(string $host)`	设置连接到 Selenium RC 服务器的主机名。
`void setPort(int $port)`	设置连接到 Selenium RC 服务器的端口号。
`void setTimeout(int $timeout)`	设置连接到 Selenium RC 服务器的超时时间。
`void setSleep(int $seconds)`	设置 Selenium RC 客户端向 Selenium RC 服务器发送动作指令之间睡眠的秒数。

你也可以使用一组浏览器运行每一个测试：在你的测试用例类中声明一个名为`$browsers`的`public static`数组代替使用`setBrowser()`设置一个浏览器。该数组中的每

一项都描述一个浏览器配置。这些浏览器中的每一个都能被不同的 Selenium RC 服务器接待：

范例 19.2: 设置多浏览器配置

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

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

```
        'port'    => 4444,
        'timeout' => 30000,
      )
    );

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

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

`PHPUnit_Extensions_SeleniumTestCase` 能够收集测试经由 Selenium 运行时的代码覆盖率信息：

1. 拷贝 *`PHPUnit/Extensions/SeleniumTestCase/phpunit_coverage.php`* 到你的 web 服务器的文件根目录中。
2. 在你的 web 服务器的 *`php.ini`* 配置文件中，配置 *`PHPUnit/Extensions/SeleniumTestCase/prepend.php`* 和 *`PHPUnit/Extensions/SeleniumTestCase/append.php`* 分别作为 `auto_prepend_file` 和 `auto_append_file`。
3. 在你的扩展了 `PHPUnit_Extensions_SeleniumTestCase` 的测试用例类中，使用

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

配置用于 *phpunit_coverage.php* 脚本的 URL。

表 19.2 列出 PHPUnit_Extensions_SeleniumTestCase 提供的各种断言方法。

表 19.2. 断言

断言	含义
`void assertAlertPresent()`	如果不存在警告（对话框）则报错。
`void assertNoAlertPresent()`	如果存在警告（对话框）则报错。
`void assertChecked(string $locator)`	如果`$locator`标识的元素未选中则报错。
`void assertNotChecked(string $locator)`	如果`$locator`标识的元素被选中则报错。
`void assertConfirmationPresent()`	如果不存在确认（对话框）则报

断言	含义
	错。
`void assertNoConfirmationPresent()`	如果存在确认（对话框）则报错。
`void assertEditable(string $locator)`	如果`$locator`标识的元素不可编辑则报错。
`void assertNotEditable(string $locator)`	如果`$locator`标识的元素可编辑则报错。
`void assertElementValueEquals(string $locator, string $text)`	如果`$locator`标识的元素的值不等于给定的`$text`则报错。
`void assertElementValueNotEquals(string $locator, string $text)`	如果`$locator`标识的元素的值等于给定的`$text`则报错。
`void assertElementContainsText(string $locator, string $text)`	如果`$locator`标识的元素不含给定的`$text`则报错。

断言	含义
`void assertElementNotContainsText(string $locator, string $text)`	如果`$locator`标识的元素包含给定的`$text`则报错。
`void assertElementPresent(string $locator)`	如果`$locator`标识的元素不存在则报错。
`void assertElementNotPresent(string $locator)`	如果`$locator`标识的元素存在则报错。
`void assertLocationEquals(string $location)`	如果当前位置（location）不等于给定的`$location`则报错。
`void assertLocationNotEquals(string $location)`	如果当前位置（location）等于给定的`$location`则报错。
`void assertPromptPresent()`	如果不存在提示（对话框）则报错。

断言	含义
`void assertNoPromptPresent()`	如果存在提示（对话框）则报错。
`void assertSelectHasOption(string $selectLocator, string $option)`	如果给定的选项不可用则报错。
`void assertSelectNotHasOption(string $selectLocator, string $option)`	如果给定的选项可用则报错。
`void assertSelected($selectLocator, $option)`	如果给定的标签未选中则报错。
`void assertNotSelected($selectLocator, $option)`	如果给定的标签被选中则报错。
`void assertIsSelected(string $selectLocator, string $value)`	如果给定的值未选中则报错。
`void assertIsNotSelected(string $selectLocator, string $value)`	如果给定的值被选中则报错。
`void assertSomethingSelected(string $selectLocator)`	如果`$selectLocator`标识的选项未选中则报错。
`void assertNothingSelected(string $selectLocator)`	如果`$selectLocator`标识的选项被选

断言	含义
	中则报错。
void assertTextPresent(string $pattern)	如果给定的 $pattern 不存在则报错。
void assertTextNotPresent(string $pattern)	如果给定的 $pattern 存在则报错。
void assertTitleEquals(string $title)	如果当前标题不等于给定的$title则报错。
void assertTitleNotEquals(string $title)	如果当前标题等于给定的$title 则报错。
void assertVisible(string $locator)	如果$locator 标识的元素不可见则报错。
void assertNotVisible(string $locator)	如果$locator 标识的元素可见则报错。

表 19.3 显示 `PHPUnit_Extensions_SeleniumTestCase` 的两个模板方法：

表 19.3. 模板方法

方法	含义
`void defaultAssertions()`	重写以执行一个测试用例中的所有测试共享的断言。该方法在发送给 Selenium RC 服务器的每个命令之后被调用。
`void sharedAssertions()`	重写以执行一个测试用例中的所有测试共享的断言。该方法在一个测试执行结束之前被调用。

请查阅 Selenium Core 的文档获得关于可用的命令以及如何使用的参考。

利用 `runSelenese($filename)` 方法，你也能从它的 Selenese/HTML 规范运行一个 Selenium 测试。此外，利用静态属性 `$seleneseDirectory`，你能从一个包含 Selenese/HTML 文件的目录自动地创建测试对象。指定的目录被递归地搜索预期包含 Selenese/HTML 的 `.htm` 文件。 范例 19.3 显示一个例子。

范例 19.3: 使用 Selenese/HTML 文件的目录作为测试

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

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

第 20 章 持续集成

> 持续集成是一种软件开发实践，团队成员频繁地合并他们的工作，通常每个人至少每天合并，导致每天多次合并。每次合并由一个自动化的构建系统（包括测试）校验，以尽快发觉合并错误。很多团队发现这种方式显著地减少了合并问题，并且允许团队更快地开发内聚的软件。
>
> --Martin Fowler

本章提供持续集成的概述，总结技术及其在 PHPUnit 下的应用。

CruiseControl

持续集成要求一个完全自动化和可再生的构建系统，包括测试且每天运行很多次。这允许每个开发者每天合并从而减少合并问题。要达此目的，可通过设置一个 cronjob 每隔一段时间重新校验项目的源代码仓库、运行测试并发布结果，仍然可能需要一个更舒适的解决方案。

这就是像 CruiseControl 这样的用于持续构建过程的框架出现的原因。它包括但并不限于用于 email 通知的插件、Apache Ant 和多种源代码控制工具。提供了 web 界面用于察看当前和以前构建（版本）的详细信息。

下面的例子假设 CruiseControl 已经安装于 */usr/local/cruisecontrol*。

1. **cd /usr/local/cruisecontrol**
2. **mkdir -p projects/BankAccount/build/logs**
3. **cd projects/BankAccount**
4. **svn co svn://svn.phpunit.de/phpunit/phpunit/branches/release/3.2/PHPUnit/Samples/BankAccount source**
5. 编辑文件 *build.xml*。

范例 20.1: projects/BankAccount/build.xml

```
<project name="BankAccount" default="build" basedir=".">
 <target name="checkout">
  <exec dir="${basedir}/source/" executable="svn">
   <arg line="up"/>
  </exec>
 </target>

 <target name="test">
  <exec dir="${basedir}/source" executable="phpunit"
failonerror="true">
   <arg line="--log-xml
${basedir}/build/logs/bankaccount.xml BankAccountTest"/>
  </exec>
 </target>

 <target name="build" depends="checkout,test"/>
</project>
```

6.**cd /usr/local/cruisecontrol**

7.编辑文件 *config.xml*。

范例 20.2: config.xml

```
<cruisecontrol>

  <project name="BankAccount" buildafterfailed="false">

    <plugin

    name="svnbootstrapper"


classname="net.sourceforge.cruisecontrol.bootstrappers.SV
NBootstrapper"/>

    <plugin

    name="svn"


classname="net.sourceforge.cruisecontrol.sourcecontrols.S
VN"/>



    <listeners>

      <currentbuildstatuslistener
file="logs/${project.name}/status.txt"/>

    </listeners>



    <bootstrappers>

      <svnbootstrapper
localWorkingCopy="projects/${project.name}/source/"/>

    </bootstrappers>



    <modificationset>
```

```
    <svn
localWorkingCopy="projects/${project.name}/source/"/>
  </modificationset>

  <schedule interval="300">
    <ant
    anthome="apache-ant-1.7.0"
    buildfile="projects/${project.name}/build.xml"/>
  </schedule>

  <log dir="logs/${project.name}">
    <merge dir="projects/${project.name}/build/logs/"/>
  </log>

  <publishers>
    <currentbuildstatuspublisher
    file="logs/${project.name}/buildstatus.txt"/>

    <email
    mailhost="localhost"

buildresultsurl="http://cruise.example.com/buildresults/$
{project.name}"
    skipusers="true"
    spamwhilebroken="true"
```

```
      returnaddress="project@example.com">

        <failure address="dev@lists.example.com"
reportWhenFixed="true"/>

      </email>

    </publishers>

  </project>

</cruisecontrol>
```

8. **./cruisecontrol.sh**

9. 在你的浏览器中打开 *http://localhost:8080/*。

Apache Maven

Apache Maven 是一个软件项目管理和综合的工具。基于概念项目对象模型（POM），Apache Maven 能够从信息的中枢位置管理项目的构建、报告和文档。

PHPUnit 的 XML 日志设备（见"XML 格式"一节）产生的单一 XML 日志文件在能被 Apache Maven 的 surefire 插件处理以前，需要被分割成单独的 XML 日志文件，分别对应每个测试套件。该插件用于在构建周期的测试阶段执行应用的单元测试。范例 20.4 显示一个实现分割的 XSLT 样式表。范例 20.3 显示一个 *pom.xml* 配置文件的例子。

范例 20.3: pom.xml

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


<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <!-- ... -->
  <prerequisites>
    <maven>2.0.7</maven>
  </prerequisites>
  <!-- ... -->
  <build>
  <!-- ... -->
    <plugins>
      <plugin>
      <dependencies>
        <dependency>
          <groupId>org.apache.ant</groupId>
          <artifactId>ant-trax</artifactId>
          <version>1.7.0</version>
        </dependency>
```

```
      <dependency>
        <groupId>net.sf.saxon</groupId>
        <artifactId>saxon</artifactId>
        <version>8.7</version>
      </dependency>
    </dependencies>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-antrun-plugin</artifactId>
    <version>1.2-SNAPSHOT</version>
    <executions>
      <execution>
        <id>codecoverage</id>
        <phase>pre-site</phase>
        <goals>
          <goal>run</goal>
        </goals>
        <configuration>
          <tasks>
            <property name="phpunit.codecoverage"

location="${project.reporting.outputDirectory}/phpunit/co
decoverage" />
            <property name="surefire.reports"

location="${project.build.directory}/surefire-reports" />
```

```
            <mkdir dir="${phpunit.codecoverage}"/>

            <mkdir dir="${surefire.reports}"/>



            <!-- ${ant.phpunit} path to PHPUnit executable
-->

            <!-- ${ant.pear};... this is the include path for
your test execution -->

            <!-- ${test.AllTests} PHPUnit cmd line args like
'AllTests de/dmc/dashboard/AllTests.php' -->

            <exec executable="${ant.phpunit}"
dir="${basedir}">

              <arg line="-d
include_path=${ant.pear};${project.build.sourceDirectory};
${project.build.testSourceDirectory}

                        --report ${phpunit.codecoverage}
${test.AllTests}" />

            </exec>



            <xslt in="${phpunit.codecoverage}/logfile.xml"

                 out="${surefire.reports}/xslt.info"


style="src/test/config/phpunit_to_surefire.xslt"

                 processor="trax">

              <!-- this is the output folder for surefire like
XML Reports -->

              <param name="outputDir"
expression="${surefire.reports}"/>
```

```
            </xslt>
          </tasks>
        </configuration>
      </execution>
    </executions>
    </plugin>
  </plugins>
 </build>
 <reporting>
   <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-report-plugin</artifactId>
      <version>2.4-SNAPSHOT</version>
      <reportSets>
        <reportSet>
          <reports>
            <report>report-only</report>
          </reports>
        </reportSet>
      </reportSets>
    </plugin>
   </plugins>
```

```
  </reporting>
</project>
```

范例 20.4: phpunit_to_surefire.xsl

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

<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
  <xsl:param name="outputDir">.</xsl:param>

  <xsl:template match="testsuites">
    <xsl:apply-templates select="testsuite"/>
  </xsl:template>

  <xsl:template match="testsuite">
    <xsl:if test="testcase">
      <xsl:variable name="outputName" select="./@name"/>
      <xsl:result-document
href="file:///{$outputDir}/{$outputName}.xml"
method="xml">
```

```
      <xsl:copy-of select="."/>
    </xsl:result-document>
  </xsl:if>

  <xsl:apply-templates select="testsuite"/>
 </xsl:template>
</xsl:stylesheet>
```

第 21 章 PHPUnit 的执行

PHPUnit 的执行有点不常见，利用了在一般应用代码中难以维护的技术。理解 PHPUnit 如何运行你的测试有助于编写它们。

单个测试被表现为一个 `PHPUnit_Framework_Test` 对象，而且需要一个 `PHPUnit_Framework_TestResult` 对象来运行。`PHPUnit_Framework_TestResult` 对象被传入 `PHPUnit_Framework_Test` 对象的 `run()` 方法中，它运行实际的测试方法并向 `PHPUnit_Framework_TestResult` 对象报告所有的异常。这是个来自 Smalltalk 世界的特性，称为*聚集参数（Collecting Parameter）*。它建议当你需要跨越多个方法收集结果时（在我们的案例中是若干对用于各种测试的 `run()` 方法的调用结果），你应该

像方法加入一个参数并传递一个将为你收集结果的对象。查阅 Erich Gamma 和 Kent Beck 的文章“JUnit: A Cook's Tour” [GammaBeck1999]和 Kent Beck 的“Smalltalk Best Practice Patterns” [Beck1997]。

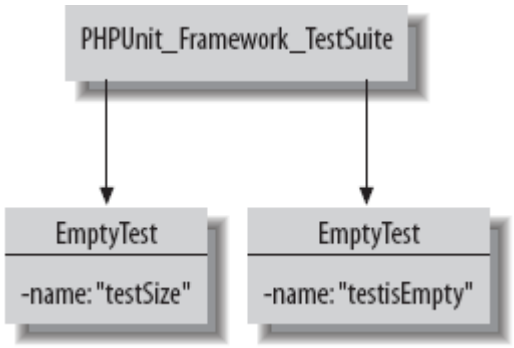
要进一步理解 PHPUnit 如何运行你的测试，考虑范例 21.1 中的测试用例类。

范例 21.1: EmptyTest 类

```
<?php
require_once 'PHPUnit/Framework.php';

class EmptyTest extends PHPUnit_Framework_TestCase
{
    private $emptyArray = array();

    public function testSize()
    {
        $this->assertEquals(0, sizeof($this->emptyArray));

    }

    public function testIsEmpty()
    {
        $this->assertTrue(empty($this->emptyArray));
    }
}
?>
```

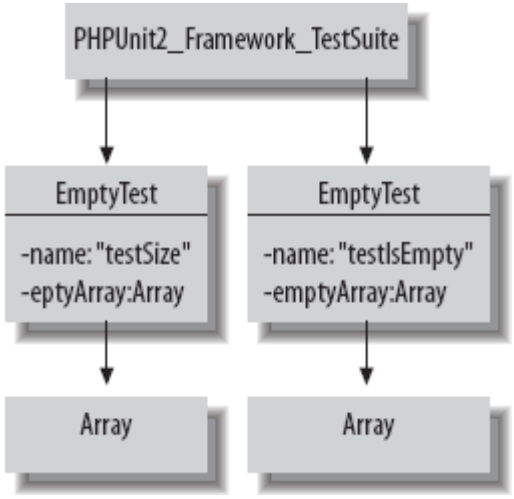
当测试运行时，PHPUnit 做的第一件事情是把测试类转变为一个 `PHPUnit_Framework_Test` 对象——此时，`PHPUnit_Framework_TestSuite` 包含两个 `EmptyTest` 实例，如图 21.1 中所示。

图 21.1. 即将被运行的测试



当 `PHPUnit_Framework_TestSuite` 运行时，它依次运行每个 `EmptyTest`。每次都运行它独有的 `setUp()` 方法，为每个测试创建一个新的 `$emptyArray`，如图 21.2 中所示。使用这种方式，如果一个测试修改数组，其他测试不受影响。即使变为全局和超全局（如 `$_ENV`）变量也不影响其他测试。

图 21.2. 运行后的测试，各自带有自己特有的 fixture



简而言之，测试运行时一个测试用例类导致一个两级的对象树。每个测试方法作用于其专有的通过 `setUp()` 创建的对象拷贝。结果是测试能够完全独立地运行。

要运行测试方法自身，PHPUnit 使用反射在实例变量 `$name` 中查找方法名并调用它。这是另一个特性，称为*插件式选择器（Pluggable Selector）*，通常用于 Smalltalk 世界。利用插件式选择器使得编写测试更简单，但也要付出代价：你不能根据代码来判定某方式是否被调用了，必须在运行时察看数据值。

第 22 章 PHPUnit API

对于多数用途，PHPUnit 具有一套简单的 API：为你的测试用例子类化 `PHPUnit_Framework_TestCase` 并调用 `assertTrue()`或 `assertEquals()`。可是，对于你们中那些

想要更深入了解 PHPUnit 的人，这儿有所有公开的方法和类。

概述

使用 PHPUnit 的多数时候，你会遇到 5 个类或接口：

`PHPUnit_Framework_Assert`

一个针对期望值来检查实际值的静态方法的集合。

`PHPUnit_Framework_Test`

所有测试对象的接口。

`PHPUnit_Framework_TestCase`

单个测试。

`PHPUnit_Framework_TestSuite`

测试集合。

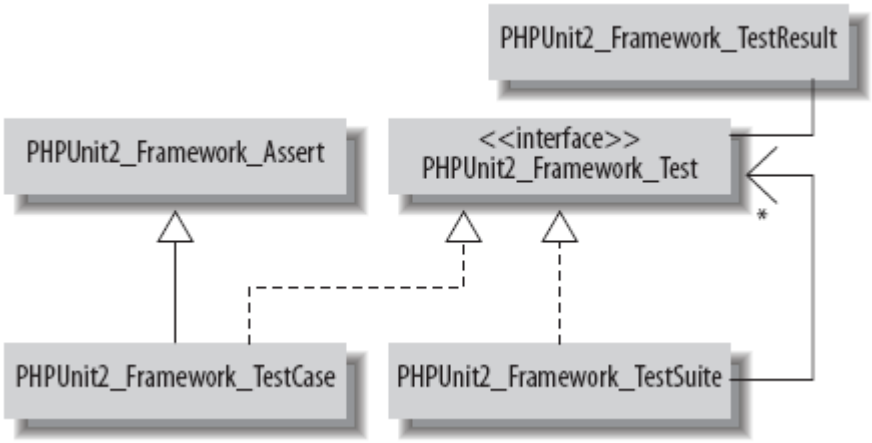
`PHPUnit_Framework_TestResult`

运行一个或更多测试的结果的摘要。

图 22.1 显示 PHPUnit 中的五个基础类和接口之间的关系：`PHPUnit_Framework_Assert`、`PHPUnit_Framework_Test`、

PHPUnit_Framework_TestCase、 PHPUnit_Framework_TestSuite 和 PHPUnit_Framework_TestResult。

图 22.1. PHPUnit 中的五个基础类和接口



PHPUnit_Framework_Assert

为 PHPUnit 编写的多数测试用例间接地衍生自 PHPUnit_Framework_Assert 类，它含有自动地检查值并报告差异的方法。这些方法被声明为 static，因此你可以在你的方法中编写契约式设计风格的断言并使它们经由 PHPUnit 报告（范例 22.1）。

范例 22.1: 契约式设计风格的断言

```
<?php
require_once 'PHPUnit/Framework.php';

class Sample
{
```

```
    public function aSampleMethod($object)
    {
        PHPUnit_Framework_Assert::assertNotNull($object);
    }
}

$sample = new Sample;
$sample->aSampleMethod(NULL);
?>
```

```
Fatal error: Uncaught exception
'PHPUnit_Framework_ExpectationFailedException'

with message 'Failed asserting that <null> is not identical
to <null>.'
```

然而，多数时候，你将在测试内部检查断言。

每个断言方法都有两个变体：其一接受一个将同错误一起显示的消息参数，另一个不是。典型地，当显示失败时，可选的消息也被显示，这使调试更加容易。

范例 22.2: 带消息使用断言

```
<?php
require_once 'PHPUnit/Framework.php';

class MessageTest extends PHPUnit_Framework_TestCase
{
    public function testMessage()
    {
        $this->assertTrue(FALSE, 'This is a custom messag
e.');
    }
```

```
}
?>
```

下面的例子显示当你运行来自范例 22.2 的带消息使用断言的测试 `testMessage()` 时的输出：

```
phpunit MessageTest

PHPUnit 3.2.10 by Sebastian Bergmann.


F


Time: 0 seconds


There was 1 failure:


1) testMessage(MessageTest)

This is a custom message.

Failed asserting that <boolean:false> is true.

/home/sb/MessageTest.php:8


FAILURES!

Tests: 1, Failures: 1.
```

表 22.1 显示断言的所有变体。

表 22.1. 断言

断言	含义
`void assertArrayHasKey(mixed $key, array $array)`	如果`$array`中没有`$key`则报错。
`void assertArrayHasKey(mixed $key, array $array, string $message)`	如果`$array`中没有`$key`则报`$message`标识的错误。
`void assertClassHasAttribute(string $attributeName, string $className)`	如果`$className::attributeName`不存在则报错。
`void assertClassHasAttribute(string $attributeName, string $className, string $message)`	如果`$className::attributeName`不存在则报`$message`标识的错误。
`void assertClassHasStaticAttribute(string $attributeName, string $className)`	如果`$className::attributeName`不存在或非`static`则报错。
`void assertClassHasStaticAttribute(string $attributeName, string $className, string $message)`	如果`$className::attributeName`不存在或非`static`则报`$message`标识的错误。

断言	含义
`void assertContains(mixed $needle, array $haystack)`	如果`$needle`不是`$haystack`的元素则报错。
`void assertContains(mixed $needle, array $haystack, string $message)`	如果`$needle`不是`$haystack`的元素则报`$message`标识的错误。
`void assertContains(mixed $needle, Iterator $haystack)`	如果`$needle`不是`$haystack`的元素则报错。
`void assertContains(mixed $needle, Iterator $haystack, string $message)`	如果`$needle`不是`$haystack`的元素则报`$message`标识的错误。
`void assertContains(string $needle, string $haystack)`	如果`$needle`不是`$haystack`的子字符串则报错。
`void assertContains(string $needle, string $haystack, string $message)`	如果`$needle`不是`$haystack`的子字符串则报`$message`标识的错误。
`assertContainsOnly(string $type, array $haystack)`	如果`$haystack`并非只含有`$type`类型的变量则报错。
`assertContainsOnly(string $type, array $haystack, NULL, string $message)`	如果`$haystack`并非只含有`$type`类型的变量则报`$message`标识的错误。

断言	含义
`assertContainsOnly(string $type, array $haystack, bool $isNativeType)`	如果`$haystack`并非只含有`$type`类型的变量则报错。`$isNativeType`是指示`$type`是否 PHP 原生类型的标志。
`assertContainsOnly(string $type, array $haystack, bool $isNativeType, string $message)`	如果`$haystack`并非只含有`$type`类型的变量则报`$message`标识的错误。`$isNativeType`是指示`$type`是否 PHP 原生类型的标志。
`assertContainsOnly(string $type, Iterator $haystack)`	如果`$haystack`并非只含有`$type`类型的变量则报错。
`assertContainsOnly(string $type, Iterator $haystack, NULL, string $message)`	如果`$haystack`并非只含有`$type`类型的变量则报`$message`标识的错误。
`assertContainsOnly(string $type, Iterator $haystack, bool $isNativeType)`	如果`$haystack`并非只含有`$type`类型的变量则报错。`$isNativeType`是指示`$type`是否 PHP 原生类型的标志。
`assertContainsOnly(string $type, Iterator $haystack, bool $isNativeType, string`	如果`$haystack`并非只含有`$type`类型的变量则报

断言	含义
$message)	$message 标识的错误。$isNativeType 是指示$type 是否 PHP 原生类型的标志。
void assertEquals(array $expected, array $actual)	如果数组$expected 和$actual 不等则报错。
void assertEquals(array $expected, array $actual, string $message)	如果数组$expected 和$actual 不等则报$message 标识的错误。
void assertEquals(float $expected, float $actual, '', float $delta = 0)	如果浮点数$expected 和 $actual 之差不在$delta 中则报错。
void assertEquals(float $expected, float $actual, string $message, float $delta = 0)	如果浮点数$expected 和 $actual 之差不在$delta 中则报$message 标识的错误。
void assertEquals(object $expected, object $actual)	如果对象$expected 和$actual 属性值不等则报错。
void assertEquals(object $expected, object $actual, string $message)	如果对象$expected 和$actual 属性值不等则报$message 标识的错误。
void assertEquals(string	如果字符串$expected 和

断言	含义
$expected, string $actual)	$actual 不等则报错。同差异一样，错误也被报告。
void assertEquals(string $expected, string $actual, string $message)	如果字符串$expected 和 $actual 不等则报$message 标识的错误。同差异一样，错误也被报告。
void assertEquals(DOMDocument $expected, DOMDocument $actual)	如果 DOMDocument 对象$expected 和$actual 描绘的 XML 文档不等则报错。
void assertEquals(DOMDocument $expected, DOMDocument $actual, string $message)	如果 DOMDocument 对象$expected 和$actual 描绘的 XML 文档不等则报$message 标识的错误。
void assertEquals(mixed $expected, mixed $actual)	如果变量$expected 和$actual 不等则报错。
void assertEquals(mixed $expected, mixed $actual, string $message)	如果变量$expected 和$actual 不等则报$message 标识的错误。
void assertFalse(bool $condition)	如果$condition 为 TRUE 则报错。

断言	含义
void assertFalse(bool $condition, string $message)	如果$condition 为 TRUE 则报 $message 标识的错误。
void assertFileEquals(string $expected, string $actual)	如果$expected指定的文件的内容与$actual 指定的文件不同则报错。
void assertFileEquals(string $expected, string $actual, string $message)	如果$expected指定的文件的内容与$actual 指定的文件不同则报$message 标识的错误。
void assertFileExists(string $filename)	如果$filename指定的文件不存在则报错。
void assertFileExists(string $filename, string $message)	如果$filename指定的文件不存在则报$message 标识的错误。
void assertGreaterThan(mixed $expected, mixed $actual)	如果值$actual 不比值 $expected 大则报错。
void assertGreaterThan(mixed $expected, mixed $actual, string $message)	如果值$actual 不比值 $expected 大则报$message 标识的错误。
void	如果值$actual 不比值

断言	含义
assertGreaterThanOrEqual(mixed $expected, mixed $actual)	$expected 大也不相等则报错。
void assertGreaterThanOrEqual(mixed $expected, mixed $actual, string $message)	如果值$actual 不比值 $expected 大也不相等则报 $message 标识的错误。
void assertLessThan(mixed $expected, mixed $actual)	如果值$actual 不比值 $expected 小则报错。
void assertLessThan(mixed $expected, mixed $actual, string $message)	如果值$actual 不比值 $expected 小则报$message 标识的错误。
void assertLessThanOrEqual(mixed $expected, mixed $actual)	如果值$actual 不比值 $expected 小也不相等则报错。
void assertLessThanOrEqual(mixed $expected, mixed $actual, string $message)	如果值$actual 不比值 $expected 小也不相等则报 $message 标识的错误。
void assertNull(mixed $variable)	如果$variable 不是 NULL 则报错。
void assertNull(mixed $variable, string $message)	如果$variable 不是 NULL 则报 $message 标识的错误。

断言	含义
`void assertObjectHasAttribute(string $attributeName, object $object)`	如果`$object->attributeName`不存在则报错。
`void assertObjectHasAttribute(string $attributeName, object $object, string $message)`	如果`$object->attributeName`不存在则报`$message`标识的错误。
`void assertRegExp(string $pattern, string $string)`	如果`$string`不匹配正则表达式`$pattern`则报错。
`void assertRegExp(string $pattern, string $string, string $message)`	如果`$string`不匹配正则表达式`$pattern`则报`$message`标识的错误。
`void assertSame(object $expected, object $actual)`	如果变量`$expected`和`$actual`引用不同的对象则报错。
`void assertSame(object $expected, object $actual, string $message)`	如果变量`$expected`和`$actual`引用不同的对象则报`$message`标识的错误。
`void assertSame(mixed $expected, mixed $actual)`	如果变量`$expected`和`$actual`的类型或值不同则报错。
`void assertSame(mixed $expected, mixed $actual,`	如果变量`$expected`和`$actual`的类型或值不同则报

断言	含义
string $message)	$message 标识的错误。
void assertTrue(bool $condition)	如果$condition 为 FALSE 则报错。
void assertTrue(bool $condition, string $message)	如果$condition 为 FALSE 则报 $message 标识的错误。
void assertType(string $expected, mixed $actual)	如果变量$actual 不是 $expected 类型则报错。
void assertType(string $expected, mixed $actual, string $message)	如果变量$actual 不是 $expected 类型则报$message 标识的错误。
void assertXmlFileEqualsXmlFile(string $expectedFile, string $actualFile)	如果$actualFile 中的 XML 文档不等于$expectedFile 中的则报错。
void assertXmlFileEqualsXmlFile(string $expectedFile, string $actualFile, string $message)	如果$actualFile 中的 XML 文档不等于$expectedFile 中的则报$message 标识的错误。
void assertXmlStringEqualsXmlString(string $expectedXml, string $actualXml)	如果$actualXml 中的 XML 文档不等于$expectedXml 中的则报错。

断言	含义
`void assertXmlStringEqualsXmlString(string $expectedXml, string $actualXml, string $message)`	如果`$actualXml`中的 XML 文档不等于`$expectedXml`中的则报`$message`标识的错误。
`void assertArrayNotHasKey(mixed $key, array $array)`	如果`$array`含有`$key`则报错。
`void assertArrayNotHasKey(mixed $key, array $array, string $message)`	如果`$array`含有`$key`则报`$message`标识的错误。
`void assertClassNotHasAttribute(string $attributeName, string $className)`	如果`$className::attributeName`存在则报错。
`void assertClassNotHasAttribute(string $attributeName, string $className, string $message)`	如果`$className::attributeName`存在则报`$message`标识的错误。
`void assertClassNotHasStaticAttribute(string $attributeName, string $className)`	如果`$className::attributeName`存在并且是`static`则报错。
`void`	如果

断言	含义
assertClassNotHasStaticAttribute(string $attributeName, string $className, string $message)	$className::attributeName 存在并且是 static 则报 $message 标识的错误。
void assertNotContains(mixed $needle, array $haystack)	如果$needle 是$haystack 中的一个元素则报错。
void assertNotContains(mixed $needle, array $haystack, string $message)	如果$needle 是$haystack 中的一个元素则报$message 标识的错误。
void assertNotContains(mixed $needle, Iterator $haystack)	如果$needle 是$haystack 中的一个元素则报错。
void assertNotContains(mixed $needle, Iterator $haystack, string $message)	如果$needle 是$haystack 中的一个元素则报$message 标识的错误。
void assertNotContains(string $needle, string $haystack)	如果$needle 是$haystack 的子字符串则报错。
void assertNotContains(string $needle, string $haystack, string $message)	如果$needle 是$haystack 的子字符串则报$message 标识的错误。
assertNotContainsOnly(string $type, array $haystack)	如果$haystack 只含有$type

断言	含义
	类型的变量则报错。
assertNotContainsOnly(string $type, array $haystack, NULL, string $message)	如果$haystack只含有$type类型的变量则报$message标识的错误。
assertNotContainsOnly(string $type, array $haystack, bool $isNativeType)	如果$haystack只含有$type类型的变量则报错。$isNativeType是指示$type是否 PHP 原生类型的标志。
assertNotContainsOnly(string $type, array $haystack, bool $isNativeType, string $message)	如果$haystack只含有$type类型的变量则报$message标识的错误。$isNativeType是指示$type是否 PHP 原生类型的标志。
assertNotContainsOnly(string $type, Iterator $haystack)	如果$haystack只含有$type类型的变量则报错。
assertNotContainsOnly(string $type, Iterator $haystack, NULL, string $message)	如果$haystack只含有$type类型的变量则报$message标识的错误。
assertNotContainsOnly(string $type, Iterator $haystack,	如果$haystack只含有$type类型的变量则报错。

断言	含义
bool $isNativeType)	$isNativeType 是指示$type 是否 PHP 原生类型的标志。
assertNotContainsOnly(string $type, Iterator $haystack, bool $isNativeType, string $message)	如果$haystack 只含有$type 类型的变量则报$message 标识的错误。 $isNativeType 是指示$type 是否 PHP 原生类型的标志。
void assertNotEquals(array $expected, array $actual)	如果数组$expected 和$actual 相等则报错。
void assertNotEquals(array $expected, array $actual, string $message)	如果数组$expected 和$actual 相等则报$message 标识的错误。
void assertNotEquals(float $expected, float $actual, '', float $delta = 0)	如果浮点数$expected 和 $actual 之差不在$delta 中则报错。
void assertNotEquals(float $expected, float $actual, string $message, float $delta = 0)	如果浮点数$expected 和 $actual 之差不在$delta 中则报$message 标识的错误。
void assertNotEquals(object $expected, object $actual)	如果对象$expected 和$actual 属性值相等则报错。

断言	含义
void assertNotEquals(object $expected, object $actual, string $message)	如果对象$expected和$actual属性值相等则报$message标识的错误。
void assertNotEquals(string $expected, string $actual)	如果字符串$expected和$actual相等则报错。
void assertNotEquals(string $expected, string $actual, string $message)	如果字符串$expected和$actual相等则报$message标识的错误。
void assertNotEquals(DOMDocument $expected, DOMDocument $actual)	如果 DOMDocument 对象$expected和$actual描绘的 XML 相等则报错。
void assertNotEquals(DOMDocument $expected, DOMDocument $actual, string $message)	如果 DOMDocument 对象$expected和$actual描绘的 XML 相等则报$message标识的错误。
void assertNotEquals(mixed $expected, mixed $actual)	如果变量$expected和$actual相等则报错。
void assertNotEquals(mixed $expected, mixed $actual, string $message)	如果变量$expected和$actual相等则报$message标识的错误。

断言	含义
`void assertFileNotEquals(string $expected, string $actual)`	如果`$expected`和`$actual`指定的文件内容相同则报错。
`void assertFileNotEquals(string $expected, string $actual, string $message)`	如果`$expected`和`$actual`指定的文件内容相同则报`$message`标识的错误。
`void assertFileNotExists(string $filename)`	如果`$filename`指定的文件存在则报错。
`void assertFileNotExists(string $filename, string $message)`	如果`$filename`指定的文件存在则报`$message`标识的错误。
`void assertNotNull(mixed $variable)`	如果`$variable`为`NULL`则报错。
`void assertNotNull(mixed $variable, string $message)`	如果`$variable`为`NULL`则报`$message`标识的错误。
`void assertNotRegExp(string $pattern, string $string)`	如果`$string`匹配正则表达式`$pattern`则报错。
`void assertNotRegExp(string $pattern, string $string, string $message)`	如果`$string`匹配正则表达式`$pattern`则报`$message`标识的错误。

断言	含义
`void assertNotSame(object $expected, object $actual)`	如果变量`$expected`和`$actual`引用相同的对象则报错。
`void assertNotSame(object $expected, object $actual, string $message)`	如果变量`$expected`和`$actual`引用相同的对象则报`$message`标识的错误。
`void assertNotSame(mixed $expected, mixed $actual)`	如果变量`$expected`和`$actual`具有相同的值和类型则报错。
`void assertNotSame(mixed $expected, mixed $actual, string $message)`	如果变量`$expected`和`$actual`具有相同的值和类型则报`$message`标识的错误。
`void assertNotType(string $expected, mixed $actual)`	如果变量`$actual`是`$expected`类型则报错。
`void assertNotType(string $expected, mixed $actual, string $message)`	如果变量`$actual`是`$expected`类型则报`$message`标识的错误。
`void assertObjectNotHasAttribute (string $attributeName, object $object)`	如果`$object->attributeName`存在则报错。
`void`	如果`$object->attributeName`

断言	含义
`assertObjectNotHasAttribute(string $attributeName, object $object, string $message)`	存在则报`$message`标识的错误。
`void assertXmlFileNotEqualsXmlFile(string $expectedFile, string $actualFile)`	如果`$actualFile`和`$expectedFile`中的XML文档相等则报错。
`void assertXmlFileNotEqualsXmlFile(string $expectedFile, string $actualFile, string $message)`	如果`$actualFile`和`$expectedFile`中的XML文档相等则报`$message`标识的错误。
`void assertXmlStringNotEqualsXmlString(string $expectedXml, string $actualXml)`	如果`$actualXml`和`$expectedXml`中的XML文档相等则则报错。
`void assertXmlStringNotEqualsXmlString(string $expectedXml, string $actualXml, string $message)`	如果`$actualXml`和`$expectedXml`中的XML文档相等则则报`$message`标识的错误。
`void assertAttributeContains(mixed $needle, string $haystackAttributeName,`	如果`$needle`不是`$haystackClassName::$haystackAttributeName`中的元素则报

断言	含义
string $haystackClassName)	错，后者可为数组、字符串或实现 Iterator 接口的对象。$haystackClassName::$haystackAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeContains(mixed $needle, string $haystackAttributeName, string $haystackClassName, string $message)	如果$needle 不是 $haystackClassName::$haystackAttributeName 中的元素则报 $message 标识的错误，后者可为数组、字符串或实现 Iterator 接口的对象。$haystackClassName::$haystackAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotContains(mixed $needle, string $haystackAttributeName, string $haystackClassName)	如果$needle 是 $haystackClassName::$haystackAttributeName 中的元素则报错，后者可为数组、字符串

断言	含义
	或实现 Iterator 接口的对象。$haystackClassName::$haystackAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotContains(mixed $needle, string $haystackAttributeName, string $haystackClassName, string $message)	如果$needle 是 $haystackClassName::$haystackAttributeName 中的元素则报 $message 标识的错误，后者可为数组、字符串或实现 Iterator 接口的对象。$haystackClassName::$haystackAttributeName 属性的可见性可为 public、protected 或 private。

断言	含义
void assertAttributeContains(mixed $needle, string $haystackAttributeName, object $haystackObject)	如果$needle不是$haystackObject->haystackAttributeName中的元素则报错，后者可为数组、字符串或实现 Iterator 接口的对象。$haystackObject->haystackAttributeName属性的可见性可为public、protected或private。
void assertAttributeContains(mixed $needle, string $haystackAttributeName, object $haystackObject, string $message)	如果$needle不是$haystackObject->haystackAttributeName中的元素则报$message标识的错误，后者可为数组、字符串或实现 Iterator 接口的对象。$haystackObject->haystackAttributeName属性的可见性可为public、protected或private。

断言	含义
void assertAttributeNotContains(mixed $needle, string $haystackAttributeName, object $haystackObject)	如果$needle是$haystackObject->haystackAttributeName中的元素则报错，后者可为数组、字符串或实现 Iterator 接口的对象。$haystackObject->haystackAttributeName属性的可见性可为public、protected或private。
void assertAttributeNotContains(mixed $needle, string $haystackAttributeName, object $haystackObject, string $message)	如果$needle不是$haystackObject->haystackAttributeName中的元素则报$message标识的错误，后者可为数组、字符串或实现 Iterator 接口的对象。$haystackObject->haystackAttributeName属性的可见性可为public、protected或private。
void assertAttributeEquals(array $expected, string $actualAttributeName, string $actualClassName)	如果数组$expected和$actualClassName::$actualAttributeName不等则报错。$actualClassName::$actualAttributeName属性的可见性可为

断言	含义
	public、protected 或 private。
void assertAttributeEquals(array $expected, string $actualAttributeName, string $actualClassName, string $message)	如果数组$expected 和 $actualClassName::$actualAttributeName 不等则报$message 标识的错误。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(array $expected, string $actualAttributeName, string $actualClassName)	如果数组$expected 和 $actualClassName::$actualAttributeName 相等则报错。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(array $expected, string $actualAttributeName, string $actualClassName, string $message)	如果数组$expected 和 $actualClassName::$actualAttributeName 相等则报$message 标识的错误。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。

断言	含义
void assertAttributeEquals(float $expected, string $actualAttributeName, string $actualClassName, '', float $delta = 0)	如果浮点数$expected和$actualClassName::$actualAttributeName之差不在$delta中则报错。$actualClassName::$actualAttributeName属性的可见性可为public、protected或private。
void assertAttributeEquals(float $expected, string $actualAttributeName, string $actualClassName, string $message, float $delta = 0)	如果浮点数$expected和$actualClassName::$actualAttributeName之差不在$delta中则报$message标识的错误。$actualClassName::$actualAttributeName属性的可见性可为public、protected或private。
void assertAttributeNotEquals(float $expected, string $actualAttributeName, string $actualClassName, '', float $delta = 0)	如果浮点数$expected和$actualClassName::$actualAttributeName之差在$delta中则报错。$actualClassName::$actualAttributeName属性的可见性可为public、protected或private。
void	如果浮点数$expected和

断言	含义
assertAttributeNotEquals(float $expected, string $actualAttributeName, string $actualClassName, string $message, float $delta = 0)	$actualClassName::$actualAttributeName之差在$delta中则报$message标识的错误。$actualClassName::$actualAttributeName属性的可见性可为public、protected或private。
void assertAttributeEquals(string $expected, string $actualAttributeName, string $actualClassName)	如果字符串$expected和$actualClassName::$actualAttributeName不等则报错。同差异一样，错误也被报告。$actualClassName::$actualAttributeName属性的可见性可为public、protected或private。
void assertAttributeEquals(string $expected, string $actualAttributeName, string $actualClassName, string $message)	如果字符串$expected和$actualClassName::$actualAttributeName不等则报$message标识的错误。同差异一样，错误也被报告。$actualClassName::$actualAttributeName属性的可见性可为public、protected或private。
void	如果字符串$expected和

断言	含义
assertAttributeNotEquals(string $expected, string $actualAttributeName, string $actualClassName)	$actualClassName::$actualAttributeName 相等则报错。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(string $expected, string $actualAttributeName, string $actualClassName, string $message)	如果字符串$expected 和 $actualClassName::$actualAttributeName 相等则报$message 标识的错误。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeEquals(mixed $expected, string $actualAttributeName, string $actualClassName)	如果变量$expected 和 $actualClassName::$actualAttributeName 不等则报错。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeEquals(mixed $expected, string $actualAttributeName, string	如果变量$expected 和 $actualClassName::$actualAttributeName 不等则报$message

断言	含义
$actualClassName, string $message)	标识的错误。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(mixed $expected, string $actualAttributeName, string $actualClassName)	如果变量 $expected 和 $actualClassName::$actualAttributeName 相等则报错。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(mixed $expected, string $actualAttributeName, string $actualClassName, string $message)	如果变量 $expected 和 $actualClassName::$actualAttributeName 相等则报 $message 标识的错误。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeEquals(array $expected, string $actualAttributeName, object $actualObject)	如果数组 $expected 和 $actualObject->actualAttributeName 不等则报错。$actualObject->actualAttributeName 属性的可见性可为

断言	含义
	public、protected 或 private。
void assertAttributeEquals(array $expected, string $actualAttributeName, object $actualObject, string $message)	如果数组$expected 和 $actualObject->actualAttributeName 不等则报$message 标识的错误。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(array $expected, string $actualAttributeName, object $actualObject)	如果数组$expected 和 $actualObject->actualAttributeName 相等则报错。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(array $expected, string $actualAttributeName, object $actualObject, string $message)	如果数组$expected 和 $actualObject->actualAttributeName 相等则报$message 标识的错误。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。

断言	含义
void assertAttributeEquals(float $expected, string $actualAttributeName, object $actualObject, '', float $delta = 0)	如果浮点数$expected和$actualObject->actualAttributeName之差不在$delta中则报错。$actualObject->actualAttributeName属性的可见性可为public、protected或private。
void assertAttributeEquals(float $expected, string $actualAttributeName, object $actualObject, string $message, float $delta = 0)	如果浮点数$expected和$actualObject->actualAttributeName之差不在$delta中则报$message标识的错误。$actualObject->actualAttributeName属性的可见性可为public、protected或private。
void assertAttributeNotEquals(float $expected, string $actualAttributeName, object $actualObject, '', float $delta = 0)	如果浮点数$expected和$actualObject->actualAttributeName之差在$delta中则报错。$actualObject->actualAttributeName属性的可见性可为public、protected或private。
void	如果浮点数$expected和

断言	含义
assertAttributeNotEquals(float $expected, string $actualAttributeName, object $actualObject, string $message, float $delta = 0)	$actualObject->actualAttributeName 之差在$delta 中则报 $message 标识的错误。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeEquals(string $expected, string $actualAttributeName, object $actualObject)	如果字符串$expected 和 $actualObject->actualAttributeName 不等则报错。同差异一样，错误也被报告。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeEquals(string $expected, string $actualAttributeName, object $actualObject, string $message)	如果字符串$expected 和 $actualObject->actualAttributeName 不等则报$message 标识的错误。同差异一样，错误也被报告。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void	如果字符串$expected 和

断言	含义
assertAttributeNotEquals(string $expected, string $actualAttributeName, object $actualObject)	$actualObject->actualAttributeName 相等则报错。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(string $expected, string $actualAttributeName, object $actualObject, string $message)	如果字符串 $expected 和 $actualObject->actualAttributeName 相等则报 $message 标识的错误。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeEquals(mixed $expected, string $actualAttributeName, object $actualObject)	如果变量 $expected 和 $actualObject->actualAttributeName 不等则报错。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeEquals(mixed $expected, string $actualAttributeName, object $actualObject, string	如果变量 $expected 和 $actualObject->actualAttributeName 不等则报 $message 标识的错误。

断言	含义
$message)	$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(mixed $expected, string $actualAttributeName, object $actualObject)	如果变量$expected 和 $actualObject->actualAttributeName 相等则报错。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotEquals(mixed $expected, string $actualAttributeName, object $actualObject, string $message)	如果变量$expected 和 $actualObject->actualAttributeName 相等则报$message 标识的错误。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeSame(object $expected, string $actualAttributeName, string $actualClassName)	如果 $actualClassName::$actualAttributeName 和$actual 引用不同的对象则报错。$actualClassName::$actualAttributeName 属性的可见性可为

断言	含义
	public、protected 或 private。
void assertAttributeSame(object $expected, string $actualAttributeName, string $actualClassName, string $message)	如果 $actualClassName::$actualAttributeName 和 $actual 引用不同的对象则报 $message 标识的错误。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeSame(mixed $expected, string $actualAttributeName, string $actualClassName)	如果 $actualClassName::$actualAttributeName 和 $actual 的值或类型不同则报错。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeSame(mixed $expected, string $actualAttributeName, string $actualClassName, string $message)	如果 $actualClassName::$actualAttributeName 和 $actual 的值或类型不同则报 $message 标识的错误。$actualClassName::$actualAttr

断言	含义
	ibuteName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotSame(object $expected, string $actualAttributeName, string $actualClassName)	如果 $actualClassName::$actualAttributeName 和 $actual 引用相同的对象则报错。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotSame(object $expected, string $actualAttributeName, string $actualClassName, string $message)	如果 $actualClassName::$actualAttributeName 和 $actual 引用相同的对象则报 $message 标识的错误。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotSame(mixed $expected, string $actualAttributeName, string $actualClassName)	如果 $actualClassName::$actualAttributeName 和 $actual 的值和类型都相同则报错。$actualClassName::$actualAttr

断言	含义
	ibuteName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotSame(mixed $expected, string $actualAttributeName, string $actualClassName, string $message)	如果 $actualClassName::$actualAttributeName 和 $actual 的值和类型都相同则报 $message 标识的错误。$actualClassName::$actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeSame(object $expected, string $actualAttributeName, object $actualObject)	如果 $actualObject->actualAttributeName 和 $actual 引用不同的对象则报错。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeSame(object $expected, string $actualAttributeName, object $actualObject, string $message)	如果 $actualObject->actualAttributeName 和 $actual 引用不同的对象则报 $message 标识的错

断言	含义
	误。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeSame(mixed $expected, string $actualAttributeName, object $actualObject)	如果 $actualObject->actualAttributeName 和 $actual 的值或类型不同则报错。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeSame(mixed $expected, string $actualAttributeName, object $actualObject, string $message)	如果 $actualObject->actualAttributeName 和 $actual 的值或类型不同则报 $message 标识的错误。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotSame(object $expected, string	如果 $actualObject->actualAttributeName 和 $actual 引用相同的

断言	含义
$actualAttributeName, object $actualObject)	对象则报错。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotSame(object $expected, string $actualAttributeName, object $actualObject, string $message)	如果 $actualObject->actualAttributeName 和$actual 引用相同的对象则报$message 标识的错误。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotSame(mixed $expected, string $actualAttributeName, object $actualObject)	如果 $actualObject->actualAttributeName 和$actual 的值和类型都相同则报错。$actualObject->actualAttributeName 属性的可见性可为 public、protected 或 private。
void assertAttributeNotSame(mixed $expected, string	如果 $actualObject->actualAttributeName 和$actual 的值和类型

断言	含义
`$actualAttributeName, object $actualObject, string $message)`	都相同则报`$message`标识的错误。`$actualObject->actualAttributeName`属性的可见性可为`public`、`protected`或`private`。

更复杂的断言可用`PHPUnit_Framework_Constraint`类表述。它们可利用`assertThat()`方法被计算，如范例 22.3 中所示。

范例 22.3: 同约束对象一起使用 assertThat()

```
<?php
require_once 'PHPUnit/Framework.php';

class ConstraintTest extends PHPUnit_Framework_TestCase
{
    public function testNotEquals()
    {
        $constraint = $this->logicalNot(
          $this->equalTo('foo')
        );

        $this->assertThat('foo', $constraint);
    }
}
?>
```

```
phpunit ConstraintTest
```

```
PHPUnit 3.2.10 by Sebastian Bergmann.

F

Time: 0 seconds

There was 1 failure:

1) testNotEquals(ConstraintTest)
Failed asserting that <string:foo> is not equal to
<string:foo>.
/home/sb/ConstraintTest.php:12

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

表 22.2 显示可用的 PHPUnit_Framework_Constraint 实现。

表 22.2. 约束

约束	含义
PHPUnit_Framework_Constraint_Attribute attribute(PHPUnit_Framework_Constraint $constraint, $attributeName)	将另一个约束应用于

约束	含义
	类或对象的某个属性的约束。
PHPUnit_Framework_Constraint_IsAnything anything()	接受任意输入值的约束。
PHPUnit_Framework_Constraint_ArrayHasKey arrayHasKey(mixed $key)	断言其所计算的数组具有指定的关键字的约束。
PHPUnit_Framework_Constraint_TraversableContains contains(mixed $value)	断言其所计算的 array

约束	含义
	或实现接口 `Iterator` 的对象含有指定值的约束。
`PHPUnit_Framework_Constraint_IsEqual` `equalTo($value, $delta = 0, $maxDepth = 10)`	检查某值是否等于另一个的约束。

约束	含义
`PHPUnit_Framework_Constraint_Attribute attributeEqualTo($attributeName, $value, $delta = 0, $maxDepth = 10)`	检查某值是否等于某类或对象的一个属性的约束。
`PHPUnit_Framework_Constraint_FileExists fileExists()`	检查其所计算的文件（名）是否存在的约束。
`PHPUnit_Framework_Constraint_GreaterThan greaterThan(mixed $value)`	断言其所计算的值比给定值大的约

约束	含义
	束。
`PHPUnit_Framework_Constraint_OrgreaterThanOrEqual(mixed $value)`	断言其所计算的值比给定值大或与之相等的约束。
`PHPUnit_Framework_Constraint_ClassHasAttributeclassHasAttribute(string $attributeName)`	断言其所计算的类具有指定属性的约束。
`PHPUnit_Framework_Constraint_ClassHasStaticAttribute classHasStaticAttribute(string $attributeName)`	断言其所计算的类具有指定 static

约束	含义
	属性的约束。
PHPUnit_Framework_Constraint_ObjectHasAttribute hasAttribute(string $attributeName)	断言其所计算的对象具有指定属性的约束。
PHPUnit_Framework_Constraint_IsIdentical identicalTo(mixed $value)	断言某值与另一个是同一个的约束。
PHPUnit_Framework_Constraint_IsInstanceOf isInstanceOf(string $className)	断言其所计算的对象是给定类的实

约束	含义
	例的约束。
PHPUnit_Framework_Constraint_IsType isType(string $type)	断言其所计算的值是指定类型的约束。
PHPUnit_Framework_Constraint_LessThan lessThan(mixed $value)	断言其所计算的值比给定值小的约束。
PHPUnit_Framework_Constraint_OrlessThanOrEqual(mixed $value)	断言其所计算的值比给定值小或与之相等

约束	含义
	的约束。
`logicalAnd()`	逻辑与。
`logicalNot(PHPUnit_Framework_Constraint $constraint)`	逻辑非。
`logicalOr()`	逻辑或。
`logicalXor()`	逻辑异或。
`PHPUnit_Framework_Constraint_PCREMatch matchesRegularExpression(string $pattern)`	断言其所计算的字符串匹配一个正则表达式的约束。
`PHPUnit_Framework_Constraint_StringContains stringContains(string $string, bool $case)`	断言其所计算

约束	含义
	的字符串包含给定字符串的约束。

也许你会发现，除了这些，你还需要其他断言来比较特定于你的项目的对象。创建你自己的 `Assert` 类来包含这些断言以便简化你的测试。

失败的断言都调用一个瓶颈方法，`fail(string $message)`，它抛出一个 `PHPUnit_Framework_AssertionFailedError`。还有个不带参数的变体。当你的测试遇到错误时显式调用 `fail()`。期望异常的测试是个例子。 表 22.3 列举了 PHPUnit 中的瓶颈方法。

表 22.3. 瓶颈方法

方法	含义
`void fail()`	报告错误。

方法	含义
`void fail(string $message)`	报告由`$message`标识的错误。

`markTestIncomplete()`和`markTestSkipped()`是标记测试为未完成或跳过的便利方法。

表 22.4. 标记测试为未完成或跳过的

方法	含义
`void markTestIncomplete(string $message)`	标记当前测试为未完成的，`$message`为可选的。
`void markTestSkipped(string $message)`	标记当前测试为跳过的，`$message`为可选的。

尽管单元测试用于测试类的 public 接口，或许有时候你想要测试非 public 属性值。`readAttribute()`方法使你能这样做，它返回给定类或对象的给定（static）属性值。

表 22.5. 访问非 public 属性

方法	含义

方法	含义
`Mixed readAttribute($classOrObject, $attributeName)`	返回类或对象的给定（static）属性（`$attributeName`）值。它也可用于声明为 `protected` 或 `private` 的属性

PHPUnit_Framework_Test

`PHPUnit_Framework_Test` 是所有具有测试作用的对象用到的通用接口。实现者可以描述一个或更多测试。表 22.6 中显示两个方法。

表 **22.6.**实现者方法

方法	含义
`int count()`	返回测试数量。
`void run(PHPUnit_Framework_TestResult $result)`	运行测试并向 `$result` 报告错误。

PHPUnit_Framework_TestCase 和 PHPUnit_Framework_TestSuite 是 PHPUnit_Framework_Test 的两个最出色的实现。你可以自己实现 PHPUnit_Framework_Test。该接口刻意保持很小因此易于实现。

PHPUnit_Framework_TestCase

你的测试用例类必须继承自 PHPUnit_Framework_TestCase。多数时候，你将运行自动创建的测试套件中的测试。既然如此，你的每个测试都应该由 test*（按照约定）指定的方法表示。

PHPUnit_Framework_TestCase 实现 PHPUnit_Framework_Test::count() 为总是返回 1。该类中实现的 PHPUnit_Framework_Test::run(PHPUnit_Framework_TestResult $result) 运行 setUp()，运行测试方法，然后运行 tearDown() 并向 PHPUnit_Framework_TestResult 报告任何错误。

表 22.7 显示 PHPUnit_Framework_TestCase 提供的方法。

表 22.7. TestCase

方法	含义
__construct()	创建一个测试用

方法	含义
	例。
`__construct(string $name)`	创建一个指定的测试用例。名字用于打印测试用例，且经常作为将要通过反射运行的测试方法的名字。
`string getName()`	返回测试用例的名字。
`void setName($name)`	设置试用例的名字。
`PHPUnit_Framework_TestResult run(PHPUnit_Framework_TestResult $result)`	运行测试用例并在`$result`中报告的便利方法。
`void runTest()`	如果你不想通过反射调用测试方法，重写测试方法。
`object getMock($className, [array $methods, [array $arguments, [string $mockClassName, [boolean $callOriginalConstructor, [boolean`	返回一个用于指定的`$className`的模拟对象（见第 11

方法	含义
`$callOriginalClone, [boolean $callAutoload]]]]]])`	章）。缺省地，给定类的所有方法都是模拟的。当提供第二个（可选）参数时，只有名字出现在数组中的方法是模拟的。第三个（可选）参数持有参数数组，用于传入模拟对象的构造函数。第四个（可选）参数可用于为模拟对象指定类名。第五个（可选）参数可用于禁用对原始对象的`__construct()`方法的调用。第六个（可选）参数可用于禁用对原始对象的`__clone()`方法的调

方法	含义
	用。第七个（可选）参数可用于在模拟对象创建期间禁用 `__autoload()`。
`void iniSet(string $varName, mixed $newValue)`	该方法是对 `ini_set()` 函数的封装，在测试运行后自动地将改过的 *php.ini* 设定重置为初始值。
`void setLocale(integer $category, string $locale, ...)`	该方法是对 `setlocale()` 函数的封装，在测试运行后自动地将 locale 重置为初始值。

有两个模板方法——`setUp()` 和 `tearDown()`——你可以重写以创建和清除为将进行的测试准备的对象。表 22.8 显示这些方法。还有第三个模板方法，

sharedAssertions()，允许定义可被一个测试用例中的所有测试执行的断言。

表 22.8. 模板方法

方法	含义
void setUp()	重写以创建为测试准备的对象。每个运行的测试都将运行于自己专有的测试用例中，并且 setUp()将为其每一个被分别调用。
void sharedAssertions()	重写以执行为一个测试用例中的所有测试共享的断言。该方法在测试执行结束前和 tearDown()被调用前调用。
void tearDown()	重写以清理一旦测试结束即不再需要的对象。通常，你只需要在 tearDown()中显式地清理外部资源（例如文件或套接字）。

PHPUnit_Framework_TestSuite

PHPUnit_Framework_TestSuite 是 PHPUnit_Framework_Test 的组合。最简单的情况，它包含一组测试用例，当套件运

行时它们都会运行。由于它是个组合体，然而，套件可以包含套件，后者又可包含套件等等，使得可以容易地结合不同来源的测试并一起运行。

除了 `PHPUnit_Framework_Test` 的方法——`run(PHPUnit_Framework_TestResult $result)`和 `count()`，`PHPUnit_Framework_TestSuite` 提供一些创建命名或未命名实例的方法。 表 22.9 显示创建 `PHPUnit_Framework_TestSuite` 实例的方法。

表 22.9. 创建命名或未命名实例

方法	含义
`__construct()`	返回一个空测试套件。
`__construct(string $theClass)`	返回一个测试套件，包含一个类名为`$theClass` 的实例，该类的每个方法都以`test*`命名。如果不存在名为`$theClass` 的类，一个名为`$theClass` 空测试套件被返回。
`__construct(string $theClass, string $name)`	返回一个名为`$name` 的测试套件，包含一个类名为

方法	含义
	$theClass 的实例，该类的每个方法都以 test* 命名。
__construct(ReflectionClass $theClass)	返回一个测试套件，包含一个类名用 $theClass 描述的实例，该类的每个方法都以 test* 命名。
__construct(ReflectionClass $theClass, $name)	返回一个名为 $name 的测试套件，包含一个类名用 $theClass 描述的实例，该类的每个方法都以 test* 命名。
string getName()	返回测试套件名字。
void setName(string $name)	设置测试套件名字。
void markTestSuiteSkipped(string $message)	标记当前测试套件为将被跳过，$message 是可选的。

PHPUnit_Framework_TestSuite 也提供方法用于增加和获取 PHPUnit_Framework_Test，如表 22.10 中所示。

表 22.10. 增加和获取测试

方法	含义
`void addTestSuite(PHPUnit_Framework_TestSuite $suite)`	向测试套件增加另一个测试套件。
`void addTestSuite(string $theClass)`	向测试套件增加一个测试套件，它包含一个类名为 `$theClass` 的实例，该类的每个方法都以 `test*` 命名。
`void addTestSuite(ReflectionClass $theClass)`	向测试套件增加一个测试套件，它包含一个类名用 `$theClass` 描述的实例，该类的每个方法都以 `test*` 命名。
`void addTest(PHPUnit_Framework_Test $test)`	将 `$test` 加入套

方法	含义
	件。
`void addTestFile(string $filename)`	向套件增加一些测试，它们由给定的源文件中的类定义。
`void addTestFiles(array $filenames)`	向套件增加一些测试，它们由给定的源文件中的类定义。
`int testCount()`	返回由该套件直接包含的测试（非递归）的数量。
`PHPUnit_Framework_Test[] tests()`	返回由该套件直接包含的测试。
`PHPUnit_Framework_Test testAt(int $index)`	返回`$index`处的测试。

范例 22.4 显示如何创建和运行测试套件。

范例 22.4: 创建和运行测试套件

```
<?php
require_once 'PHPUnit/Framework.php';

require_once 'ArrayTest.php';

// 创建一个包含源自类 ArrayTest 的测试的测试套件。
$suite = new PHPUnit_Framework_TestSuite('ArrayTest');

// 运行测试。
$suite->run();
?>
```

第 7 章显示如何使用类 `PHPUnit_Framework_TestSuite` 通过分级组合测试用例来组织测试套件。

类 `PHPUnit_Framework_TestSuite` 提供两个模板方法——`setUp()` 和 `tearDown()`，分别在一个测试套件的（所有）测试运行前后被调用。

表 22.11. 模板方法

方法	含义

方法	含义
`void setUp()`	在测试套件的第一个测试运行前调用。
`void tearDown()`	在测试套件的最后一个测试运行后调用。

PHPUnit_Framework_TestResult

当你在运行所有这些测试时，你需要在某处存储所有结果：运行了多少测试，哪个失败了，以及他们耗时多久。`PHPUnit_Framework_TestResult` 收集这些结果。单个 `PHPUnit_Framework_TestResult` 被传遍测试树。当某个测试运行或失败，实际情况被记录在 `PHPUnit_Framework_TestResult` 中。在运行的结尾，`PHPUnit_Framework_TestResult` 含有所有测试的摘要。

`PHPUnit_Framework_TestResult` 也可被其他想要报告测试进程的对象观测。例如，一个图形测试启动器可能观测 `PHPUnit_Framework_TestResult` 并在每次某个测试启动时更新进度条。

表 22.12 概述 `PHPUnit_Framework_TestResult` 的 API。

表 22.12. TestResult

方法	含义
`void addError(PHPUnit_Framework_Test $test, Exception $e)`	记录`$test`的运行引发了意外抛出`$e`。
`void addFailure(PHPUnit_Framework_Test $test, PHPUnit_Framework_AssertionFailedError $e)`	记录`$test`的运行引发了意外抛出`$e`。
`PHPUnit_Framework_TestFailure[] errors()`	返回记录的错误。
`PHPUnit_Framework_TestFailure[] failures()`	返回记录的失败。
`PHPUnit_Framework_TestFailure[] notImplemented()`	返回记录的未完成的测试用例。
`int errorCount()`	返回错误的数量。
`int failureCount()`	返回失败的数量。

方法	含义
`int notImplementedCount()`	返回未完成的测试用例的数量。
`int count()`	返回运行的测试用例的总数。
`boolean wasSuccessfull()`	返回是否所有测试运行成功。
`boolean allCompletlyImplemented()`	返回是否所有测试都完全实现了。
`void collectCodeCoverageInformation(bool $flag)`	启用或禁用代码覆盖率信息的收集。

方法	含义
`array getCodeCoverageInformation()`	返回收集的代码覆盖率信息。

如果你要注册一个 `PHPUnit_Framework_TestResult` 的观察器，需要实现 `PHPUnit_Framework_TestListener`。要注册，调用 `addListener()`，如表 22.13 中所示。

表 22.13. TestResult 和 TestListener

方法	含义
`void addListener(PHPUnit_Framework_TestListener $listener)`	注册 `$listener`，当记录测试结果时接收更新信息。
`void removeListener(PHPUnit_Framework_TestListener $listener)`	从接收更新信息取消注册 `$listener`。

表 22.14 显示测试监听器实现的方法；又见范例 23.3。

表 22.14. TestListener 回到例程

方法	含义
`void addError(PHPUnit_Framework_Test $test, Exception $e)`	`$test` 抛出了 `$e`。
`void addFailure(PHPUnit_Framework_Test $test, PHPUnit_Framework_AssertionFailedError $e)`	`$test` 放弃了一个断言，抛出一种 `PHPUnit_Framework_AssertionFailedError`。
`void addIncompleteTest(PHPUnit_Framework_Test $test, Exception $e)`	`$test` 是未实现的测试。
`void addSkippedTest(PHPUnit_Framework_Test $test, Exception $e)`	`$test` 是已跳过的测试。
`void startTestSuite(PHPUnit_Framework_TestSuite $suite)`	`$suite` 将要运行。
`void`	`$suite` 已完成运行。

方法	含义
endTestSuite(PHPUnit_Framework_TestSuite $suite)	
void startTest(PHPUnit_Framework_Test $test)	$test 将要运行。
void endTest(PHPUnit_Framework_Test $test)	$test 已完成运行。

包结构

迄今为止，本书中提到的类大多出自 PHPUnit/Framework。这儿是 PHPUnit 中的所有包：

- PHPUnit/Framework

  PHPUnit 中的基础类。

- PHPUnit/Extensions

  PHPUnit 框架的扩展。

- PHPUnit/Runner

  运行测试支持的抽象。

- PHPUnit/TextUI

基于文本的测试启动器。

- `PHPUnit/Util`

  被其他包用到的工具类。

第 23 章 扩展 PHPUnit

PHPUnit 能以各种不同的方式扩展，以使编写测试更容易并且定制来自运行测试的反馈。这是扩展 PHPUnit 的共同起点。

子类化 PHPUnit_Framework_TestCase

在抽象子类 `PHPUnit_Framework_TestCase` 中编写工具方法并从该类得到你的测试用例类。这是扩展 PHPUnit 的最简单的方法之一。

断言类

编写你自己的类，它们带有专用于你的目的的断言。

子类化 PHPUnit_Extensions_TestDecorator

你可以在 `PHPUnit_Extensions_TestDecorator` 的子类中中封装测试用例或测试套件，并使用装饰者（Decorator）模式在测试运行前后执行某些动作。

PHPUnit 自带两个具体的测试装饰者：`PHPUnit_Extensions_RepeatedTest` 和 `PHPUnit_Extensions_TestSetup`。前一个用于重复运行一个测试，并且只当所有迭代都成功时才算成功。后面一个在第 6 章中讨论过。

范例 23.1 显示一个删节版的 `PHPUnit_Extensions_RepeatedTest` 测试装饰者，例解如何编写自己的测试装饰者。

范例 23.1: RepeatedTest 装饰者

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

class PHPUnit_Extensions_RepeatedTest extends PHPUnit_Ext
ensions_TestDecorator
{
    private $timesRepeat = 1;

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

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

    public function count()
    {
```

```
        return $this->timesRepeat * $this->test->count();
    }

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

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

        return $result;
    }
}
?>
```

实现 PHPUnit_Framework_Test

接口 `PHPUnit_Framework_Test` 很有限且易于实现。你能够编写一个比 `PHPUnit_Framework_TestCase` 简单而且运行（比如）*数据驱动测试*的 `PHPUnit_Framework_Test` 的实现。

范例 23.2 显示一个数据驱动的测试用例类，用来比较来自文件的逗号分隔的值（CSV）。这种文件的每一行类似 `foo;bar`，其第一个值是我们期望的，第二个值是实际值。

范例 23.2: 一个数据驱动测试

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'PHPUnit/Util/Timer.php';
require_once 'PHPUnit/TextUI/TestRunner.php';

class DataDrivenTest implements PHPUnit_Framework_Test
{
    private $lines;

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

    public function count()
    {
        return 1;
    }

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

        foreach ($this->lines as $line) {
            $result->startTest($this);
            PHPUnit_Util_Timer::start();

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

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

```
            catch (PHPUnit_Framework_AssertionFailedError
 $e) {
                $result->addFailure($this, $e, PHPUnit_Ut
il_Timer::stop());
            }

            catch (Exception $e) {
                $result->addError($this, $e, PHPUnit_Util
_Timer::stop());
            }

            $result->endTest($this, PHPUnit_Util_Timer::s
top());
        }

        return $result;
    }
}

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

```
PHPUnit 3.2.10 by Sebastian Bergmann.



.F



Time: 0 seconds



There was 1 failure:



1) DataDrivenTest

Failed asserting that two strings are equal.

expected string <bar>
```

```
difference      <  x>

got string      <baz>

/home/sb/DataDrivenTest.php:32

/home/sb/DataDrivenTest.php:53


FAILURES!

Tests: 2, Failures: 1.
```

子类化 PHPUnit_Framework_TestResult

通过向 `run()` 方法传入一个特殊用途的 `PHPUnit_Framework_TestResult` 对象，你能改变测试运行的方式和收集的结果数据。

实现 PHPUnit_Framework_TestListener

要定制 `PHPUnit_Framework_TestResult`，没必要编写它的整个子类。大多时候，实现一个新 `PHPUnit_Framework_TestListener`（见表 22.14）并在运行测试前附在 `PHPUnit_Framework_TestResult` 对象上就够了。

范例 23.3 显示 `PHPUnit_Framework_TestListener` 接口的一个简单实现。

范例 23.3:一个简单的测试监听器

```php
<?php
require_once 'PHPUnit/Framework.php';

class SimpleTestListener
implements PHPUnit_Framework_TestListener
{
  public function
  addError(PHPUnit_Framework_Test $test,
           Exception $e,
           $time)
  {
    printf(
      "Error while running test '%s'.\n",
      $test->getName()
    );
  }

  public function
  addFailure(PHPUnit_Framework_Test $test,
             PHPUnit_Framework_AssertionFailedError $e,
             $time)
  {
    printf(
      "Test '%s' failed.\n",
      $test->getName()
    );
  }

  public function
  addIncompleteTest(PHPUnit_Framework_Test $test,
                    Exception $e,
                    $time)
  {
    printf(
      "Test '%s' is incomplete.\n",
      $test->getName()
    );
  }

  public function
```

```
  addSkippedTest(PHPUnit_Framework_Test $test,
                 Exception $e,
                 $time)
  {
    printf(
      "Test '%s' has been skipped.\n",
      $test->getName()
    );
  }

  public function startTest(PHPUnit_Framework_Test $test)
  {
    printf(
      "Test '%s' started.\n",
      $test->getName()
    );
  }

  public function endTest(PHPUnit_Framework_Test $test, $
time)
  {
    printf(
      "Test '%s' ended.\n",
      $test->getName()
    );
  }

  public function
  startTestSuite(PHPUnit_Framework_TestSuite $suite)
  {
    printf(
      "TestSuite '%s' started.\n",
      $suite->getName()
    );
  }

  public function
  endTestSuite(PHPUnit_Framework_TestSuite $suite)
  {
    printf(
```

```
      "TestSuite '%s' ended.\n",
      $suite->getName()
    );
  }
}
?>
```

范例 23.4 显示如何运行和观测测试套件。

范例 23.4: 运行和观测测试套件

```
<?php
require_once 'PHPUnit/Framework.php';

require_once 'ArrayTest.php';
require_once 'SimpleTestListener.php';

// 创建一个包括测试套件，来自类 ArrayTest 的测试。
$suite = new PHPUnit_Framework_TestSuite('ArrayTest');

// 创建一个测试结果，并附上一个 SimpleTestListener 对象作为对它的观测者。
$result = new PHPUnit_Framework_TestResult;
$result->addListener(new SimpleTestListener);

// 运行测试。
$suite->run($result);
?>
```

```
TestSuite 'ArrayTest' started.

Test 'testNewArrayIsEmpty' started.

Test 'testNewArrayIsEmpty' ended.
```

```
Test 'testArrayContainsAnElement' started.

Test 'testArrayContainsAnElement' ended.

TestSuite 'ArrayTest' ended.
```

新测试运行器

如果你需要从测试运行得到不同的反馈，编写你自己的测试启动器，交互式的或非交互的。为类 `PHPUnit_TextUI_TestRunner`（PHPUnit 命令行测试启动器）所继承的抽象类 `PHPUnit_Runner_BaseTestRunner` 可作为起点。

附录 A XML 配置文件

...

测试套件

...

```
<phpunit>
  <testsuite name="My Test Suite">
    <directory>/path/to/*Test.php files</directory>
    <file>/path/to/MyTest.php</file>
  </testsuite>
</phpunit>
```

分组

...

```
<phpunit>
  <groups>
    <include>
      <group>name</group>
    </include>
    <exclude>
      <group>name</group>
    </exclude>
  </groups>
</phpunit>
```

上面的XML配置对应带下面的参数调用TextUI测试启动器：

- `--group name`
- `--exclude-group name`

包含及排除用于代码覆盖率的文件

...

```
<phpunit>
  <filter>
    <blacklist>
```

```
    <directory suffix=".php">/path/to/files</directory>
    <file>/path/to/file</file>
    <exclude>
      <directory suffix=".php">/path/to/files</directory>
      <file>/path/to/file</file>
    </exclude>
  </blacklist>
  <whitelist>
    <directory suffix=".php">/path/to/files</directory>
    <file>/path/to/file</file>
    <exclude>
      <directory suffix=".php">/path/to/files</directory>
      <file>/path/to/file</file>
    </exclude>
  </whitelist>
 </groups>
</phpunit>
```

上面的 XML 配置对应如下使用类 `PHPUnit_Util_Filter`:

```
PHPUnit_Util_Filter::addDirectoryToFilter(
 /path/to/files', '.php'
);

PHPUnit_Util_Filter::addFileToFilter('/path/to/file');
```

```
PHPUnit_Util_Filter::removeDirectoryFromFilter(
  /path/to/files', '.php'
);

PHPUnit_Util_Filter::removeFileFromFilter('/path/to/file')
;

PHPUnit_Util_Filter::addDirectoryToWhitelist(
  /path/to/files', '.php'
);

PHPUnit_Util_Filter::addFileToWhitelist('/path/to/file');

PHPUnit_Util_Filter::removeDirectoryFromWhitelist(
  /path/to/files', '.php'
);

PHPUnit_Util_Filter::removeFileFromWhitelist('/path/to/fi
le');
```

日志

...

```
<phpunit>
  <logging>
```

```
    <log type="coverage-html" target="/tmp/report"
charset="UTF-8"
         yui="true" highlight="false"
         lowUpperBound="35" highLowerBound="70"/>
    <log type="coverage-xml" target="/tmp/coverage.xml"/>
    <log type="graphviz" target="/tmp/logfile.dot"/>
    <log type="json" target="/tmp/logfile.json"/>
    <log type="metrics-xml" target="/tmp/metrics.xml"/>
    <log type="plain" target="/tmp/logfile.txt"/>
    <log type="pmd-xml" target="/tmp/pmd.xml"
cpdMinLines="5" cpdMinMatches="70"/>
    <log type="tap" target="/tmp/logfile.tap"/>
    <log type="test-xml" target="/tmp/logfile.xml"
logIncompleteSkipped="false"/>
    <log type="testdox-html" target="/tmp/testdox.html"/>
    <log type="testdox-text" target="/tmp/testdox.txt"/>
  </logging>
</phpunit>
```

上面的XML配置对应带下面的参数调用TextUI测试启动器：

- `--coverage-html /tmp/report`
- `--coverage-xml /tmp/coverage.xml`
- `--log-graphviz /tmp/logfile.dot`
- `--log-json /tmp/logfile.json`
- `--log-metrics /tmp/metrics.xml`

- > /tmp/logfile.txt
- --log-pmd /tmp/pmd.xml
- --log-tap /tmp/logfile.tap
- --log-xml /tmp/logfile.xml
- --testdox-html /tmp/testdox.html
- --testdox-text /tmp/testdox.txt

PMD 规则

...

```
<phpunit>
  <logging>
    <pmd>
      <rule class="PHPUnit_Util_Log_PMD_Rule_Project_CRAP"
            threshold="5,30"/>
      <rule
class="PHPUnit_Util_Log_PMD_Rule_Class_DepthOfInheritance
Tree"
            threshold="6"/>
      <rule
class="PHPUnit_Util_Log_PMD_Rule_Class_EfferentCoupling"
            threshold="20"/>
      <rule
class="PHPUnit_Util_Log_PMD_Rule_Class_ExcessiveClassLeng
th"
            threshold="1000"/>
```

```
    <rule
class="PHPUnit_Util_Log_PMD_Rule_Class_ExcessivePublicCou
nt"
          threshold="45"/>
    <rule
class="PHPUnit_Util_Log_PMD_Rule_Class_TooManyFields"
          threshold="15"/>
    <rule
class="PHPUnit_Util_Log_PMD_Rule_Function_CodeCoverage"
          threshold="35,70"/>
    <rule class="PHPUnit_Util_Log_PMD_Rule_Function_CRAP"
          threshold="30"/>
    <rule
class="PHPUnit_Util_Log_PMD_Rule_Function_CyclomaticCompl
exity"
          threshold="20"/>
    <rule
class="PHPUnit_Util_Log_PMD_Rule_Function_ExcessiveMethod
Length"
          threshold="100"/>
    <rule
class="PHPUnit_Util_Log_PMD_Rule_Function_ExcessiveParame
terList"
          threshold="10"/>
    <rule
class="PHPUnit_Util_Log_PMD_Rule_Function_NPathComplexity
"
          threshold="200"/>
  </pmd>
```

```
  </logging>
</phpunit>
```

上面的<rule>对应内建的 PMD 规则的默认设置。

设置 PHP INI 和全局变量

...

```
<phpunit>
  <php>
    <ini name="foo" value="bar"/>
    <var name="foo" value="bar"/>
  </php>
</phpunit>
```

上面的 XML 配置对应下面的 PHP 代码：

```
ini_set('foo', 'bar');
$GLOBALS['foo'] = 'bar';
```

附录 B. 用于 PHP 4 的 PHPUnit

存在一些用于 PHP 4 而不需要 PHP 5 的 PHPUnit 发行版。由于 PHP 4 的受限对象模型，用于它的 PHPUnit 不像用于 PHP 5 的 PHPUnit 那样是 JUnit 的完全对等 实现 。它也缺少用于 PHP 5 的 PHPUnit 的某些特性，例如代码覆盖率分析。

下面的命令行显示如何利用 PEAR 安装程序安装用于 PHP 4 的 PHPUnit：

```
pear install -f phpunit/PHPUnit-1.3.3
```

用于 PHP 4 的 PHPUnit 的测试用例类类似于用于 PHP 5 的 PHPUnit 的测试用例类。本质区别在于该类扩展 `PHPUnit_TestCase`（它自己扩展提供断言方法的 `PHPUnit_Assert`）。

范例 B.1 显示用于 PHP 4 的 PHPUnit 下的 `ArrayTest` 测试用例的一个版本。

范例 B.1: 编写一个用于 PHPUnit 1.x 的测试用例

```
<?php
require_once 'PHPUnit/TestCase.php';

class ArrayTest extends PHPUnit_TestCase
{
    var $_fixture;

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

    function testNewArrayIsEmpty()
    {
        $this->assertEquals(0, sizeof($this->_fixture));
    }
```

```
    function testArrayContainsAnElement()
    {
        $this->_fixture[] = 'Element';
        $this->assertEquals(1, sizeof($this->_fixture));
    }
}
?>
```

用于 PHP 4 的 PHPUnit 不提供 TextUI 测试启动器，最一般的运行测试方法是编写测试套件并手工运行它，如范例 B.2 中所示。

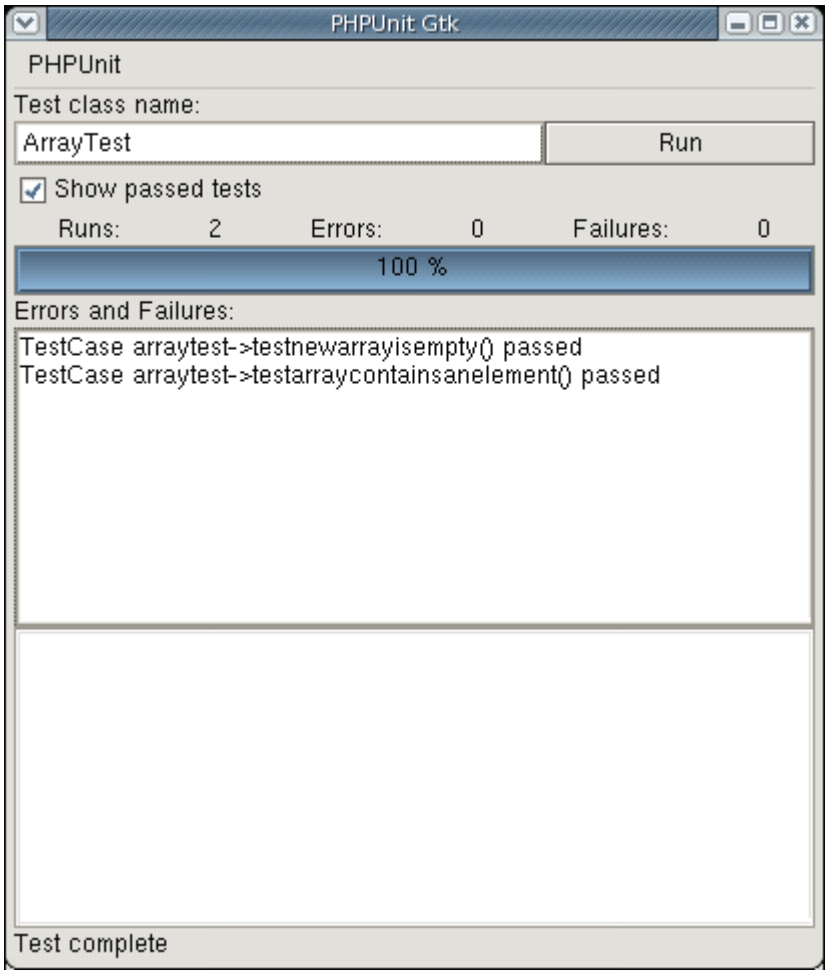
范例 B.2: 用 PHPUnit 1.x 运行一个测试用例

```
<?php
require_once 'ArrayTest.php';
require_once 'PHPUnit.php';

$suite  = new PHPUnit_TestSuite('ArrayTest');
$result = PHPUnit::run($suite);

print $result->toString();
?>
```

```
TestCase arraytest->testnewarrayisempty() passed

TestCase arraytest->testarraycontainsanelement() passed
```

图 B.1 显示一个用于 PHP 4 的 PHPUnit 具有而用于 PHP 5 的 PHPUnit 没有的特性：基于 PHP-GTK 的带图形用户界面的测试启动器。

图 **B.1. PHP-GTK** 测试启动器



附录 C. 索引

符号

@assert, 注解

@codeCoverageIgnoreEnd, 忽略代码块

@codeCoverageIgnoreStart, 忽略代码块

@covers, 指定覆盖的方法

A

assertArrayNotHasKey(),
PHPUnit_Framework_Assert
assertAttributeContains(),
PHPUnit_Framework_Assert
assertAttributeEquals(),
PHPUnit_Framework_Assert
assertAttributeNotContains(),
PHPUnit_Framework_Assert
assertAttributeNotEquals(),
PHPUnit_Framework_Assert
assertAttributeNotSame(),
PHPUnit_Framework_Assert
assertAttributeSame(),
PHPUnit_Framework_Assert
assertClassHasAttribute(),
PHPUnit_Framework_Assert
assertClassHasStaticAttribute(),
PHPUnit_Framework_Assert
assertClassNotHasAttribute(),
PHPUnit_Framework_Assert
assertClassNotHasStaticAttribute(),
PHPUnit_Framework_Assert

assertContains(), PHPUnit_Framework_Assert
assertContainsOnly(),
PHPUnit_Framework_Assert
assertEquals(), PHPUnit_Framework_Assert
assertFalse(), PHPUnit_Framework_Assert
assertFileEquals(),
PHPUnit_Framework_Assert
assertFileExists(),
PHPUnit_Framework_Assert
assertFileNotEquals(),
PHPUnit_Framework_Assert
assertFileNotExists(),
PHPUnit_Framework_Assert
assertGreaterThan(),
PHPUnit_Framework_Assert
assertGreaterThanOrEqual(),
PHPUnit_Framework_Assert
断言, 自动化测试, PHPUnit_Framework_Assert,
断言类
assertLessThan(),
PHPUnit_Framework_Assert

assertLessThanOrEqual(),
PHPUnit_Framework_Assert
assertNotContains(),
PHPUnit_Framework_Assert
assertNotContainsOnly(),
PHPUnit_Framework_Assert
assertNotEquals(),
PHPUnit_Framework_Assert
assertNotNull(), PHPUnit_Framework_Assert
assertNotRegExp(),
PHPUnit_Framework_Assert
assertNotSame(),
PHPUnit_Framework_Assert
assertNotType(), PHPUnit_Framework_Assert
assertNull(), PHPUnit_Framework_Assert
assertObjectHasAttribute(),
PHPUnit_Framework_Assert
assertObjectNotHasAttribute(),
PHPUnit_Framework_Assert
assertRegExp(), PHPUnit_Framework_Assert
assertSame(), PHPUnit_Framework_Assert
assertThat(), PHPUnit_Framework_Assert

assertTrue(), PHPUnit_Framework_Assert

assertType(), PHPUnit_Framework_Assert

assertXmlFileEqualsXmlFile(), PHPUnit_Framework_Assert

assertXmlFileNotEqualsXmlFile(), PHPUnit_Framework_Assert

assertXmlStringEqualsXmlString(), PHPUnit_Framework_Assert

assertXmlStringNotEqualsXmlString(), PHPUnit_Framework_Assert

attribute(), PHPUnit_Framework_Assert

attributeEqualTo(), PHPUnit_Framework_Assert

自动化文档，敏捷文档

自动化测试，自动化测试

B

黑名单，包含和排除文件，包含及排除用于代码覆盖率的文件

C

D

E

F

G

H

I

J

JSON, 命令行测试启动器

L

lessThan(), PHPUnit_Framework_Assert
lessThanOrEqual(),
PHPUnit_Framework_Assert
日志文件, 命令行测试启动器
日志, 日志, 日志
logicalAnd(), PHPUnit_Framework_Assert
logicalNot(), PHPUnit_Framework_Assert
logicalOr(), PHPUnit_Framework_Assert
logicalXor(), PHPUnit_Framework_Assert

M

markTestIncomplete(),
PHPUnit_Framework_Assert
markTestSkipped(),
PHPUnit_Framework_Assert
matchesRegularExpression(),
PHPUnit_Framework_Assert

O

观测者模式, PHPUnit_Framework_TestResult

P

PHPUnit_Extensions_OutputTestCase, 测试输出

PHPUnit_Extensions_PerformanceTestCase, 测试性能

PHPUnit_Extensions_RepeatedTest, 子类化 PHPUnit_Extensions_TestDecorator

PHPUnit_Extensions_TestDecorator, 子类化 PHPUnit_Extensions_TestDecorator

PHPUnit_Extensions_TestSetup, 子类化 PHPUnit_Extensions_TestDecorator

PHPUnit_Framework_Assert, BankAccount 范例, PHPUnit_Framework_Assert, 断言类

PHPUnit_Framework_IncompleteTest, 未完成的测试

PHPUnit_Framework_IncompleteTestError, 未完成的测试

PHPUnit_Framework_Test, PHPUnit 的执行, PHPUnit_Framework_Test, 实现 PHPUnit_Framework_Test

PHPUnit_Framework_TestCase, 编写 PHPUnit 测试, PHPUnit_Framework_TestCase, 子类化 PHPUnit_Framework_TestCase

PHPUnit_Framework_TestListener, PHPUnit_Framework_TestResult, 实现 PHPUnit_Framework_TestListener

PHPUnit_Framework_TestResult, PHPUnit 的执行, PHPUnit_Framework_TestResult, 子类化 PHPUnit_Framework_TestResult, 实现 PHPUnit_Framework_TestListener

PHPUnit_Framework_TestSuite, PHPUnit 的执行, PHPUnit_Framework_TestSuite, 实现 PHPUnit_Framework_TestListener

PHPUnit_Runner_TestSuiteLoader, 命令行测试启动器

PHPUnit_Util_Filter, 包含和排除文件, 包含及排除用于代码覆盖率的文件

插件式选择器, PHPUnit 的执行

R

S

T

U

W

附录 D. 参考书目

[Astels2003] *Test Driven Development*. David Astels. Copyright © 2003. Prentice Hall. ISBN 0131016490.

[Beck1997] *Smalltalk Best Practice Patterns*. Kent Beck. Copyright © 1997. Prentice Hall. ISBN 0-13-476904-X.

[Beck2002] *Test Driven Development by Example*. Kent Beck. Copyright © 2002. Addison-Wesley. ISBN 0-321-14653-0.

[Beck2004] *JUnit Pocket Guide*. Quick Lookup and Advice. Kent Beck. Copyright © 2004. O'Reilly Media. ISBN 0-596-00743-4.

[Bergmann2005] *Professionelle Softwareentwicklung mit PHP 5*. Objektorientierung, Entwurfsmuster, Modellierung, Fortgeschrittene Datenbankprogrammierung. Sebastian Bergmann. Copyright © 2005. dpunkt.verlag. ISBN 3-89864-229-1.

[GammaBeck1999] *JUnit: A Cook's Tour*. Erich Gamma and Kent Beck. Copyright © 1999. http://junit.sourceforge.net/doc/cookstour/cookstour.htm.

[GuBaRe2005] *PHP 5 Power Programming*. Andi Gutmans, Stig Bakken, and Derick Rethans. Copyright © 2005. Prentice Hall. ISBN 0-131-47149-X.

附录 E. 版权

所有权。你必须给出原作者的署名。

* 对于任何得再利用或分发物，你必须向他人解释本作品的许可条款。

* 如果得到作者许可，这些条件中的任何一项都可被放弃。

你的正当使用和其他权利不能以任何方式影响上述条款。

以上是下面的完整许可的摘要。

==========

Creative Commons Legal Code
Attribution 2.0

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. THE LICENSOR GRANTS
YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF
SUCH TERMS AND CONDITIONS.

1. Definitions

a. "Collective Work" means a work, such as a periodical issue,
anthology or encyclopedia, in which the Work in its entirety in
unmodified form, along with a number of other contributions,
constituting separate and independent works in themselves, are

assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

c. "Licensor" means the individual or entity that offers the Work

under the terms of this License.

d. "Original Author" means the individual or entity who created the Work.

e. "Work" means the copyrightable work of authorship offered under the terms of this License.

f. "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.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner 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
Collective Works, and to reproduce the Work as incorporated in
the Collective Works;

b. to create and reproduce Derivative Works;

c. to distribute copies or phonorecords of, display publicly,
perform publicly, and perform publicly by means of a digital
audio transmission the Work including as incorporated in
Collective Works;

d. to distribute copies or phonorecords of, display publicly,
perform publicly, and perform publicly by means of a digital
audio transmission Derivative Works.

e.

For the avoidance of doubt, where the work is a musical composition:

i. Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.

ii. Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).

f. Webcasting Rights and Statutory Royalties. For the avoidance of

doubt, where the Work is a sound recording, Licensor waives the
exclusive right to collect, whether individually or via a
performance-rights society (e.g. SoundExchange), royalties for
the public digital performance (e.g. webcast) of the Work,
subject to the compulsory license created by 17 USC Section 114
of the US Copyright Act (or the equivalent in other
jurisdictions).

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. 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, publicly display, publicly perform, or

publicly digitally perform the Work only under the terms of this
License, and You must include a copy of, or the Uniform Resource
Identifier for, this License with every copy or phonorecord of
the Work You distribute, publicly display, publicly perform, or
publicly digitally perform. You may not offer or impose any
terms on the Work that alter or restrict the terms of this
License or the recipients' exercise of the rights granted
hereunder. You may not sublicense the Work. You must keep intact
all notices that refer to this License and to the disclaimer of
warranties. You may not distribute, publicly display, publicly
perform, or publicly digitally perform the Work with any
technological measures that control access or use of the Work in
a manner inconsistent with the terms of this License
Agreement. The above applies to the Work as incorporated in a
Collective Work, but this does not require the Collective Work
apart from the Work itself to be made subject to the terms of

this License. If You create a Collective Work, upon notice from
any Licensor You must, to the extent practicable, remove from
the Collective Work any reference to such Licensor or the
Original Author, as requested. If You create a Derivative Work,
upon notice from any Licensor You must, to the extent
practicable, remove from the Derivative Work any reference to
such Licensor or the Original Author, as requested.

b. If you distribute, publicly display, publicly perform, or
publicly digitally perform the Work or any Derivative Works or
Collective Works, You must keep intact all copyright notices for
the Work and give the Original Author credit reasonable to the
medium or means You are utilizing by conveying the name (or
pseudonym if applicable) of the Original Author if supplied; the
title of the Work if supplied; to the extent reasonably
practicable, the Uniform Resource Identifier, 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 in the case of a Derivative Work,

a credit identifying the use of the Work in the Derivative Work

(e.g., "French translation of the Work by Original Author," or

"Screenplay based on original Work by Original Author"). Such

credit may be implemented in any reasonable manner; provided,

however, that in the case of a Derivative Work or Collective

Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as

prominent as such other comparable authorship credit.

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 Derivative
Works or Collective Works 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 digitally perform the Work or a Collective Work, 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 digitally perform a Derivative Work, 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.

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

==========
==========