使用Symfony2的组件创建自己的PHP框架（第八部分：用PHPUnit做单元测试）

3条回复

英文原文地址：http://fabien.potencier.org/article/57/create-your-own-framework-on-top-of-the-symfony2-components-part-8

一些细心的读者已经发现，前一章完成的框架里还存在难以发现却很严重的bug。创建框架，你必须得保证它能像当初设计的那样工作，否则的话使用它做出来的程序都会出现它导致的问题。当然好消息是：如果你修改了一个bug，你就等于修改了一大堆（译者加：使用它的）的应用程序。

今天我们的任务是，利用PHPUnit来为我们的框架写一些单元测试。首先创建一个PHPUnit的配置文件`example.com/phpunit.xml.dist`:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <phpunit backupGlobals="false"
4           backupStaticAttributes="false"
5           colors="true"
6           convertErrorsToExceptions="true"
7           convertNoticesToExceptions="true"
8           convertWarningsToExceptions="true"
9           processIsolation="false"
10          stopOnFailure="false"
11          syntaxCheck="false"
12          bootstrap="vendor/.composer/autoload.php"
13 >
14      <testsuites&gt;
15          <testsuite name="Test Suite">
16              <directory>./tests</directory>
17          </testsuite>
18      </testsuites>
19 </phpunit>
20
```

此配置定义了PHPUnit大部分设置很好的默认选项；更有趣的是，自动加载器将会用来启动测试，而所有的测试都将放在`example.com/tests/`目录。

现在，让我们写一个“资源无法找到”的响应的测试。为了防止要测试的对象的依赖组件对测试的影响，我们打算使用test doubles（译者注：对于test doubles我自己的理解是，在测试某一个组件的时候，有可能这个组件需要其他组件才能工作，比如方向盘。当然为了测试方向盘能否正常工作，不见得非要把方向盘装车上，若有专门的仪器可以模拟真实的环境也可以测试）。如果我们的组件都依赖于接口而不是实际的类，我们将很容易使用模拟环境（test doubles）来做测试。还好Symfony2的核心组件都提供了这样的接口，比如url匹配器以及控制器解析器。让我们修改一下框架来使用这些接口：

```
1  <?php
2
3  // example.com/src/Simplex/Framework.php
4
5  namespace Simplex;
6
7  // ...
8
9  use Symfony\Component\Routing\Matcher\UrlMatcherInterface;
10 use Symfony\Component\HttpKernel\Controller\ControllerResolverInte
11
12 class Framework
13 {
14     protected $matcher;
15     protected $resolver;
16
17     public function __construct(
18         UrlMatcherInterface $matcher,
19         ControllerResolverInterface $resolver
20     ) {
21         $this->matcher = $matcher;
22         $this->resolver = $resolver;
23     }
24
25     // ...
26 }
27
```

我们现在便可以写我们的第一个测试了：

```
1  <?php
2
3  // example.com/tests/Simplex/Tests/FrameworkTest.php
```

```
4
5  namespace Simplex\Tests;
6
7  use Simplex\Framework;
8  use Symfony\Component\HttpFoundation\Request;
9  use Symfony\Component\Routing\Exception\ResourceNotFoundException;
10
11 class FrameworkTest extends \PHPUnit_Framework_TestCase
12 {
13     public function testNotFoundHandling()
14     {
15         $framework = $this->getFrameworkForException(new ResourceN
16
17         $response = $framework->handle(new Request());
18
19         $this->assertEquals(404, $response->getStatusCode());
20     }
21
22     protected function getFrameworkForException($exception)
23     {
24         $matcher = $this->getMock('Symfony\Component\Routing\Match
25         $matcher
26             ->expects($this->once())
27             ->method('match')
28             ->will($this->throwException($exception))
29         ;
30         $resolver = $this->getMock(
31             'Symfony\Component\HttpKernel\Controller\ControllerRes
32         );
33
34         return new Framework($matcher, $resolver);
35     }
36 }
37
```

此测试模拟了一个不匹配任何路由规则的请求，接着`match()`方法将产生`ResourceNotFoundException`的意外。我们将测试我们的框架类能否将此意外转变成一个404相应。

执行测试程序非常简单:

```
$ phpunit

```

我将不会详细解释代码执行的细节，因为这不是本章的重点，但如果你实在被这些代码搞的崩溃，我强烈推荐你们看看*PHPUnit*关于*test doubles*的文档

执行完上面的命令，你应该能看到一条绿色背景提示（在windows下面应该是看不到的，绿色背景的提示表示测试很OK），如果没有，说明你的框架代码可能有问题哦。

添加测试控制器抛出的异常的代码也是so easy的事情：

```
public function testErrorHandling()
{
    $framework = $this->getFrameworkForException(new \RuntimeExcept

    $response = $framework->handle(new Request());

    $this->assertEquals(500, $response->getStatusCode());
}

```

最后很重要的一步，添加一个返回正常相应的测试：

```
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpKernel\Controller\ControllerResolver;

public function testControllerResponse()
{
    $matcher = $this->getMock('Symfony\Component\Routing\Matcher\U
    $matcher
        ->expects($this->once())
        ->method('match')
        ->will($this->returnValue(array(
            '_route' => 'foo',
            'name' => 'Fabien',
            '_controller' => function ($name) {
                return new Response('Hello '.$name);
            }
        )))
```

```
17      ;
18      $resolver = new ControllerResolver();
19
20      $framework = new Framework($matcher, $resolver);
21
22      $response = $framework->handle(new Request());
23
24      $this->assertEquals(200, $response->getStatusCode());
25      $this->assertContains('Hello Fabien', $response->getContent())
26  }
27
```

在此测试中，我们模拟了一个能匹配的路由，并返回了一个简单的控制器。我们测试了框架返回的相应代码是否200，并且相应内容是否是我们期待的那样工作。

要查看此测试代码是否测试了所有的运行事例（译者注：也就是代码测试率，即测试运行过的代码行数占整个代码行数的比率），请使用PHPUnit的代码测试率功能（你需要安装php的XDebug插件）：

```
1  $ phpunit --coverage-html=cov/
2
```

在浏览器中打开`example.com/cov/src_Simplex_Framework.php.html`，检查所有关于框架类的代码是不是都是绿色的（绿色的代码表示这些它们已经被测试过了）

得益于我们写的面向对象的代码，我们能够写出能测试全部使用事例的框架测试代码。test doubles测试可以保证我们测试的是只自己的代码而不是Symfony2的。

现在我们（又一次）对自己的代码充满信心，我们可以安心的考虑我们接下来要加什么功能了