使用Symfony2的组件创建自己的PHP框架（第十一部分：使用事件处理异常）

发表回复

英文原文地址：http://fabien.potencier.org/article/60/create-your-own-framework-on-top-of-the-symfony2-components-part-11

如果你正在使用我们的框架，你或许想让框架支持自定义错误页面。目前，我们可以处理404和500错误，不过此功能都是硬编码在框架里面的。但让错误信息变得可自定义也非常容易：分发一个新事件然后监听它。要做到这点也就意味着监听器（listener）需要调用一个控制器。

但要是这个控制器也抛异常，那不是会发生无限循环调用？应该有更方便的方法是吧。

下面看看`HttpKernel`类。`HttpKernel`类是对`HttpKernelInterface`接口的一个可被广泛使用的，灵活的，可扩展的实现。我们就是用它而不用反反复复的发明轮子（译者注：又来了作者真讨厌发明轮子）。

这个类跟我们目前写的类非常的类似：它在一个请求进来的时候，在某些策略点（strategic point）分发事件，并使用控制器分析器来选择某一个控制器去处理请求。它会非常的细心处理各种问题，并且在有问题发生的时候，会有非常棒的回馈。

下面是我们的新框架代码：

```
1  <?php
2
3  // example.com/src/Simplex/Framework.php
4
5  namespace Simplex;
6
7  use Symfony\Component\HttpKernel\HttpKernel;
8
9  class Framework extends HttpKernel
10 {
11 }
12
```

以及新版控制器：

```
<?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;
use Symfony\Component\HttpKernel;
use Symfony\Component\EventDispatcher\EventDispatcher;

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

$context = new Routing\RequestContext();
$matcher = new Routing\Matcher\UrlMatcher($routes, $context);
$resolver = new HttpKernel\Controller\ControllerResolver();

$dispatcher = new EventDispatcher();
$dispatcher->addSubscriber(new HttpKernel\EventListener\RouterList

$framework = new Simplex\Framework($dispatcher, $resolver);

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

```

RouterListener是一个跟我们之前实现的一样逻辑的类：匹配请求的路由，并且根据路由参数为请求生成更多的属性。

现在我们的代码既精简，功能又强大，比以前更加的给力了。比如说现在可以利用ExceptionListener监听器来让你的错误处理变得能够配置。

```
$errorHandler = function (HttpKernel\Exception\FlattenException $ex
    $msg = 'Something went wrong! ('.$exception->getMessage().')';

    return new Response($msg, $exception->getStatusCode());
});

```

```
7 $dispatcher->addSubscriber(new HttpKernel\EventListener\ExceptionLi
8
```

ExceptionListener使用FlattenException来代替抛出Exception，让你更加容易地操作和显示异常。它能使用任何有效的控制器作为异常的处理器，所以你可以建立一个错误控制器（ErrorController类）来代替闭包

```
1 $listener = new HttpKernel\EventListener\ExceptionListener(
2     'Calendar\\Controller\\ErrorController::exceptionAction');
3 $dispatcher->addSubscriber($listener);
4
```

错误控制器代码如下：

```
1  <?php
2
3  // example.com/src/Calendar/Controller/ErrorController.php
4
5  namespace Calendar\Controller;
6
7  use Symfony\Component\HttpFoundation\Response;
8  use Symfony\Component\HttpKernel\Exception\FlattenException;
9
10 class ErrorController
11 {
12     public function exceptionAction(FlattenException $exception)
13     {
14         $msg = 'Something went wrong! ('.$exception->getMessage().
15
16         return new Response($msg, $exception->getStatusCode());
17     }
18 }
19
```

怎样？轻易得得到一个简洁而又可自定义的错误管理功能！如果你的控制器抛出一个异常，HttpKernel会很好去处理它。

在第二章，我们曾经聊过Response::prepare()这个方法，此方法能保证响应是严格遵守http协议的.如果能在

响应客户端之前，都调用一次这个方法，那有多好。其实这便是ResponseListener所做的事情：

```
1 $dispatcher->addSubscriber(new HttpKernel\EventListener\ResponseLis
2
```

是不是很方便呢？让我们再添加一个新功能，让响应对象支持响应流（streamed response）功能如何？只用注册一个StreamedResponseListener就行了：

```
1 $dispatcher->addSubscriber(new HttpKernel\EventListener\StreamedRes
2
```

然后在你的控制器里返回一个StreamedResponse实例来替代以前的Response实例

请阅读*Symfony2*组件的*Internal*这一章，了解更多利用*HttpKernel*做事件分发的知识点，以及他们是如何改变对请求的处理流程的

现在让我们在添加一个可以允许控制器仅返回字符串，而非必须返回一个完整的响应对象的监听器：

```
1  class LeapYearController
2  {
3      public function indexAction(Request $request, $year)
4      {
5          $leapyear = new LeapYear();
6          if ($leapyear->isLeapYear($year)) {
7              return 'Yep, this is a leap year! ';
8          }
9
10         return 'Nope, this is not a leap year.';
11     }
12 }
13
```

为了实现此功能，我们需要监听内核里面的kernel.view事件，此事件会在控制器刚被调用结束后触发。它的作用是仅在需要的时候，将控制器返回的字符串转化成一个完全的响应对象：

```
1 <?php
2
```

```
3  // example.com/src/Simplex/StringResponseListener.php
4
5  namespace Simplex;
6
7  use Symfony\Component\EventDispatcher\EventSubscriberInterface;
8  use Symfony\Component\HttpKernel\Event\GetResponseForControllerRes
9  use Symfony\Component\HttpFoundation\Response;
10
11 class StringResponseListener implements EventSubscriberInterface
12 {
13     public function onView(GetResponseForControllerResultEvent $ev
14     {
15         $response = $event->getControllerResult();
16
17         if (is_string($response)) {
18             $event->setResponse(new Response($response));
19         }
20     }
21
22     public static function getSubscribedEvents()
23     {
24         return array('kernel.view' => 'onView');
25     }
26 }
27
```

只有当控制器返回不是响应对象的时候，此事件才会被触发，另外如果在此事件上设置了响应对象，便停止了此事件的继续传播（原文还有一句，说正因为如此所以代码很简单，但是我觉得这代码简不简单没什么直接逻辑，不知道作者什么意思）

当然别忘记在前端控制器里面注册它

```
1  $dispatcher->addSubscriber(new Simplex\StringResponseListener());
2
```

如果你忘记了注册（译者注：*StringResponseListener*），*HttpKernel*会抛出一个带有很好错误提示的异常：*The controller must return a response (Nope, this is not a leap year. given)..*

至此，我们的框架代码是越来越紧凑，而且都是用现成的库组成的。扩展的本质其实就是事件监听+订阅事

件。

但愿你现在能够理解为什么这看起来那么简单的`HttpKernelInterface`会这么的强大。它默认实现`HttpKernel`，能让你感受到很多的很酷的特性，而且能毫不费劲得使用它。又因为`HttpKernel`是Symfony2以及Silex框架的核心，所以你可以有两全其美的选择：在一个稳固的，持续维护的，并且在许多网站都已验证的地基架构上，做一个可定制的，满足你需求的框架，以及写出做过安全审计而又能很好扩展的代码