路由

当您的应用程序收到请求时，它会调用控制器操作来生成响应。路由配置定义对每个传入 URL 运行哪个操作。它还提供其他有用的功能，例如生成 SEO 友好的 URL（例如 /read/intro-to-symfony而不是 index.php?article_id=57）。

创建路由

路由可以在 YAML、XML、PHP 中或使用属性进行配置。所有格式都提供相同的功能和性能，因此请选择您最喜欢的格式。Symfony 推荐 attributes，因为把 route 和 controller 放在同一个地方很方便。

创建路由作为属性

PHP 属性允许在与这些路由关联的控制器代码旁边定义路由。属性在 PHP 8 及更高版本中是原生的，因此您可以立即使用它们。

在使用它们之前，您需要向项目添加一些配置。如果你的项目使用 Symfony Flex，那么这个文件已经为你创建了。否则，请手动创建以下文件：

```yaml
# config/routes/attributes.yaml
controllers:
    resource:
        path: ../../src/Controller/
        namespace: App\Controller
    type: attribute

kernel:
    resource: App\Kernel
    type: attribute
```

这个配置告诉 Symfony 在声明在 App\Controller 命名空间中的类上查找定义为属性的路由，并存储在遵循 PSR-4 标准的 src/Controller/ 目录中。内核也可以充当控制器，这对于使用 Symfony 作为微框架的小型应用程序特别有用。

假设您要在应用程序中为 /blog URL 定义路由。为此，请创建一个如下所示的 controller 类：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    #[Route('/blog', name: 'blog_list')]
    public function list(): Response
    {
        // ...
    }
}
```

此配置定义了一个名为 blog_list 的路由，该路由在用户请求 /blog URL 时匹配。当匹配发生时，应用程序运行 BlogController 类的 list() 方法。

路由名称（blog_list）目前并不重要，但在以后生成 URL 时将是必不可少的。您只需记住，每个路由名称在应用程序中必须是唯一的。

在 YAML、XML 或 PHP 文件中创建路由

您可以在单独的 YAML、XML 或 PHP 文件中定义路由，而不是在控制器类中定义路由。主要优点是它们不需要任何额外的依赖。主要缺点是，在检查某些控制器动作的路由时，您必须处理多个文件。

以下示例展示了如何在 YAML/XML/PHP 中定义一个名为 blog_list 的路由，该路由将 /blog URL 与 BlogController 的 list() 操作相关联：

```
# config/routes.yaml
blog_list:
    path: /blog
    # the controller value has the format 'controller_class::method_name'
    controller: App\Controller\BlogController::list

    # if the action is implemented as the __invoke() method of the
    # controller class, you can skip the '::method_name' part:
    # controller: App\Controller\BlogController
```

匹配 HTTP 方法

默认情况下，路由匹配任何 HTTP 动词（ GET、POST、PUT 等）使用 methods 选项来限制每个路由应响应的动词：

```
// src/Controller/BlogApiController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogApiController extends AbstractController
{
    #[Route('/api/posts/{id}', methods: ['GET', 'HEAD'])]
    public function show(int $id): Response
    {
        // ... return a JSON response with the post
    }

    #[Route('/api/posts/{id}', methods: ['PUT'])]
    public function edit(int $id): Response
    {
        // ... edit a post
    }
}
```

匹配表达式

如果您需要基于某些任意匹配逻辑进行匹配的某些路由，请使用 condition 选项：

```
// src/Controller/DefaultController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class DefaultController extends AbstractController
{
```

```
    #[Route(
        '/contact',
        name: 'contact',
        condition: "context.getMethod() in ['GET', 'HEAD'] and request.headers.get('User-Agent') matches
'/firefox/i'",
        // expressions can also include config parameters:
        // condition: "request.headers.get('User-Agent') matches '%app.allowed_browsers%'"
    )]
    public function contact(): Response
    {
        // ...
    }

    #[Route(
        '/posts/{id}',
        name: 'post_show',
        // expressions can retrieve route parameter values using the "params" variable
        condition: "params['id'] < 1000"
    )]
    public function showPost(int $id): Response
    {
        // ... return a JSON response with the post
    }
}
```

condition 选项的值是一个使用任何有效表达式语言语法的表达式，并且可以使用 Symfony 创建的这些变量中的任何一个：

- context RequestContext 的一个实例，它保存有关正在匹配的路由的最基本信息。
- request 表示当前请求的 Symfony Request 对象。
- params 当前路由的匹配路由参数数组。

您还可以使用以下函数：

- env(string $name) 使用 Environment Variable Processors 返回变量的值
- service(string $alias) 返回路由条件服务。首先，将 #[AsRoutingConditionService] 属性或 routing.condition_service 标记添加到要在路由条件中使用的服务：``` use Symfony\Bundle\FrameworkBundle\Routing\Attribute\AsRoutingConditionService; use Symfony\Component\HttpFoundation\Request; #[AsRoutingConditionService(alias: 'route_checker')] class RouteChecker { public function check(Request $request): bool { // ... } }

然后，使用 service() 函数在 conditions 中引用该服务：

```
// Controller (using an alias):
#[Route(condition: "service('route_checker').check(request)")]
// Or without alias:
#[Route(condition: "service('App\\Service\\RouteChecker').check(request)")]
```

在后台，表达式被编译为原始 PHP。因此，使用 condition 键不会产生超出底层 PHP 执行时间的额外开销。

调试路由

随着应用程序的增长，您最终将拥有大量路由。Symfony 包含一些命令来帮助您调试路由问题。首先，debug:router 命令按照 Symfony 评估它们的顺序列出所有应用程序路由：

```
php bin/console debug:router

---------------  -------  -------  -----  ----------------------------------------
Name             Method   Scheme   Host   Path
---------------  -------  -------  -----  ----------------------------------------
homepage         ANY      ANY      ANY    /
contact          GET      ANY      ANY    /contact
contact_process  POST     ANY      ANY    /contact
article_show     ANY      ANY      ANY    /articles/{_locale}/{year}/{title}.{_format}
blog             ANY      ANY      ANY    /blog/{page}
blog_show        ANY      ANY      ANY    /blog/{slug}
---------------  -------  -------  -----  ----------------------------------------
```

将某个路由的名称（或名称的一部分）传递给此参数以打印路由详细信息：

```
php bin/console debug:router app_lucky_number

+-------------+---------------------------------------------------------+
| Property    | Value                                                   |
+-------------+---------------------------------------------------------+
| Route Name  | app_lucky_number                                        |
| Path        | /lucky/number/{max}                                     |
| ...         | ...                                                     |
```

```
| Options     | compiler_class: Symfony\Component\Routing\RouteCompiler |
|             | utf8: true                                              |
+-------------+---------------------------------------------------------+
```

另一个命令称为 router: match，它显示哪个路由将匹配给定的 URL。找出某些 URL 未执行您期望的控制器操作的原因非常有用：

```
php bin/console router:match /lucky/number/8

  [OK] Route "app_lucky_number" matches
```

路由参数

前面的示例定义了 URL 永远不会更改的路由（例如 /blog）。但是，定义某些部分可变的路由是很常见的。例如，显示某些博客文章的 URL 可能包含标题或 slug（例如 /blog/my-first-post 或 /blog/all-about-symfony）。

在 Symfony 路由中，变量部分被 { } 包裹起来。例如，显示博客文章内容的路由定义为 /blog/{slug}：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    // ...

    #[Route('/blog/{slug}', name: 'blog_show')]
    public function show(string $slug): Response
    {
        // $slug will equal the dynamic part of the URL
        // e.g. at /blog/yay-routing, then $slug='yay-routing'

        // ...
```

```
    }
}
```

变量部分的名称（本例中为 {slug}）用于创建 PHP 变量，该路由内容被存储并传递给控制器。如果用户访问 /blog/my-first-post URL，Symfony 会执行 BlogController 类中的 show() 方法，并将 $slug = 'my-first-post' 参数传递给 show() 方法。

路由可以定义任意数量的参数，但每个参数只能在每个路由上使用一次（例如 /blog/posts-about-{category}/page/{pageNumber}）。

参数验证

假设您的应用程序有一个 blog_show 路由（URL：/blog/{slug}）和一个 blog_list 路由（URL：/blog/{page}）。鉴于路由参数接受任何值，因此无法区分这两个路由。

如果用户请求 /blog/my-first-post，则两个路由将匹配，并且 Symfony 将使用首先定义的路由。要解决此问题，请使用 requirements 选项向 {page} 参数添加一些验证：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    #[Route('/blog/{page}', name: 'blog_list', requirements: ['page' => '\d+'])]
    public function list(int $page): Response
    {
        // ...
    }

    #[Route('/blog/{slug}', name: 'blog_show')]
    public function show($slug): Response
    {
        // ...
    }
}
```

requirements 选项定义了路由参数必须匹配的 PHP 正则表达式，才能匹配整个路由。在此示例中，\d+ 是匹配任意长度的数字的正则表达式。现在：

```
URLRoute路线 Parameters 参数
/blog/2blog_list$page = 2
/blog/my-first-postblog_show$slug = my-first-post $slug 乇的首发
```

如果您愿意，可以使用语法 {parameter_name} . 此功能能使配置更简洁，但在需求复杂时，它可能会降低路由的可读性：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    #[Route('/blog/{page<\d+>}', name: 'blog_list')]
    public function list(int $page): Response
    {
        // ...
    }
}
```

可选参数

在前面的示例中，blog_list 的 URL 为 /blog/{page}。如果用户访问 /blog/1，它将匹配。但是如果他们访问 /blog，则不会匹配。将参数添加到路由后，它必须具有值。

您可以通过为 {page} 参数添加默认值，在用户访问 /blog 时再次 blog_list匹配。使用属性时，默认值在 controller 动作的参数中定义。在其他配置格式中，它们使用 defaults 选项定义：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
```

```
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    #[Route('/blog/{page}', name: 'blog_list', requirements: ['page' => '\d+'])]
    public function list(int $page = 1): Response
    {
        // ...
    }
}
```

现在，当用户访问 /blog 时，blog_list路由将匹配，$page 将默认为值 1。

如果要始终在生成的 URL 中包含一些默认值（例如，在上一个示例中强制生成 /blog/1 而不是 /blog），请在参数名称前添加 ! 字符：/blog/{!page}

与 requirements 一样，也可以使用语法 {parameter_name?default_value} 在每个参数中内联默认值。此功能与内联要求兼容，因此您可以在单个参数中内联两者：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    #[Route('/blog/{page<\d+>?1}', name: 'blog_list')]
    public function list(int $page): Response
    {
        // ...
    }
}
```

优先级参数

Symfony 按照路由定义的顺序来评估路由。如果路由的路径与许多不同的模式匹配，则可能会阻止其他路由匹配。在 YAML 和 XML 中，您可以在配置文件中上下移动路由定义以控制其优先级。在定义为 PHP

属性的路由中，这要困难得多，因此你可以在这些路由中设置可选的 priority 参数来控制它们的优先级：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    /**
     * This route has a greedy pattern and is defined first.
     */
    #[Route('/blog/{slug}', name: 'blog_show')]
    public function show(string $slug): Response
    {
        // ...
    }

    /**
     * This route could not be matched without defining a higher priority than 0.
     */
    #[Route('/blog/list', name: 'blog_list', priority: 2)]
    public function list(): Response
    {
        // ...
    }
}
```

priority 参数需要一个整数值。优先级较高的路由排序在优先级较低的路由之前。未定义时，默认值为 0。

参数转换

一个常见的路由需求是将存储在某个参数中的值（例如，充当用户 ID 的整数）转换为另一个值（例如，代表用户的对象）。此功能称为“param converter”。

现在，保留之前的路由配置，但更改 controller 操作的参数。添加 BlogPost $post，而不是字符串 $slug

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

```
use App\Entity\BlogPost;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    // ...

    #[Route('/blog/{slug}', name: 'blog_show')]
    public function show(BlogPost $post): Response
    {
        // $post is the object whose slug matches the routing parameter

        // ...
    }
}
```

如果你的控制器参数包含对象的类型提示（在本例中为 BlogPost），则 “param converter” 会发出数据库请求，以使用请求参数（在本例中为 slug）查找对象。如果未找到对象，Symfony 会自动生成 404 响应。

查看 Doctrine param conversion 文档，了解 #[MapEntity] 属性，该属性可用于自定义用于从 route 参数获取对象的数据库查询。

支持的枚举参数

你可以使用 PHP 支持的枚举作为路由参数，因为 Symfony 会自动将它们转换为它们的标量值。

```
// src/Controller/OrderController.php
namespace App\Controller;

use App\Enum\OrderStatusEnum;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class OrderController extends AbstractController
{
```

```
    #[Route('/orders/list/{status}', name: 'list_orders_by_status')]
    public function list(OrderStatusEnum $status = OrderStatusEnum::Paid): Response
    {
        // ...
    }
}
```

特殊参数

除了你自己的参数外，路由还可以包括 Symfony 创建的以下任何特殊参数:

- _controller 该参数用于确定路由匹配时执行哪个控制器和动作。
- _format 匹配的值用于设置 Request 对象的“请求格式”。这用于设置响应的 Content-Type 等操作（例如，json 格式转换为 application/json 的 Content-Type）。
- _fragment 用于设置片段标识符，该标识符是 URL 的可选最后一部分，以 # 字符开头，用于标识文档的一部分。
- _locale 用于设置请求的区域设置。

您可以在单个路由和路由导入中包含这些属性（_fragment 除外）。Symfony 定义了一些同名的特殊属性（除了前导下划线），这样你就可以更容易地定义它们了：

```
// src/Controller/ArticleController.php
namespace App\Controller;

// ...
class ArticleController extends AbstractController
{
    #[Route(
        path: '/articles/{_locale}/search.{_format}',
        locale: 'en',
        format: 'html',
        requirements: [
            '_locale' => 'en|fr',
            '_format' => 'html|xml',
        ],
    )]
    public function search(): Response
    {
    }
}
```

额外参数

在路由的`defaults`选项中，您可以选择定义路由配置中未包含的参数。这对于将额外的参数传递给路由的控制器很有用：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    #[Route('/blog/{page}', name: 'blog_index', defaults: ['page' => 1, 'title' => 'Hello world!'])]
    public function index(int $page, string $title): Response
    {
        // ...
    }
}
```

路由参数中的斜杠字符

路由参数可以包含除`/`斜杠字符之外的任何值，因为该字符用于分隔`URL`的不同部分。例如，如果`/share/{token}`路由中的`token`值包含`/`字符，则此路由将不匹配。

一种可能的解决方案是将`parameter requirements`更改为更宽松：

```
// src/Controller/DefaultController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class DefaultController extends AbstractController
{
    #[Route('/share/{token}', name: 'share', requirements: ['token' => '.+'])]
    public function share($token): Response
```

```
    {
        // ...
    }
}
```

路由别名

Route alias 允许您为同一路由设置多个名称:

```
# config/routes.yaml
new_route_name:
    alias: original_route_name
```

在此示例中， original_route_name 和 new_route_name 路由都可以在应用程序中使用，并且将产生相同的结果。

弃用路由别名

如果某个路由别名不应该再使用（因为它已过时或您决定不再维护它），您可以弃用其定义:

```
new_route_name:
    alias: original_route_name

    # this outputs the following generic deprecation message:
    # Since acme/package 1.2: The "new_route_name" route alias is deprecated. You should stop using it, as it will be removed in the future.
    deprecated:
        package: 'acme/package'
        version: '1.2'

    # you can also define a custom deprecation message (%alias_id% placeholder is available)
    deprecated:
        package: 'acme/package'
        version: '1.2'
        message: 'The "%alias_id%" route alias is deprecated. Do not use it anymore.'
```

在此示例中，每次使用 new_route_name 别名时，都会触发弃用警告，建议您停止使用该别名。

该消息实际上是一个消息模板，它将 %alias_id% 占位符的出现替换为路由别名。模板中必须至少出现一次 %alias_id% 占位符。

路由组和前缀

一组路由共享一些选项是很常见的（例如，所有与博客相关的路由都以 /blog 开头）这就是为什么 Symfony 包含共享路由配置的功能。

当定义路由为属性时，将通用配置放在 controller 类的 #[Route] 属性中。在其他路由格式中，在导入路由时使用选项定义通用配置。

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

#[Route('/blog', requirements: ['_locale' => 'en|es|fr'], name: 'blog_')]
class BlogController extends AbstractController
{
    #[Route('/{_locale}', name: 'index')]
    public function index(): Response
    {
        // ...
    }

    #[Route('/{_locale}/posts/{slug}', name: 'show')]
    public function show(string $slug): Response
    {
        // ...
    }
}
```

在此示例中，index () 操作的路由将调用 blog_index，其 URL 将为 /blog/{_locale}。show () 操作的路由将 blog_show调用，其 URL 将为 /blog/{_locale}/posts/{slug}。这两个路由还将验证 _locale 参数是否与 class 属性中定义的正则表达式匹配。

获取路由名称和参数

Symfony 创建的 Request 对象将所有路由配置（比如 name 和 parameters）都存储在 “request attributes” 中。您可以通过 Request 对象在控制器中获取此信息：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class BlogController extends AbstractController
{
    #[Route('/blog', name: 'blog_list')]
    public function list(Request $request): Response
    {
        $routeName = $request->attributes->get('_route');
        $routeParameters = $request->attributes->get('_route_params');

        // use this to get all the available attributes (not only routing ones):
        $allAttributes = $request->attributes->all();

        // ...
    }
}
```

在 services 中，您可以通过注入 RequestStack 服务来获取此信息。在模板中，使用 Twig 全局 app 变量获取当前路由名称（ app.current_route）及其参数（ app.current_route_parameters）。

特殊路由

Symfony 定义了一些特殊的控制器来渲染模板并从路由配置中重定向到其他路由，因此你不必创建控制器动作。

直接从路由渲染模板

请阅读 Symfony 模板 主文章 中关于从 route 渲染模板的部分。

直接从路由重定向到 URL 和路由

使用 RedirectController 重定向到其他路由和 URL:

```
# config/routes.yaml
doc_shortcut:
    path: /doc
    controller: Symfony\Bundle\FrameworkBundle\Controller\RedirectController
    defaults:
        route: 'doc_page'
        # optionally you can define some arguments passed to the route
        page: 'index'
        version: 'current'
        # redirections are temporary by default (code 302) but you can make them permanent (code 301)
        permanent: true
        # add this to keep the original query string parameters when redirecting
        keepQueryParams: true
        # add this to keep the HTTP method when redirecting. The redirect status changes
        # * for temporary redirects, it uses the 307 status code instead of 302
        # * for permanent redirects, it uses the 308 status code instead of 301
        keepRequestMethod: true
        # add this to remove all original route attributes when redirecting
        ignoreAttributes: true
        # or specify which attributes to ignore:
        # ignoreAttributes: ['offset', 'limit']

legacy_doc:
    path: /legacy/doc
    controller: Symfony\Bundle\FrameworkBundle\Controller\RedirectController
    defaults:
        # this value can be an absolute path or an absolute URL
        path: 'https://legacy.example.com/doc'
        permanent: true
```

重定向带有尾部斜杠的 URL

从历史上看, URL 遵循 UNIX 约定，为目录添加尾部斜杠（例如 https://example.com/foo/ ）并删除它们以引用文件 （ https://example.com/foo）。尽管为两个 URL

提供不同的内容是可以的，但现在将两个 URL 视为同一 URL 并在它们之间重定向是很常见的。

Symfony 遵循这个逻辑在有和没有尾部斜杠的 URL 之间重定向（但仅限于 GET 和 HEAD 请求）：

子域路由

路由可以配置 host 选项，以要求传入请求的 HTTP 主机与某个特定值匹配。在以下示例中，两个路由都匹配相同的路径（ / ），但其中一个路由仅响应特定的主机名：

```
// src/Controller/MainController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class MainController extends AbstractController
{
    #[Route('/', name: 'mobile_homepage', host: 'm.example.com')]
    public function mobileHomepage(): Response
    {
        // ...
    }

    #[Route('/', name: 'homepage')]
    public function homepage(): Response
    {
        // ...
    }
}
```

host 选项的值可以包含参数（这在多租户应用程序中很有用），并且这些参数也可以根据需要进行验证：

```
// src/Controller/MainController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

```
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class MainController extends AbstractController
{
    #[Route(
        '/',
        name: 'mobile_homepage',
        host: '{subdomain}.example.com',
        defaults: ['subdomain' => 'm'],
        requirements: ['subdomain' => 'm|mobile'],
    )]
    public function mobileHomepage(): Response
    {
        // ...
    }

    #[Route('/', name: 'homepage')]
    public function homepage(): Response
    {
        // ...
    }
}
```

在上面的示例中，subdomain 参数定义了一个默认值，因为否则，每次使用这些路由生成 URL 时，您都需要包含一个 subdomain 值。

本地化路由（i18n）

如果您的应用程序被翻译成多种语言，则每个路由可以为每个翻译区域设置定义不同的 URL。这避免了复制路由的需要，这也减少了潜在的错误：

```
// src/Controller/CompanyController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

```

```
class CompanyController extends AbstractController
{
    #[Route(path: [
        'en' => '/about-us',
        'nl' => '/over-ons'
    ], name: 'about_us')]
    public function about(): Response
    {
        // ...
    }
}
```

当匹配到本地化的路由时，Symfony 在整个请求过程中会自动使用相同的 locale。

国际化应用程序的一个常见要求是为所有路由添加 locale 前缀。这可以通过为每个 locale 定义不同的前缀来完成（如果您愿意，还可以为默认 locale 设置一个空前缀）：

```
# config/routes/attributes.yaml
controllers:
    resource: '../../src/Controller/'
    type: attribute
    prefix:
        en: '' # don't prefix URLs for English, the default locale
        nl: '/nl'
```

另一个常见要求是根据区域设置将网站托管在不同的域上。这可以通过为每个 locale 定义不同的主机来完成。

```
# config/routes/attributes.yaml
controllers:
    resource: '../../src/Controller/'
    type: attribute
    host:
        en: 'www.example.com'
        nl: 'www.example.nl'
```

无状态路由

有时，当应该缓存 HTTP 响应时，确保这种情况可能发生是很重要的。但是，每当在请求期间启动会话时，Symfony 都会将响应转换为私有的不可缓存响应。

有关详细信息，请参阅 HTTP 缓存。

路由可以配置一个无状态的布尔选项，以声明在匹配请求时不应使用该会话：

```
// src/Controller/MainController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Attribute\Route;

class MainController extends AbstractController
{
    #[Route('/', name: 'homepage', stateless: true)]
    public function homepage(): Response
    {
        // ...
    }
}
```

现在，如果使用了会话，应用程序将根据您的 kernel.debug 参数报告它：

- enabled：将引发 UnexpectedSessionUsageException 异常
- disabled：将记录警告

它将帮助您了解并希望修复应用程序中的意外行为。

生成 URL

路由系统是双向的：

- 它们将 URL 与控制器相关联（如前几节所述）；
- 它们为给定路由生成 URL。

通过从路由生成 URL，您可以不在 HTML 模板中手动写入 `<a href="...">` 值。此外，如果某些路由的 URL 发生变化，您只需更新路由配置，所有链接都会更新。

要生成 URL，您需要指定路由的名称（例如 blog_show）和路由定义的参数值（例如 slug = my-blog-post）。

因此，每个路由都有一个内部名称，该名称在应用程序中必须是唯一的。如果你没有使用 name 选项显式设置路由名称，Symfony 会根据控制器和动作自动生成一个名称。

如果目标类有一个添加路由的 __invoke() 方法，并且目标类恰好添加了一条路由，那么 Symfony 就会基于 FQCN 声明路由别名。 Symfony 还会自动为每个只定义一个路由的方法添加一个别名。请考虑以下类：

```
// src/Controller/MainController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Attribute\Route;

final class MainController extends AbstractController
{
    #[Route('/', name: 'homepage')]
    public function homepage(): Response
    {
        // ...
    }
}
```

Symfony 将添加一个名为 App\Controller\MainController::homepage .

在 Controller 中生成 URL

如果你的控制器是从 AbstractController 继承而来的，请使用 generateUrl() 辅助函数：

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

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

class BlogController extends AbstractController
{
    #[Route('/blog', name: 'blog_list')]
    public function list(): Response
    {
        // generate a URL with no route arguments
        $signUpPage = $this->generateUrl('sign_up');

        // generate a URL with route arguments
```

```
        $userProfilePage = $this->generateUrl('user_profile', [
            'username' => $user->getUserIdentifier(),
        ]);

        // generated URLs are "absolute paths" by default. Pass a third optional
        // argument to generate different URLs (e.g. an "absolute URL")
        $signUpPage = $this->generateUrl('sign_up', [], UrlGeneratorInterface::ABSOLUTE_URL);

        // when a route is localized, Symfony uses by default the current request locale
        // pass a different '_locale' value if you want to set the locale explicitly
        $signUpPageInDutch = $this->generateUrl('sign_up', ['_locale' => 'nl']);

        // ...
    }
}
```

如果你的控制器不是从 AbstractController 扩展而来的，你需要在你的控制器中获取服务，并按照下一节的说明进行操作。

在 Services 中生成 URL

将路由器 Symfony 服务注入到你自己的服务中，并使用它的 generate() 方法。当使用服务自动装配时，您只需在服务构造函数中添加一个参数，并使用 UrlGeneratorInterface 类对其进行类型提示：

```
// src/Service/SomeService.php
namespace App\Service;

use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

class SomeService
{
    public function __construct(
        private UrlGeneratorInterface $router,
    ) {
    }

    public function someMethod(): void
    {
        // ...
```

```
        // generate a URL with no route arguments
        $signUpPage = $this->router->generate('sign_up');

        // generate a URL with route arguments
        $userProfilePage = $this->router->generate('user_profile', [
            'username' => $user->getUserIdentifier(),
        ]);

        // generated URLs are "absolute paths" by default. Pass a third optional
        // argument to generate different URLs (e.g. an "absolute URL")
        $signUpPage = $this->router->generate('sign_up', [], UrlGeneratorInterface::ABSOLUTE_URL);

        // when a route is localized, Symfony uses by default the current request locale
        // pass a different '_locale' value if you want to set the locale explicitly
        $signUpPageInDutch = $this->router->generate('sign_up', ['_locale' => 'nl']);
    }
}
```

在模板中生成 URL

请阅读 Symfony 模板主条目中关于在页面之间创建链接的部分。

在 JavaScript 中生成 URL

如果您的 JavaScript 代码包含在 Twig 模板中，则可以使用 path () 和 url () Twig 函数生成 URL 并将其存储在 JavaScript 变量中。需要 escape () 过滤器来转义任何非 JavaScript 安全的值：

```
<script>
    const route = "{{ path('blog_show', {slug: 'my-blog-post'})|escape('js') }}";
</script>
```

如果您需要动态生成 URL 或者使用纯 JavaScript 代码，则此解决方案不起作用。在这些情况下，请考虑使用 FOSJsRoutingBundle。

在命令中生成 URL

在命令中生成 URL 的工作方式与在服务中生成 URL 相同。唯一的区别是命令不在 HTTP 上下文中执行。因此，如果您生成绝对 URL，您将获得 http://localhost/ 作为主机名，而不是您的真实主机名。

解决方案是配置 default_uri 选项来定义命令在生成 URL 时使用的 “request context”：

```yaml
# config/packages/routing.yaml
framework:
    router:
        # ...
        default_uri: 'https://example.org/my/path/'
```

□□□□□□□□□ URL □□□□□□□□□□□□

```php
// src/Command/SomeCommand.php
namespace App\Command;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;
// ...

class SomeCommand extends Command
{
    public function __construct(private UrlGeneratorInterface $urlGenerator)
    {
        parent::__construct();
    }

    protected function execute(InputInterface $input, OutputInterface $output): int
    {
        // generate a URL with no route arguments
        $signUpPage = $this->urlGenerator->generate('sign_up');

        // generate a URL with route arguments
        $userProfilePage = $this->urlGenerator->generate('user_profile', [
            'username' => $user->getUserIdentifier(),
        ]);

        // by default, generated URLs are "absolute paths". Pass a third optional
        // argument to generate different URIs (e.g. an "absolute URL")
        $signUpPage = $this->urlGenerator->generate('sign_up', [], UrlGeneratorInterface::ABSOLUTE_URL);

        // when a route is localized, Symfony uses by default the current request locale
```

```
        // pass a different '_locale' value if you want to set the locale explicitly
        $signUpPageInDutch = $this->urlGenerator->generate('sign_up', ['_locale' => 'nl']);

        // ...
    }
}
```

检查路由是否存在

在高度动态的应用程序中，可能需要先检查路由是否存在，然后再使用它来生成 URL。在这些情况下，请不要使用 getRouteCollection() 方法，因为这会重新生成路由缓存并减慢应用程序的速度。

相反，请尝试生成 URL 并捕获路由不存在时引发的 RouteNotFoundException：

```
use Symfony\Component\Routing\Exception\RouteNotFoundException;

// ...

try {
    $url = $this->router->generate($routeName, $routeParameters);
} catch (RouteNotFoundException $e) {
    // the route is not defined...
}
```

在生成的 URL 上强制使用 HTTPS

默认情况下，生成的 URL 使用与当前请求相同的 HTTP 方案。在没有 HTTP 请求的控制台命令中，URL 默认使用 http。你可以通过每个命令（通过路由器的 getContext() 方法）或使用以下配置参数全局更改它：

```
# config/services.yaml
parameters:
    router.request_context.scheme: 'https'
    asset.request_context.secure: true
```

在控制台命令之外，使用 schemes 选项显式定义每个路由的方案：

```
// src/Controller/SecurityController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

```
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class SecurityController extends AbstractController
{
    #[Route('/login', name: 'login', schemes: ['https'])]
    public function login(): Response
    {
        // ...
    }
}
```

为登录路由生成的 URL 将始终使用 HTTPS。这意味着，当使用 path () Twig 函数生成 URL 时，如果原始请求的 HTTP 方案与路由使用的方案不同，则可能会得到绝对 URL 而不是相对 URL：

```
{# if the current scheme is HTTPS, generates a relative URL: /login #}
{{ path('login') }}

{# if the current scheme is HTTP, generates an absolute URL to change
   the scheme: https://example.com/login #}
{{ path('login') }}
```

对传入请求也强制执行 scheme 要求。如果您尝试使用 HTTP 访问 /login URL，您将自动重定向到相同的 URL，但使用 HTTPS 方案。

如果要强制一组路由使用 HTTPS，可以在导入路由时定义默认方案。以下示例在定义为属性的所有路由上强制使用 HTTPS：

```
# config/routes/attributes.yaml
controllers:
    resource: '../../src/Controller/'
    type: attribute
    schemes: [https]
```

对 URI 进行签名

签名 URI 是包含依赖于 URI 内容的哈希值的 URI。这样，您以后可以通过重新计算签名 URI 的哈希值并将其与 URI 中包含的哈希值进行比较来检查签名 URI 的完整性。

Symfony 提供了一个通过 UriSigner 服务对 URI 进行签名的工具，你可以将其注入到你的服务或控制器中：

```
// src/Service/SomeService.php
namespace App\Service;

use Symfony\Component\HttpFoundation\UriSigner;

class SomeService
{
    public function __construct(
        private UriSigner $uriSigner,
    ) {
    }

    public function someMethod(): void
    {
        // ...

        // generate a URL yourself or get it somehow...
        $url = 'https://example.com/foo/bar?sort=desc';

        // sign the URL (it adds a query parameter called '_hash')
        $signedUrl = $this->uriSigner->sign($url);
        // $url = 'https://example.com/foo/bar?sort=desc&_hash=e4a21b9'

        // check the URL signature
        $uriSignatureIsValid = $this->uriSigner->check($signedUrl);
        // $uriSignatureIsValid = true

        // if you have access to the current Request object, you can use this
        // other method to pass the entire Request object instead of the URI:
        $uriSignatureIsValid = $this->uriSigner->checkRequest($request);
    }
}
```

出于安全原因，通常会使签名的 URI 在一段时间后过期（例如，当使用它们重置用户凭证时）。默认情况下，已签名的 URI 不会过期，但您可以使用 sign() 的 $expiration 参数定义过期日期/时间:

```
// src/Service/SomeService.php
namespace App\Service;
```

```
use Symfony\Component\HttpFoundation\UriSigner;

class SomeService
{
    public function __construct(
        private UriSigner $uriSigner,
    ) {
    }

    public function someMethod(): void
    {
        // ...

        // generate a URL yourself or get it somehow...
        $url = 'https://example.com/foo/bar?sort=desc';

        // sign the URL with an explicit expiration date
        $signedUrl = $this->uriSigner->sign($url, new \DateTimeImmutable('2050-01-01'));
        // $signedUrl = 'https://example.com/foo/bar?sort=desc&_expiration=2524608000&_hash=e4a21b9'

        // if you pass a \DateInterval, it will be added from now to get the expiration date
        $signedUrl = $this->uriSigner->sign($url, new \DateInterval('PT10S'));  // valid for 10 seconds from now
        // $signedUrl = 'https://example.com/foo/bar?sort=desc&_expiration=1712414278&_hash=e4a21b9'

        // you can also use a timestamp in seconds
        $signedUrl = $this->uriSigner->sign($url, 4070908800); // timestamp for the date 2099-01-01
        // $signedUrl = 'https://example.com/foo/bar?sort=desc&_expiration=4070908800&_hash=e4a21b9'
    }
}
```

故障排除

以下是您在使用路由时可能会看到的一些常见错误:

```
Controller "App\\Controller\\BlogController::show()" requires that you
provide a value for the "$slug" argument.
```

当你的控制器方法有一个参数（例如 $slug）时，就会发生这种情况:

```
public function show(string $slug): Response
{
    // ...
}
```

但是你的路由路径没有 {slug} 参数（例如，它是 /blog/show）。将 {slug} 添加到路由路径：/blog/show/{slug} 或为参数指定默认值（即 $slug = null）。

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

这意味着您正在尝试生成指向 blog_show 路由的 URL，但您没有传递 slug 值（这是必需的，因为它在路由路径中有一个 {slug} 参数）。要解决此问题，请在生成路由时传递 slug 值：

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

or, in Twig: 或者，在 Twig 中：

```
{{ path('blog_show', {slug: 'slug-value'}) }}
```

Revision #1
Created 20 November 2024 00:36:55 by David Wang
Updated 20 November 2024 01:23:40 by David Wang