Symfony2 VS 原生的 PHP

为什么说 Symfony2 比打开一个文件，然后直接开始写 php 代码要好？

如果你从没有用过框架、或者不熟悉 MVC 概念、或者只是被 Symfony2 的一些讨论所吸引，那么，这章是为你准备的。我们不会直接告诉你 Symfony2 比用原生的 PHP 使得开发更快速、软件质量更好，你自己会体会到的。

在这章中，你将会用原生的 PHP 写一个小应用，然后把它重构，使之更有组织性。

你将会穿越时空，看到在过去的这些年里，web 开发进化到如今这个地步的背后的一些决策。

最后，你会发现 Symfony2 是如何把你从庸俗的任务中解救出来，然后可以掌控自己的代码。

原生 php 的小博客

在这章，你将会用原生的 php 搭建一个象征性的博客应用。首先，创建一个页面用来展示存在数据库中的所有博客条目。使用原生的 php 是很快的，同时也很混乱。

```
1.  <?php
2.  // index.php
3.  $link = mysql_connect('localhost', 'myuser', 'mypassword');
4.  mysql_select_db('blog_db', $link);
5.
6.  $result = mysql_query('SELECT id, title FROM post', $link);
7.  ?>
8.  <!DOCTYPE html>
9.  <html>
10.     <head>
11.         <title>List of Posts</title>
12.     </head>
```

```
13.         <body>
14.             <h1>List of Posts</h1>
15.             <ul>
16.                 <?php while ($row = mysql_fetch_assoc($result)): ?>
17.                 <li>
18.                     <a href="/show.php?id=<?php echo $row['id'] ?>">
19.                         <?php echo $row['title'] ?>
20.                     </a>
21.                 </li>
22.                 <?php endwhile; ?>
23.             </ul>
24.         </body>
25. </html>
26.
27. <?php
28.     mysql_close($link);
29. ?>
```

上面的代码很快就可以写好，执行的也很快，但是随着应用的增长，维护几乎是不可能的。

有几个主要的问题需要说明：

- 没有错误检查；如果连接数据库失败怎么办？
- 代码缺乏组织性；随着应用的增长，一个文件将变的不可能维护。在哪处理表单提交？怎么样验证数据？发送邮件的代码应该写在哪？
- 无法复用代码；由于任何逻辑都是在一个文件里，应用的其他页面无法复用任何一部分代码

另外一个没提到的问题是数据库写死了使用 MySql。虽然本章不介绍，Symfony2 整合了 Doctrine，这是一个专注于数据库抽象和映射的类库

让我们继续去解决这些问题

独立表现层

代码很容易从分离应用程序逻辑和表现层 HTML 中获益：

```
<?php
// index.php
$link = mysql_connect('localhost', 'myuser', 'mypassword');
mysql_select_db('blog_db', $link);

$result = mysql_query('SELECT id, title FROM post', $link);

$posts = array();
while ($row = mysql_fetch_assoc($result)) {
    $posts[] = $row;
}

mysql_close($link);

// include the HTML presentation code
require 'templates/list.php';
```

现在 HTML 代码写在单独的文件中(template/list.php)，天本质上就是一个使用 php 模板语言的 HTML 文件。

```
<!DOCTYPE html>
<html>
    <head>
        <title>List of Posts</title>
    </head>
    <body>
        <h1>List of Posts</h1>
            <ul>
                <?php foreach ($posts as $post): ?>
                <li>
                    <a href="/read?id=<?php echo $post['id'] ?>">
                        <?php echo $post['title'] ?>
                    </a>
                </li>
                <?php endforeach; ?>
            </ul>
    </body>
</html>
```

按照惯例，包含应用程序逻辑的文件-index.php-被称作控制器。不管你使用什么语言的框架，控制器这个词你将会经常见到。它连接处理用户输入和准备响应两个过程。

在这个例子中，控制器从数据库中准备数据，然后包含一个模板去展示这些数据。通过和控制器分离，如果你想使用其他格式渲染博客条目只需要改变包含的模板文件就可以了（比如 json 格式使用 list.json.php）。

分离应用程序逻辑

目前为止，这个应用只有一个页面。如果第二个页面也需要使用相同的数据库连接，或者相同的博客文章数组要怎么办呢？重构代码使核心行为和应用的数据访问方法独立到单独的文件中，我们叫 model.php:

```
1.  <?php
2.  // model.php
3.  function open_database_connection()
4.  {
5.      $link = mysql_connect('localhost', 'myuser', 'mypassword');
6.      mysql_select_db('blog_db', $link);
7.      return $link;
8.  }
9.  function close_database_connection($link)
10. {
11.     mysql_close($link);
12. }
13. function get_all_posts()
14. {
15.     $link = open_database_connection();
16.     $result = mysql_query('SELECT id, title FROM post', $link);
17.     $posts = array();
18.     while ($row = mysql_fetch_assoc($result)) {
19.         $posts[] = $row;
20.     }
21.     close_database_connection($link);
22.     return $posts;
23. }
```

文件名使用 model.php 是因为，应用逻辑和数据访问习惯上被称作模型层。在一个组织性良好的项目中，业务逻辑的主要代码都在模型中，而不是在控制器中。不像这个例子，只有一部分（或没有）模型关心访问数据库

控制器现在非常简单：

```
1.  <?php
2.  require_once 'model.php';
3.
4.  $posts = get_all_posts();
5.
6.  require 'templates/list.php';
```

现在，控制器的唯一任务就是从模型层获取数据，然后调用模板去渲染数据。这是一个非常简单的模型-视图-控制器模式。

分离布局

现在，应用被重构成 3 个独立的部分，这有很多的优点、而且在不同的页面中复用也可行了。

代码中唯一不能复用的是视图层。通过创建一个 layout.php 文件完善：

```
1.  <!-- templates/layout.php -->
2.  <!DOCTYPE html>
3.  <html>
4.      <head>
5.          <title><?php echo $title ?></title>
6.      </head>
7.  <body>
8.       <?php echo $content ?>
9.  </body>
10. </html>
```

template/list.php）现在可以通过继承布局得到简化：

```
1.  <?php $title = 'List of Posts' ?>
2.
3.  <?php ob_start() ?>
4.      <h1>List of Posts</h1>
5.      <ul>
6.          <?php foreach ($posts as $post): ?>
7.          <li>
8.              <a href="/read?id=<?php echo $post['id'] ?>">
9.                  <?php echo $post['title'] ?>
10.             </a>
```

```
            </li>
            <?php endforeach; ?>
        </ul>
<?php $content = ob_get_clean() ?>

<?php include 'layout.php' ?>
```

你现在被传授了一种复用模板的方法。但是，为了实现这点，你必须要在模板中使用一些不太优雅的 php 函数（ob_start()和 ob_get_clean（ ））

Symfony2 使用模板组件让你优雅而且简单的完成这个需求。稍后你将会看到这步操作。

增加博客展示页面

博客列表页面已经被重构的结构更好，而且可以复用。为了验证这点，添加一个博客展示页面，它通过一个 id 请求参数展示一篇博客。首先，在 model.php 文件中新创建一个方法，它通过提供的 id 检索一个博客。

```
// model.php
function get_post_by_id($id)
{
    $link = open_database_connection();

    $id = intval($id);
    $query = 'SELECT date, title, body FROM post WHERE id = '.$id;
    $result = mysql_query($query);
    $row = mysql_fetch_assoc($result);

    close_database_connection($link);
    return $row;
}
```

接着，创建一个新文件叫 show.php -- 这个页面的控制器

```
<?php
require_once 'model.php';

$post = get_post_by_id($_GET['id']);
require 'templates/show.php';
```

最后，创建一个新的模板文件 - templates/show.php，用来渲染这个单独的博客文章：

```
<?php $title = $post['title'] ?>

<?php ob_start() ?>
    <h1><?php echo $post['title'] ?></h1>

    <div class="date"><?php echo $post['date'] ?></div>
    <div class="body">
        <?php echo $post['body'] ?>
    </div>
<?php $content = ob_get_clean() ?>

<?php include 'layout.php' ?>
```

创建第二个页面非常简单，也没有重复使用代码。但是，这个页面还是引入了一些框架可以解决的问题。比如，没有或者非法的 id 会导致页面崩溃。如果出现 404 页面会更好，但是这个不太容易实现。更糟的是，如果你忘记了使用 intval()函数过滤 id 参数，你的数据库就有被 sql 注入攻击的危险。

另一个主要问题是每个独立的控制器都必须要引入 model.php 文件。如果每个控制器突然都要包含另一个文件或者执行一些全局的任务（安全监测）怎么办？当前这个情况，代码必须加到每个控制器文件中。如果你忘记了其中的一个，祈祷不会产生安全问题吧

前端控制器来营救

解决方法是使用一个[前端控制器](),这个一个文件，所有的请求都被它处理。通过前端控制器，应用程序的 uri 会稍微发生些变化，但是更灵活了。

```
Without a front controller
/index.php          => Blog post list page (index.php executed)
/show.php           => Blog post show page (show.php executed)

With index.php as the front controller
/index.php          => Blog post list page (index.php executed)
/index.php/show     => Blog post show page (index.php executed)
```

uri 中 index.php 部分可以使用 apache rewrite 模块移除掉。在这个例子中，博客展示页面的 uri 就会变成"/show"

当使用了前端控制器，一个文件（index.php）处理所有请求。对于博客展示页面，/index.php/show 会执行 index.php 文件，它现在负责根据 uri 路由所有请求。稍后你会看到，前端控制器是一个非常有用的工具。

创建前端控制器

你将对应用做一次大改进。通过使用一个文件处理所有请求，你可以把一些事情，比如安全控制、载入配置和路由等统一处理。在这个应用中，index.php 必须足够聪明，能够根据请求的 uri 渲染博客列表或者博客展示页面：

```
1.  <?php
2.  // index.php
3.
4.  // load and initialize any global libraries
5.  require_once 'model.php';
6.  require_once 'controllers.php';
7.
8.  // route the request internally
9.  $uri = $_SERVER['REQUEST_URI'];
10. if ('/index.php' == $uri) {
11.         list_action();
12. } elseif ('/index.php/show' == $uri && isset($_GET['id'])) {
13.         show_action($_GET['id']);
14. } else {
15.         header('Status: 404 Not Found');
16.         echo '<html><body><h1>Page Not Found</h1></body></html>';
17. }
```

为了结构更好，两个控制器（index.php 和 show.php）现在是 php 方法了，而且被移到一个单独的文件中，controller.php：

```
1.  function list_action()
2.  {
3.          $posts = get_all_posts();
4.          require 'templates/list.php';
5.  }
6.
7.  function show_action($id)
8.  {
9.          $post = get_post_by_id($id);
10.         require 'templates/show.php';
11. }
```

作为前端控制器，index.php 担当了一个新的角色，包括载入核心类库和路由，以使两个控制器被调用。事实上，这个前端控制器开始有点类似 Symfony2 处理和路由请求的机制了。

前端控制器的另一个有点是灵活的 url。注意，博客展示页面的 url 可以从"/show"改为"/read"，而这只需要修改一个地方的代码。在之前，需要修改一个文件名。在 Symfony2 中，url 比这更灵活。

到目前为止，这个项目已经从单个 php 文件改进到结构有组织性而且代码可以复用的结果。你应该高兴一点，但远没到满意的地步。比如路由系统是经常变化的，不能通过"/"识别列表为列表页面。花了大量时间在代码的结构组织上，而不是开发博客系统上。更多的时间应该被用在处理表单提交、输入验证、日志和安全上。你为什么要在这些日常问题上重复造轮子呢？

接触 Symfony2

让 Symfony2 解救你。在开始使用 Symfony2 之前，你要先下载它。这可以通过 composer 完成，它会下载正确的版本、解决依赖关系并提供自动载入工具。自动载入是可以不需要提前引入一个类就直接使用它的工具。

在你的根目录，用下面的内容创建一个 composer.json 文件：

```
{
    "require": {
        "symfony/symfony": "2.4.*"
    },
    "autoload": {
        "files": ["model.php","controllers.php"]
    }
}
```

下一步，下载 composer，然后运行下面的命令，它会把 Symfony 下载到 vendor 目录：

```
$ php composer.phar  install
```

在开始下载依赖之前，composer 会创建一个 vendor/auotload.php 文件，它负责自动载入 symfony 框架的所有文件以及在 composer.json 中指定的文件。

Symfony 的核心概念是：应用程序的主要工作是接收请求并返回响应。为了这个目的，Symfony2 提供了 Request 和 Responce 类。这些类是处理原生 HTTP 请求和响应的面向对象展现。使用他们来改善博客系统：

```
<?php
// index.php
require_once 'vendor/autoload.php';

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
$request = Request::createFromGlobals();

$uri = $request->getPathInfo();
if ('/' == $uri) {
    $response = list_action();
} elseif ('/show' == $uri && $request->query->has('id')) {
    $response = show_action($request->query->get('id'));
}else{
    $html = '<html><body><h1>Page Not Found</h1></body></html>';
    $response = new Response($html, Response::HTTP_NOT_FOUND);
}

// echo the headers and send the response
$response->send();
```

V2.4 新特性：在 Symfony2.4 中加入了 HTTP 状态码的支持

控制器现在负责返回 responce 对象。为了看起来更简单，你可以加入一个新的 render_template()函数，作用就像 Symfony2 的模板引擎：

```
1.  // controllers.php
2.  use Symfony\Component\HttpFoundation\Response;
3.
4.  function list_action()
5.  {
6.      $posts = get_all_posts();
7.      $html = render_template('templates/list.php', array('posts' => $posts));
8.      return new Response($html);
9.  }
10.
11. function show_action($id)
12. {
13.     $post = get_post_by_id($id);
14.     $html = render_template('templates/show.php', array('post' => $post));
15.
16.     return new Response($html);
17. }
18.
19. // helper function to render templates
20. function render_template($path, array $args)
21. {
22.     extract($args);
23.     ob_start();
24.     require $path;
25.     $html = ob_get_clean();
26.
27.     return $html;
28. }
```

通过引入 Symfony2 的小部分，应用更灵活和可靠了。Request 对象提供了可靠的方法去访问 HTTP 请求。尤其是，getPathInfo()方法返回清除过的 uri（总是返回/show 而不是/index.php/show）。因此，即使用户访问"/index.php/show"，应用还是可以智能的把请求路由到 show_action()函数。

Response 对象提供灵活的方式构建 HTTP 响应，允许使用面向对象的方式添加 HTTP 头和内容。

虽然这个应用中的响应很简单，但是随着应用的增长你会不断得到灵活性带来的好处。

Symfony2 的简单应用

改善博客系统已经很长时间了，但是这个小应用包含了很多的代码。在这个过程中，你开发了一个小型的路由系统和一个使用 ob_start()和 ob_get_clean()的模板渲染的方法。如果，出于某些原因，你需要继续开发这个框架，你至少可以使用 Symfony 中独立的 Routing 和 Template 组件，他们可以解决这些问题。

不用再去处理这些问题了，你可以让 Symfony 帮你做。这是一个用 Symfony2 构建的类似项目：

```php
// src/Acme/BlogBundle/Controller/BlogController.php
namespace Acme\BlogBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class BlogController extends Controller
{
    public function listAction()
    {
        $posts = $this->get('doctrine')->getManager()
            ->createQuery('SELECT p FROM AcmeBlogBundle:Post p')
            ->execute();

        return $this->render(
            'AcmeBlogBundle:Blog:list.html.php',
            array('posts' => $posts)
        );
    }

    public function showAction($id)
    {
        $post = $this->get('doctrine')
            ->getManager()
            ->getRepository('AcmeBlogBundle:Post')
            ->find($id);

        if (!$post) {
            // cause the 404 page not found to be displayed
            throw $this->createNotFoundException();
        }

        return $this->render(
            'AcmeBlogBundle:Blog:show.html.php',
            array('post' => $post)
        );
    }
}
```

这两个控制器仍然是轻量级的。他们都使用 Doctrine ORM library 从数据库检索数据，然后模板组件渲染模板，返回 Response 对象。列表的模板现在更简单了：

```
<!-- src/Acme/BlogBundle/Resources/views/Blog/list.html.php -->
<?php $view->extend('::layout.html.php') ?>

<?php $view['slots']->set('title', 'List of Posts') ?>

<h1>List of Posts</h1>
<ul>
    <?php foreach ($posts as $post): ?>
    <li>
        <a href="<?php echo $view['router']->generate(
            'blog_show',
            array('id' => $post->getId())
        ) ?>">
            <?php echo $post->getTitle() ?>
        </a>
    </li>
    <?php endforeach; ?>
</ul>
```

布局还是差不多的样子：

```
<!-- app/Resources/views/layout.html.php -->
<!DOCTYPE html>
<html>
    <head>
        <title><?php echo $view['slots']->output(
            'title',
            'Default title'
        ) ?></title>
    </head>
    <body>
        <?php echo $view['slots']->output('_content') ?>
    </body>
</html>
```

博客展示页的模板当做练习了，因为它对列表模板来说不是必须的。

当 Symfony2 的引擎（Kernel）启动，它需要一个映射用来判断根据请求的信息哪个控制器该被执行。路由系统提供了一个具有可读性格式的映射信息：

```
# app/config/routing.yml
blog_list:
    path:         /blog
    defaults: { _controller: AcmeBlogBundle:Blog:list }
blog_show:
    path:         /blog/show/{id}
    defaults: { _controller: AcmeBlogBundle:Blog:show }
```

现在，Symfony2 处理所有的任务，前端控制器非常之简单。由于它做的事很少，一旦它建好了，你基本不需要改动它。（如果你使用 Symfony2 发行版，你都不需要去创建它）

```
// web/app.php
require_once __DIR__.'/../app/bootstrap.php';
require_once __DIR__.'/../app/AppKernel.php';

use Symfony\Component\HttpFoundation\Request;

$kernel = new AppKernel('prod', false);
$kernel->handle(Request::createFromGlobals())->send();
```

前端控制器的唯一工具就是实例化 Symfony2 的引擎，然后传给它 Request 对象去处理。

Symfony2 的核心使用路由映射觉得哪个控制器被调用。

和之前一样，控制器方法还是负责返回 Response 对象。这个没什么不同。

想要直观的了解 Symfony 是如何处理请求的，请查阅 request flow diagram。

Symfony2 带来的好处

在下面的章节中，你将会学习到 Symfony 的每个模块是如何工作的，以及推荐的项目组织结构。而现在，让我们看看把博客系统从原生的 php 移植到 Symfony2 的优点：

- 你的应用现在更清晰，代码结构更统一（尽管 Symfony 没有强制要求你这么做）。代码复用性提高了，新手也能很快的上手项目。
- 你的代码都是为了你的应用逻辑而写。你无需开发或维护一些底层的公共工具，比如自动载入类、路由、控制器。
- Symfony2 提供给你开源的工具，像 Doctrine、模板引擎、安全、表单、验证期、多

语言化组件。

- 由于路由组件，应用拥有非常灵活的 url。
- Symfony2 以 HTTP 为中心的架构提供我们一些强大的工具，比如基于 Symfony2 内部 HTTP 缓存的 HTTP 缓存功能或 Varnish。这些将在稍后的缓存一章讲到。

可能，通过使用 Symfony2，最大的好处就是你接触到了 Symfony2 社区提供的大量高质量的开源工具。Symfony2 社区的好工具可以在 knpbundles.com 找到。

更好的模板

Symfony2 自带了一个使模板写起来更快、更易读的模板引擎 - twig。

这意味着同一个应用使用更少的代码。比如，列表模板使用 twig 这样写：

```
1.  {# src/Acme/BlogBundle/Resources/views/Blog/list.html.twig #}
2.  {% extends "::layout.html.twig" %}
3.
4.  {% block title %}List of Posts{% endblock %}
5.
6.  {% block body %}
7.      <h1>List of Posts</h1>
8.      <ul>
9.          {% for post in posts %}
10.             <li>
11.                 <a href="{{ path('blog_show', {'id': post.id}) }}">
12.                     {{ post.title }}
13.                 </a>
14.             </li>
15.             {% endfor %}
16.     </ul>
17. {% endblock %}
```

相应的 layout.html.twig 同样很简单：

```
{# app/Resources/views/layout.html.twig #}
<!DOCTYPE html>
<html>
    <head>
    <title>{% block title %}Default title{% endblock %}</title>
</head>
<body>
    {% block body %}{% endblock %}
</body>
</html>
```

Symfony2 很好的支持 Twig。PHP 模板也一直被 Symfony2 支持，Twig 的更多优点在接下来会继续讨论。想要获取更多信息，请查看模板。

从 CookBook 了解更多

- 如何使用 php 代替 twig
- 如何将控制器定义为服务