Symfony 文档

- 入门
  - 安装和设置 Symfony 框架
  - 在 Symfony 中创建您的首页
  - 路由
  - 控制器

Symfony

##

Symfony

- PHP 8.2 PHP PHP 8 Ctype iconv PCRE Session SimpleXML Tokenizer
- Composer PHP

Symfony CLI symfony Symfony

symfony

```
symfony check:requirements
```

Symfony

Symfony

```
symfony new my_project_directory --version="7.1.*" --webapp

symfony new my_project_directory --version="7.1.*"
```

--webapp
Web

Symfony Composer Symfony

```
composer create-project symfony/skeleton:"7.1.*" my_project_directory
cd my_project_directory
composer require webapp
```

```
composer create-project symfony/skeleton:"7.1.*" my_project_directory
```

无论您运行哪个命令来创建 Symfony 应用程序。所有这些都将创建一个新的 my_project_directory/ 目录，将一些依赖项下载到其中，甚至生成入门所需的基本目录和文件。换句话说，您的新应用程序已准备就绪!

设置现有的 Symfony 项目

除了创建新的 Symfony 项目之外，您还将处理其他开发人员已经创建的项目。在这种情况下，您只需要获取项目代码并使用 Composer 安装依赖项即可。假设您的团队使用 Git，请使用以下命令设置您的项目:

```
cd projects/
git clone ...

cd my-project/
composer install
```

您可能还需要自定义 .env 文件并执行一些其他特定于项目的任务（例如创建数据库）。当第一次使用现有的 Symfony 应用程序时，运行此命令可能会很有用，该命令显示有关项目的信息:

```
php bin/console about
```

运行 Symfony 应用程序

在生产中，您应该安装 Nginx 或 Apache 等 Web 服务器并将其配置为运行 Symfony 。如果您不使用 Symfony 本地 Web 服务器进行开发，也可以使用此方法。

然而，对于本地开发，运行 Symfony 最方便的方法是使用 symfony二进制文件提供的本地 Web 服务器。该本地服务器除其他外还提供对 HTTP/2、并发请求、 TLS/SSL 和自动生成安全证书的支持。

打开控制台终端，进入新项目目录并启动本地 Web 服务器，如下所示:

```
cd my-project/
symfony server:start
```

打开浏览器并导航到 http://localhost:8000/ 。如果一切正常，您将看到一个欢迎页面。稍后，当您完成工作时，通过从终端按 Ctrl+C来停止服务器。

Symfony Docker 集成

如果您想将 Docker 与 Symfony 结合使用，请参阅将 Docker 与 Symfony 结合使用。

[illegible]

[illegible] Symfony [illegible] Symfony [illegible]

[illegible] Symfony [illegible] Symfony Flex [illegible] Symfony [illegible] /[illegible] Symfony Flex [illegible] Composer [illegible] Symfony [illegible] Symfony [illegible]

Symfony Flex [illegible] require [illegible] update[illegible] remove Composer [illegible]

```
cd my-project/
composer require logger
```

[illegible] Flex [illegible] Symfony [illegible] Composer [illegible] logger [illegible] Symfony Flex[illegible] Symfony [illegible]

[illegible] Symfony [illegible] /[illegible] “[illegible] ” [illegible] Symfony [illegible] Flex [illegible] symfony.lock [illegible]

Symfony Flex [illegible]

- [illegible] Symfony Flex [illegible]
- Contrib [illegible] Symfony Flex [illegible]

[illegible] Symfony Recipes [illegible]

Packs

[illegible] Symfony [illegible] packs [illegible] Composer [illegible]

[illegible] composer require --dev debug[illegible] symfony/debug-pack [illegible] symfony/debug-bundle [illegible] symfony/monolog-bundle [illegible] symfony/var-dumper[illegible]

[illegible] composer.json[illegible] symfony/debug-pack[illegible] Flex [illegible] require-dev[illegible] symfony/var-dumper [illegible]

[illegible]

[illegible] Symfony CLI [illegible] symfony [illegible]

```
symfony check:security
```

[illegible] PHP [illegible] composer.lock [illegible]

[illegible] check:security [illegible]

Symfony LTS [illegible]

[illegible] Symfony [illegible] “[illegible]” [illegible] LTS [illegible] Symfony [illegible] LTS [illegible]

[illegible] Symfony [illegible] LTS [illegible] --version [illegible]

```
symfony new my_project_directory --version=lts



symfony new my_project_directory --version=next



symfony new my_project_directory --version="6.4.*"
```

lts [illegible] next [illegible] Symfony [illegible] Composer [illegible]

```
composer create-project symfony/skeleton:"6.4.*" my_project_directory
```

Symfony [illegible]

Symfony [illegible] Symfony [illegible] Symfony [illegible]

[illegible] Symfony Demo [illegible]

```
symfony new my_project_directory --demo
```

[illegible] Symfony [illegible]

[illegible] HTML [illegible] JSON [illegible]

- [illegible] PHP [illegible] Symfony Response[illegible] HTML [illegible] JSON [illegible] PDF [illegible]
- [illegible] URL[illegible] /about [illegible]

[illegible]

[illegible] - /lucky/number - [illegible] “[illegible]”[illegible]“[illegible]”[illegible]

```
<?php
// src/Controller/LuckyController.php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;

class LuckyController
{
    public function number(): Response
    {
        $number = random_int(0, 100);

        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
```

[illegible] URL[illegible] /lucky/number [illegible] number()[illegible] #[Route][illegible] PHP [illegible]

```
// src/Controller/LuckyController.php

 // ...
```

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

  class LuckyController
  {
+     #[Route('/lucky/number')]
      public function number(): Response
      {
          // this looks exactly the same
      }
  }
```

[illegible] Symfony Web [illegible] http://localhost:8000/lucky/number[illegible]

[illegible]

- [illegible] Response [illegible] JSON [illegible]
- [illegible] config/routes.yaml[illegible] URL[illegible] path [illegible] controller [illegible] URL[illegible]

bin/console [illegible]

[illegible] bin/console[illegible]

```
php bin/console
```

[illegible] debug:router[illegible]

```
php bin/console debug:router
```

[illegible] app_lucky_number[illegible]

```
----------------  -------  -------  -----  --------------
Name              Method   Scheme   Host   Path
----------------  -------  -------  -----  --------------
app_lucky_number  ANY      ANY      ANY    /lucky/number
----------------  -------  -------  -----  --------------
```

[illegible] app_lucky_number[illegible] ——[illegible]

[illegible]

Web [illegible]

Symfony[illegible] Web [illegible] symfony/profiler-pack [illegible] Symfony [illegible]

[illegible]

[illegible]

[illegible] HTML[illegible] Symfony [illegible] Twig [illegible]

[illegible] twig [illegible]

```
composer require twig
```

[illegible] LuckyController[illegible] Symfony [illegible] AbstractController[illegible]

```
// src/Controller/LuckyController.php


 // ...
+ use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;


- class LuckyController
+ class LuckyController extends AbstractController
 {
     // ...
 }
```

[illegible] render()[illegible] number[illegible] Twig [illegible]

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


use Symfony\Component\HttpFoundation\Response;
// ...


class LuckyController extends AbstractController
{
    #[Route('/lucky/number')]
```

```
    public function number(): Response
    {
        $number = random_int(0, 100);

        return $this->render('lucky/number.html.twig', [
            'number' => $number,
        ]);
    }
}
```

模板文件位于 templates/目录中，该目录是在您安装 Twig 时自动创建的。创建一个新的 templates/lucky 目录，其中包含新的 number.html.twig文件：

```
{# templates/lucky/number.html.twig #}
<h1>Your lucky number is {{ number }}</h1>
```

{{ number }}语法用于在 Twig 中打印变量。刷新您的浏览器即可获得新的幸运数字！

现在您可能想知道 Web 调试工具栏去了哪里：那是因为当前模板中没有

URL
SEO URL /read/intro-to-symfony
index.php?article_id=57

YAML XML PHP
Symfony
attributes route controller

PHP PHP 8

Symfony Flex

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

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

Symfony App\Controller
PSR-4 src/Controller/ Symfony

/blog URL controller

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

[illegible] blog_list [illegible] /blog URL [illegible] BlogController [illegible] list[illegible]

[illegible] blog_list[illegible] URL [illegible]

[illegible] YAML[illegible] XML [illegible] PHP [illegible]

[illegible] YAML[illegible] XML [illegible] PHP [illegible]

[illegible] YAML/XML/PHP [illegible] blog_list [illegible] /blog URL [illegible] BlogController [illegible] list[illegible]

```
# 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 [illegible] Symfony [illegible]

- context RequestContext [illegible]
- request [illegible] Symfony Request [illegible]
- params [illegible]

[illegible]

- env(string $name) [illegible] Environment Variable Processors [illegible]
- service(string $alias) [illegible] #[AsRoutingConditionService] [illegible] routing.condition_service [illegible] ``` use Symfony\Bundle\FrameworkBundle\Routing\Attribute\AsRoutingConditionService; use Symfony\Component\HttpFoundation\Request;

#[AsRoutingConditionService(alias: 'route_checker')] class RouteChecker { public function check(Request $request): bool { // ... } }

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

[illegible] PHP[illegible] condition [illegible] PHP [illegible]

[illegible]

[illegible] Symfony [illegible] debug [illegible] router [illegible] Symfony [illegible]

```
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}
----------------  -------  -------  -----  --------------------------------------------
```

[illegible]

```
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                                              |
+-------------+---------------------------------------------------------+
```

[illegible] router[illegible] match[illegible] URL[illegible] URL [illegible]

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

  [OK] Route "app_lucky_number" matches
```

[illegible]

[illegible] URL [illegible] /blog [illegible] URL [illegible] slug[illegible] /blog/my-first-post [illegible] /blog/all-about-symfony[illegible]

[illegible] Symfony [illegible] { } [illegible] /blog/{slug}[illegible]

```
// 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'
```

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

[illegible] {slug}[illegible] PHP [illegible] /blog/my-first-post URL[illegible] Symfony [illegible] BlogController [illegible] show[illegible] $slug = 'my-first-post' [illegible] show[illegible]

[illegible] /blog/posts-about-{category}/page/{pageNumber} [illegible]

[illegible]

[illegible] blog_show [illegible] URL[illegible] /blog/{slug}[illegible] blog_list [illegible] URL[illegible] /blog/{page}[illegible]

[illegible] /blog/my-first-post[illegible] Symfony [illegible] requirements [illegible] {page} [illegible]

```
// 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：

URL（Route路线） Parameters 参数
/blog/2 blog_list $page = 2
/blog/my-first-post blog_show $slug = my-first-post $slug 后的首发

如果您愿意，可以使用语法 {parameter_name} .
此功能使配置更简洁，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
    {
        // ...
    }
}
```

[illegible] /blog [illegible] blog_list[illegible] $page [illegible] 1[illegible]

[illegible] URL [illegible] /blog/1 [illegible] /blog [illegible] [illegible] /blog/{[illegible] page}

[illegible] requirements [illegible] {parameter_name?default_value} [illegible]
[illegible]

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

[illegible]

Symfony
[illegible]

YAML [illegible] XML [illegible] PHP [illegible] priority [illegible]

```php
// 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 [illegible] 0[illegible]

[illegible]

[illegible] ID [illegible] “param converter”[illegible]

[illegible] controller [illegible] BlogPost $post[illegible] $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

        // ...
    }
}
```

[illegible] BlogPost[illegible] “param converter” [illegible] slug[illegible] Symfony [illegible] 404 [illegible]

[illegible] Doctrine param conversion [illegible] #[MapEntity] [illegible] route [illegible]

[illegible]

[illegible] PHP [illegible] Symfony [illegible]

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

[illegible]

[illegible] Symfony [illegible]

- _controller [illegible]
- _format [illegible] Request [illegible] “[illegible]”[illegible] Content-Type [illegible] json [illegible] application/json [illegible] Content-Type[illegible]
- _fragment [illegible] URL [illegible] # [illegible]
- _locale [illegible]

[illegible] _fragment [illegible] Symfony [illegible]

```
// 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
    {
```

```
    }
}
```

[illegible]

[illegible] defaults
[illegible]

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

[illegible]

[illegible] / [illegible] URL [illegible] /share/{token} [illegible] token [illegible] / [illegible]

[illegible] parameter requirements [illegible]

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

[illegible]

Route alias [illegible]

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

[illegible] original_route_name [illegible] new_route_name [illegible]

[illegible]

[illegible]

```
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.'
```

[illegible] new_route_name [illegible]

[illegible] %alias_id% [illegible] %alias_id% [illegible]

[illegible]

[illegible] /blog [illegible] Symfony [illegible]

[illegible] controller [illegible] #[Route] [illegible]

```php
// 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
[illegible]

[illegible]

[illegible] Symfony [illegible] [illegible] [illegible] route [illegible]

[illegible] URL [illegible]

[illegible] RedirectController [illegible] URL[illegible]

```yaml
# 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
```

URL

URL UNIX https://example.com/foo/ https://example.com/foo URL

URL URL

Symfony URL GET HEAD

host HTTP

/

```
// 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

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

□□□□□□□□□□ Symfony □□□□□□□□□□□□□ locale□

□□□□□□□□□□□□□□□□□□□ locale □□□□□□□□□ locale □□□□□□□□□□□□□□□□□□□□ locale □□□□□□□□

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

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ locale □□□□□□□□□

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

[illegible]

[illegible] HTTP [illegible]
Symfony [illegible]

[illegible] HTTP [illegible]

[illegible]

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

[illegible] kernel.debug [illegible]

- enabled[illegible] UnexpectedSessionUsageException [illegible]
- disabled[illegible]

[illegible]

[illegible] URL

[illegible]

- [illegible] URL [illegible] ;
- [illegible] URL[illegible]

[illegible] URL[illegible] HTML [illegible] `<a href=“...”>` [illegible] URL [illegible]

[illegible] URL [illegible] blog_show [illegible] slug = my-blog-post [illegible]

[illegible] name [illegible] Symfony [illegible]

[illegible] __invoke [illegible] Symfony [illegible] FQCN [illegible] Symfony [illegible]

```
// 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 [illegible] App\Controller\MainController::homepage .

[illegible] Controller [illegible] URL

[illegible] AbstractController [illegible] generateUrl [illegible]

```
// 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']);

        // ...
    }
}
```

[illegible] AbstractController [illegible]

[illegible] Services [illegible] URL

[illegible] Symfony [illegible] generate [illegible] UrlGeneratorInterface [illegible]

```
// 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”：

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

现在，在命令中生成 URL 时，您将获得预期的结果：

```
// 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']);

        // ...
    }
}
```

[illegible]

[illegible] URL [illegible] getRouteCollection [illegible]

[illegible] URL [illegible] RouteNotFoundException [illegible]

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

// ...

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

[illegible] URL [illegible] HTTPS

[illegible] URL [illegible] HTTP [illegible] HTTP [illegible] URL [illegible] http [illegible] getContext [illegible]

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

[illegible] schemes [illegible]

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

URL HTTPS path Twig URL HTTP URL 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') }}
```

scheme HTTP /login URL URL HTTPS

HTTPS HTTPS

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

URI

[illegible] URI [illegible] URI [illegible] URI[illegible] URI [illegible] URI [illegible] URI [illegible]

Symfony [illegible] UriSigner [illegible] URI [illegible]

```
// 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);
    }
}
```

[illegible] URI [illegible] URI [illegible] sign[illegible] $expiration [illegible] /[illegible]

```php
// 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'
    }
}
```

```
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'}) }}
```

控制器

控制器是您创建的 PHP 函数，它从 Request 对象中读取信息并创建并返回 Response 对象。响应可以是 HTML 页面、 JSON 、 XML、文件下载、重定向、 404 错误或其他任何内容。控制器运行应用程序呈现页面内容所需的任意逻辑。

基础控制器

虽然控制器可以是任何 PHP 可调用的（函数、对象上的方法或 Closure ），但控制器通常是控制器类中的一个方法：

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

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

class LuckyController
{
    #[Route('/lucky/number/{max}', name: 'app_lucky_number')]
    public function number(int $max): Response
    {
        $number = random_int(0, $max);

        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
```

控制器是 number() 方法，它位于控制器类 LuckyController 中。

这个控制器非常简单：

- 第 2 行： Symfony 利用 PHP 的命名空间功能来命名空间整个控制器类。
- 第 4 行： Symfony 再次利用了 PHP 的命名空间功能： use 关键字导入了控制器必须返回的 Response 类。