# 使用Symfony2的组件创建自己的PHP框架（第四部分：使用路由组件）

5条回复

英文原文地址：http://fabien.potencier.org/article/53/create-your-own-framework-on-top-of-the-symfony2-components-part-4

在开始我们今天的话题前，我们先重构一下我们的框架，让我们的模板文件更加易读：

```php
<?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();

$map = array(
    '/hello' => 'hello',
    '/bye'   => 'bye',
);

$path = $request->getPathInfo();
if (isset($map[$path])) {
    ob_start();
    extract($request->query->all(), EXTR_SKIP);
    include sprintf(__DIR__.'/../src/pages/%s.php', $map[$path]);
    $response = new Response(ob_get_clean());
} else {
    $response = new Response('Not Found', 404);
}

$response->send();

```

由于我们将请求里面的get参数解压（extract）出来了，我们就可以简化模板代码：

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

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

```

现在我们的代码将可以以更好的状态来添加新的功能了。

任何一个网站都有一个重要的要素，那就是他们的URL的形式。多亏了有$map变量这个映射表，我们将URL以及跟他关联的响应这两部分代码从我们的代码里面解耦出来了，但目前还是不够灵活。举个例子，如果你想动态生成URL，并将URL中的一个部分来代替GET参数：

```
# 之前
/hello?name=Fabien

# 之后
/hello/Fabien

```

想添加这个功能？请使用sf2的路由组件吧。如同以往，先在`composer.json`文件里面添加它，然后执行composer的`update`命令安装它。（译者注：composer.json文件不能写注释；另组件版本不一定参考本文，可尝试安装最新版本）

```
{
    "require": {
        "symfony/class-loader": "2.1.*",
        "symfony/http-foundation": "2.1.*",
        "symfony/routing": "2.1.*"
    }
}

```

现在开始，我们开始使用composer的autoloader，来代替我们之前所用的sf2自己的autoloader。删除之前写得`autoload.php`文件，然后把`front.php`引用它的代码改成引用composer的autoloader：

```
<?php
```

```
2
3 // example.com/web/front.php
4
5 require_once __DIR__.'/../vendor/.composer/autoload.php';
6
7 // ...
8
```

（译者注：composer版本更新以后，autoloader.php直接放在了vendor目录下面，因为不知道以后是否还会变化，请以实际目录为准）

用来代替之前$map这个映射关系数组的，是路由组件，它依赖于RouteCollection实例来描述映射关系：

```
1 use Symfony\Component\Routing\RouteCollection;
2
3 $routes = new RouteCollection();
4
```

让我添加两条路由规则，一条是/hello/SOMETHING，另一条是简单的/bye：

```
1 use Symfony\Component\Routing\Route;
2
3 $routes->add('hello', new Route('/hello/{name}', array('name' => 'W
4 $routes->add('bye', new Route('/bye'));
5
```

每一条路由规则都由一个名字（hello）以及一个Route实例来定义，而一条路由实例又由一条路由规则（/hello/{name})以及默认值数组（array('name' => 'World')）来定义。

请阅读官方文档——最近就会上线——学会路由组件其他更多功能，比如*URL*生成器，属性限制，*HTTP*方法限制，*YAML*，*XML*配置载入器，规则转储为*PHP*文件甚至*Apache*的*URL*重写规则来获取性能上的提升，以及其他更多功能。

基于RouteCollection实例里存储的信息，UrlMatch对象可实现对URL的匹配：

```
1 use Symfony\Component\Routing\RequestContext;
2 use Symfony\Component\Routing\Matcher\UrlMatcher;
```

```
3
4 $context = new RequestContext();
5 $context->fromRequest($request);
6 $matcher = new UrlMatcher($routes, $context);
7
8 $attributes = $matcher->match($request->getPathInfo());
9
```

match方法接受请求路径为参数，然后返回相关的路由属性数组（注意路由的名字已经自动赋值给_route属性了）：

```
1  print_r($matcher->match('/bye'));
2  array (
3    '_route' => 'bye',
4  );
5
6  print_r($matcher->match('/hello/Fabien'));
7  array (
8    'name' => 'Fabien',
9    '_route' => 'hello',
10 );
11
12 print_r($matcher->match('/hello'));
13 array (
14   'name' => 'World',
15   '_route' => 'hello',
16 );
17
```

我们并不严格要求要使用*$context*参数，但在真正的项目中最好还是加上，因为需要他来匹配*HTTP*方法以及其他属性（译者注：你可以定义一个*URL*只能用*HTTP*的*GET*方法访问，不使用*context*而只传*pathinfo*参数，是做不到这一点的，所以作者说最好还是使用*context*。但我认为*context*的存在让*api*看上去不舒服，因为第一时间很难判断*context*的作用是什么，为什么需要这个参数，既然已经使用了*context*，为什么还需要单独传入*pathinfo*）。

如果匹配器找不到任何一个匹配规则，他会抛出一个意外：

```
1 $matcher->match('/not-found');
2
```

```
3 // throws a Symfony\Component\Routing\Exception\ResourceNotFoundExc
4
```

利用上面的知识，我们将框架代码重写一下：

```
1  <?php
2
3  // example.com/web/front.php
4
5  require_once __DIR__.'/../vendor/.composer/autoload.php';
6
7  use Symfony\Component\HttpFoundation\Request;
8  use Symfony\Component\HttpFoundation\Response;
9  use Symfony\Component\Routing;
10
11 $request = Request::createFromGlobals();
12 $routes = include __DIR__.'/../src/app.php';
13
14 $context = new Routing\RequestContext();
15 $context->fromRequest($request);
16 $matcher = new Routing\Matcher\UrlMatcher($routes, $context);
17
18 try {
19     extract($matcher->match($request->getPathInfo()), EXTR_SKIP);
20     ob_start();
21     include sprintf(__DIR__.'/../src/pages/%s.php', $_route);
22
23     $response = new Response(ob_get_clean());
24 } catch (Routing\Exception\ResourceNotFoundException $e) {
25     $response = new Response('Not Found', 404);
26 } catch (Exception $e) {
27     $response = new Response('An error occurred', 500);
28 }
29
30 $response->send();
31
```

此段代码改进以下一些事情：

- 使用route名字作为模板的文件名

- 500错误也可以进行控制和管理了
- 解压后的请求变量让我们的模板文件代码简单许多

```
1 <!-- example.com/src/pages/hello.php -->
2
3 Hello <?php echo htmlspecialchars($name, ENT_QUOTES, 'UTF-8')
4
```

- 路由管理被单独分配到一个文件里面

```
1  <?php
2
3  // example.com/src/app.php
4
5  use Symfony\Component\Routing;
6
7  $routes = new Routing\RouteCollection();
8  $routes->add('hello', new Routing\Route('/hello/{name}', arra
9  $routes->add('bye', new Routing\Route('/bye'));
10
11 return $routes;
12
```

现在我们的框架（front.php里的代码）和配置文件（所有都在app.php文件里配置）有了很明确的分工。

我们用不到30行的代码便写好了我们新的框架，比之前那个更灵活更强大了。

使用路由组件还有一个很大的好处：利用路由规则生成URL。如果你使用路由组件来匹配你的URL，又使用路由组件来生成你的URL，那么你想更换某个路由的规则，可毫无顾忌对系统的影响。想知道如何利用这个功能生成链接？小菜一碟：

```
1 use Symfony\Component\Routing;
2
3 $generator = new Routing\Generator\UrlGenerator($routes, $context);
4
5 echo $generator->generate('hello', array('name' => 'Fabien'));
6 // outputs /hello/Fabien
```

```
7
```

代码非常明了，根本不用再重新说明过程了；然后，多亏了有context你甚至可以生成全路径：

```
1 echo $generator->generate('hello', array('name' => 'Fabien'), true)
2 // outputs something like http://example.com/somewhere/hello/Fabier
3
```

是否在关注路由的性能问题？基于你指定的路由规则，你可以创建一个被强力优化过的匹配类，来代替之前的*UrlMatcher()*：

```
1 $dumper = new Routing\Matcher\Dumper\PhpMatcherDumper($routes);
2
3 echo $dumper->dump();
4
```

还不满足？你还可以将路由规则转储为*apache*的重写规则：

```
1 $dumper = new Routing\Matcher\Dumper\ApacheMatcherDumper($routes);
2
3 echo $dumper->dump();
4
```

译者注：任何前段控制器框架，或者说单点入口框架，都会面对路由器性能问题，这个问题甚至被PHP之父作为“反对使用框架”的论点之一。事实上，如果一个项目有几十个甚至上百个路由规则，路由器性能的确是一个头痛的问题。sf2的路由转存组件将路由转存为Apache的URL改写规则，将本来PHP就不擅长的路由工作交给特别擅长此工作的web服务器，的确是个很靠谱的创新。对性能要求较高的同学可以考虑尝试一下。另外我想既然Apache的改写能做，Nginx的改写规则也应该不远了。