Routing

2.8 version

edit this page (https://github.com/symfony/sym docs/edit/2.8/routing.rst)

Beautiful URLs are an absolute must for any serious web application. This means leaving behind ugly URLs like `index.php?article_id=57` in favor of something like `/read/intro-to-symfony`.

Having flexibility is even more important. What if you need to change the URL of a page from `/blog` to `/news`? How many links should you need to hunt down and update to make the change? If you're using Symfony's router, the change is simple.

The Symfony router lets you define creative URLs that you map to different areas of your application. By the end of this chapter, you'll be able to:

- Create complex routes that map to controllers
- Generate URLs inside templates and controllers
- Load routing resources from bundles (or anywhere else)
- Debug your routes

Routing Examples ¶

A *route* is a map from a URL path to a controller. For example, suppose you want to match any URL like `/blog/my-post` or `/blog/all-about-symfony` and send it to a controller that can look up and render that blog entry. The route is simple:

Annotations YAML XML PHP

```php
// src/AppBundle/Controller/BlogController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class BlogController extends Controller
{
    /**
     * Matches /blog exactly
     *
     * @Route("/blog", name="blog_list")
     */
    public function listAction()
    {
        // ...
    }

    /**
     * Matches /blog/*
     *
     * @Route("/blog/{slug}", name="blog_show")
     */
    public function showAction($slug)
    {
        // $slug will equal the dynamic part of the URL
        // e.g. at /blog/yay-routing, then $slug='yay-routing'

        // ...
    }
}
```

Thanks to these two routes:

- If the user goes to `/blog`, the first route is matched and `listAction()` is executed;
- If the user goes to `/blog/*`, the second route is matched and `showAction()` is executed.

Because the route path is `/blog/{slug}`, a `$slug` variable is passed to `showAction()` matching that value. For example, if the user goes to `/blog/yay-routing`, then `$slug` will equal `yay-routing`.

Whenever you have a `{placeholder}` in your route path, that portion becomes a wildcard: it matches *any* value. Your controller can now *also* have an argument called `$placeholder` (the wildcard and argument names *must* match).

Each route also has an internal name: `blog_list` and `blog_show`. These can be anything (as long as each is unique) and don't have any meaning yet. Later, you'll use it to generate URLs.

Routing in Other Formats

The `@Route` above each method is called an *annotation*. If you'd rather configure your routes in YAML, XML or PHP, that's no problem!

In these formats, the `_controller` "defaults" value is a special key that tells Symfony which controller should be executed when a URL matches this route. The `_controller` string is called the logical name. It follows a pattern that points to a specific PHP class and method, in this case the `AppBundle\Controller\BlogController::listAction` and `AppBundle\Controller\BlogController::showAction` methods.

This is the goal of the Symfony router: to map the URL of a request to a controller. Along the way, you'll learn all sorts of tricks that make mapping even the most complex URLs easy.

Adding {wildcard} Requirements ¶

Imagine the `blog_list` route will contain a paginated list of blog posts, with URLs like `/blog/2` and `/blog/3` for pages 2 and 3. If you change the route's path to `/blog/{page}`, you'll have a problem:

- blog_list: `/blog/{page}` will match `/blog/*`;
- blog_show: `/blog/{slug}` will *also* match `/blog/*`.

When two routes match the same URL, the *first* route that's loaded wins. Unfortunately, that means that `/blog/yay-routing` will match the `blog_list`. No good!

To fix this, add a *requirement* that the `{page}` wildcard can *only* match numbers (digits):

Annotations YAML XML PHP

```
// src/AppBundle/Controller/BlogController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class BlogController extends Controller
{
    /**
     * @Route("/blog/{page}", name="blog_list", requirements={"page": "\d+"})
     */
    public function listAction($page)
    {
        // ...
    }

    /**
     * @Route("/blog/{slug}", name="blog_show")
     */
    public function showAction($slug)
    {
        // ...
    }
}
```

The `\d+` is a regular expression that matches a *digit* of any length. Now:

URL	Route	Parameters
/blog/2	blog_list	$page = 2
/blog/yay-routing	blog_show	$slug = yay-routing

To learn about other route requirements - like HTTP method, hostname and dynamic expressions - see How to Define Route Requirements (routing/requirements.html).

Giving {placeholders} a Default Value ¶

In the previous example, the `blog_list` has a path of `/blog/{page}`. If the user visits `/blog/1`, it will match. But if they visit `/blog`, it will **not** match. As soon as you add a `{placeholder}` to a route, it *must* have a value.

So how can you make `blog_list` once again match when the user visits `/blog`? By adding a *default* value:

Annotations YAML XML PHP

```
// src/AppBundle/Controller/BlogController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class BlogController extends Controller
{
    /**
     * @Route("/blog/{page}", name="blog_list", requirements={"page": "\d+"})
     */
    public function listAction($page = 1)
    {
        // ...
    }
}
```

Now, when the user visits `/blog`, the `blog_list` route will match and `$page` will default to a value of `1`.

Advanced Routing Example ¶

With all of this in mind, check out this advanced example:

Annotations YAML XML PHP

```
// src/AppBundle/Controller/ArticleController.php

// ...
class ArticleController extends Controller
{
    /**
     * @Route(
     *     "/articles/{_locale}/{year}/{slug}.{_format}",
     *     defaults={"_format": "html"},
     *     requirements={
     *         "_locale": "en|fr",
     *         "_format": "html|rss",
     *         "year": "\d+"
     *     }
     * )
     */
    public function showAction($_locale, $year, $slug)
    {
    }
}
```

As you've seen, this route will only match if the `{_locale}` portion of the URL is either `en` or `fr` and if the `{year}` is a number. This route also shows how you can use a dot between placeholders instead of a slash. URLs matching this route might look like:

- `/articles/en/2010/my-post`
- `/articles/fr/2010/my-post.rss`
- `/articles/en/2013/my-latest-post.html`

The Special _format Routing Parameter

This example also highlights the special `_format` routing parameter. When using this parameter, the matched value becomes the "request format" of the `Request` object.

Ultimately, the request format is used for such things as setting the `Content-Type` of the response (e.g. a `json` request format translates into a `Content-Type` of `application/json`). It can also be used in the controller to render a different template for each value of `_format`. The `_format` parameter is a very powerful way to render the same content in different formats.

In Symfony versions previous to 3.0, it is possible to override the request format by adding a query parameter named `_format` (for example: `/foo/bar?_format=json`). Relying on this behavior not only is considered a bad practice but it will complicate the upgrade of your applications to Symfony 3.

Sometimes you want to make certain parts of your routes globally configurable. Symfony provides you with a way to do this by leveraging service container parameters. Read more about this in "How to Use Service Container Parameters in your Routes (routing/service_container_parameters.html)".

Special Routing Parameters ¶

As you've seen, each routing parameter or default value is eventually available as an argument in the controller method. Additionally, there are three parameters that are special: each adds a unique piece of functionality inside your application:

`_controller`
As you've seen, this parameter is used to determine which controller is executed when the route is matched.

`_format`
Used to set the request format (read more).

`_locale`
Used to set the locale on the request (read more (translation/locale.html#translation-locale-url)).

Controller Naming Pattern ¶

If you use YAML, XML or PHP route configuration, then each route must have a `_controller` parameter, which dictates which controller should be executed when that route is matched. This parameter uses a simple string pattern called the *logical controller name*, which Symfony maps to a specific PHP method and class. The pattern has three parts, each separated by a colon:

bundle*:*controller*:*action

For example, a `_controller` value of `AppBundle:Blog:show` means:

Bundle	Controller Class	Method Name
`AppBundle`	`BlogController`	`showAction()`

The controller might look like this:

```php
// src/AppBundle/Controller/BlogController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class BlogController extends Controller
{
    public function showAction($slug)
    {
        // ...
    }
}
```

Notice that Symfony adds the string `Controller` to the class name (`Blog => BlogController`) and `Action` to the method name (`show => showAction()`).

You could also refer to this controller using its fully-qualified class name and method: `AppBundle\Controller\BlogController::showAction`. But if you follow some simple conventions, the logical name is more concise and allows more flexibility.

> In addition to using the logical name or the fully-qualified class name, Symfony supports a third way of referring to a controller. This method uses just one colon separator (e.g. `service_name:indexAction`) and refers to the controller as a service (see How to Define Controllers as Services (controller/service.html)).

Loading Routes ¶

Symfony loads all the routes for your application from a *single* routing configuration file: `app/config/routing.yml`. But from inside of this file, you can load any *other* routing files you want. In fact, by default, Symfony loads annotation route configuration from your AppBundle's `Controller/` directory, which is how Symfony sees our annotation routes:

YAML XML PHP

```yaml
# app/config/routing.yml
app:
    resource: "@AppBundle/Controller/"
    type:     annotation
```

For more details on loading routes, including how to prefix the paths of loaded routes, see How to Include External Routing Resources (routing/external_resources.html).

Generating URLs ¶

The routing system should also be used to generate URLs. In reality, routing is a bidirectional system: mapping the URL to a controller and a route back to a URL.

To generate a URL, you need to specify the name of the route (e.g. `blog_show`) and any wildcards (e.g. `slug = my-blog-post`) used in the path for that route. With this information, any URL can easily be generated:

```php
class MainController extends Controller
{
    public function showAction($slug)
    {
        // ...

        // /blog/my-blog-post
        $url = $this->generateUrl(
            'blog_show',
            array('slug' => 'my-blog-post')
        );
    }
}
```

The `generateUrl()` method defined in the base `Controller` (http://api.symfony.com/2.8/Symfony/Bundle/FrameworkBundle/Controller/Controller.html) class is just a shortcut for this code:

```
$url = $this->container->get('router')->generate(
    'blog_show',
    array('slug' => 'my-blog-post')
);
```

Generating URLs with Query Strings ¶

The `generate` method takes an array of wildcard values to generate the URI. But if you pass extra ones, they will be added to the URI as a query string:

```
$this->get('router')->generate('blog', array(
    'page' => 2,
    'category' => 'Symfony'
));
// /blog/2?category=Symfony
```

Generating URLs from a Template ¶

To generate URLs inside Twig, see the templating chapter: Linking to Pages (templating.html#templating-pages). If you also need to generate URLs in JavaScript, see How to Generate Routing URLs in JavaScript (routing/generate_url_javascript.html).

Generating Absolute URLs ¶

By default, the router will generate relative URLs (e.g. `/blog`). From a controller, pass `UrlGeneratorInterface::ABSOLUTE_URL` to the third argument of the `generateUrl()` method:

```
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

$this->generateUrl('blog_show', array('slug' => 'my-blog-post'), UrlGeneratorInterface::ABSO
// http://www.example.com/blog/my-blog-post
```

> The host that's used when generating an absolute URL is automatically detected using the current `Request` object. When generating absolute URLs from outside the web context (for instance in a console command) this doesn't work. See How to Generate URLs from the Console (console/request_context.html) to learn how to solve this problem.

Troubleshooting ¶

Here are some common errors you might see while working with routing:

> *Controller "AppBundleControllerBlogController::showAction()" requires that you provide a value for the "$slug" argument.*

This happens when your controller method has an argument (e.g. `$slug`):

```
public function showAction($slug)
{
    // ..
}
```

But your route path does *not* have a `{slug}` wildcard (e.g. it is `/blog/show`). Add a `{slug}` to your route path: `/blog/show/{slug}` or give the argument a default value (i.e. `$slug = null`).

> *Some mandatory parameters are missing ("slug") to generate a URL for route "blog_show".*

This means that you're trying to generate a URL to the `blog_show` route but you are *not* passing a `slug` value (which is required, because it has a `{slug}`) wildcard in the route path. To fix this, pass a `slug` value when generating the route:

```
$this->generateUrl('blog_show', array('slug' => 'slug-value'));

// or, in Twig
// {{ path('blog_show', {'slug': 'slug-value'}) }}
```

Summary ¶

Routing is a system for mapping the URL of incoming requests to the controller function that should be called to process the request. It both allows you to specify beautiful URLs and keeps the functionality of your application decoupled from those URLs. Routing is a bidirectional mechanism, meaning that it should also be used to generate URLs.

Keep Going! ¶

Routing, check! Now, uncover the power of controllers (controller.html).

Learn more about Routing ¶

- How to Restrict Route Matching through Conditions (routing/conditions.html)
- How to Create a custom Route Loader (routing/custom_route_loader.html)
- How to Visualize And Debug Routes (routing/debug.html)
- How to Include External Routing Resources (routing/external_resources.html)
- How to Pass Extra Information from a Route to a Controller (routing/extra_information.html)
- How to Generate Routing URLs in JavaScript (routing/generate_url_javascript.html)
- How to Match a Route Based on the Host (routing/hostname_pattern.html)
- How to Define Optional Placeholders (routing/optional_placeholders.html)
- How to Configure a Redirect without a custom Controller (routing/redirect_in_config.html)
- Redirect URLs with a Trailing Slash (routing/redirect_trailing_slash.html)
- How to Define Route Requirements (routing/requirements.html)
- Looking up Routes from a Database: Symfony CMF DynamicRouter (routing/routing_from_database.html)
- How to Force Routes to always Use HTTPS or HTTP (routing/scheme.html)
- How to Use Service Container Parameters in your Routes (routing/service_container_parameters.html)
- How to Allow a "/" Character in a Route Parameter (routing/slash_in_parameter.html)

« The Routing Component (components/routing.html) **Controller » (controller.html)**

(/blog/a-week-of-symfony-510-3-9-october-2016)

500Million Symfony downloads contest results!
October 07, 2016
(/blog/500million-symfony-downloads-contest-results)

Symfony 3.1.5 released
October 03, 2016
(/blog/symfony-3-1-5-released)

Visit The Symfony Blog (/blog/)

(https://sensiolabs.com/symfony-certification)

Symfony 3 Certification now available in 4,000 centers around the world!

GET CERTIFIED
(https://sensiolabs.com/en/symfony/certification/order)

q=SF3C3&from=10/17/2016&to=10/18/2016)

Web Development with Symfony 3
Paris - 2016-10-24
(http://training.sensiolabs.com/en/courses?q=SF3C4&from=10/24/2016&to=10/27/2016)

Web Development with Symfony 3
Online America - 2016-10-24
(http://training.sensiolabs.com/en/courses?q=SF3C4&from=10/24/2016&to=10/27/2016)

View all sessions (https://training.sensiolabs.com/)

What is Symfony? (/what-is-symfony)
- Symfony at a Glance (/at-a-glance)
- Symfony Components (/components)
- Projects using Symfony (/projects)
- Case Studies (/blog/category/case-studies)
- Symfony Roadmap (/roadmap)
- Security Policy (/doc/current/contributing/code/security.html)
- Logo & Screenshots (/logo)
- Trademark & Licenses (/license)
- symfony1 Legacy (/legacy)

Learn Symfony (/doc/current/index.html)
- Book (/doc/2.8/book/index.html)
- Cookbook (/doc/2.8/cookbook/index.html)
- Components (/doc/2.8/components/index.html)
- Best Practices (/doc/2.8/best_practices/index.html)
- Bundles (/doc/bundles/)
- Reference (/doc/2.8/reference/index.html)
- Training (https://training.sensiolabs.com/en/courses?q=symfony)
- Certification (https://sensiolabs.com/certification)

Community (/community)
- SensioLabs Connect (https://connect.sensiolabs.com/)
- Support (/support)
- How to be Involved (/doc/current/contributing/index.html)
- Code Stats (/stats/code)
- Downloads Stats (/stats/downloads)
- Contributors (/contributors)

Blog (/blog/)
- A week of symfony (/blog/category/a-week-of-symfony)
- Case studies (/blog/category/case-studies)
- Community (/blog/category/community)
- Documentation (/blog/category/documentation)
- Living on the edge (/blog/category/living-on-the-edge)
- Meet the Bundle (/blog/category/meet-the-bundle)
- Releases (/blog/category/releases)
- Security Advisories (/blog/category/security-advisories)
- Symfony CMF (/blog/category/symfony-cmf)
- *Community Events* (/events/)

Services (/services)
- Our services (/services)
- Train developers (https://training.sensiolabs.com/en)
- Start a project (/services#before)
- Manage your project quality (https://insight.sensiolabs.com/)
- Improve your project performance (https://blackfire.io/)
- Struggling with your project (/services)
- Support (http://sensiolabs.com/en/support/sensiolabs_support.html)
- Contact us (/services)

About (/about)
- SensioLabs (http://sensiolabs.com/recrutement/rej
- Contributors (/contributors)
- Jobs (http://sensiolabs.com/recrutement/rej
- Support (/support)