

学到了这里你可能还有疑问,为甚麽要使用**Symfony** 呢,打开一个php文件直接去编写代码不是更好吗?

如果你从来没有使用过任何的PHP框架,并且对MVC的理念一点都不熟悉,仅仅只是听说很多人都在用Symfony,所依你也打算用,那末这一章就是为你准备的.在这一章里,我并不会教你如何用Symfony去开发一个比原生的php脚本运行速度更快,更好地软件,因为那完全时取决与你自己的.

在这一章里,我将会带着你用原生的php来写一些简单的应用,然后用Symfony重构他,让它变得更加优秀!!我们将回溯历史,你会看到web开发在过去的几十年来的进化史.

通过前面几章的学习,相信你已经看到了Symfony是如何拯救你平凡的项目的,并且让你重新获取对你的代码的控制!!

使用原生的PHP创建一个简单博客程序

在这一章中,我们将先使用原生的php来构建一个简单的博客程序!那末,现在开始吧!让我们创建一个简单的博客入口程序,并且这个入口极有可能也已经连接数据库.用原生的php来完成这些事这是非常快速,但也非常繁琐的一项工作!

```

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', $link);
7.  ?>
8.
9.  <!DOCTYPE html>
10. <html>
11.     <head>
12.         <title>List of Posts</title>
13.     </head>
14.     <body>
15.         <h1>List of Posts</h1>
16.         <ul>
17.             <?php while ($row = mysql_fetch_assoc($result)): ?>
18.                 <li>
19.                     <a href="/show.php?id=<?php echo $row['id'] ?>">
20.                         <?php echo $row['title'] ?>
21.                     </a>
22.                 </li>
23.             <?php endwhile ?>
24.         </ul>
25.     </body>
26. </html>
27.
28. <?php
29. mysql_close($link);
30. ?>

```

这样编写起来速度是相当快的,但是随着你的应用程序成长,这种开发效率是不可能保持的!有迹象表明,需要解决如下几个问题:

1. 没有错误检查:万一数据库连接失败了该怎莫办!
2. 组织不力:如果应用功能和需求一旦开始增长,这种简单的文件将会快速增加从而导致项目不能正常维护.你应该将你处理表单提交的代码放置在哪里呢?你怎莫去验证数据呢?你应该将发送电子邮件的代码放在哪里呢?
3. 很难重用代码:因为一切都被写在了同一个文件中,没有办法重用任何部分的代码在其他的页面中!

在这里还有一个问题,整个程序已经完全依赖于mysql了!虽然在这里没有提及,但是万一你以后想到更换一个更高效的付费数据库的时候你该怎莫办呢?Symfony全面整合了[Doctrine](#) 类库,这个类库致力于数据库的抽象和映射!

展现分离

代码可以从已经准备用来做html展示的应用逻辑中获取:

```
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. $posts = array();
9. while ($row = mysql_fetch_assoc($result)) {
10.     $posts[] = $row;
11. }
12.
13. mysql_close($link);
14.
15. // include the HTML presentation code
16. require 'templates/list.php';
```

可以看到,html代码现在被存储在一个分离的文件中(`templates/list.php`),这是一个在模板中使用类似于php语法结构的文件:

```
1. <!DOCTYPE html>
2. <html>
3.     <head>
4.         <title>List of Posts</title>
5.     </head>
6.     <body>
7.         <h1>List of Posts</h1>
8.         <ul>
9.             <?php foreach ($posts as $post): ?>
10.                <li>
11.                    <a href="/read?id=<?php echo $post['id'] ?>">
12.                        <?php echo $post['title'] ?>
13.                    </a>
14.                </li>
15.            <?php endforeach ?>
16.        </ul>
17.    </body>
18. </html>
```

通过转换,这个文件已经在 `index.php` 中包含了所有的应用逻辑,这就是所谓的"控制器(`controller`)"!!不管你用什莫编程语言,使用什莫框架,关于术语控制器(`controller`)你可能已经听过很多次了!他是用来处理用户输入和响应的一部分代码,说的笼统一些就是做逻辑处理的!

在这种情况下,控制器(`controller`)从数据库获取数据,并且渲染一个模板来展示数据,通过控制器的孤立,当你想要用其他的格式(比如 `JSON` 格式: `list.json.php`)来展示博客入口的时候,你只需要简单的改变模板文件就好了!

业务逻辑分离

迄今为止,应用仅仅只包含了一个页面!但是如果有第二个页面想要去使用相同的数据库链接甚至于想要相同的博客文章排列呢?这时,我们需要重构代码,使应用程序的核心行为和数据访问功能被隔离在一个新的名为 `model.php` 文件中:

```

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.
8.      return $link;
9.  }
10.
11. function close_database_connection($link)
12. {
13.     mysql_close($link);
14. }
15.
16. function get_all_posts()
17. {
18.     $link = open_database_connection();
19.
20.     $result = mysql_query('SELECT id, title FROM post', $link);
21.     $posts = array();
22.     while ($row = mysql_fetch_assoc($result)) {
23.         $posts[] = $row;
24.     }
25.     close_database_connection($link);
26.
27.     return $posts;
28. }

```

文件名 `model.php` 被使用是因为逻辑和数据访问在传统开发中都被称之为模块层!在一个组织良好的程序中,表示商业逻辑的代码应该被放在模块(`model`)中(可以将它放在一个控制器(`controller`)里面).不同的是,在这个例子中,模块的功能仅仅是用来访问链接数据库。

这个例子中的控制器的实现(`index.php`)是相当简单的!

```

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应用的模块层中获取数据,并且调用模板来渲染数据.这是一个非常简单的 `model-view-controller` 模式的示例。

布局分离

在这一点上,应用程序已被重构为三个不同的部分,提供各种优点,并有机会在不同的页面中重用所有代码。

唯一不能重复使用的代码部分是页面布局,可以通过新增一个 `layout.php` 文件来弥补:

```

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>

```

模板(`templates/list.php`)可以简化为一个扩展的布局:

```

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>
11.              </li>
12.          <?php endforeach ?>
13.      </ul>
14.  <?php $content = ob_get_clean() ?>
15.
16.  <?php include 'layout.php' ?>

```

你现在已经有了一个完整的程序,并且它允许你去重用的模板.

不幸的是,要做到这一点,你不得不在模板中使用了一些丑陋的PHP函数 (`ob_start()`, `ob_get_clean()`)。Symfony的模板组件可以让你非常简洁轻松的实现这一点,弥补原生php的缺陷!你马上就会看到Symfony的威力!!

添加博客”show”页面

博客的展示页面已经被重构,并且代码已经变得组织良好并且重用性很高!现在我们来证明这个问题!添加一个展示(show)页面,用来显示一个根据id查询参数来确定个人博客文章的页面:

现在开始!在 `model.php` 中创建一个新的函数,用来基于给定的id来检索一个个人博客:

```

1.
2.
3.  // model.php
4.  function get_post_by_id($id)
5.  {
6.      $link = open_database_connection();
7.
8.      $id = intval($id);
9.      $query = 'SELECT date, title, body FROM post WHERE id = '.$id;
10.     $result = mysql_query($query);
11.     $row = mysql_fetch_assoc($result);
12.
13.     close_database_connection($link);
14.
15.     return $row;
16. }

```

现在,创建一个新的文件 `show.php`,这是这个新的页面的控制器!

```

1.  <?php
2.  require_once 'model.php';
3.
4.  $post = get_post_by_id($_GET['id']);
5.
6.  require 'templates/show.php';

```

最后,创建一个模板文件 `templates/show.php` 去渲染个人博客文章:

```

1. <?php $title = $post['title'] ?>
2.
3. <?php ob_start() ?>
4.     <h1><?php echo $post['title'] ?></h1>
5.
6.     <div class="date"><?php echo $post['date'] ?></div>
7.     <div class="body">
8.         <?php echo $post['body'] ?>
9.     </div>
10. <?php $content = ob_get_clean() ?>
11.
12. <?php include 'layout.php' ?>

```

创建第二个页面是非常容易的,这期间没有任何的代码复制!不过,这里介绍了一个更加挥之不去的问题,可以使用一个框架完美的解决他。例如,丢失或无效的ID查询参数会导致页面崩溃。一中更好地方式是渲染并返回一个404页面,但是要想做到这个这确是相当不容易的!更糟糕的是,你可能已经忘记去使用 `intval()` 函数来清除id查询参数里面的非法字符,你的站点入口的数据库链接很有可能会遭受sql注入攻击!

还有一个主要问题就是 个人的控制器必须被包含在 `model.php` 文件里.如果每一个控制器都需要包含一个其他的文件,并且作为一个全局的任务去执行会怎莫样呢?这安全吗?目前的情况是,代码需要被添加到每一个控制器文件里面,如果你忘记添加一些东西到一个文件里面,希望这不涉及到安全性!

“Front Controller”解决方案

一个简单的解决方案就是使用一个前端控制器,用它来处理所有的请求。随着前端控制器加入,这些URI应用程序略有变化,但开始变得更加灵活:

```

1. Without a front controller
2. /index.php      => Blog post list page (index.php executed)
3. /show.php      => Blog post show page (show.php executed)
4.
5. With index.php as the front controller
6. /index.php      => Blog post list page (index.php executed)
7. /index.php/show => Blog post show page (index.php executed)

```

当使用一个前端控制器时,一个简单的php文件(`index.php`)将处理每一个请求,对于博客文章展示页面, `/index.php/show` 将会执行 `index.php` 文件,这个文件将基于内部的url来充分的处理每一个路由请求.正如你所看到的,一个前段控制器是一个强有力的工具.

创建Front Controller

自此以后,你将在应用程序开发上迈出巨大的一步!!通过路由所有的请求,你可以集中精力于一些事情,如安全处理,配置加载,以及路由!在这个应用程序中, `index.php` 必须能够智能的根据请求的链接来渲染博客文章展示页面:

```

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. $uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
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. }

```

为了让程序更加的有组织,我们将 `index.php` 和 `show.php` 里的控制器函数每一个都移动到一个分离的文件 `controllers.php` 中:

```

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. }

```

作为前段控制器, `index.php` 已经扮演了一个全新的角色,它为了让两个控制器函数(`list_action()` 和 `show_action()`)被调用载入了所有的核心库并且路由整个应用程序,这看起来和Symfony的机制越来越像了.

前端控制器的另一优点是灵活的网址 `URLs`,注意,这个博客文章展示页只能在一个位置被改变,在此之前,一个入口文件必须要被重定义!在Symfony中,这个入口文章将变得更加灵活!

到目前为止,整个应用已经由简单的 `php` 文件被转换为一个有组织性,代码可重用的一个结构.你现在应该很开心了,但是先等等,高兴的有点儿早了.当前还有很多问题.例如,路由系统是很无情的,他并不会通过 `/` (如果 `apache rewrite` 拟静态规则被添加)来识别那些列表页(`/index.php`),同样,如果开发的不是博客,那末大量的时间将会被花费在系统的代码架构上(例如:路由,如何调用控制器,模板渲染等)!更多的时间将会花费在处理表单提交,输入验证,数据记录,以及系统安全性上.为甚麽你要去为这些常规的问题去发明解决方案呢?

初识Symfony

Symfony是可以被重用的,在使用symfony之前,你需要去下载它!它可以通过 `Composer` (它负责下载正确的版本和所有的依赖关系,并提供了自动加载机制)来获取. `Composer` 作为一个自动加载工具,她让在使用php而无须在意是否明确包含一些依赖文件成为可能!

在你的根目录,创建一个 `composer.json` 文件并写入如下内容:

```

1. {
2.     "require": {
3.         "symfony/symfony": "2.6.*"
4.     },
5.     "autoload": {
6.         "files": [ "model.php", "controllers.php" ]
7.     }
8. }

```

接下来,下载`Composer` 并且运行如下命令,他将会将Symfony下载到 `vendor/` 目录下:

```
1. $ composer install
```

除了下载你需要的依赖包以外, **Composer** 还会创建一个 **vendor/autoload.php** 文件,它用来处理在Symfony框架中所有文件的自动载入以及自动加载在 **composer.json** 中提到的文件。

Symfony的核心理念就是去由应用程序去解释,每一个请求并且返回一个响应.为此,Symfony提供了一个 **Request** 和 **Response** 类,这些类是用于做原生 **http** 解析和响应的面向对象的封装,用他们可以进一步的提升我们的blog应用:

```
1. <?php
2. // index.php
3. require_once 'vendor/autoload.php';
4.
5. use Symfony\Component\HttpFoundation\Request;
6. use Symfony\Component\HttpFoundation\Response;
7.
8. $request = Request::createFromGlobals();
9.
10. $uri = $request->getPathInfo();
11. if ('/' == $uri) {
12.     $response = list_action();
13. } elseif ('/show' == $uri && $request->query->has('id')) {
14.     $response = show_action($request->query->get('id'));
15. } else {
16.     $html = '<html><body><h1>Page Not Found</h1></body></html>';
17.     $response = new Response($html, Response::HTTP_NOT_FOUND);
18. }
19.
20. // echo the headers and send the response
21. $response->send();
```

现在,控制器负责返回 **Response** 对象,为了更为便捷,你可以使用一个 **render_template()** 函数,它的行为颇有点像Symfony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' => $posts));
8.
9.     return new Response($html);
10. }
11.
12. function show_action($id)
13. {
14.     $post = get_post_by_id($id);
15.     $html = render_template('templates/show.php', array('post' => $post));
16.
17.     return new Response($html);
18. }
19.
20. // helper function to render templates
21. function render_template($path, array $args)
22. {
23.     extract($args);
24.     ob_start();
25.     require $path;
26.     $html = ob_get_clean();
27.
28.     return $html;
29. }
```

显而易见,通过Symfony的一小部分,程序已经变的相当灵活可靠了! **Request** 类提供了一个可信的方式去访问信息通过http请求.更为特别的是,可以通过 **getPathInfo()** 方法来返回一个干净的url(通常是类似于 **/show**,但绝不是 **/index.php/show**).所以,及时用户使用 **/index.php/show** 来访问,应用也是相当智能的,,他完全可以通过 **show_action()** 来路由请求.

Response 类在构建http响应的时候给予了用户一种灵活性,它允许 **http** 报头和内容通过面向对象的接口被添加到对象中.这个应用程序中的响应是相当简单的,这种灵活性将会随着你的应用程序的增长 变得更加显著!

Symfony简单示例

我们已经带着这个blog应用走出很长的一段路了,但是就这样一个简单的程序,他仍然包含了大量的代码. 你已经实现了一个简单的路由系统并且使用 **ob_start()** 和 **ob_get_clean()** 来 渲染模板.如果因为一些原因,你可能需要重头开始构建这个框架,你已经使用了 Symfony的独立路由和模板机制这些已经解决了这个问题.

对于这些常见问题,你可以让symfony为你关注他们.这里有一个示例程序,现在你可以通过symfony来构建他:

```
1. // src/AppBundle/Controller/BlogController.php
2. namespace AppBundle\Controller;
3.
4. use Symfony\Bundle\FrameworkBundle\Controller\Controller;
5.
6. class BlogController extends Controller
7. {
8.     public function listAction()
9.     {
10.         $posts = $this->get('doctrine')
11.             ->getManager()
12.             ->createQuery('SELECT p FROM AcmeBlogBundle:Post p')
13.             ->execute();
14.
15.         return $this->render('Blog/list.html.php', array('posts' => $posts));
16.     }
17.
18.     public function showAction($id)
19.     {
20.         $post = $this->get('doctrine')
21.             ->getManager()
22.             ->getRepository('AppBundle:Post')
23.             ->find($id);
24.
25.         if (!$post) {
26.             // cause the 404 page not found to be displayed
27.             throw $this->createNotFoundException();
28.         }
29.
30.         return $this->render('Blog/show.html.php', array('post' => $post));
31.     }
32. }
```

这两个控制器仍然是轻量级的,他们每一个都用 **Doctrine ORM library** 来从数据库检索对象,使用模板组件来渲染模板,以及返回一个 **Response** 对象,该列表模板现在要简单一些:

```
1. <!-- app/Resources/views/Blog/list.html.php -->
2. <?php $view->extend('layout.html.php') ?>
3.
4. <?php $view['slots']->set('title', 'List of Posts') ?>
5.
6. <h1>List of Posts</h1>
7. <ul>
8.     <?php foreach ($posts as $post): ?>
9.         <li>
10.             <a href="<?php echo $view['router']->generate(
11.                 'blog_show',
12.                 array('id' => $post->getId())
13.             ) ?>">
14.                 <?php echo $post->getTitle() ?>
15.             </a>
16.         </li>
17.     <?php endforeach ?>
18. </ul>
```

布局模板几乎是相同的:


```

1. <!-- app/Resources/views/layout.html.php -->
2. <!DOCTYPE html>
3. <html>
4.     <head>
5.         <title><?php echo $view['slots']->output(
6.             'title',
7.             'Default title'
8.         ) ?></title>
9.     </head>
10.    <body>
11.        <?php echo $view['slots']->output('_content') ?>
12.    </body>
13. </html>

```

该展示模板用作练习.这是相当简单的,基本上微不足道.

当 Symfony 称之为 **Kernel** 的引擎启动时,他需要知道控制器和请求的一个映射关系.一个提前约定好的可读的路由配置需要被提供:

```

1. # app/config/routing.yml
2. blog_list:
3.     path:          /blog
4.     defaults: { _controller: AppBundle:Blog:list }
5.
6. blog_show:
7.     path:          /blog/show/{id}
8.     defaults: { _controller: AppBundle:Blog:show }

```

现在.Symfony已经可以处理一些日常任务了.前段控制器真的时特别简单,当他被创建后,你就再也不用去管他了(如果你使用Symfony的分布,你甚至不会需要创建它).

```

1. // web/app.php
2. require_once __DIR__.'../../app/bootstrap.php';
3. require_once __DIR__.'../../app/AppKernel.php';
4.
5. use Symfony\Component\HttpFoundation\Request;
6.
7. $kernel = new AppKernel('prod', false);
8. $kernel->handle(Request::createFromGlobals())->send();

```

前段控制器的唯一工作就是去初始化Symfony的模板引擎(**Kernel**),并且传递一个请求(**Request**)交由内核处理.Symfony通过使用路由映射来决定调用哪一个控制器.就象以前一样,作为响应,控制器方法是用来返回 **Response** 对象的.

Symfony的优点

在即将到来的章节中,你将会了解更多关于Sfmfony的工作机制以推荐的项目组织.现在,看看如何从原生的php迁移使用symfony来提升blog的质量:

- 你的应用程序现在已经明确了一贯有组织的代码(虽然Symfony的不强迫你使用)。这促进了可重用性,并允许新的开发人员在你的项目更快速地成为生产力;
- 你写的代码1000%都是你的应用程序,你不许要去建立或维护低级别的实用程序,例如:自动载入,路由,以及模板渲染控制
- 在Symfony中,你可以访问开源工具,如 **Doctrine** 和 **Templating**,安全(**Security**),表单(**Form**),验证(**Validation**)和翻译(**Translation**)等组件(仅举几例);
- 应用有着完全灵活的url,感谢路由组件
- Symfony以HTTP为核心的架构,让你可以访问功能强大的工具,如HTTP缓存搭载的Symfony的内部HTTP缓存或更强大的工具,如 **Varnish**。关于缓存这将会在后面的章节提到。

最重要的是,通过使用Symfony的,你有机会获得一整套高质量的由Symfony的社区开发的开源工具!Symfony的社区工具可以在 Knplabs.com 找到。

更好用的模板引擎

如果你选择去使用它,Symfony有一个标配的名为Twig的模板引擎,,它可以让模板更快地编写,更加易于阅读. 这意味着示例应用程序可以包含更少的代码.例如,展示模板可以用Twig这样写:

```
1.  {# app/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` 也是非常容易写出的:

```
1.  {# app/Resources/views/layout.html.twig #}  
2.  <!DOCTYPE html>  
3.  <html>  
4.      <head>  
5.          <title>{% block title %}Default title{% endblock %}</title>  
6.      </head>  
7.      <body>  
8.          {% block body %}{% endblock %}  
9.      </body>  
10. </html>
```

我的主页:<http://aifei8.net>:



服务项目.

专注于移动互联网以及嵌入式设备的研发



android应用定制开发

使用google官方的java语言开发框架，效率高，占用资源少，运行速度快。也可采用最近风靡一时的node技术。



IOS应用定制开发

使用node和html5,开发速度快，运行速度快，低耦合，低成本，几乎接近C语言的运行速度



PC桌面应用定制开发

支持windows、linux以及apple的Mac Book三大主流操作系统，良好的跨平台开发技术，给你不一样的体验



美工设计

采用html5超文本标记语言和css样式表控制页面内容，ECMAScript与用户进行交互，高效的ajax技术实现实时更新站点内容。



网站建设

采用开源免费的linux、apache、mysql、php的lamp架构。占用资源少，运行速度快




SEO优化

搜索引擎爆炸式的成长是Internet的代表，用户每次对搜索引擎的访问都潜在的产生了对特定厂商的业务。

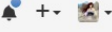
分享

我的博客:<http://blog.aifei8.net>


github主页:<http://github.com/Pengfei-Gao>



Pull requests Issues Gist



Contributions Repositories Public activity Edit profile








Love Princess Studio
Pengfei-Gao

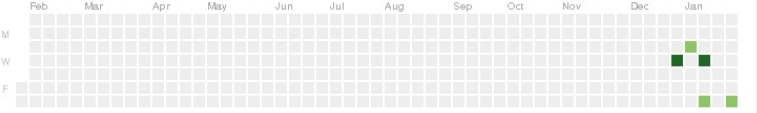
Love Princess Studio
吉林省延吉市延边大学
net.aifei8@gmail.com
http://aifei8.net
Joined on 30 Dec 2015

0 Followers 4 Starred 0 Following


Popular repositories

 YBUInformation	一个延边大学urp教务系统学生信息查询的示例	1 ★
 develop-reference-data	一些常用的开发文档	1 ★
 ionic	Advanced HTML5 mobile development framework and SDK. Build incredible mobile apps with web technologies y...	0 ★
 source-insight-3-for-centos7	centos7 下集成的source Insight 3	0 ★
 Symfony-doc-translate	Symfony的中文文档	0 ★

Contributions



Summary of pull requests, issues opened, and commits. [Learn how we count contributions.](#)

Less  More

中 简



