第 4 章　建立页面的基础知识

很奇怪，每次学习新语言或者框架的第一个例子都是在屏幕上显示"Hello, world!"。目前为止所有利用人工智能来实现交谈的尝试的结果都很差，所以电脑能问候整个世界这种想法实在有些古怪。但是 symfony 并不比其他程序笨，证据是，你可以用 symfony 创建一个说"Hello, <你的名字>"的页面。

本章会告诉你如何创建一个模块，也就是一组页面的集合体。你还将了解到如何建立一个页面，由于 MVC，页面由一个动作和一个模板构成。链接和表单是 web 交互的基础，你将在这一章了解如何在模板里增加他们，如何用动作处理他们。

建立模块框架

在第二章中我们介绍过，symfony 把页面组织成模块。建立页面之前，你必须先建立一个模块，也就是一个 symfony 能识别的目录结构的一个空壳。

symfony 命令行工具能自动建立模块。你只需要用 symfony 命令行工具执行 init-module 任务并传应用程序名与模块名这两个参数给它就可以了。在前一章里，你建立了 myapp 应用程序。如果要在这个应用程序里增加一个 mymodule 模块，只需要在命令行下输入下面的命令：

```
> cd ~/myproject
> symfony init-module myapp mymodule

>> dir+      ~/myproject/apps/myapp/modules/mymodule
>> dir+      ~/myproject/apps/myapp/modules/mymodule/actions
>> file+
~/myproject/apps/myapp/modules/mymodule/actions/actions.class.php
>> dir+      ~/myproject/apps/myapp/modules/mymodule/config
>> dir+      ~/myproject/apps/myapp/modules/mymodule/lib
>> dir+      ~/myproject/apps/myapp/modules/mymodule/templates
>> file+
~/myproject/apps/myapp/modules/mymodule/templates/indexSuccess.php
>> dir+      ~/myproject/apps/myapp/modules/mymodule/validate
>> file+
~/myproject/test/functional/myapp/mymoduleActionsTest.php
>> tokens
~/myproject/test/functional/myapp/mymoduleActionsTest.php
>> tokens
~/myproject/apps/myapp/modules/mymodule/actions/actions.class.php
```

```
>> tokens
~/myproject/apps/myapp/modules/mymodule/templates/indexSuccess.php
```

除了 actions/, config/, lib/, templates/, 与 validate/目录，这条命令只建立了三个文件。test/目录里的文件与单元测试有关，在第 15 章之前你都不用管它。actions.class.php(见例 4-1)做了一个到默认模块的成功页面的跳转。templates/indexSuccess.php 文件是空的。

例 4-1 - 默认的自动生成的动作 actions/actions.class.php

```
<?php

class mymoduleActions extends sfActions
{
  public function executeIndex()
  {
    $this->forward('default', 'module');
  }
}
```

NOTE 如果你看一下实际的 actions.class.php 文件，你会注意到除了上面的这几行之外还有其他的内容，包括一些注释。这是因为 symfony 推荐使用 PHP 注释来为你的项目生成文档，所以每个类文件都与 phpDocumentor 工具(http://www.phpdoc.org/)兼容。

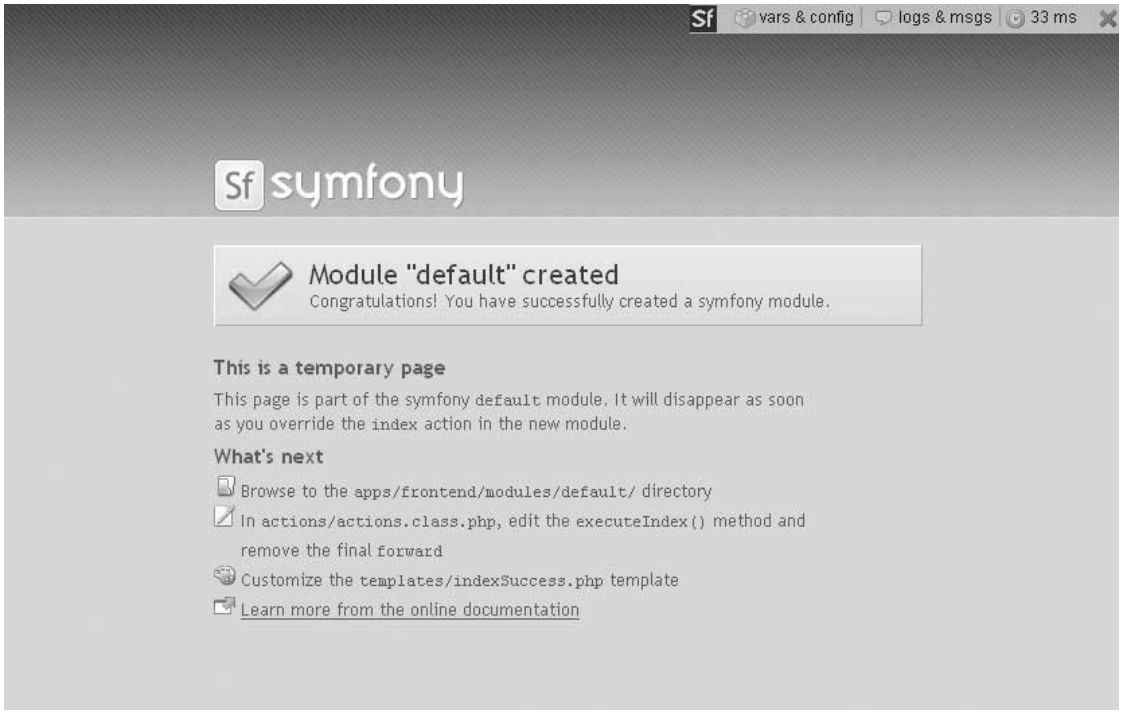
symfony 为每一个新模块建立一个 index 动作。它是由一个 execueIndex 的方法与一个叫 indexSuccess.php 的模板组成的。execute 前缀与 Success 后缀的含义会在第 6 章与第 7 章中分别解释。现在你可以认为这是一种命名习惯。在浏览器中输入下面的网址就可以看到这个页面(图 4-1)：

```
http://localhost/myapp_dev.php/mymodule/index
```

本章不会用到这个默认的 index 动作,所以你可以把 executeIndex()方法从 actions.clas.hpp 文件中去掉，并把 indexSuccess.php 文件从 templates/目录中删除。

NOTE 除了命令行，symfony 还提供了其他的建立模块的方法。其中之一是你自己来建立这些文件与目录。很多时候，模块中的动作和模板用来处理一个表里面的数据。由于建立、获取、更新与删除所需的代码往往是一样的，symfony 提供一种称之为脚手架(scaffolding)的机制来自动生成一个模块。这种技术详见第 14 章。

图 4-1 - 自动生成的默认 index 页



增加一个页面

symfony 里面，页面背后的逻辑放在动作里面，表现放在模板里。不需要逻辑的页面也需要一个空的动作。

增加一个动作

我们需要一个通过 myAction 动作来访问"Hello, world!"的页面。要建立这个页面，只要在 mymyduleActions 类里面增加一个 executeMyAction 方法，如例 4-2 。

例 4-2 - 增加一个动作就是给动作类增加一个执行方法

```
<?php

class mymoduleActions extends sfActions
{
  public function executeMyAction()
  {
  }
}
```

动作方法的名字永远是 execute``XXX``()，方法名字的第二部分的第一个字母总是大写。

现在，如果你访问下面的网址：

http://localhost/myapp_dev.php/mymodule/myAction

symfony 会抱怨缺少`myActionSuccess.php'模板。这很正常；在 symfony 里，一个页面永远是由一个动作与一个模板组成。

NOTE URL(不是域名)是区分大小写的，symfony 也区分大小写(虽然在 PHP 里方法名不区分大小写)。这就是说，如果你增加一个 executemyaction()方法，或者 executeMyaction(),然后你在浏览器里访问 myAction，symfony 会返回 404 错误信息。

SIDEBAR URL 是响应的一部分

symfony 包含一个路由系统，这个系统可以把真正的动作名与 URL 的形式分开来。这样就可以实现特殊 URL 格式。你可以不受文件结构或者请求参数的限制；动作的 URL 可以是你想要的样子。例如，请求一个 article 模块的 index 动作的 URL 常常是这样的：

http://localhost/myapp_dev.php/article/index?id=123

这个 URL 从数据库里面取出指定的文章。在这个例子里，这篇文章(id=123)是欧洲(europe)栏目里的一篇关于法国金融(finance in France)的文章。但是通过修改 routing.yml 配置文件，这个 URL 可以完全改成另外一种直观的形式：

http://localhost/articles/europe/france/finance.html

这个 URL 不仅对搜索引擎更友好，也对用户更有意义，用户可以像使用命令行一样通过在地址栏执行特定的查找，例如：

http://localhost/articles/tagged/finance+france+euro

symfony 知道如何为用户解析与生成漂亮的 URL。路由系统自动地从一个漂亮的 URL 中剥离出参数然后传给动作。它也能格式化回应的超链接使他们看起来更"漂亮"。这个功能详见第 9 章。

总之，这意味着应用程序的动作的命名可以和他们的 URL 不一致，但是动作方法的命名必须与动作名相统一。动作名说明动作要做的事情，它通常是一个不定式动词(例如 show,list,edit 等)。动作名可以隐藏起来不让用户知道，所以请放心的使用动作的名字(例如 listByName 或者 showWithComments)。这样可以有效地节省注释，另外代码的可读性也大大增强了。

增加一个模板

动作需要一个模板来表现自己。模板是模块的 templates/目录里的一个文件，模板名字由动作名与动作终止组成。默认的动作终止是"success"也就是成功，所以 myAction 动作的模板名是 myActionSuccess.php。

理想的模板只包含显示代码，所以 PHP 代码越少越好。显示"Hello, world!"的页面的模板可以如同例 4-3 中的那么简单。

例 4-3 - mymodule/templates/myActionSuccess.php 模板

```
<p>Hello, world!</p>
```

如果需要在模板里执行一些 PHP 代码，你应该避免使用通常的 PHP 语法(如例 4-4)。相反，你应该在模板里面使用特殊的 PHP 语法，如例 4-5 所示，这样不是 PHP 程序员的人也能理解。这样不仅最终生成的代码的缩进格式正确，而且可以让你把复杂的代码放在动作里面，因为只有控制语句(if,foreach,while 等)有特殊语法。

例 4-4 - 通常的 PHP 语法，对于动作没问题，对于模板就很糟糕

```
<p>Hello, world!</p>
<?php

if ($test)
{
  echo "<p>".time()."</p>";
}

?>
```

例 4-5 - 另类 PHP 语法，适合于模板

```
<p>Hello, world!</p>
<?php if ($test): ?>
<p><?php echo time(); ?></p>
<?php endif; ?>
```

TIP 一般来说模板语法的可读性是否够强是看这个文件是否不包含 PHP 的 echo 语句或者"{}"。大多数时候，开始的<?php 应该与结束的?>在同一行。

从动作传递信息给模板

动作要做的事情是所有的复杂计算，取出数据，测试，为模板设定显示或者测试用的变量。symfony 让动作类的属性(动作里的可以通过$this->variableName

访问)能够直接在模板里面的全局命名空间里面访问得到(通过$variableName)。例 4-6 与 4-7 演示如何从动作传递信息给模板。

例 4-6 - 设定动作的一个属性，把它传给模板

```
<?php

class mymoduleActions extends sfActions
{
  public function executeMyAction()
  {
    $today = getdate();
    $this->hour = $today['hours'];
  }
}
```

例 4-7 - 模板能直接访问动作的属性

```
<p>Hello, world!</p>
<?php if ($hour >= 18): ?>
<p>Or should I say good evening? It's already <?php echo $hour ?
>.</p>
<?php endif; ?>
```

NOTE 有几个数据可以直接在模板中访问而不需要在动作里面设置。每个模板都可以执行$sf_contex,$sf_request,$sf_params 还有$sf_user 对象的方法。它们包含当前上下文、请求、请求参数还有 session 的信息。不久你就能学会怎么有效的利用它们。

从用户表单取得数据

表单是从用户取得信息的好方法。用 HTML 写表单的元素有时会很麻烦，特别是你想要 XHTML 兼容时。你可以按照平常的方式在 symfony 模板里面使用表单元素，如例 4-8 所示，不过 symfony 提供了一些辅助函数来简化这个任务。

例 4-8 - 模板可以包含普通的 HTML 代码

```
<p>Hello, world!</p>
<?php if ($hour >= 18): ?>
<p>Or should I say good evening? It's already <?php echo $hour ?
>.</p>
<?php endif; ?>
<form method="post" target="/myapp_dev.php/mymodule/anotherAction">
```

```
  <label for="name">What is your name?</label>
  <input type="text" name="name" id="name" value="" />
  <input type="submit" value="Ok" />
</form>
```

辅助函数是 symfony 定义的用在模板里的函数。它输出 HTML 代码从而节省你写 HTML 代码的时间。使用 symfony 辅助函数，你可以用例 4-9 的代码达到与例 4-8 同样的结果。

例 4-9 - 用辅助函数比写 HTML 标签更快更容易

```
<p>Hello, world!</p>
<?php if ($hour >= 18): ?>
<p>Or should I say good evening? It's already <?php echo $hour ?
>.</p>
<?php endif; ?>
<?php echo form_tag('mymodule/anotherAction') ?>
  <?php echo label_for('name', 'What is your name?') ?>
  <?php echo input_tag('name') ?>
  <?php echo submit_tag('Ok') ?>
</form>
```

SIDEBAR 辅助函数是来帮助你的。

如果，你认为在例 4-9 的例子里，辅助函数的版本没有写 HTML 快，看看这个例子：

```
     $card_list = array(
>       'VISA' => 'Visa',
>       'MAST' => 'MasterCard',
>       'AMEX' => 'American Express',
>       'DISC' => 'Discover');
>     echo select_tag('cc_type', options_for_select($card_list,
'AMEX'));
>     ?>
```

上面的代码的 HTML 输出如下：

```
<select name="cc_type" id="cc_type">
  <option value="VISA">Visa</option>
  <option value="MAST">MasterCard</option>
  <option value="AMEX" selected="selected">American Express</option>
  <option value="DISC">Discover</option>
</select>
```

在模板里使用辅助函数使编写代码的速度提高，代码更清晰，更简洁。唯一的代价是需要花时间学习他们，学习过程将一直持续到本书完结，到你在你习惯的编辑器中用快捷键写的时候。所以如果不会用 symfony 的辅助函数，你仍然可以继续使用 HTML 标签，不过这很浪费也很枯燥。

注意我们不推荐专业 web 开发者使用短开始标签(<?=,等效于 <?php echo)，因为你的正式服务器可能能够支持多种脚本语言而把短标签与别的脚本语言混淆。另外，短开始标签不是 PHP 的默认设置，需要打开才能使用。最后，如果你需要处理 XML 与验证，短标签会有问题因为<?在 XML 里有特殊含义。

由于 symfony 提供了很多辅助函数简化表单，表单处理需要一整章来讲解。表单处理详见第 10 章。

链接到另一个动作

我们已经讲到动作名与访问这个动作的 URL 之间需要有一个转换过程。所以如果你建立一个到 anotherAction 的链接，如例 4-10 所示，它只适用于默认的路由设置。如果以后你决定修改 URL 格式，那你还要修改所有包含这个链接的模板。

例 4-10 - 传统的超链接

```
<a href="/myapp_dev.php/mymodule/anotherAction?name=anonymous">
  I never say my name
</a>
```

为了避免这样的麻烦，请使用 link_to()辅助函数来建立所有的链接到应用程序内部的动作的超链接。例 4-11 演示了如何使用超链接辅助函数。

例 4-11 - link_to() 辅助函数

```
<p>Hello, world!</p>
<?php if ($hour >= 18): ?>
<p>Or should I say good evening? It's already <?php echo $hour ?
>.</p>
<?php endif; ?>
<?php echo form_tag('mymodule/anotherAction') ?>
  <?php echo label_for('name', 'What is your name?') ?>
  <?php echo input_tag('name') ?>
  <?php echo submit_tag('Ok') ?>
  <?php echo link_to('I never say my name','mymodule/anotherAction?
name=anonymous') ?>
</form>
```

上面的代码生成的HTML与前一个例子完全一样，但是如果修改路由规则，所有的模板会根据规则重新格式URL。

link_to()辅助函数，与很多辅助函数类似，接受另一个特殊的参数，这个参数用来传递HTML标签属性。例4-12是一个option属性的例子还有生成的HTML。option参数可以是一个数组或者一个简单的由几个key=value与空格组成的字符串。

例 4-12 - 大多数辅助函数有Option参数

```
// 用数组作option参数
<?php echo link_to('I never say my name', 'mymodule/anotherAction?
name=anonymous',
  array(
    'class'    => 'special_link',
    'confirm'  => 'Are you sure?',
    'absolute' => true
)) ?>

// 用字符串作option参数
<?php echo link_to('I never say my name', 'mymodule/anotherAction?
name=anonymous',
  'class=special_link confirm=Are you sure? absolute=true') ?>

// 结果一样
 => <a class="special_link" onclick="return confirm('Are you
sure?');"
    href="http://localhost/myapp_dev.php/mymodule/anotherAction/name/
anonymous">
    I never say my name</a>
```

任何使用symfony辅助函数输出HTML标签的时候，都可以在option参数中加入额外的属性(例如例4-12中的class属性)。你甚至可以用HTML 4.0的"快速而肮脏(quick-and-dirty)"的方式(不写双引号)，symfony会用漂亮的XHTML方式输出。这是用辅助函数比写HTML快的又一个原因。

NOTE 由于需要额外的解析与转换，字符串形式比数组要慢。

与其它辅助函数类似，链接辅助函数有好几种形式与参数。第9章将向你详细介绍这些内容。

从请求中取得信息

无论用户通过表单(通常是POST请求)还是通过URL(GET请求) 取得信息，你都可以在动作中通过sfActions对象的getRequestParameter()方法取得相关的数据。例4-13演示了如何在actionAction中取得name参数的值。

例 4-13 - 在动作中取得请求参数的值

```
<?php

class mymoduleActions extends sfActions
{
  ...

  public function executeAnotherAction()
  {
    $this->name = $this->getRequestParameter('name');
  }
}
```

如果数据操作很简单，你甚至不必用动作来取得参数值。模板可以直接通过$sf_params的get()方法来取得参数的值，类似于动作中的getRequestParameter()方法。

如果 executeAnotherAction() 方法是空的, 例 4-14 中的这种方法也可以从anotherActionSuccess.php模板中取到name参数的值。

例 4-14 - 直接从模板中取得参数的值

```
<p>Hello, <?php echo $sf_params->get('name') ?>!</p>
```

NOTE 为什么不直接使用$_POST,$_GET, 或 $_REQUEST 变量呢？因为如果你的URL的格式会变化(例如http://localhost/articles/europe/france/finance.html ，没有?或者=)，这样这些PHP变量就不管用了，只有路由系统能够取得请求参数。还有你可能需要输入过滤器来防止恶意代码注入，只有保持所有的参数使用一个干净的参数存储器的时候才能实现。

$sf_params对象的作用仅仅是数组的替代品。例如，如果你想判断一个请求参数是否存在，你可以只用$sf_params->has()方法而不必用get()方法取得实际的值，如例4-15。

例 4-15 - 在模板中判断一个参数是否存在

```
<?php if ($sf_params->has('name')): ?>
  <p>Hello, <?php echo $sf_params->get('name') ?>!</p>
<?php else: ?>
```

```
  <p>Hello, John Doe!</p>
<?php endif; ?>
```

你可能已经猜到这用一行代码就可以完成。与 symfony 里面的大多数 getter 方法一样，动作里的 getRequestParameter()还有模板里的$sf_params->get()方法(实际上两者调用的是同一个对象的同一个方法)可以有第二个参数：默认值，在参数不存在的时候起作用。

```
<p>Hello, <?php echo $sf_params->get('name', 'John Doe') ?>!</p>
```

总结

在 symfony 里面，页面由一个动作(actions/actions.class.php 文件里的一个方法，以 execute 开头)还有一个模板(templates/目录里的一个文件，通常以 Success.php 结尾)组成。功能有关联的页面组成模块。写模板有辅助函数帮忙，辅助函数是 symfony 提供的返回 HTML 代码的函数。并且你需要把 URL 考虑成回应的一部分，URL 也可以根据需要重新安排格式，所以你需要避免绕过超链接辅助方法直接写动作的 URL。

一旦了解了这些基本原理，你就可以开始用 symfony 写一个完整的 web 应用程序了。但是这需要花很长时间，因为几乎所有的功能都可以通过 symfony 的某种功能来简化开发……，所以这本书还没结束。