Symfony 6 Routing

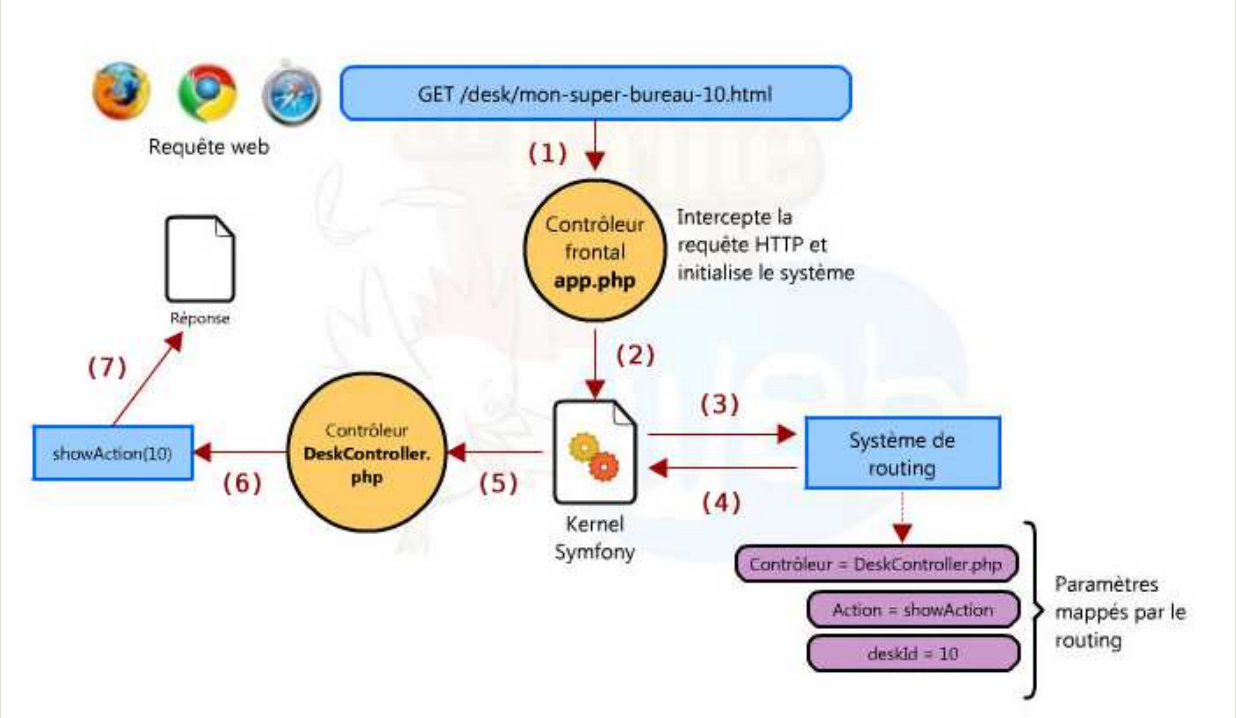
AYMEN SELLAOUTI

Introduction

Système permettant de

- gérer les liens internes du projet
- avoir des URLs plus explicites
- associer une URL à une action

Introduction



Cette architecture illustre le fonctionnement de Symfony.
1- Requête de l'utilisateur
2- La requête est envoyé vers le noyau de Symfony.
3- Le Noyau consulte le routeur afin de connaitre quel Action exécuter.
4- Le routeur envoi les informations concernant l'URI
5- Le noyau invoque l'action exécuter.
6- Exécution de l'action
7- L'action retourne la réponse.

Routing (http://www.lafermeduweb.net/)

Format de gestion du routing

Les fichiers de routing peuvent être de quatre formats différents :

- YAML
- XML
- PHP
- Annotations
- **Attributs**

Squellete d'une route

- Jusqu'à la version 3, Sensio recommande l'utilisation du format Yaml au sein des applications. Les bundles développés sont partagés entre deux formats : le XML e le Yaml.
- Sensio a décidé de recommander Yaml parece qu'il est « *user-friendly* ».
- A partir de la version 3.4, la documentation s'est focalisé essentiellement sur les annotation et la recommande. Nous allons donc voir ces deux formats.
- Dans la version 8 de PHP, une nouvelle syntaxe plus lisible et plus fonctionnelle a été introduite : les **attributs**.

Squelette d'une route en utilisant les Annotations et les attributs

```php
/**
 *
 * @Route("/blog", name="blog_list")
 */
public function list()
{
    // ...
}
```

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

Squelette d'une route en utilisant YAML

```yaml
# config/routes.yaml
blog_list:
    # Matches /blog exactly
    path:       /blog
    controller: App\Controller\BlogController::list
```

https://symfony.com/doc/current/routing.html

Squelette d'une route en utilisant XML

```xml
<!-- config/routes.xml -->
<?xml version="1.0" encoding="UTF-8" ?>
<routes xmlns="http://symfony.com/schema/routing"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://symfony.com/schema/routing
        https://symfony.com/schema/routing/routing-1.0.xsd">

    <!-- Matches /blog exactly -->
    <route id="blog_list" path="/blog"
controller="App\Controller\BlogController::list">
        <!-- settings -->
    </route>
</routes>
```

https://symfony.com/doc/current/routing.html

Squelette d'une route en utilisant PHP

```php
<?php
// config/routes.php
use App\Controller\BlogController;
use
Symfony\Component\Routing\Loader\Configurator\RoutingConfigurator;

return function (RoutingConfigurator $routes) {
// Matches /blog exactly
$routes->add('blog_list', '/blog')->controller(
[BlogController::class, 'list']
);
};
```

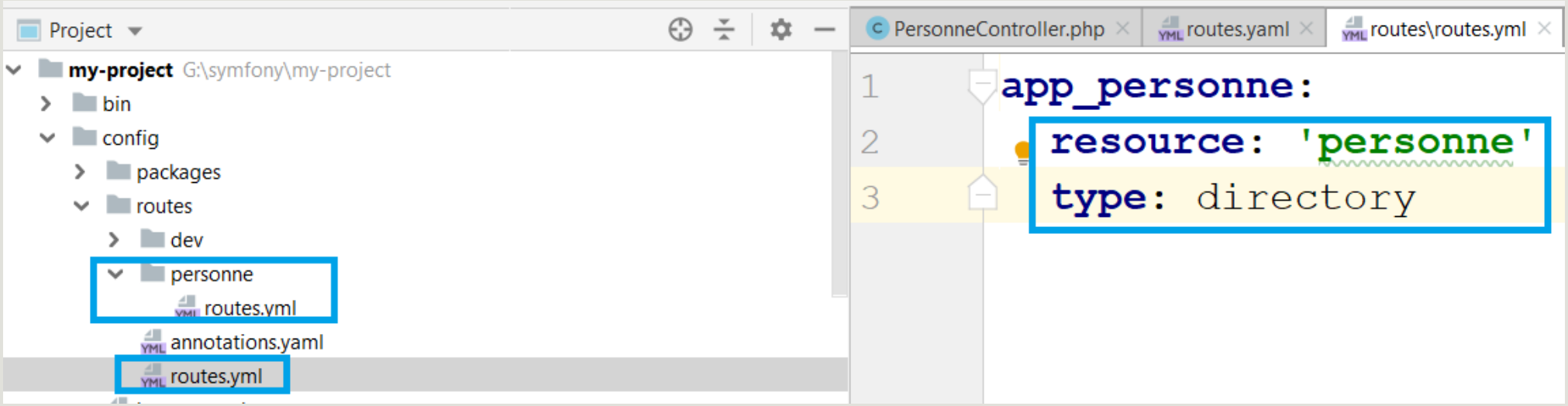
https://symfony.com/doc/current/routing.html

Organisation des routes YAML

- Lorsque vous utilisez YAML, il est préférable de ne pas centraliser l'ensemble de vos routes dans un même fichier. Ceci peut nuire à la lisibilité de vos routes.
- Décomposer vos routes en des fichiers logiques. Exemple si vous avez une partie Back et une partie front, ayez deux fichiers back.yaml et front.yaml.
- Garder le fichier principal de vos routes pour appeler ces fichiers la.

Organisation des routes YAML

- Afin d'identifier un fichier de ressources « route» utiliser la clé **ressource** afin d'informer le chemin de la ressource et la clé **type** afin de spécifier le type de votre ressource (dans notre cas c'est directory).



Les préfixes

- Dans certains cas vous avez besoins d'avoir des **endpoints particuliers pour un ensemble de fonctionnalités**. Par exemple lorsque vous aller réaliser une partie administration vous aurez généralement des routes qui commencent par /admin.

- Afin de gérer ca, le routeur de Symfony offre la possibilité d'ajouter des **préfixes** à vos routes.

Les préfixes YAML

- Afin de **préfixer** une route YAML aller dans le fichier ou vous avez appelé la ressource et ajouter la clé prefix :

```yaml
app_personne:
  resource: 'personne'
  type: directory
  prefix: /personne
```

https://symfony.com/doc/4.2/routing/external_resources.html

Préfixe annotation

- Une annotation Route sur une classe Contrôleur définit un préfixe sur toutes les routes des actions de ce contrôleur

```
/**
 * @Route("/personne")
 */
class PersonneController extends AbstractController
{
}
```

```
#[Route('/first']
class FirstController extends AbstractController
```

Paramétrage de la route : Yaml

- Nous pouvons ajouter autant de paramètre dans la route
- Le séparateur est '/'

- Exemple

```
front_article:
    path:       /article/{year}/{langue}/{slug}/{format}
    controller: App\Controller\BlogController::add
```

- Ici l'url doit contenir l'année de l'article {year}, la langue de l'article{langue} les mots clefs {slug} ainsi que le format {fomrat}

Paramétrage de la route : Annotation

```
/**
 * @Route("/hello/{name}", name="front_homepage")
 */
public function test ($name){}

/**
 * @Route("/article/{year}/{locale}/{slug}/{format}",
name="front_article")
 */
public function showArticle($year,$locale,$slug,$format){}
```

Paramétrage de la route : Attributs

```php
#[Route('/second/{name}', name: 'app_second')]
public function index($name): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
    ]);
}
```

Paramétrage de la route : valeurs par défaut Annotations

- En utilisant les annotations, nous ajoutons un champ defaults qui contiendra les valeurs par défaut.

```
/**
 * @Route(
 *      "/article/{year}/{locale}/{slug}/{format}",
 *       name="front_article",
 *       defaults={"format":"html", "slug":"Symfony"}
 * )
 */
public function showArticleAction($year,$_locale,$slug,$_format){
    dump($year,$_locale,$slug,$_format);
    die();
}
```

Important : Seul les paramètres optionnels terminant la route pourront être absents de l'URL

- Maintenant l'URL suivante devient correcte : http://127.0.0.1:8000/article/2005/fr et les variables slug et format prendront leur valeur par défaut

Paramétrage de la route : valeurs par défaut Annotations Raccourci

- Vous pouvez utiliser le raccourci { attribut**?defaultValue**}.
- Evitez ce type de raccourci si vous avez des routes complexes afin d'avoir une bonne lisibilité de vos routes.

```
/**
 * @Route(
 *      "/article/{year}/{locale}/{slug?symfony}/{format?html}",
 *       name="front_article",
 * )
 */
public function showArticleAction($year,$locale,$slug,$format){
    dump($year,$_locale,$slug,$_format);
    die();
}
```

Important : Seul les paramètres optionnels terminant la route pourront être absents de l'URL

- Maintenant l'URL suivante devient correcte : http://127.0.0.1:8000/article/2005/fr et les variables slug et _format prendront leur valeur par défaut

Paramétrage de la route : valeurs par défaut Attributs

```
#[Route(
    '/second/{name}',
    name: 'app_second',
    defaults: ['name' => 'aymen']
)]
public function index($name): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
    ]);
}
```

Paramétrage de la route : valeurs par défaut Attributs

```php
#[Route(
    '/second/{name?skander}',
    name: 'app_second',
)]
public function index($name): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
    ]);
}
```

Paramétrage de la route : valeurs par défaut Yaml

- Afin d'avoir des valeurs par défauts nous utilisons la syntaxe suivante :

Syntaxe

```
front_article:
    path:       /article/{year}/{locale}/{slug}/{format}
    controller:  App\Controller\BlogController::add
    defaults:
        attribut: defaultValue
```

Paramétrage de la route : Requirements Annotation

- En utilisant les annotations, nous ajoutons un champ requirements qui contiendra les différentes contraintes.

```
/**
 * @Route(
 *      "/article/{year}/{locale}/{slug}/{format}",
 *       name="front_article",
 *      defaults={"_format":"html", "slug":"Symfony"},
 *      requirements={
 *           "locale" : "fr|en",
 *           "year" : "\d{4}"
 *      }
 * )
 */
public function showArticleAction($year,$_locale,$slug,$_format){}
```

Paramétrage de la route : Requirements Annotation, le raccourci

- Vous pouvez utiliser le raccourci { attribut<requirement>}.
- Evitez ce type de raccourci si vous avez des routes complexes afin d'avoir une bonne lisibilité de vos routes.

```
/**
 * @Route(
 *      "/article/{year<\d+>}/{locale}/{slug}.{format}",
 *       name="front_article")
 */
public function showArticleAction($year,$_locale,$slug,$_format){

}
```

Paramétrage de la route : Requirements Attributs

- En utilisant les annotations, nous ajoutons un champ requirements qui contiendra les différentes contraintes.

```php
#[Route(
    '/second/{name}/{age}',
    name: 'app_second',
    requirements: ['age'=> '\d+']
)]
public function index($name, $age): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
        'myAge'
    ]);
}
```

Paramétrage de la route : Requirements Attributs

- En utilisant les annotations, nous ajoutons un champ requirements qui contiendra les différentes contraintes.

```php
#[Route(
    '/second/{name}/{age<\d+>}',
    name: 'app_second',
)]
public function index($name, $age): Response
{
    return $this->render('second/index.html.twig', [
        'myName' => $name,
        'myAge'=> $age
    ]);
}
```

Requirements autres exemples

\d équivalente à \d{1}
\d+ ensemble d'entiers

Ordre de traitement des routes (1)

Le traitement des routes se fait de la première route vers la dernière.

Attention à l'ordre d'écriture de vos routes.

front: Comment accéder au path front_pages ? Quel est le problème avec ces 2 routes

path: /front/{page}

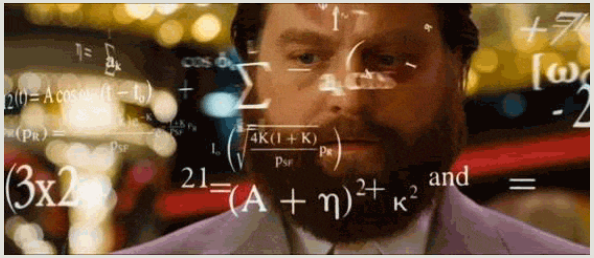
controller: App\Controller\BlogController::front

front_pages:

path: /front/{Keys}

controller: App\Controller\BlogController::show

Exercice



- Reprenez les deux routes précédentes.
- Testez l'accessibilité à la deuxième route.
- Ajouter ce qu'il faut pour remédier au problème

Ordre de traitement des routes (2)

```
front:
    path:   /front/{page}
    controller:   App\Controller\BlogController::front
}
front_pages:
    path:   /front/{Keys}
    controller:   App\Controller\BlogController::show
```

Les deux routes sont de la forme /front/* donc n'importe quel route de cette forme sera automatiquement transféré au Default controller pour exécuter l'index action.

Proposer une solution

Ordre de traitement des routes (3)

front:

path: /front/{page}

controller: App\Controller\BlogController::front

requirements:

page: \d+

front_pages:

path: /front/{Keys}

controller: App\Controller\BlogController::show

Tester le fonctionnement des routes suivantes : /front/1234

/front/test-ordre-de-routeing

Ordre de traitement des routes (4)

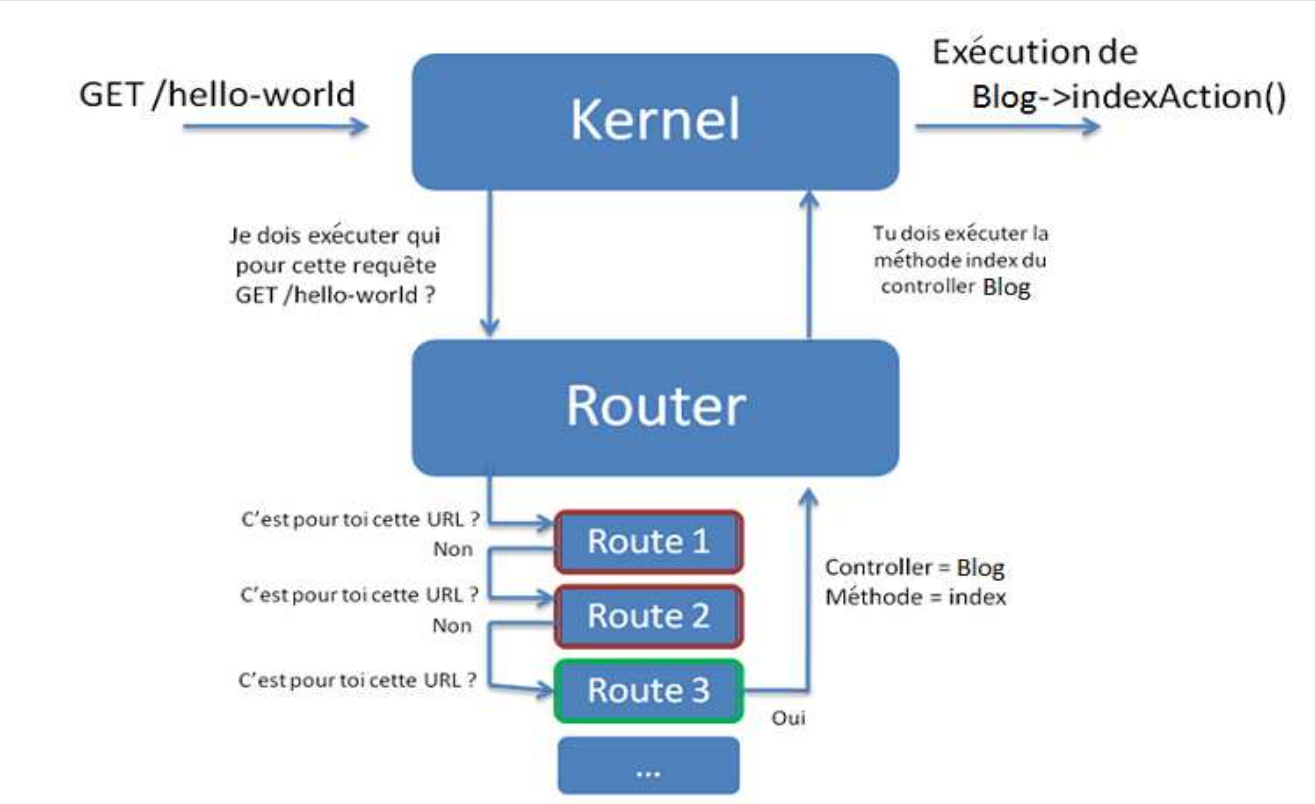
Que se passe t-il si on inverse l'ordre des deux routes ? Est-ce que la solution persiste?

```
front_pages:
    path:   /front/{Keys}
    controller:   App\Controller\BlogController::show
front:
    path:   /front/{page}
    controller:   App\Controller\BlogController::front
    requirements:
        page: \d+
```

Ordre de traitement des routes (5)

- Les Routes précédentes Gagnent toujours
- L'ordre des routes est très important.
- En utilisant un ordre clair et intelligent, vous pouvez accomplir tout ce que vous voulez.
- http://symfony.com/fr/doc/current/book/routing.html

Ordre de traitement des routes (6)



Débogage des routes

- Afin de visualiser l'ensemble des routes utiliser la debug toolbar
- Vous pouvez aussi utiliser la commande :

  symfony console debug:router

  symfony console debug:router

- On peut aussi vérifier quelle route correspond à une URL spécifique

  symfony console router:match URI

  php bin/console router:match URI