使用Symfony2的组件创建自己的PHP框架（第六部分：控制器分析器）

发表回复

英文原文地址：http://fabien.potencier.org/article/55/create-your-own-framework-on-top-of-the-symfony2-components-part-6

可能你觉得我们的框架已经非常的稳定了，但其实我们仍然可以继续改进它。

目前，我们所有的例子，都是以面向过程的方式实现的，但是别忘了，我们的控制器可以是任何一种合法的php回调函数。让我们把控制器改写成一个类：

```
1  class LeapYearController
2  {
3      public function indexAction($request)
4      {
5          if (is_leap_year($request->attributes->get('year'))) {
6              return new Response('Yep, this is a leap year!');
7          }
8
9          return new Response('Nope, this is not a leap year.');
10     }
11 }
12
```

再修改相应的路由配置：

```
1  $routes->add('leap_year', new Routing\Route('/is_leap_year/{year}',
2      'year' => null,
3      '_controller' => array(new LeapYearController(), 'indexAction')
4  )));
5
```

当你使用这种方法创建更多的新页面的时候，你会觉得更加的合理和直观，但是这种方式也有一个副作用：`LeapYearController`对象总是会被创建，无论当前的url是不是跟`leap_year`相匹配。这样做非常糟糕，因为性能受到很大影响。无论什么样的请求发送到服务器，所有的控制器类都会初始化一个对象。如果控制器

会按照指定的路由匹配规则“迟载入（lazy-loaded）”，性能将会好很多。

要解决这个问题，我们再安装一个“Http核心（HttpKernel）”组件：

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

```

HttpKernel组件有许多有意思的功能，但目前我们需要的是“控制器分析器（controller resolver）”。根据传过来的请求对象，controller resolver知道那一个控制器将要被执行，以及将要传给它什么参数。所有的controller resolver需要实现以下接口：

```
namespace Symfony\Component\HttpKernel\Controller;

interface ControllerResolverInterface
{
    function getController(Request $request);

    function getArguments(Request $request, $controller);
}

```

`getController()`方法实现依赖跟之前同样的方式：_controller属性必然是跟某个请求对应关联的。除了php内置的回调函数方式，`getController()`也支持像`"class::method"`这样的字符串作为合法的回调函数。

```
$routes->add('leap_year', new Routing\Route('/is_leap_year/{year}',
    'year' => null,
    '_controller' => 'LeapYearController::indexAction',
)));

```

要让这段代码生效，我们要让框架代码使用`HttpKernel`组件的控制器分析器：

```
use Symfony\Component\HttpKernel;

$resolver = new HttpKernel\Controller\ControllerResolver();

$controller = $resolver->getController($request);
$arguments = $resolver->getArguments($request, $controller);

$response = call_user_func_array($controller, $arguments);

```

*Controller resolver*还有一个好处是，它将会为你合理的处理错误，比如说，如果你忘记了给路由规则设定*_controller*属性

现在我们来看看控制器所需要的参数是怎么被猜测出来的。`getArguments()`方法将使用PHP的反射来决定什么样的参数才需要传给控制器。

`indexAction()`方法需要一个Request对象作为参数。`getArguments()`知道如何根据参数类型（type-hint）来传入正确的参数：

```
public function indexAction(Request $request)

// 译者注：如果没有限定参数类型是Request类型，那么参数就不会被正确传递
// 所以以下代码不会正常运行
// public function indexAction($request)

```

更有趣的是，`getArguments()`方法可以传递任何请求对象的属性值，只要参数的名字跟属性名称一样就行：

```
public function indexAction($year)

```

你甚至可以在传入属性指的同时也传入请求对象（因为已经通过参数类型和参数名称做好了匹配，所以参数顺序无所谓）

```
public function indexAction(Request $request, $year)

public function indexAction($year, Request $request)

```

```
4
```

最后，你可以使用方法参数默认值的方式，来给一个请求的可选参数设置默认值：

```
1 public function indexAction($year = 2012)
2
```

让我们把year参数传递给控制器：

```
1  class LeapYearController
2  {
3      public function indexAction($year)
4      {
5          if (is_leap_year($year)) {
6              return new Response('Yep, this is a leap year!');
7          }
8
9          return new Response('Nope, this is not a leap year.');
10     }
11 }
12
```

控制器分离器还有验证控制器回调函数以及其参数的功能，如果出现问题，它将抛出一个详细的意外来解释发生了什么错误（比如控制器类不存在，控制器方法不存在，某个参数没有相应的请求属性）

默认的控制器分析器已经非常的灵活，你可能会好奇怎么可能还有人去再实现一个分析器（为什么需要提供一个接口）？举两个（译者注：其他分析器的实现的）例子：在*sf2*里，*getController()*功能得到了进一步强化，让其可实现将控制器作为服务，而在*FrameworkExtraBundle*里，*getArguments()*方法也得到了增强，让请求参数自动转化为对象。

让我们最后整理一下我们的新框架：

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

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

function render_template($request)
{
    extract($request->attributes->all());
    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);
$resolver = new HttpKernel\Controller\ControllerResolver();

try {
    $request->attributes->add($matcher->match($request->getPathInf

    $controller = $resolver->getController($request);
    $arguments = $resolver->getArguments($request, $controller);

    $response = call_user_func_array($controller, $arguments);
} catch (Routing\Exception\ResourceNotFoundException $e) {
    $response = new Response('Not Found', 404);
} catch (Exception $e) {
    $response = new Response('An error occurred', 500);
}

$response->send();

```

现在我们框架现在更加的健壮，更加的灵活，而且实现它仍然没有超过40行代码。