使用Symfony2的组件创建自己的PHP框架（第三部分：前端控制器）

发表回复

英文原文地址：http://fabien.potencier.org/article/52/create-your-own-framework-on-top-of-the-symfony2-components-part-3

直到现在，我们的应用程序还只有一个页面，非常简单。为了再增加一点点乐趣，我们再添加一个“再会”页面。

```
1  <?php
2
3  // framework/bye.php
4
5  require_once __DIR__.'/autoload.php';
6
7  use Symfony\Component\HttpFoundation\Request;
8  use Symfony\Component\HttpFoundation\Response;
9
10 $request = Request::createFromGlobals();
11
12 $response = new Response('Goodbye!');
13 $response->send();
14
```

如你所见，这段代码跟我们第一个页面的代码写法都差不多。让我们把相同的代码提取出来，这样我们就可以让这段代码在所有要创建的页面里共享了。对于创建我们“真正的”框架来说，代码共享听起来是个不错的想法！

以PHP的方式来完成上面目标的重构，就是创建一个包含文件：

```
1  <?php
2
3  // framework/init.php
4
5  require_once __DIR__.'/autoload.php';
6
```

```
7  use Symfony\Component\HttpFoundation\Request;
8  use Symfony\Component\HttpFoundation\Response;
9
10 $request = Request::createFromGlobals();
11 $response = new Response();
12
```

首页改成：

```
1  <?php
2
3  // framework/index.php
4
5  require_once __DIR__.'/init.php';
6
7  $input = $request->get('name', 'World');
8
9  $response->setContent(sprintf(
10     'Hello %s',
11     htmlspecialchars($input, ENT_QUOTES, 'UTF-8')
12 ));
13 $response->send();
14
```

以及我们的“再会”页：

```
1  <?php
2
3  // framework/bye.php
4
5  require_once __DIR__.'/init.php';
6
7  $response->setContent('Goodbye!');
8  $response->send();
9
```

这样我们已将公共代码挪到了单独的文件，但这样做感觉上还是不够好，是吧？首先，我们每个页面代码依然还存在共同的send方法，其次我们的页面代码任然看起来不像模版代码，而且我们依然不能很好的测试这段代码。

再其次，添加一个新的页面，意味着我们要添加一个新的PHP脚本文件，并且此文件是通过URL完全暴露给终端用户的（http://example.com/bye.php），PHP脚本文件名和客户端URL是完全直接相互映射的，这是因为请求的调度工作完全是由web服务器直接完成的。但为了灵活性，把调度工作直接交给我们自己的代码来做似乎是一个更好的想法。将所有的请求交给一个php脚本来做路由处理的话，这个想法是很容易实现的。

只暴露一个脚本给终端用户这种方式，是一种叫“前段控制器*(front controller)*”的设计模式

这个文件如下所示：

```
1  <?php
2
3  // framework/front.php
4
5  require_once __DIR__.'/autoload.php';
6
7  use Symfony\Component\HttpFoundation\Request;
8  use Symfony\Component\HttpFoundation\Response;
9
10 $request = Request::createFromGlobals();
11 $response = new Response();
12
13 $map = array(
14     '/hello' => __DIR__.'/hello.php',
15     '/bye'   => __DIR__.'/bye.php',
16 );
17
18 $path = $request->getPathInfo();
19 if (isset($map[$path])) {
20     require $map[$path];
21 } else {
22     $response->setStatusCode(404);
23     $response->setContent('Not Found');
24 }
25
26 $response->send();
27
```

经过改造后的新页面代码，举个例子，`hello.php`，应该是这个样子：

```
1 <?php
2
3 // framework/hello.php
4
5 $input = $request->get('name', 'World');
6 $response->setContent(sprintf(
7     'Hello %s', htmlspecialchars($input, ENT_QUOTES, 'UTF-8')
8 ));
9
```

在front.php文件中，$map定义了URL路径和相关PHP页面脚本路径的对应关系。

这么做还有一个好处是，如果用户输入了一个$map里未定义的URL，我们可以返回一个自定义的404页面。现在你可以用代码完全控制你的网站了

你现在必须使用`front.php`来访问一些页面：

- `http://example.com/front.php/hello?name=Fabien`
- `http://example.com/front.php/bye`

`/hello`和`/bye`是页面的路径

大部分*web*服务器，比如*Nginx*和*Apache*，都有改写*URL*的功能，可以将前段控制器脚本从*URL*里面去掉，比如可以让用户直接输入*`http://example.com/hello?name=Fabien`*来进入上面的连接，这样看起来*URL*干净了许多。

能实现上面功能的诀窍便是使用`Request::getPathInfo()`方法，此方法将返回不包括前段控制器文件名，但包含它的子目录路径的路径信息。

你甚至都不用建立一个*web*服务器来做测试，你只用将*`Request::createFromGlobals()`*方法替换成比如*`Request::create('/hello?name=Fabien')`*，你就可以模拟任何请求了。

目前所有的请求都是通过`front.php`来访问的，所以我们可以将其他PHP文件挪到web目录的外面去，以达到保护其他代码的作用。

```
1 example.com/
```

```
    composer.json
    src/
        autoload.php
        pages/
            hello.php
            bye.php
    vendor/
    web/
        front.php

```

现在，更改一下web服务器的配置，把访问的根目录设置到web目录下，这样web目录外面的代码，客户端就不能直接访问到了。

更改目录结构以后，有些包含路径是需要手工去调整的，这些就留给读者自己去修改了

最后一个被重复写过多次的代码是`Response::setContent()`方法。我们可以把所有的页面代码变成直接echo出来的模版代码，这样我们就可以在前段控制器脚本中直接调用`setContent`方法了：

```php
<?php

// example.com/web/front.php

// ...

$path = $request->getPathInfo();
if (isset($map[$path])) {
    ob_start();
    include $map[$path];
    $response->setContent(ob_get_clean());
} else {
    $response->setStatusCode(404);
    $response->setContent('Not Found');
}

// ...

```

`hello.php`也需要修改成模版页：

```
<!-- example.com/src/pages/hello.php -->

<?php $name = $request->get('name', 'World') ?>

Hello <?php echo htmlspecialchars($name, ENT_QUOTES, 'UTF-8') ?>

```

最终我们得到了我们今天的框架代码：

```
<?php

// example.com/web/front.php

require_once __DIR__.'/../src/autoload.php';

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

$request = Request::createFromGlobals();
$response = new Response();

$map = array(
    '/hello' => __DIR__.'/../src/pages/hello.php',
    '/bye'   => __DIR__.'/../src/pages/bye.php',
);

$path = $request->getPathInfo();
if (isset($map[$path])) {
    ob_start();
    include $map[$path];
    $response->setContent(ob_get_clean());
} else {
    $response->setStatusCode(404);
    $response->setContent('Not Found');
}

$response->send();

```

要在添加一个新页面，目前只需要两步完成：在`$map`变量里面添加一个映射，然后在`src/pages/`目录添加一

个模版页面。在模版文件里面，通过`$request`对象获得请求参数，然后利用`$response`变量，修改响应内容。

如果你决定在此停止阅读，你最好是把*map*的映射关系表提取出来放在单独的文件里面