使用Symfony2的组件创建自己的PHP框架（第二部分：接受请求与生成响应）

发表回复

英文原文地址：http://fabien.potencier.org/article/51/create-your-own-framework-on-top-of-the-symfony2-components-part-2

在开始重构我们的代码之前，我打算先再谈谈为什么您最好使用一个框架来替代用PHP直接书写这种方式来创建一个应用程序。即使写一个很小的代码片段，使用框架也是一个好主意，而使用sf2组件库来创建一个框架比直接写一个框架更好。

这里我不会讨论在有多个开发者共同开发的大项目中，那些经典的或者明显的使用框架的好处。网上已经有很多关于这个话题的资源信息。

即使昨天我们写的“应用程序”非常的简单，但也会遇到很多问题：

```
1 <?php
2
3 // framework/index.php
4
5 $input = $_GET['name'];
6
7 printf('Hello %s', $input);
8
```

首先，如果URL里面没有GET参数，PHP会抛出一个警告，让我们修正它：

```
1 <?php
2
3 // framework/index.php
4
5 $input = isset($_GET['name']) ? $_GET['name'] : 'World';
6
7 printf('Hello %s', $input);
8
```

其次，这段代码并不安全。你能相信吗？即使是这么简单的代码也有普遍的网络安全问题XSS（跨域脚本攻击，译者注：即攻击者构造一个name参数带有javascript脚本内容的URL让另外一个人访问，网页打开后脚本就会在被攻击人的浏览器上运行）而易被攻击。下面是一个更安全的版本：

```
1 <?php
2
3 $input = isset($_GET['name']) ? $_GET['name'] : 'World';
4
5 header('Content-Type: text/html; charset=utf-8');
6
7 printf('Hello %s', htmlspecialchars($input, ENT_QUOTES, 'UTF-8'));
8
```

正如你所见，使用*htmlspecialchars*来加强安全性的话，既感觉很烦又容易疏漏，这便是为什么要使用类似*Twig*等模版引擎的原因，它的自动转义功能是默认开启的。即使关闭自动转义，使用自带*e*方法也会相对使用*htmlspecialchars*方便不少

正如你所见到的那样，为了不出现PHP警告以及加强安全性，我们的代码已经不像当初那样简单了。

除了安全性，这段代码也很难测试。虽然这段代码也没什么好测试的，但即使是这么简单的代码，我也觉得很难优雅而自然的进行测试。这是我利用PHPUnit来写的一个测试：

```
1  <?php
2
3  // framework/test.php
4
5  class IndexTest extends \PHPUnit_Framework_TestCase
6  {
7      public function testHello()
8      {
9          $_GET['name'] = 'Fabien';
10
11         ob_start();
12         include 'index.php';
13         $content = ob_get_clean();
14
15         $this->assertEquals('Hello Fabien', $content);
16     }
17 }
```

```
18
```

如果我们的项目再大一些，我们还会发现更多的问题。如果你非常想知道还有哪些问题，请阅读*sf2*文档的《*sf2 VS* 纯*PHP*》部分

从这几点来看，如果你还觉得安全性和可测性不是使用框架的好理由，你可以不用继续往下阅读了。

当然，使用框架给你带来的不只是安全性和可测试性。请记住你选择的框架必须能够让你更快的写出更好的代码来

使用**HttpFoundation**组件迈向面向对象之路

写web程序都是有关HTTP交互的，所以我们写框架的基本原则都是围绕着《HTTP协议》展开的

HTTP协议描述了客户端（比如说浏览器）和服务器端（比如网络服务器）的交互方式。客户端和服务器端的对话是通过良好定义的“消息”——即请求和响应的传递来实现的。客户端向服务器发起一个请求，服务器根据这个请求再向客户端返回一个响应

在php的世界里，请求由全局变量（$_GET,$_POST,$_FILE等）来表示，响应又由一些函数来生成（echo,header,setcookie等，译者注：echo严格意义上来说不是一个函数，不过你懂作者的意思就行了）

写出更好代码的第一步最好是使用面向对象来实现。这是sf2的`HttpFoundation`组件的主要目的：利用面向对象层来代替PHP默认使用的全局变量。

要使用此组件，打开composer.js然后修改代码为下面内容：

```
1 // framework/composer.json
2 {
3     "require": {
4         "symfony/class-loader": "2.1.*",
5         "symfony/http-foundation": "2.1.*"
6     }
7 }
8
```

然后运行Composer的update命令：

```
$ php composer.phar update

```

最后，在autoloader.php文件的最后添加以下代码，让HttpFoundation组件能自动加载：

```
<?php

// framework/autoload.php

$loader->registerNamespace(
    'Symfony\\Component\\HttpFoundation',
    __DIR__.'/vendor/symfony/http-foundation'
);

```

现在，让我们用请求类（Request class）和响应类（Response class）来重写我们的程序：

```
<php

// framework/index.php

require_once __DIR__.'/autoload.php';

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

$request = Request::createFromGlobals();

$input = $request->get('name', 'World');

$response = new Response(sprintf(
    'Hello %s',
    htmlspecialchars($input, ENT_QUOTES, 'UTF-8')
));

$response->send();

```

createFromGlobals方法会根据当前PHP的全局变量生成一个Request对象。

send方法返回Response对象里的内容到客户端（他会先根据内容发送HTTP头信息）。

其实在调用send方法之前，我们应该先调用一下prepare方法（$response->prepare($request)）来确认发送一个与标准http协议兼容的相应。举个例子，如果客户端对一个页面发送HEAD请求，那么服务器是不应该返回响应正文的（译者注：HEAD请求类似GET请求，除了要求服务器只返回头信息）。

对比之前的代码，你现在能完全控制HTTP消息。你能创建任何任何你想创建的请求，你也能发送任何你想返回的响应。

我们并没有显式设置*Content-type*为*UTF-8*，因为*Response*对象默认发送的就是*UTF-8*编码的内容。

得益于优雅而简单的API，你能很轻易的通过Request对象获得请求的各种信息：

```php
<?php

// 请求的URI (e.g. /about)
$request->getPathInfo();

// 分别得到GET参数或POST参数
$request->query->get('foo'); // GET
$request->request->get('bar', '如何没有bar的默认值'); // POST

// 得到服务器变量
$request->server->get('HTTP_HOST');

// 得到上传文件对象
$request->files->get('foo');

// 得到cookie值
$request->cookies->get('PHPSESSID');

// 得到http请求头信息
$request->headers->get('host');
$request->headers->get('content_type');

$request->getMethod();    // GET, POST, PUT, DELETE, HEAD
$request->getLanguages(); // 得到客户端接收语言数组

```

你也能够模拟一个请求：

```
1 $request = Request::create('/index.php?name=Fabien');
2
```

你也能利用Response对象很轻易的改变响应：

```
1  <?php
2
3  $response = new Response();
4
5  $response->setContent('Hello world!');
6  $response->setStatusCode(200);
7  $response->headers->set('Content-Type', 'text/html');
8
9  // configure the HTTP cache headers
10 $response->setMaxAge(10);
11
```

要*debug*一个相应对象，使用*(string)*让其变成一个字符串，他将返回*HTTP*响应的全部信息（包括头信息和正文）

最后也是最重要的一点，这些类以及sf2组件库其他的类，安全性方面都被一家专门做安全评审的公司评估过（译者注：这里说的公司应该是指SektionEins，一家德国专门做安全评估的公司），而且作为开源项目就意味着许多志愿程序员已经将许多潜在的安全问题解决了。你上一次给你的框架做专业的安全评估是什么时候？（译者注：意思是说一般人自己在家鼓捣的框架不可能去搞什么专业的安全评估的）

即使像获取用户IP这么简单的操作，也会出现安全隐患：

```
1 <?php
2
3 if ($myIp == $_SERVER['REMOTE_ADDR']) {
4     // 被信任的IP! 给一些权限!
5 }
6
```

当你在你的投产web服务器前加了反相代理的时候，你会发现这段代码不会正常的工作。为了让你的代码能够在你的投产服务器和开发服务器都能够使用，你需要把代码改成下面这个样子：

```php
<?php

if ($myIp == $_SERVER['HTTP_X_FORWARDED_FOR']
    || $myIp == $_SERVER['REMOTE_ADDR']) {
    // 被信任的IP！给一些权限！
}

```

而使用`Request::getClientIp()`方法将让你始终可以得到正确的用户IP（无论请求过来之前经过了多少代理服务器）

```php
<?php

$request = Request::createFromGlobals();

if ($myIp == $request->getClientIp()) {
    // 被信任的IP！给一些权限！
}

```

另外还有一个好处：默认情况下它很安全，这里的安全我指的是：`$_SERVER['HTTP_X_FORWARDED_FOR']`这个参数并不可信，因为在没有代理服务器的情况下，终端用户是可以模仿出这么一个值的。所以，如果你在投产机里使用了这个参数，那么是很容易遭到黑客攻击而让你的服务器被滥用。即使是`getClientIp`方法也并非那么安全，为了解决刚才所说的情况，你需要显式调用`trustProxyData()`方法来处理：

```php
<?php

Request::trustProxyData();

if ($myIp == $request->getClientIp(true)) {
    // 被信任的IP！给一些权限！
}

```

此刻`getClientIp`方法在各种情况下都会安全的工作，你能在你所有的项目里使用它，无论你用什么样的服务器配置，他都会安全而正确的工作。这便是使用框架的目的之一，如果你要从头到尾自己琢磨一个框架，这些问题你都得自己去考虑。所以为何不使用一个已经好用的技术呢？

如果你想知道关于*HttpFoundation*组件更多的信息，你可以查看它的*API*，或者查看*sf2*网站上写得非常专业的文档

不管你信不信，我们已经写好我们的第一个框架了。如果你愿意的话你也可不必再继续往下阅读，因为使用sf2的HttpFoundation组件已经能让你写出更好更适合测试的代码了，另外它也能让你的代码写得更快，因为那些每次开发都需要注意的问题它都已经帮你解决了。

事实上，像Drupal等一些项目已经开始采用HttpFoundation组件了（从第8个版本开始）。如果这个组件能在这些项目里正常运行，那么它也可以在你的项目里正常运行。所以.....不要重复发明轮子。

我差点忘了还有一个使用HttpFoundation组件的好处：它会让你在那些使用过它的项目和框架中有更好的互相操作性(interoperability)（目前这样的项目有Symfony2, Drupal 8, phpBB 4, Silex, Midgard CMS, Zikula ...）