如果
luoln
1111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111

第 9 章 - 链接和路由系统

链接和 URL 在 web 应用程序框架中需要特别关注。因为这是应用程序的唯一进入点（前端控制器）；在模板中使用辅助函数可以让 URL 工作方式和他们的外观完全分离。这就叫做路由。路由可以使建立的应用程序对用户更友好更安全。本章会讲述在 symfony 应用程序里处理 URL 所需要了解的内容：

- 什么是路由系统及它是如何工作的
- 如何在模板中使用链接辅助函数让路由处理外部 URL
- 如何配置路由规则来改变 URL 的组成

你还会找到一些控制路由的性能和增加 最后的修饰小技巧。

什么是路由？

路由是一种为了显示更友好的 URL 而重写 URL 的机制。但要去了解他为何如此重要的之前，先需要花几分钟来理解什么是 URL。

URL 是服务器指令

URL 包含了从浏览器到服务器用来请求执行一个用户需求行为的信息。例如，一个传统的 URL 包含了脚本文件路径和一些请求所必要的参数，就像下面这个例子：

```
http://www.example.com/web/controller/article.php?
id=123456&format_code=6532
```

这个 URL 显示了关于应用程序结构和数据库的信息。开发者通常在界面里隐藏了应用程序的架构（例如，他们设置页标题“个人资料页”而不是“QZ7.65”）。应用程序内部重要信息的泄露是违背 URL 本意的，并会带来严重弊端：

- 技术数据出现在 URL 中会造成潜在的安全隐患。在前面的例子中，如果一个不怀好意的用户更改了 id 参数的值会发生什么呢？这是否意味着应用程序可以提供一个直接访问数据库的方法？或者要是用户为了好玩而尝试其他脚本名字，例如 admin.php？总而言之，原始的 URL 会让黑客轻易的破坏你的应用程序，这样无法进行安全控管。
- 不管在哪里出现费解的 URL 都会让人不安，他们会弱化内容的效果。如今，URL 不止出现在地址栏里。当用户把鼠标悬停在链接上，或者在搜

索结果中也会出现。当用户要查找信息时，他们希望得到的反馈是易于理解的，而不是如图 9-1 所示的混乱的 URL。

图 9-1 - URL 在很多地方显示，例如在搜索结果中

- 如果一条 URL 必须被改变（例如，如果脚本名或者它的一个参数需要改变），每一个链接到它的 URL 都要相应的更改。这就意味着修改控制结构是非常累人的和昂贵的，这不符合理想的敏捷开发。

如果 symfony 不使用前端控制器概念也许会变得更糟，更确切地说，如果应用程序包含了许多可以从 Internet 访问的脚本，在许多目录中，就如这些：

```
http://www.example.com/web/gallery/album.php?name=my%20holidays
http://www.example.com/web/weblog/public/post/list.php
http://www.example.com/web/general/content/page.php?name=about%20us
```

在这种情况下，开发者需要用文件结构来匹配 URL 结构，结果就会导致当结构改变的时候，维护将会非常困难。

URL 是界面的一部分

路由概念还有一层含义就是把 URL 看作界面的一部分。应用程序可以通过格式化的 URL 带给用户信息，用户也可以使用 URL 来访问应用程序的资源。

~~Symfony~~symfony 应用程序中这是可能实现的，因为展示给最终用户的 URL 与执行请求需要的服务器指令完全无关。相反，它涉及到资源的需求，并且他们可以自由的格式化。例如，symfony 可以理解下面的 URL 并使其显示和本章第一个 URL 同样的页面：

```
http://www.example.com/articles/finance/2006/activity-breakdown.html
```

这样有很大的优势：

- URL 实际上没有意义，但他们可以帮助用户来决定连接后的页面是否有他们期待的内容。一个连接能包含关于它所返回资源的额外细节。这对搜索引擎结果特别有用。此外，URL 有时候不会随着页面标题出现（试想在 email 里复制 URL），在这种情况下，你必须让它有一些含义。图 9-2 就是一个有意义的 URL 的例子。

图 9-2 URL 能传达关于页面的额外信息，如发布日期

symfony PHP5 framework » **AJAX pagination** made simple
The **AJAX pagination** demo uses two very simple actions, both passing a pager to their template: class pagerActions extends sfActions { public function ...
www.symfony-project.com/weblog/2006/07/17/**ajax-pagination**-made-simple.html - 12k - Cached - Similar pages

- 在纸上的URL会更容易输入和记住。如果你的公司网站在你的名片上写了http://www.example.com/controller/web/index.jsp?id=ERD4，这也许不会带来很多的访问者。
- URL本身可以成为一个命令行工具去执行命令或者获得信息。应用程序给高级用户提供了这种可以更快捷的可能性。

  ```
  // 结果列表：新增一个新的tag来限制结果列表
  http://del.icio.us/tag/symfony+ajax
  // 用户信息页：改变用户名来获得其他用户的信息
  http://www.askeet.com/user/francois
  ```

- 只要用一个简单的修改就能改变URL格式和动作名/参数。这就意味着你可以先开发程序，然后再确定URL格式，而不会把应用程序搞得一团糟。
- 甚至在你重组应用程序内部时，对外的URL依旧保持原样。这就可以使动态页面URL可以被收藏了。
- 搜索引擎在索引网站时倾向于忽略动态页面（结尾是.php,.asp之类）。所以格式化后的URL可以让搜索引擎认为是静态的内容，尽管实际上是动态的页面，因此就能更好的索引你的应用程序页面。
- 这会更安全。用户不能通过测试URL来浏览网页根文件结构，任何不能识别的URL会转向到一个开发者指定的页面，请求呼叫实际的脚本和它的参数都是隐藏的。

用户访问的URL和实际的脚本名和请求的参数都是由路由系统根据在配置文件中定义的规则来转义的。

NOTE 网页资源文件怎么办？幸运的是，网页资源文件的URL（图片、样式表和JavaScript)在浏览中不会出现很多，所以不需要对它们设置路由。在symfony中，所有的网页资源文件都位于web/目录下，他们的URL对应了他们在文件系统中的位置。然后，你能使用网页资源文件辅助函数来生成URL去控制动态网页资源（通过动作）。例如，要显示一个动态生成的图片，使用image_tag(url_for('captcha/image?key='.$key))。

它是如何工作的

~~Symfony~~symfony把外部URL和内部URL分离了。他们之间的联系是由路由系统定义的。为了简化处理，symfony为内部URL使用了和通用URL相似的语法。例9-1是一个示例：

例 9-1 - 外部URL和内部URL

```
// 内部URL语法
<module>/<action>[?param1=value1][&param2=value2][&param3=value3]...

// 内部URL示例，永远不会让最终用户看到
article/permalink?year=2006&subject=finance&title=activity-breakdown

// 外部URL示例， 显示给最终用户看的
http://www.example.com/articles/finance/2006/activity-breakdown.html
```

路由系统使用了一个特殊的配置文件routing.yml来定义路由规则。看一下例9-2所示的规则，它定义了用来把显示内容转换为articles/*/*/*的模式。

例 9-2 一个路由规则示例

```
article_by_title:
  url:    articles/:subject/:year/:title.html
  param:  { module: article, action: permalink }
```

每一个发送给symfony应用程序的请求首先会被路由系统分析（这很方便，因为每一个请求都由一个前端控制器来处理）。路由系统在请求URL和路由系统定义的规则模式中寻找匹配规则。如果匹配，通配符的名字变成请求的参数，并被合并为param:定义的关键字。例9-3显示了它是如何工作的。

例 9-3 - 路由系统解释输入的请求URL

```
// 用户输入（或点击）这个外部URL
http://www.example.com/articles/finance/2006/activity-breakdown.html

// 前端控制器确认它和article_by_title规则匹配
// 路由系统建立了以下请求参数
  'module'  => 'article'
  'action'  => 'permalink'
  'subject' => 'finance'
  'year'    => '2006'
  'title'   => 'activity-breakdown'
```

TIP 外部URL的.html扩展名只是一个简单的装饰，会被路由系统忽略掉。他只是让动态页面看上去和静态页面一样。你会在本章稍后的“路由配置”部分看到如何激活这个扩展。

接下来请求会传递给article模块的permalink动作，它用所有请求参数中的请求信息来决定显示哪个文章。

但是这个机制也能反向工作。为了让应用程序在自己的链接中显示外部URL，你必须提供给路由系统足够的数据让它确定使用哪个规则。你可以通过一个特

殊的辅助函数而不能直接用<a>标签来写超链接--这会完全绕过路由系统，如例 9-4 所示。

例 9-4 路由系统格式化模板中对外的URL

```
// 辅助函数 url_for()把内部URL转换为外部URL
<a href="<?php echo url_for('article/permalink?
subject=finance&year=2006&title=activity-breakdown') ?>">click here</
a>

// 辅助函数确认URL匹配article_by_title规则
// 路由系统由此建立了一个外部URL
 => <a href="http://www.example.com/articles/finance/2006/activity-
breakdown.html">click here</a>

// 辅助函数link_to()直接输出一个超链接
// 并避免PHP 和 HTML 的混淆
<?php echo link_to(
  'click here',
  'article/permalink?subject=finance&year=2006&title=activity-
breakdown'
) ?>

// 内部处理的时候，link_to()会调用url_for()，所以结果是一样的
=> <a href="http://www.example.com/articles/finance/2006/activity-
breakdown.html">click here</a>
```

因此，路由是一个双向机制，他只会在你使用link_to辅助函数来格式化你的链接时工作。

URL重写

有一个问题必须在更深入了解路由系统之前澄清一下。在前面部分的示例，并没有在内部URL中提到前端控制器（index.php或myapp_dev.php）。前端控制器决定环境，而不是应用程序的元素。所以，所有的链接必须是独立于环境之外的，同时前端控制器名永远不应该出现在内部URL中。

在生成的URL示例中也没有脚本名出现。这是因为生成的URL在默认生产环境中不包含任何脚本名。在settings.yml文件中的no_script_name参数精确的控制了前台控制器名是否显示在生成的URL中。如例 9-5 设置它为off状态，则每一个通过链接辅助函数输出的URL都不会有前端控制器名字。

例 9-5 - 是否在URL中显示前端控制器名字，设置在

apps/myapp/settings.yml

```
prod:
  .settings
    no_script_name:  off
```

现在，生成的URL看上去像这样：

http://www.example.com/index.php/articles/finance/2006/activity-breakdown.html

除了生产环境外的其他所有环境中，no_script_name参数默认是设置为off的。所以当你在开发环境中浏览应用程序的时候，URL中总是会有前端控制器名字。

http://www.example.com/myapp_dev.php/articles/finance/2006/activity-breakdown.html

在生产环境中，no_script_name设置为on，所以URL只显示了路由信息，这样看上去更友好。并且也没有技术信息显示。

http://www.example.com/articles/finance/2006/activity-breakdown.html

但是应用程序事如何知道去调用哪个前端控制器？这就是URL重写要做的事。网页服务器可以配置为如果URL中什么都没有的时候去调用一个指定的脚本。

在Apache中，当mod_rewrite扩展激活的时候是可以这么做的。每一个symfony项目都有一个.htaccess文件用来配置服务器web目录的mod_rewrite设置。默认的文件内容如例9-6所示。

例 9-6 在myproject/web/.htaccess中的Apache默认重写规则

```
<IfModule mod_rewrite.c>
  RewriteEngine On

  # we skip all files with .something
  RewriteCond %{REQUEST_URI} \..+$
  RewriteCond %{REQUEST_URI} !\.html$
  RewriteRule .* - [L]

  # we check if the .html version is here (caching)
  RewriteRule ^$ index.html [QSA]
  RewriteRule ^([^.]+)$ $1.html [QSA]
  RewriteCond %{REQUEST_FILENAME} !-f

  # no, so we redirect to our front web controller
```

```
  RewriteRule ^(.*)$ index.php [QSA,L]
</IfModule>
```

网页服务器检查收到的URL形态。如果URL不包含后缀或者如果没有此页面的缓存（第12章讲述了缓存），就交由index.php处理。

无论如何，symfony项目的web/目录是让项目中所有应用程序共享的。这就意味在web目录中通常有不止一个前端控制器。例如，项目有前台和后台应用程序，dev和prod环境，这样在web目录下就有四个前端控制器：

```
index.php          // 生产环境前台
frontend_dev.php   // 开发环境前台
backend.php        // 生产环境后台
backend_dev.php    // 开发环境后台
```

mod_rewrite设置只能指定一个默认的脚本名。如果在全部应用程序和环境中设置no_script_name为on，所有的URL都会解析给在prod环境中的frontend应用程序。这就是为什么在一个项目中URL重写只能针对一个环境中的一个应用程序。

TIP 还有一种方法可以允许多个应用程序没有脚本名。那就是在网页根目录下建立子目录，并把前端控制器放在里面。然后相应的更改SF_ROOT_DIR常量，并为每个需要的应用程序建立 .htaccess URL重写配置。

链接辅助函数

因为路由系统的原因，所以在你的模板中要使用链接辅助函数替换掉<a>标签。别为这个烦恼，这是一个让你的应用程序保持干净，并易于维护的机会。除此之外，链接辅助函数提供了一些你不应该忽视的非常有用的快捷方式。

超链接，按钮和表单

你已经知道link_to()辅助函数。它输出一个XHTML兼容的超链接，它有2个参数：可以点击的元素和所指向的资源的内部URL。如果你想用一个按钮来替代超链接，应该使用button_to()辅助函数。表单也有一个辅助函数来管理action属性的值。你会在下一段学习表单。例9-7展示了一些链接辅助函数的例子。

例 9-7 - <a>, <input>和<form>标签的链接辅助函数

```
// 字符串的超链接
<?php echo link_to('my article', 'article/read?
title=Finance_in_France') ?>
 => <a href="/routed/url/to/Finance_in_France">my article</a>
```

```
// 图片的超链接
<?php echo link_to(image_tag('read.gif'), 'article/read?
title=Finance_in_France') ?>
 => <a href="/routed/url/to/Finance_in_France"><img
src="/images/read.gif" /></a>

// 按钮标签
<?php echo button_to('my article', 'article/read?
title=Finance_in_France') ?>
 => <input value="my article"
type="button"onclick="document.location.href='/routed/url/to/Finance_
in_France';" />

// 表单标签
<?php echo form_tag('article/read?title=Finance_in_France') ?>
 => <form method="post" action="/routed/url/to/Finance_in_France" />
```

链接辅助函数能接受内部URL和绝对URL(http://开始，被路由系统忽略)还有定位符。注意在真正的应用程序中，内部URL是由动态参数建立的。例9-8是这些情况的例子。

例 9-8 - 链接辅助函数接受的URL

```
// 内部URL
<?php echo link_to('my article', 'article/read?
title=Finance_in_France') ?>
 => <a href="/routed/url/to/Finance_in_France">my article</a>

// 带有动态参数的内部URL
<?php echo link_to('my article', 'article/read?title='.$article-
>getTitle()) ?>

// 带有定位符的内部URL
<?php echo link_to('my article', 'article/read?
title=Finance_in_France#foo') ?>
 => <a href="/routed/url/to/Finance_in_France#foo">my article</a>

// 绝对URL
<?php echo link_to('my article',
'http://www.example.com/foobar.html') ?>
 => <a href="http://www.example.com/foobar.html">my article</a>
```

链接辅助函数选项

如第 7 章所述，辅助函数能接受额外的选项参数，甚至可以是一个数组或者一个字符串。这对链接辅助函数也一样，如例 9-9 所示。

例 9-9 - 链接辅助函数接受额外的选项

```
// 数组作为额外选项
<?php echo link_to('my article', 'article/read?
title=Finance_in_France', array(
  'class'  => 'foobar',
  'target' => '_blank'
)) ?>

// 字符串作为额外选项（同样结果）
<?php echo link_to('my article', 'article/read?
title=Finance_in_France','class=foobar target=_blank') ?>
 => <a href="/routed/url/to/Finance_in_France" class="foobar"
target="_blank">my article</a>
```

也可以给链接辅助函数增加一个 symfony 特有的选项：confirm 和 popup。第一个会使链接在被点击时弹出一个 JavaScript 确认对话框，第二个会让连接在新窗口中打开，如例 9-10 所示。

例 9-10 - 链接辅助函数的'confirm'和'popup'选项

```
<?php echo link_to('delete item', 'item/delete?id=123', 'confirm=Are
you sure?') ?>
 => <a onclick="return confirm('Are you sure?');"
       href="/routed/url/to/delete/123.html">add to cart</a>

<?php echo link_to('add to cart', 'shoppingCart/add?id=100',
'popup=true') ?>
 => <a onclick="window.open(this.href);return false;"
       href="/fo_dev.php/shoppingCart/add/id/100.html">add to
cart</a>

<?php echo link_to('add to cart', 'shoppingCart/add?id=100', array(
  'popup' => array('Window title',
'width=310,height=400,left=320,top=0')
)) ?>
 => <a onclick="window.open(this.href,'Window
title','width=310,height=400,left=320,top=0');return false;"
       href="/fo_dev.php/shoppingCart/add/id/100.html">add to
cart</a>
```

可以一起使用这些选项。

伪装的 GET 和 POST 选项

有时候网页开发者使用一个 GET 请求来执行一个 POST 操作。例如，看一下下面的 URL：

http://www.example.com/index.php/shopping_cart/add/id/100

这个请求会改变应用程序中的数据，他会在购物车对象中增加一个项目并保存在用户会话或者数据库中。这个 URL 可以被收藏，缓存，或者被搜索引擎索引。想一下使用这个技术会对数据库产生作用的负面影响。实际上这个请求应被当作 POST 来处理，因为搜索引擎机器人不会在索引的时候执行一个 POST 的请求。

~~Symfony~~symfony 提供了一个方法用来转换 link_to()或者 button_to()辅助函数的调用为一个实际的 POST。只要增加 post=true 选项，如例 9-11 所示。

例 9-11 - 让一个链接调用一个 POST 请求

```
<?php echo link_to('go to shopping cart', 'shoppingCart/add?id=100',
'post=true') ?>
 => <a onclick="f = document.createElement('form');
document.body.appendChild(f);
                f.method = 'POST'; f.action = this.href;
f.submit();return false;"
       href="/shoppingCart/add/id/100.html">go to shopping cart</a>
```

这个<a>标签有一个 href 属性同时浏览器没有 Javascript 支持，如搜索引擎机器人，会根据这个链接执行一个默认的 GET 动作。因此你必须通过增加一些限制让你的动作只会对 POST 方法有响应，如接下来所示：

```
$this->forward404If($request->getMethod() != sfRequest::POST);
```

请确定不要在表单里使用这个选项，这是因为它会生成自己的<form>标签。

把会修改数据的链接标记成 POST 是一个好习惯。

强制设置请求参数为 GET 变量

根据你的路由规则，变量通过 link_to()转换为参数，同时被模式化。如果在 routing.yml 文件中无法找到能匹配内部 URL 的规则时，默认的转换规则会把 module/action?key=value 转换为 /module/action/key/value，如例 9-12 所示。

例 9-12 - 默认的路由规则

```
<?php echo link_to('my article', 'article/read?
title=Finance_in_France') ?>
=> <a href="/article/read/title/Finance_in_France">my article</a>
```

如果你确实需要保留 GET 语法（通过?key=val 方式传递参数的话）你应该把参数放在 URL 参数外，也就是 query_string 选项中。所有的链接辅助函数都支持这个选项，就如例 9-13 所示。

例 9-13 - 用 query_string 选项强制指定 GET 变量

```
<?php echo link_to('my article', 'article/read?
title=Finance_in_France', array(
  'query_string' => 'title=Finance_in_France'
)) ?>
=> <a href="/article/read?title=Finance_in_France">my article</a>
```

一个包含表现为 GET 变量的请求参数的 URL 可以被客户端的脚本解释，在服务端可以通过$_GET 和$_REQUEST 变量访问。

SIDEBAR 网页资源文件辅助函数（Asset helpers）

第 7 章介绍了网页资源文件辅助函数 image_tag(), stylesheet_tag(), 和 javascript_include_tag()，分别用来让你在回应中包含图片，样式表，或者 JavaScript 文件。这些网页资源文件路径并不包含在路由系统中，因为他们链接的资源实际上都在公共网页目录中。

你不用去介意网页资源文件的文件扩展名。~~Symfony~~symfony 会在调用图片，JavaScript,或者样式表辅助函数时自动增加.png, .js, 或 .css 后缀。同样的，symfony 会自动在 web/images/, web/js/, 和 web/css/目录中自动搜索网页资源文件。当然，如果你想要包含一个特殊的文件格式或者位于一个特殊位置的文件，只要使用文件全名或者文件完全路径作为变量即可。如果你的媒体文件有一个清楚名字的话，不要去管 alt 属性，因为 symfony 会为你处理的。

```
<?php echo image_tag('test') ?>
<?php echo image_tag('test.gif') ?>
<?php echo image_tag('/my_images/test.gif') ?>

 => <img href="/images/test.png" alt="Test" />
    <img href="/images/test.gif" alt="Test" />
    <img href="/my_images/test.gif" alt="Test" />
```

用 size 属性来固定图像的大小。他需要用像素来定义长和宽，用 x 分割。

```
<?php echo image_tag('test', 'size=100x20')) ?>

 => <img href="/images/test.png" alt="Test" width="100" height="20"/>
```

如果你想让网页资源文件包含在<head>部分中（JavaScript 文件和样式表），你在模板中应该使用 use_stylesheet() 和 use_javascript()辅助函数，而不是_tag()。他们会为回应增加网页资源文件，这些网页资源文件会在</head>之间被包含，并发送给浏览器。

使用绝对路径

链接和网页资源文件辅助函数默认生成的是相对路径。要强制输出绝对路径的话，需要设置 absolute 选项为 true，如例 9-14 所示。在链接一个 email 信息，RSS 种子或者一个 API 的时候这个技巧非常有用。

例 9-14 - 获得绝对 URL 来替代相对 URL

```
<?php echo url_for('article/read?title=Finance_in_France') ?>
 => '/routed/url/to/Finance_in_France'
<?php echo url_for('article/read?title=Finance_in_France', true) ?>
 => 'http://www.example.com/routed/url/to/Finance_in_France'

<?php echo link_to('finance', 'article/read?title=Finance_in_France')
?>
 => <a href="/routed/url/to/Finance_in_France">finance</a>
<?php echo link_to('finance', 'article/read?
title=Finance_in_France','absolute=true') ?>
 => <a href="
http://www.example.com/routed/url/to/Finance_in_France">finance</a>

// 同样的情况也适合用 asset 辅助函数处理
<?php echo image_tag('test', 'absolute=true') ?>
<?php echo javascript_include_tag('myscript', 'absolute=true') ?>
```

SIDEBAR 邮件辅助函数

如今，网络中到处都是 email 搜集机器人，所以你不能直接把 email 地址显示在网页上，那会让你很快就变为垃圾邮件的受害者。这就是为什么 symfony 提供了 mail_to()辅助函数。

mail_to()辅助函数使用了 2 个参数：实际的 email 地址和显示的字符串。附加选项有 encode 参数，用来输出一个无法在 HTML 源码中阅读的，机器人不认识

的，但是可以被浏览器阅读的字符串。

```
<?php echo mail_to('myaddress@mydomain.com', 'contact') ?>
 => <a href="mailto:myaddress@mydomain.com'>contact</a>
<?php echo mail_to('myaddress@mydomain.com', 'contact',
'encode=true') ?>
 => <a href="&#109;&#x61;... &#111;&#x6d;">&#x63;&#x74;...
e&#115;&#x73;</a>
```

经过编码的 email 信息是由随机十进制和十六进制转换过的字符组成的。这个技巧让大多数如今的地址收集器无法工作，但是要注意的是搜集技术更新的非常快。

路由配置

路由系统做两件事：

- 他分析输入请求中的外部 URL 并转换到内部 URL 格式来决定使用哪个模快/动作和请求参数。
- 他把内部 URL 格式的链接格式化成为外部 URL（通过使用链接辅助函数）

转换基于一系列的路由规则。这些规则放在应用程序的 config/目录中的 routing.yml 配置文件中。例 9-15 展示了每一个 symfony 项目中都有的默认路由规则。

例 9-15 - 在 myapp/config/routing.yml 中的默认的路由规则

```
# 默认规则
homepage:
  url:   /
  param: { module: default, action: index }

default_symfony:
  url:   /symfony/:action/*
  param: { module: default }

default_index:
  url:   /:module
  param: { action: index }

default:
  url:   /:module/:action/*
```

规则和模式

路由规则是在外部URL和内部URL之间双向工作的。基本的规则有以下几点组成：

- 一个唯一的，易于识别的，快速的，可被链接辅助函数使用的标记
- 一个可以被匹配的模式(url 键)
- 一个请求参数数组(param 键)

模式能包含通配符（用*来表示）模式名也能为通配符（由冒号：开始）。一条和模式名通配符匹配的记录会转换为请求参数。例如，在例9-15中定义的default规则会匹配任何包含/foo/bar的URL，并设置module参数为foo，action参数为bar。在default_symfony规则中，symfony是关键字，action是具名通配符参数。

路由系统由顶至底来分析routing.yml文件，遇到匹配的就停止。这就是为什么要把自定的规则放在默认规则前面。例如，URL_/foo/123与例9-16中的两个定义都匹配，但是symfony会用my_rule:，因为他甚至都没有测试到default:规则。这个请求由mymodule/myaction动作处理，设置bar为123（而不是foo/123动作）。

例 9-16 - 规则由顶至底解析

```
my_rule:
  url:   /foo/:bar
  param: { module: mymodule, action: myaction }

# 默认规则
default:
  url:   /:module/:action/*
```

NOTE 当一个新的动作建立的时候，这并不意味着你必须为它建立一个路由规则。如果默认的模块/动作模式适合你的话，就不需要考虑routing.yml文件了。不管怎样，如果你想自定义动作的外部URL，就在默认的规则上面增加一条新的规则。

例9-17显示了一个为article/read 动作更改外部URL格式的过程。

例 9-17 - 为acticle/read 动作更改外部URL格式

```
<?php echo url_for('my article', 'article/read?id=123) ?>
 => /article/read/id/123        // 默认格式

// 要改为/article/123的话
// 在你的routing.yml文件开头增加一个新的规则
article_by_id:
```

```
  url:   /article/:id
  param: { module: article, action: read }
```

问题是例 9-17 中的 article_by_id 规则打破了所有 article 模组的默认路由规则。事实上，一个类似 article/delete 的 url 会匹配这个规则而不是 default，并会调用 read 动作并把 id 设置为 delete 而不是调用 delete 动作。要处理这个问题的话，你必须增加一个模式来限制 article_by_id 规则，让其只在 URL 的 id 是一个数字的时候才会去匹配。

模式限制

当一个 URL 可以匹配多于一个规则的时候，你必须为模式增加限制或者需求来重定义规则。一个需求是一组正则表达式组成的，他必须能用通配符来匹配规则。

例如，要修改 article_by_id 规则，让它只匹配 URL 的 id 参数是数字的情况，则需要在规则中增加一项，如例 9-18 所示。

例 9-18 - 路由规则中增加一个需求

```
article_by_id:
  url:   /article/:id
  param: { module: article, action: read }
  requirements: { id: \d+ }
```

现在，article/delete_URL 不会再匹配 article_by_id 规则了，因为'delete'字符串与需求不匹配。因此，路由系统会继续寻找其他的规则，最终会找到 default 规则。

SIDEBAR 永久链接（Permalinks）

一个好的路由安全原则是隐藏主键并把他们替换为尽可能有意义的字符串。要是你想用文章名字而不是他们的 ID 来显示文章内容会怎么样？这会让外部 URL 看上去像这样：

```
http://www.example.com/article/Finance_in_France
```

为了这个扩展，你必须建立一个新的 permalink 动作，它使用了一个 slug 参数替代了 id 参数，并为此增加一个新的规则：

```
article_by_id:
  url:          /article/:id
  param:        { module: article, action: read }
  requirements: { id: \d+ }
```

```
article_by_slug:
  url:          /article/:slug
  param:        { module: article, action: permalink }
```

permalink 动作需要由文章主题来确定请求，因此你的模型必须提供相应的方法。

```
public function executePermalink()
{
  $article = ArticlePeer::retrieveBySlug($this->getRequestParameter('slug');
  $this->forward404Unless($article);  // 如果没有文章匹配 slug 则显示 404 错误
  $this->article = $article;          // 把对象传递给模板
}
```

你同时需要在模板中把 read 动作的链接替换为 permalink 的链接，并激活相应的内部 URL 格式。

```
// 替换
<?php echo link_to('my article', 'article/read?id='.$article->getId()) ?>

// 为
<?php echo link_to('my article', 'article/permalink?slug='.$article->getSlug()) ?>
```

由于有了 requirements 行，尽管 article_by_id 规则在前面，一个外部的 URL 类似_/article/Finance_in_France 还是会匹配 article_by_slug 规则。

注意 slug 将获得文章，因此你必须在 Article 模型描述中增加一个索引给 slug 列来优化数据库性能。

设置默认的值

你可以给具名通配符一个默认值让规则工作，甚至这个参数是没有定义的。在 param:数组中设置默认的值。

例如，如果没有设置 id 参数则不会匹配 article_by_id 规则。你能强制设置它，如例 9-19 所示。

例 9-19 - 为通配符设置一个默认值

```
article_by_id:
  url:          /article/:id
  param:        { module: article, action: read,
id: 1
 }
```

在模式中默认参数不需要找到通配符。在例 9-20 中，display 参数设置为 true，尽管他不会出现在 URL 中。

例 9-20 - 为请求参数设置默认值

```
article_by_id:
  url:          /article/:id
  param:        { module: article, action: read, id: 1, display: true
}
```

如果仔细观察，你能看到 article 和 read 是 module 和 action 的默认值，虽然他们没有在模式中出现。

TIP 你可以在 sf_routing_default 配置参数中为所有路由规则定义一个默认的参数。例如，如果你想让所有的规则都有一个 theme 参数并被默认设置为 default 的话，在你的应用程序的 config.php 中增加一行 sfConfig::set('sf_routing_defaults', array('theme' => 'default')); 。

使用规则名字来加快路由速度

如果规则记号带有标记（@）的话，链接辅助函数接受一个规则记号来替换掉模块/动作，如例 9-21 所示。

例 9-21 - 用规则标签来代替模块/动作

```
<?php echo link_to('my article', 'article/read?id='.$article-
>getId()) ?>

// 也能这样写
<?php echo link_to('my article', '@article_by_id?id='.$article-
>getId()) ?>
```

这个技巧既有优点也有缺点。这样写有以下好处：

- 内部 URL 格式完成的更快，因为 symfony 不必去浏览所有的规则来匹配到这个链接。在一个有大量路由格式的超链接的页面，如果使用规则标签来替代模块/动作的话就会快很多。
- 使用规则标签有助于抽象动作后的逻辑。如果决定改变动作名但要保持

URL 不变，只要稍微修改一下 routing.yml 文件即可。所有的 link_to() 调用依旧可以工作而不用做其他修改。

- 用规则名会让调用的逻辑更显而易见。尽管你的模块和动作有一个详细的名字，调用@display_article_by_slug 还是比 article/display 要好。

另一方面，一个不好的地方是新增超链接不是那么显而易见，因为你总是需要参考 routing.yml 文件来找到动作需要使用哪个标签。

最优的选择取决于项目。最终，这取决于你。

TIP 在测试过程中（在 dev 环境中），如果你想在浏览器中检查哪个规则匹配一个请求（request）， 可以在网页调试工具条中的_logs and msgs_查找类似 matched route XXX 的日志信息 。你会在第 16 章找到关于网页调试模式更多的信息。

增加.html 扩展名

比较这两个 URL：

```
http://myapp.example.com/article/Finance_in_France
http://myapp.example.com/article/Finance_in_France.html
```

尽管这是同一个页面，用户和（机器人）也许会因为他们的 URL 不同而认为他们是不同的页面。第二个 URL 是一个深层次的组织完好的静态页面网页目录，可以让搜索引擎知道如何去索引的一个结构。

用路由系统给每一个外部 URL 生成时增加一个后缀，要在应用程序的 settings.yml 中设置 suffix 值，如例 9-22 所示。

例 9-22 - 在 myapp/config/settings.yml 为所有 URL 设置一个后缀

```
prod:
  .settings
    suffix:         .html
```

默认的后缀设置是一个点(.)，这意味着路由系统除非特别指定一般不需要增加后缀。

有时候需要为一个特定的路由规则指定一个后缀。在这种情况下，在 routing.yml 文件中相对 url:行中直接设置一个后缀，如例 9-23 所示。这样全局后缀设置就会被忽略。

例 9-23 - 在 myapp/config/routing.yml 为一个 URL 设置后缀

```
article_list:
```

```
  url:          /latest_articles
  param:        { module: article, action: list }

article_list_feed:
  url:          /latest_articles.rss
  param:        { module: article, action: list, type: feed }
```

不使用 routing.yml 创建规则

与大多数配置文件一样，routing.yml 是一个定义路由规则的解决方案，但不是唯一的方案。你可以在 PHP 中定义规则，也可以在应用程序的 config.php 文件中定义，或者在前端控制器脚本中定义，但必须在调用 dispatch()函数前定义，因为这个方法根据现行的路由规则来确定执行的动作。在 PHP 中允许建立动态规则，这取决于你的配置和其他参数。

处理路由规则的对象是 sfRouting 单例（singleton）。只要加载 sfRouting::getInstance()，那么每一段代码中都会有路由工作。它的 prependRoute()方法在 routing.yml 文件的已存定义上面增加了一条规则。它需要 4 个参数，就是用来定义一条规则需要的参数：一个路由标签，模式，数组的默认值和所需的其他数组。例如，例 9-18 中的 routing.yml 规则定义和例 9-24 中的 PHP 代码是一样的。

例 9-24 - 在 PHP 中定义一个规则

```
sfRouting::getInstance()->prependRoute(
  'article_by_id',                                  // 路由名
  '/article/:id',                                   // 路由模式
  array('module' => 'article', 'action' => 'read'), // 默认值
  array('id' => '\d+'),                             // 需求
);
```

sfRouting 单例（singleton）还有其他有用的方法来手动处理路由：clearRoutes(), hasRoutes(), getRoutesByName()和其他的。要了解更多资料，可以参考 API 文档(http://www.symfony-project.com/api/symfony.html)。

TIP 当你开始完全理解本书中所展示的理念时，你能通过浏览在线 API 文档或者其他更好的 symfony 源文件来增加对框架的理解。本书中没有描述所有 symfony 的 tweaks 和参数。在线文档却有更详尽的叙述。

在动作中处理路由

如果你需要获得关于当前路由的信息--例如，准备一个功能"返回到 xxx 页"链接--你需要使用 sfRouting 对象的方法。getCurrentInternalUri()方法返回的

URL 可以被 link_to()辅助函数调用，就如在例 9-25 中显示的。

例 9-25 - 使用 sfRouting 来获得关于当前路由的信息

```
// 如果这是你想要的一个 URL
http://myapp.example.com/article/21

// 在 article/read 动作中使用下面的语句
$uri = sfRouting::getInstance()->getCurrentInternalUri();
 => article/read?id=21

$uri = sfRouting::getInstance()->getCurrentInternalUri(true);
 => @article_by_id?id=21

$rule = sfRouting::getInstance()->getCurrentRouteName();
 => article_by_id

// 如果只需要当前的模块/动作名，
// 记住他们只是真实 request 参数
$module = $this->getRequestParameter('module');
$action = $this->getRequestParameter('action');
```

如果需要在一个动作中转换一个内部 URL 为外部 URL（就如 url_for()在模板中做的一样）在 sfController 对象中用 genUrl()方法，就如在例 9-26 中显示的。

例 9-26 - 使用 sfController 来转换一个内部 URI

```
$uri = 'article/read?id=21';

$url = $this->getController()->genUrl($uri);
 => /article/21

$url = $this->getController()->genUrl($uri, true);
=> http://myapp.example.com/article/21
```

总结

路由是一种双向的机制,目的是为了格式化外部 URL 使他们更友好。URL 重写允许在每个项目中的一个应用程序的 URL 中省略前端控制器名字。如果想要在路由系统双向工作的话，你必须在每次模板中需要输出一个 URL 的时候都使用链接辅助函数。在 routing.yml 文件中使用按照流程顺序和规则需求的方式来配置路由系统的规则。settings.yml 文件中包含了关于前端控制器名字和在外部

URL 中可能的后缀的附加配置。