Simple Test for PHP

Новости: Начался цикл релиза 1.0.1. Функции включают загрузку включаемых файлов, улучшенную поддержку PHP5 для фиктивных объектов и поддержку HTML-меток. Также проведена большая очистка имен методов и некоторый внутренний рефакторинг.

Далее предполагается, что вы знакомы с концепцией модульного тестирования, а также с языком веб-разработки PHP. Это руководство для нетерпеливого нового пользователя SimpleTest . Для более полной документации, особенно если вы новичок в модульном тестировании, см. текущую документацию , а для примеров тестовых случаев см. учебник по модульному тестированию .

Быстрое использование тестера

Среди инструментов тестирования программного обеспечения модульный тестер является наиболее близким к разработчику. В контексте гибкой разработки тестовый код находится прямо рядом с исходным кодом, поскольку оба пишутся одновременно. В этом контексте SimpleTest стремится стать полным решением для тестирования PHP-разработчиков и называется «Simple», потому что он должен быть простым в использовании и расширении. На самом деле это был не очень удачный выбор названия. Он включает в себя все типичные функции, которые можно ожидать от JUnit и портов PHPUnit , но также добавляет фиктивные объекты . Он также имеет некоторую функциональность JWebUnit . Сюда входит навигация по веб-страницам, тестирование cookie-файлов и отправка форм.

Самый быстрый способ продемонстрировать это — привести пример.

Предположим, мы тестируем простой класс регистрации файла, который называется Log in classes/log.php . Начнем с создания тестового скрипта, который назовем tests/log_test.php и заполним его следующим образом...

```
<?php
require_once('simpletest/unit_tester.php');
require_once('simpletest/reporter.php');
require_once('../classes/log.php');

class TestOfLogging extends UnitTestCase {
}
?>
```

Здесь папка *simpletest* либо локальная, либо находится в пути. Вам придется отредактировать эти расположения в зависимости от того, где вы разместили набор инструментов. <color #22b14c>TestOfLogging</color> — наш первый тестовый случай, и в настоящее время он пуст. Теперь у нас есть пять строк кода scaffolding и все еще нет тестов. Однако с этой части мы получаем возврат наших инвестиций очень быстро. Предположим, что класс <color #22b14c>Log</color> принимает имя файла для записи в конструкторе, и у нас есть временная папка, в которую можно поместить этот файл...

```
<?php
require_once('simpletest/unit_tester.php');
require_once('simpletest/reporter.php');
require_once('../classes/log.php');

class TestOfLogging extends UnitTestCase {

    function testCreatingNewFile() {
        @unlink('/temp/test.log');
        $log = new Log('/temp/test.log');
        $this->assertFalse(file_exists('/temp/test.log'));
        $log->message('Should write this to a file');
        $this->assertTrue(file_exists('/temp/test.log'));
    }
}
?>
```

При запуске тестового случая он ищет любой метод, начинающийся со строки <color #22b14c>test</color>, и выполняет этот метод. Обычно у нас, конечно, больше одного тестового метода. Утверждения в тестовых методах запускают сообщения в тестовую среду, которая немедленно отображает результат. Этот немедленный ответ важен не только в случае, если код вызывает сбой, но и для того, чтобы операторы <color #22b14c>print</color> могли отображать свое содержимое прямо рядом с соответствующим тестовым случаем.

Чтобы увидеть эти результаты, нам нужно запустить тесты. Если это единственный тестовый случай, который мы хотим запустить, мы можем добиться этого с помощью...

```
<?php
require_once('simpletest/unit_tester.php');
require_once('simpletest/reporter.php');
require_once('../classes/log.php');

class TestOfLogging extends UnitTestCase {

    function testCreatingNewFile() {
        @unlink('/temp/test.log');
        $log = new Log('/temp/test.log');
        $this->assertFalse(file_exists('/temp/test.log'));
        $log->message('Should write this to a file');
        $this->assertTrue(file_exists('/temp/test.log'));
    }
}

$test = &new TestOfLogging();
$test->run(new HtmlReporter());
?>
```

В случае сбоя дисплей выглядит так...

testoflogging
Fail: testcreatingnewfile→True assertion failed.
<color white/red>1/1 test cases complete. 1 passes and 1 fails.</color>

...и если это пройдет вот так...

testoflogging
<color white/green>1/1 test cases complete. 2 passes and 0 fails.</color>

И если вы это получите...

```
Fatal error: Failed opening required '../classes/log.php' (include_path='')
in
/home/marcus/projects/lastcraft/tutorial_tests/Log/tests/log_test.php on
line 7
```

это означает, что у вас отсутствует файл *classes/Log.php*, который может выглядеть так...

```
<?php
class Log {

        function Log($file_path) {
        }

        function message() {
        }
}
?>;
```

Групповые тесты

Маловероятно, что в реальном приложении мы когда-либо запустим только один тестовый случай. Это означает, что нам нужен способ группировки случаев в тестовый сценарий, который может, при необходимости, запустить каждый тест в приложении.

Наш первый шаг — удалить включения и отменить наш предыдущий взлом...

```
<?php
require_once('../classes/log.php');

class TestOfLogging extends UnitTestCase {

    function testCreatingNewFile() {
        @unlink('/temp/test.log');
        $log = new Log('/temp/test.log');
        $this->assertFalse(file_exists('/temp/test.log'));
```

```
        $log->message('Should write this to a file');
        $this->assertTrue(file_exists('/temp/test.log'));
    }
}
?>
```

Далее мы создаем новый файл с именем `tests/all_tests.php` и вставляем следующий код...

```
<?php
require_once('simpletest/unit_tester.php');
require_once('simpletest/reporter.php');

$test = &new GroupTest('All tests');
$test->addTestFile('log_test.php');
$test->run(new HtmlReporter());
?>
```

Метод <color #22b14c>GroupTest::addTestFile()</color> включит файл тестового случая и прочитает все новые созданные классы, которые являются потомками <color #22b14c>SimpleTestCase</color>, одним из примеров которых является <color #22b14c>UnitTestCase</color>. На данный момент сохраняются только имена классов, чтобы тестовый исполнитель мог создать экземпляр класса, когда он будет проходить через ваш тестовый набор. Чтобы это работало правильно, файл тестового случая не должен слепо включать какие-либо другие расширения тестового случая, которые на самом деле не запускают тесты. Это может привести к подсчету дополнительных тестовых случаев во время тестового прогона. Вряд ли это серьезная проблема, но чтобы избежать этого неудобства, просто добавьте директиву <color #22b14c>SimpleTestOptions::ignore()</color> где-нибудь в файле тестового случая. Кроме того, файл тестового случая не должен был быть включен в другом месте, иначе в этот групповой тест не будут добавлены случаи. Это будет более серьезной ошибкой, так как если классы тестового случая уже загружены PHP, метод <color #22b14c>GroupTest::addTestFile()</color> не обнаружит их.

Для отображения результатов необходимо только вызвать *tests/all_tests.php* с веб-сервера.

Использование фиктивных объектов

Давайте перенесемся еще дальше в будущее.

Предположим, что наш класс логирования протестирован и завершен. Предположим также, что мы тестируем другой класс, который требуется для записи сообщений журнала, скажем, <color #22b14c>SessionPool</color>. Мы хотим протестировать метод, который, вероятно, в конечном итоге будет выглядеть так...

```
class SessionPool {
    ...
    function logIn($username) {
        ...
        $this->_log->message("User $username logged in.");
```

```
        ...
    }
    ...
}
```

В духе повторного использования мы используем наш класс <color #22b14c>Log</color>. Обычный тестовый случай может выглядеть так...

```
<?php
require_once('../classes/log.php');
require_once('../classes/session_pool.php');

class TestOfSessionLogging extends UnitTestCase {

    function setUp() {
        @unlink('/temp/test.log');
    }

    function tearDown() {
        @unlink('/temp/test.log');
    }

    function testLogInIsLogged() {
        $log = new Log('/temp/test.log');
        $session_pool = &new SessionPool($log);
        $session_pool->logIn('fred');
        $messages = file('/temp/test.log');
        $this->assertEqual($messages[0], "User fred logged in.\n");
    }
}
?>
```

Этот дизайн тестового случая не совсем плох, но его можно улучшить. Мы тратим время на возню с файлами журналов, которые не являются частью нашего теста. Хуже того, мы создали тесные связи между классом Log и этим тестом. Что, если мы больше не будем использовать файлы, а вместо этого будем использовать библиотеку syslog ? Вы заметили дополнительный возврат каретки в сообщении? Это было добавлено регистратором? Что, если он также добавил временную метку или другие данные? Единственная часть, которую мы действительно хотим проверить, — это то, что определенное сообщение было отправлено в регистратор. Мы уменьшаем связанность, если можем передать фальшивый класс регистрации, который просто записывает вызовы сообщения для тестирования, но не предпринимает никаких действий. Однако он должен выглядеть точно так же, как наш оригинал.

Если поддельный объект не пишет в файл, то мы экономим на удалении файла до и после каждого теста. Мы могли бы сэкономить еще больше тестового кода, если бы поддельный объект любезно выполнил утверждение для нас.

Слишком хорошо, чтобы быть правдой? К счастью, мы можем легко создать такой объект...

```
<?php
```

```
require_once('../classes/log.php');
require_once('../classes/session_pool.php');
Mock::generate('Log');

class TestOfSessionLogging extends UnitTestCase {

    function testLogInIsLogged() {
        $log = &new MockLog();
        $log->expectOnce('message', array('User fred logged in.'));
        $session_pool = &new SessionPool($log);
        $session_pool->logIn('fred');
    }
}
?>
```

Тест будет запущен, когда вызов <color #22b14c>message()</color> будет вызван на объекте <color #22b14c>MockLog</color>. Вызов mock запустит сравнение параметров, а затем отправит результирующее событие pass или fail на дисплей теста. Здесь также можно включить подстановочные знаки, чтобы тесты не стали слишком конкретными. Если мок достигает конца тестового случая без вызова метода, ожидание <color #22b14c>expectOnce()</color> вызовет сбой теста. Другими словами, мок может обнаружить как отсутствие поведения, так и его наличие.

Для фиктивных объектов в наборе SimpleTest можно задать произвольные возвращаемые значения, последовательности возвращаемых значений, возвращаемые значения, выбранные в соответствии с входящими аргументами, последовательности ожидаемых параметров и ограничения на количество вызовов метода.

Для запуска этого теста библиотека фиктивных объектов должна быть включена в тестовый набор, например, в *all_tests.php*.

Тестирование веб-страниц

Одним из требований к веб-сайтам является то, что они производят веб-страницы. Если вы создаете проект сверху вниз и хотите полностью интегрировать тестирование по ходу дела, то вам понадобится способ автоматической навигации по сайту и проверки вывода на корректность. Это работа веб-тестера.

Веб-тестирование в SimpleTest довольно примитивно, например, нет JavaScript. Чтобы дать представление, вот тривиальный пример, где извлекается домашняя страница, с которой мы переходим на страницу «about», а затем тестируем некоторый определенный клиентом контент.

```
<?php
require_once('simpletest/web_tester.php');
require_once('simpletest/reporter.php');

class TestOfAbout extends WebTestCase {
```

```
    function setUp() {
        $this->get('http://test-server/index.php');
        $this->click('About');
    }

    function testSearchEngineOptimisations() {
        $this->assertTitle('A long title about us for search engines');
        $this->assertPattern('/a popular keyphrase/i');
    }
}
$test = &new TestOfAbout();
$test->run(new HtmlReporter());
?>
```

Используя этот код в качестве приемочного теста, вы можете быть уверены, что контент всегда соответствует спецификациям как разработчиков, так и других заинтересованных сторон проекта.

Дополнения и Файлы

- Ссылка на оригинальную статью
- simpletest.sourceforge.net
- github_simpletest

- SimpleTest