# 使用Symfony2的组件创建自己的PHP框架（第七部分：封装框架代码）

10条回复

英文原文地址：http://fabien.potencier.org/article/56/create-your-own-framework-on-top-of-the-symfony2-components-part-7

目前我们的框架还有一个不足之处：当我们要创建新的网站的时候，我们都需要将front.php的代码复制一份。虽然40行代码并不是很多，但是如果我们能将这些代码写成一个合适的类，将会更给力一些，比如更好的复用性以及更好的可测试性。

更进一步研究你会发现，`front.php`包含一个输入，即一个请求Request，以及一个输出，即一个相应Response。我们的框架将遵循一个简单的原则：生成与请求对应的响应。

因为Symfony2本身PHP5.3的支持，所以我们可以为框架设置一个自己的命名空间：`Simplex`

将处理请求的逻辑代码移动到我们的框架类中：

```
1  <?php
2
3  // example.com/src/Simplex/Framework.php
4
5  namespace Simplex;
6
7  use Symfony\Component\HttpFoundation\Request;
8  use Symfony\Component\HttpFoundation\Response;
9  use Symfony\Component\Routing\Matcher\UrlMatcher;
10 use Symfony\Component\Routing\Exception\ResourceNotFoundException;
11 use Symfony\Component\HttpKernel\Controller\ControllerResolver;
12
13 class Framework
14 {
15     protected $matcher;
16     protected $resolver;
17
18     public function __construct(UrlMatcher $matcher, ControllerRes
```

```
19     {
20         $this->matcher = $matcher;
21         $this->resolver = $resolver;
22     }
23 
24     public function handle(Request $request)
25     {
26         try {
27             $request->attributes->add($this->matcher->match($reque
28 
29             $controller = $this->resolver->getController($request)
30             $arguments = $this->resolver->getArguments($request, $
31 
32             return call_user_func_array($controller, $arguments);
33         } catch (ResourceNotFoundException $e) {
34             return new Response('Not Found', 404);
35         } catch (\Exception $e) {
36             return new Response('An error occurred', 500);
37         }
38     }
39 }
40 
```

相应的我们也需要更新一下front.php的代码：

```
1  <?php
2 
3  // example.com/web/front.php
4 
5  // ...
6 
7  $request = Request::createFromGlobals();
8  $routes = include __DIR__.'/../src/app.php';
9 
10 $context = new Routing\RequestContext();
11 $context->fromRequest($request);
12 $matcher = new Routing\Matcher\UrlMatcher($routes, $context);
13 $resolver = new HttpKernel\Controller\ControllerResolver();
14 
15 $framework = new Simplex\Framework($matcher, $resolver);
16 $response = $framework->handle($request);
```

```
17
18 $response->send();
19
```

让我们把除了路由定义的代码挪到另外一个命名空间Calendar下，以完成我们对重构代码的继续封装：

为了让在Simplex以及Calendar这两个命名空间下的代码文件能够自动加载，我们需要更新一下composer.json文件：

```
1  {
2      "require": {
3          "symfony/class-loader": "2.1.*",
4          "symfony/http-foundation": "2.1.*",
5          "symfony/routing": "2.1.*",
6          "symfony/http-kernel": "2.1.*"
7      },
8      "autoload": {
9          "psr-0": { "Simplex": "src/", "Calendar": "src/" }
10     }
11 }
12
```

为了让自动加载生效，需要运行

```
1 $ php composer.phar update
2
```

将控制器代码挪到Calendar\Controller\LeapYearController：

```
1  <?php
2
3  // example.com/src/Calendar/Controller/LeapYearController.php
4
5  namespace Calendar\Controller;
6
7  use Symfony\Component\HttpFoundation\Request;
8  use Symfony\Component\HttpFoundation\Response;
9  use Calendar\Model\LeapYear;
```

```
10
11 class LeapYearController
12 {
13     public function indexAction(Request $request, $year)
14     {
15         $leapyear = new LeapYear();
16         if ($leapyear->isLeapYear($year)) {
17             return new Response('Yep, this is a leap year!');
18         }
19
20         return new Response('Nope, this is not a leap year.');
21     }
22 }
23
```

然后把`is_leap_year()`方法挪到它应该存在的类中：

```
1  <?php
2
3  // example.com/src/Calendar/Model/LeapYear.php
4
5  namespace Calendar\Model;
6
7  class LeapYear
8  {
9      public function isLeapYear($year = null)
10     {
11         if (null === $year) {
12             $year = date('Y');
13         }
14
15         return 0 == $year % 400 || (0 == $year % 4 && 0 != $year %
16     }
17 }
18
```

别忘了`example.com/src/app.php`也需要在相应的地方做下更新：

```
1  $routes->add('leap_year', new Routing\Route('/is_leap_year/{year}',
```

```
    'year' => null,
    '_controller' => 'Calendar\\Controller\\LeapYearController::inc
)));

```

最终，我们得到一个新的目录结构：

```
example.com/
    composer.json
    src/
        app.php
        Simplex/
            Framework.php
        Calendar/
            Controller/
                LeapYearController.php
            Model/
                LeapYear.php
    vendor/
    web/
        front.php

```

没错！我们的应用程序目前有4个不同的部分，而且每一个部分都有自己特有的责任：

- `web/front.php`：前段控制器，这是唯一一处没有被封装的php代码文件，并且唯一与客户端交互的接口（它接受请求并发送响应），并且提供了初始化框架的代码模版（boil-plate，译者注：这个单词很有意思，来源于印刷工业，指的是不能拆卸的一整块印刷母板，你可以想成是活字印刷发明之前用来印书的东西）
- `src/Simplex`：可供复用的框架代码，作为处理请求的抽象化接口（另外，他使你的控制器/模板文件更容易测试，更多信息请看下一章）
- `src/Calendar`：我们程序的特定代码（译者注：指一个程序的具体实现）
- `src/app.php`：程序配置/框架自定义