使用Symfony2的组件创建自己的PHP框架（第十二部分：使用依赖注入容器）

1条回复

英文原文地址：http://fabien.potencier.org/article/62/create-your-own-framework-on-top-of-the-symfony2-components-part-12

在前一篇教程的最后，我们扩展了Symfony的`HttpKernel`类，而将`Simplex\Framework`的代码清空了。让我们把前端控制器的一些代码挪过来：

```
1  <?php
2
3  // example.com/src/Simplex/Framework.php
4
5  namespace Simplex;
6
7  use Symfony\Component\Routing;
8  use Symfony\Component\HttpKernel;
9  use Symfony\Component\EventDispatcher\EventDispatcher;
10
11 class Framework extends HttpKernel\HttpKernel
12 {
13     public function __construct($routes)
14     {
15         $context = new Routing\RequestContext();
16         $matcher = new Routing\Matcher\UrlMatcher($routes, $contex
17         $resolver = new HttpKernel\Controller\ControllerResolver()
18
19         $dispatcher = new EventDispatcher();
20         $dispatcher->addSubscriber(new HttpKernel\EventListener\Rc
21         $dispatcher->addSubscriber(new HttpKernel\EventListener\Re
22
23         parent::__construct($dispatcher, $resolver);
24     }
25 }
26
```

这样前端控制器的代码看起来也更加精炼：

```
<?php

// example.com/web/front.php

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

use Symfony\Component\HttpFoundation\Request;

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

$framework = new Simplex\Framework($routes);

$framework->handle($request)->send();

```

如果您能写出一个更加精简的前端控制器，那也意味着你能让一个应用程序轻易拥有更多的前端控制器，不过多个前端控制器有啥好处呢？比如说，你可以让开发环境和投产环境拥有不同的配置。在开发环境中，你得打开错误报告并把错误显示出来方便调试：

```
ini_set('display_errors', 1);
error_reporting(-1);

```

但是你肯定不想让投产环境也这么设置。所以使用不同的前端控制器可以让你拥有不同的配置环境。

将前端控制器的代码移到框架类里面可以让框架更容易配置，但同时也产生了一下一些问题：

- 因为分发器在框架外无法访问，所以我们无法添加自定义的监听器（一个简单的解决办法是添加一个`Framework::getEventDispatcher()`方法）
- 我们也无法修改`UrlMatcher`以及`ControllerResolver`的实现方式了
- 我们也无法轻易地测试框架，因为我们无法模拟框架内部的对象*
- 我们无法利用`ResponseListener`修改charset（一个简单的解决办法是让charset作为构造函数的一个参数）

由于之前我们使用依赖注入，所以当时没有这些问题。所有的依赖对象都是通过参数传递的方式“注入”到构造

体里的（举个栗子：事件分发器便是注入在框架对象里面的，所以我们能全面控制他的构造以及配置）

这是否意味着我们将在灵活性，可定制性，易测试性，以及避免在不同的前端控制器里复制粘贴代码这些美好的事情里面做出艰难的决定？答案是否定的，我们还有终极解决方案。利用Symfony2的依赖注入容器（dependency injection container，译者注：直译不太自然但是简短，其实叫做依赖注入服务管理器更贴切一点）即可解决这些问题。

```
{
    "require": {
        "symfony/class-loader": "2.1.*",
        "symfony/http-foundation": "2.1.*",
        "symfony/routing": "2.1.*",
        "symfony/http-kernel": "2.1.*",
        "symfony/event-dispatcher": "2.1.*",
        "symfony/dependency-injection": "2.1.*"
    },
    "autoload": {
        "psr-0": { "Simplex": "src/", "Calendar": "src/" }
    }
}

```

建立一个新文件来描述容器的配置：

```
<?php

// example.com/src/container.php

use Symfony\Component\DependencyInjection;
use Symfony\Component\DependencyInjection\Reference;

$sc = new DependencyInjection\ContainerBuilder();
$sc->register('context', 'Symfony\Component\Routing\RequestContext
$sc->register('matcher', 'Symfony\Component\Routing\Matcher\UrlMat
    ->setArguments(array($routes, new Reference('context')))
;
$sc->register('resolver', 'Symfony\Component\HttpKernel\Controller

$sc->register('listener.router', 'Symfony\Component\HttpKernel\Eve
    ->setArguments(array(new Reference('matcher')))
```

```
17 ;
18 $sc->register('listener.response', 'Symfony\Component\HttpKernel\E
19     ->setArguments(array('UTF-8'))
20 ;
21 $sc->register('listener.exception', 'Symfony\Component\HttpKernel\
22     ->setArguments(array('Calendar\\Controller\\ErrorController::e
23 ;
24 $sc->register('dispatcher', 'Symfony\Component\EventDispatcher\Eve
25     ->addMethodCall('addSubscriber', array(new Reference('listener
26     ->addMethodCall('addSubscriber', array(new Reference('listener
27     ->addMethodCall('addSubscriber', array(new Reference('listener
28 ;
29 $sc->register('framework', 'Simplex\Framework')
30     ->setArguments(array(new Reference('dispatcher'), new Referenc
31 ;
32
33 return $sc;
34
```

此文件的目的就是描述你需要的对象以及它们依赖的对象。在配置阶段是不会有任何对象被创建的。此文件是一个纯粹的，仅仅对如何创建以及操作对象做静态描述的文件。对象将在您或者容器需要创建它的时候才会被生成。

举个栗子，要创建路由监听器，我们只需要告诉Symfony他的类名是`Symfony\Component\HttpKernel\EventListener\RouterListeners`，以及他需要一个`Matcher`作为他的依赖组建（new Reference(‘matcher’)）。如你所见，每一个对象都被一个唯一的名字表示，此名字让我们能获取相应的对象以及被其他对象所引用。

默认情况下，每当你从容器里获取对象的时候，容器都将返回同一个实例。所以利用容器可以用来管理你的“全局”对象。

现在前端控制器只用将一切黏在一块儿：

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

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

$routes = include __DIR__.'/../src/app.php';
$sc = include __DIR__.'/../src/container.php';

$request = Request::createFromGlobals();

$response = $sc->get('framework')->handle($request);

$response->send();

因为目前所有的对象都由依赖注入容器来生成了，所以框架代码又回到了之前那最简洁的

<?php

// example.com/src/Simplex/Framework.php

namespace Simplex;

use Symfony\Component\HttpKernel\HttpKernel;

class Framework extends HttpKernel
{
}

```

如果你想要一个轻量级的容器，可以考虑下*Pimple*，一个只有*60*行代码的依赖注入容器类（译者注：我就喜欢这种简洁有效的小工具，啥都不说，强烈推荐！）

现在你可以这样注册你自己的监听器：

```
$sc->register('listener.string_response', 'Simplex\StringResponseLi
$sc->getDefinition('dispatcher')
    ->addMethodCall('addSubscriber', array(new Reference('listener.
;

```

除了描述对象，依赖注入容器也可以通过参数来做配置。我们定义一个是否在debug模式的参数：

```
1 $sc->setParameter('debug', true);
2
3 echo $sc->getParameter('debug');
4
```

所有参数将在定义对象的时候用到。我们可以让charset也能配置：

```
1 $sc->register('listener.response', 'Symfony\Component\HttpKernel\Ev
2     ->setArguments(array('%charset%'))
3 ;
4
```

之后，你需要在使用响应监听器之前指定好charset：

```
1 $sc->setParameter('charset', 'UTF-8');
2
```

参数也可以用在指定路由配置上：

```
1 $sc->register('matcher', 'Symfony\Component\Routing\Matcher\UrlMatc
2     ->setArguments(array('%routes%', new Reference('context')))
3 ;
4
```

相应地在前端控制器中添加：

```
1 $sc->setParameter('routes', include __DIR__.'/../src/app.php');
2
```

相对于整个依赖注入容器的功能来说以上的介绍还只是冰山一角，除了上面说的那些功能，还能覆写已存在的对象定义，甚至将定义转存为php类等等（译者注，还提到一个scope support，~~不知道是什么意思……~~好吧我现在知道了，scope是指某个服务“命名空间”，详情请见官方文档）。Symfony的依赖注入容器很好很强大，可以用来管理任意类型的php类。

如果你不想在你的框架中使用依赖注入容器，请别冲着哥斯巴达，别用就是了。这是您的框架，不是我的。

这文章是此系列的最后一篇了，虽然还有好多好多功能还没有提到，但也希望各位看官能从这些文章学到新姿

势，并且开始自己再深入研究Symfony2框架是如何工作的。

如果你想了解更多，我强烈建议你读一读Silex微框架的代码，特别是Application类

祝大家学得开心，玩得开心，写得开心

~~ 全剧终 ~~