使用Symfony2的组件创建自己的PHP框架（第十部分：页面缓存）

发表回复

英文原文地址：http://fabien.potencier.org/article/59/create-your-own-framework-on-top-of-the-symfony2-components-part-10

我曾在第二章的结论里谈到过，使用Symfony2组件库的好处之一是：让所有使用他们的的框架或者应用程序都有很好的互用性（interoperability, 译者注：这个单词比较难翻译。举一个例子，如果电视机插头按照国内标准的两线插头去生产，那么国内所有的两线插座都可以给这台电视机供电，这就是interoperability，简单说，就是一种让各种不同的模块或者系统在一起工作的能力）。为了向此目标再迈出一大步，我们得让框架实现`HttpKernelInterface`这个接口：

```
1  namespace Symfony\Component\HttpKernel;
2
3  interface HttpKernelInterface
4  {
5      /**
6       * @return Response A Response instance
7       */
8      function handle(Request $request, $type = self::MASTER_REQUEST
9  }
10
```

`HttpKernelInterface`算是`HttpKernel`组件里面最重要的代码了，真的。实现了此接口的框架或者应用程序会立马具有互操作性，并且随之会带来更多的好处

更新你的框架代码，让他实现`HttpKernelInterface`接口

```
1  <?php
2
3  // example.com/src/Framework.php
4
5  // ...
6
7  use Symfony\Component\HttpKernel\HttpKernelInterface;
8
```

```
class Framework implements HttpKernelInterface
{
    // ...

    public function handle(
        Request $request,
        $type = HttpKernelInterface::MASTER_REQUEST,
        $catch = true
    ) {
        // ...
    }
}

```

虽然看起来这是一个微不足道的改动，但其实此改动给我们带来了很多特性。首当其冲便是令人形象深刻的http缓存功能。

HttpCache是使用php实现的，一个功能完整的反向代理。它也实现了HttpKernelInterface并可以将另外一个用HttpKernelInterface实现的类包起来：

```
use Symfony\Component\HttpKernel\HttpCache\HttpCache;
use Symfony\Component\HttpKernel\HttpCache\Store;

$framework = new Simplex\Framework($dispatcher, $matcher, $resolver
$framework = new HttpCache($framework, new Store(__DIR__.'/../cache

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

```

这便是我们要为我们的框架添加http缓存功能的所有代码，是不是很给力？

我们需要通过改变相应头来对缓存策略进行配置。比如要将一个页面缓存10秒，我们需要调用Response::setTtl方法

```
// example.com/src/Calendar/Controller/LeapYearController.php

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

```

```
5       $leapyear = new LeapYear();
6       if ($leapyear->isLeapYear($year)) {
7           $response = new Response('Yep, this is a leap year!');
8       } else {
9           $response = new Response('Nope, this is not a leap year.')
10      }
11
12      $response->setTtl(10);
13
14      return $response;
15  }
16
```

如果你跟我一样，喜欢在命令行下面模拟请求来调试代码，你可以轻而易举的使用*echo $response;*的方式将所有的响应头信息以及内容全部显示出来（译者注：如果入口文件是front.php，你可以用类似Request::create('/is_leap_year/2012')的方式模拟一个请求，然后在框架handle了这个请求并返回了响应对象之后，不直接调用响应的send方法，而是直接echo出这个响应对象，你就可以通过命令行下执行php front.php来查看response的详细信息）

让我们写个随机数来验证我们的缓存功能是否好用：

```
1   $response = new Response('Yep, this is a leap year! '.rand());
2
```

在你部署代码到投产机的时候，记得使用*Symfony*的反相代理来提升性能，或者用更加专业的反相代理软件*Varnish*

利用http头信息来管理缓存很好很强大，它既可以使用过期策略，也可以使用验证策略来管理你的缓存。如果你对这些概念不熟悉话，最好先阅读一下Symfony2文档的http缓存部分

Response类包含了很多方法来让你方便管理http缓存，setCache方法便是其中最强大的方法之一，它可以通过一个数组来设定所有的缓存参数：

```
1   $date = date_create_from_format('Y-m-d H:i:s', '2005-10-15 10:00:0
2
3   $response->setCache(array(
4       'public'        => true,
```

```
5      'etag'          => 'abcde',
6      'last_modified' => $date,
7      'max_age'       => 10,
8      's_maxage'      => 10,
9  ));
10
11 // 等同于以下代码
12 $response->setPublic();
13 $response->setEtag('abcde');
14 $response->setLastModified($date);
15 $response->setMaxAge(10);
16 $response->setSharedMaxAge(10);
17
```

在验证模式下，你可以通过**isNotModified**方法来判断是否可将响应缓存直接返回，从而节省响应时间。

```
1  $response->setETag('whatever_you_compute_as_an_etag');
2
3  if ($response->isNotModified($request)) {
4      return $response;
5  }
6
7  $response->setContent('The computed content of the response');
8
9  return $response;
10
```

虽然HTTP缓存很给力，但万一你不能缓存整个页面而只是一部分怎么办？比如说页头用户信息？没事，ESI（Edge Side Includes）来帮您！它可以用“子请求”的方式生成页面部分缓存，来代替整个页面的缓存。

```
1 This is the content of your page
2
3 Is 2012 a leap year? <esi:include src="/leapyear/2012" />
4
5 Some other content
6
```

你需要在**HttpCache**对象里传入ESI对象，来开启**HttpCache**类对ESI功能的支持，设置以后**HttpCache**便可以自动识别ESI标签并生成一个子请求。

```
use Symfony\Component\HttpKernel\HttpCache\ESI;

$framework = new HttpCache($framework, new Store(__DIR__.'/../cache

```

要让*ESI*正常工作，你需要一个支持*ESI*的反相代理，比如*Symfony2*的实现，或者使用*Varnish*

如果设置了太多的缓存机制，或者太多的ESI标签，你可能非常难看得出那些应该缓存而那些不应该，这个时候你可以将debug设置打开进行调试：

```
$framework = new HttpCache(
    $framework,
    new Store(__DIR__.'/../cache'),
    new ESI(),
    array('debug' => true)
);

```

调试器将会在头信息中加入`X-Symfony-Cache`信息来表述缓存层都做了些什么：

```
X-Symfony-Cache:  GET /is_leap_year/2012: stale, invalid, store

X-Symfony-Cache:  GET /is_leap_year/2012: fresh

```

`HttpCache`实现了很多功能比如`stale-while-revalidate`、`stale-if-error`等RFC 5861协议中定义的Http Cache-Control的扩展功能

在一个简单的接口的帮助下，我们的框架可以享受`HttpKernel`组件的许多功能。Http缓存只是其中的一个重要部分，但已经能让你的框架运行如飞了！