使用Symfony2的组件创建自己的PHP框架（第五部分：控制器和视图分离）

发表回复

趁周末还能有点空闲，一鼓作气做到跟英文原文同步吧。

英文原文地址：http://fabien.potencier.org/article/54/create-your-own-framework-on-top-of-the-symfony2-components-part-5

细心的读者可能已经发现，之前我们“硬编码”了一些代码在模板文件里面（译者注：比如说之前的`$input = $request->get('name', 'World')`这种代码）。对于目前我们写的这些小页面，问题倒也不大。但如果你想写更多的逻辑代码，那你只能把它们写在模板文件里面，这是非常不好的做法，特别是对于我们本来就是为了达到分工这个目的而做这个框架的，更不能这么搞。

为了把逻辑代码从模板文件里面分离出来，我们再加一个“控制器”层。控制器的主要作用是根据得到的客户端请求，生成相应的响应内容。

修改模板渲染代码如下：

```
1  <?php
2
3  // example.com/web/front.php
4
5  // ...
6
7  try {
8      $request->attributes->add($matcher->match($request->getPathInf
9      $response = call_user_func('render_template', $request);
10 } catch (Routing\Exception\ResourceNotFoundException $e) {
11     $response = new Response('Not Found', 404);
12 } catch (Exception $e) {
13     $response = new Response('An error occurred', 500);
14 }
15
```

目前模板渲染已经交给了外部的函数处理（这里是`render_template()`函数），我们需要把url里面带的参数解

析出来，并作为参数传递给这个函数。我们可以这么做，但是我们可以利用Request类的另外一个叫属性的功能：Request类的属性可以让你添加其他的信息，而且这些信息不一定非要跟请求的数据有直接的联系。

现在我们开始写render_template()函数了，让它作为一个没有其他任何逻辑代码的通用控制器。为了让我们之前写的模板代码还是能如之前一样工作，我们还是将请求对象的属性值都“解压”（译者注：利用PHP的extract函数）出来：

```
1 function render_template($request)
2 {
3     extract($request->attributes->all(), EXTR_SKIP);
4     ob_start();
5     include sprintf(__DIR__.'/../src/pages/%s.php', $_route);
6 
7     return new Response(ob_get_clean());
8 }
9 
```

这里我们用render_template作为call_user_func函数的一个参数。它也完全可以由任何合法的php回调函数来代替。这便可以随意让我们使用函数，或者匿名函数，甚至类方法来做回调函数。

为了使用方便，我们把每个路由规则都加上控制器属性：

```
1  $routes->add('hello', new Routing\Route('/hello/{name}', array(
2      'name' => 'World',
3      '_controller' => 'render_template',
4  )));
5 
6  try {
7      $request->attributes->add($matcher->match($request->getPathInf
8      $response = call_user_func($request->attributes->get('_control
9  } catch (Routing\Exception\ResourceNotFoundException $e) {
10     $response = new Response('Not Found', 404);
11 } catch (Exception $e) {
12     $response = new Response('An error occurred', 500);
13 }
14 
```

这样你就可以将任何控制器函数和路由规则联系起来了。当然，在控制器代码里面，你还是可以使用写好的

render_template方法：

```
$routes->add('hello', new Routing\Route('/hello/{name}', array(
    'name' => 'World',
    '_controller' => function ($request) {
        return render_template($request);
    }
)));

```

这样做会更加灵活，因为你可以在生成模板之前以及之后对Response对象进行修改，你甚至可以传更多的参数给模板：

```
$routes->add('hello', new Routing\Route('/hello/{name}', array(
    'name' => 'World',
    '_controller' => function ($request) {
        // $foo将在模板里可见
        $request->attributes->set('foo', 'bar');

        $response = render_template($request);

        // 改变一些头信息
        $response->headers->set('Content-Type', 'text/plain');

        return $response;
    }
)));

```

以下是更新改进过后的框架代码：

```
<?php

// example.com/web/front.php

require_once __DIR__.'/../vendor/.composer/autoload.php';

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

```
use Symfony\Component\Routing;

function render_template($request)
{
    extract($request->attributes->all(), EXTR_SKIP);
    ob_start();
    include sprintf(__DIR__.'/../src/pages/%s.php', $_route);

    return new Response(ob_get_clean());
}

$request = Request::createFromGlobals();
$routes = include __DIR__.'/../src/app.php';

$context = new Routing\RequestContext();
$context->fromRequest($request);
$matcher = new Routing\Matcher\UrlMatcher($routes, $context);

try {
    $request->attributes->add($matcher->match($request->getPathInf
    $response = call_user_func($request->attributes->get('_control
} catch (Routing\Exception\ResourceNotFoundException $e) {
    $response = new Response('Not Found', 404);
} catch (Exception $e) {
    $response = new Response('An error occurred', 500);
}

$response->send();

```

为庆祝我们有一个新框架的诞生，让我们再创建一个新的，带较多逻辑的程序。我们的新程序只有一个页面，告诉我们某一年是不是闰年。如果访问`/is_leap_year`，我们会被告知今年是不是闰年，当然你也可以指定查询特定的年份，比如`/is_leap_year/2009`。我们的框架不用做太大的修改，只用在app.php文件稍加变动：

```
<?php

// example.com/src/app.php

use Symfony\Component\Routing;
use Symfony\Component\HttpFoundation\Response;
```

```
7
8  function is_leap_year($year = null) {
9      if (null === $year) {
10         $year = date('Y');
11     }
12
13     return 0 == $year % 400 || (0 == $year % 4 && 0 != $year % 100
14 }
15
16 $routes = new Routing\RouteCollection();
17 $routes->add('leap_year', new Routing\Route('/is_leap_year/{year}'
18     'year' => null,
19     '_controller' => function ($request) {
20         if (is_leap_year($request->attributes->get('year'))) {
21             return new Response('Yep, this is a leap year!');
22         }
23
24         return new Response('Nope, this is not a leap year.');
25     }
26 )));
27
28 return $routes;
29
```

`is_leap_year()`函数会告知我们某一年是不是闰年，如果传入参数，那么就检查今年是不是闰年，然后根据它返回的结果，我们能分别创造出不同的响应结果。

一如往常，如果你在此打算停止继续往下阅读了并且打算开始使用当前的框架，那你最好能利用这框架创建一个简单的网站，比如那些有趣的“单页”网站（译者注：“单页”网站，one-page website，是指一种一个网站就一个页面的网站，读者们可以点击单页网站的链接参观参观，的确非常“有趣”，它们比起今天做的判断是否闰年的功能还要简单n倍……很蛋疼）。