

Routing

1-Présentation

1. introduction
2. Intégration avec symfony2
3. Les extensions

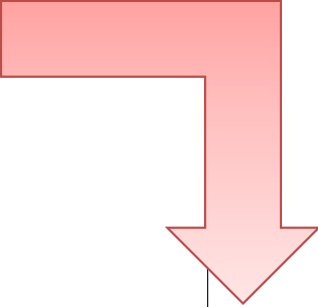
1 – introduction

Extrait du fichier app/routing.yml

```
backoffice_personal:
  resource: "@BackofficePersonalBundle/Controller/"
  type:     annotation
  prefix:   /personal

backoffice_admin:
  resource: "@BackofficeAdminBundle/Controller/"
  type:     annotation
  prefix:   /admin

# Internal routing configuration to handle ESI
#_internal:
#  resource: "@FrameworkBundle/Resources/config/routing/internal.xml"
#  prefix:   /_internal
```



```
namespace Backoffice\AdminBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Template;
use Symfony\Component\HttpFoundation\Response;
use Backoffice\AdminBundle\Entity\Produit;

class DefaultController extends Controller {

    /**
     * @Route("/", name="admin.index")
     * @Template()
     */
    public function indexAction()
    {
        $name = 'world';
        return array('name' => $name);
    }
}
```

1 – Le routage avec des annotations

Si vous souhaitez utiliser les annotations pour le routage, s'est dans le fichier de **app/config/routing.yml** que nous définissons le type des routes « **annotation** ».

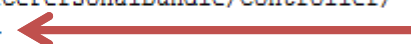
Les routes se trouveront alors renseignés directement dans les contrôleurs.

Lorsque vous déclarez vos routes de cette façon, vous devez indiquer dans **@ressources** le dossier des contrôleurs de votre bundle.

Ci-dessous un résumé complet d'annotations qui sont équivalentes au système de routage par fichier **routing.yml**

app/config/routing.yml

```
backoffice_personal:
  resource: "@BackofficePersonalBundle/Controller/"
  type:     annotation
  prefix:   /personal
```



Src/Backoffice/personalBundle/Resources/config/config.yml

```
/**
 *
 * @Route("/news", name="news_list", defaults={"page" = 1})
 * @Route("/news/{page}", name="news_list_page", requirements={"page" = "\d+"})
 * @Method("POST")
 * @Template("BackofficeAdminBundle:Produit:new.html.twig")
 */
public function indexAction($page)
{
    return array();
}
```

1 – introduction

Extrait du fichier app/routing.yml

```
backoffice_personal:  
  resource: "@BackofficePersonalBundle/Resources/config/routing.yml"  
  prefix:   /personal
```

Nous gérons cette fois ci les routes à l'aide d'un fichier externe présent dans le bundle.

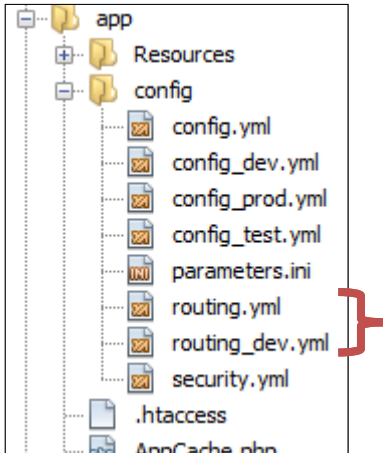
Attention les annotations ne sont pas prises en charge pour le routage

```
/**  
* @Route()  
*/
```

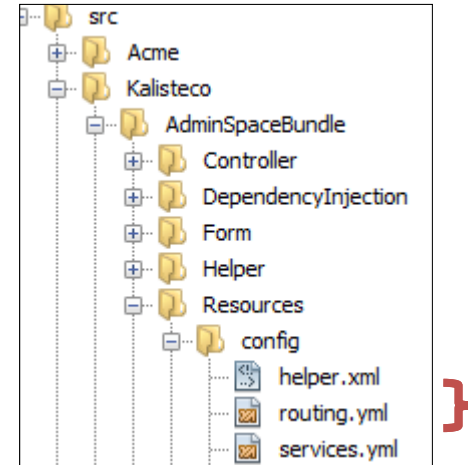
1 – introduction

Le fichier de routage principal se trouve dans le répertoire **/app/config**
et dans le répertoire **/config** de chaque bundle

app/config



src/AdminSpacBundle:Resources/config/routing.yml



Par défaut, symfony2 dispose de 2 fichiers de routage **routing.yml** et **routing_dev.yml**.

Les 2 fichiers correspondent aux 2 environnements par default de symfony ('prod' et 'dev')

Ex: L'url `.../app.php/admin` chargera **routing.yml**

L'url `.../app_dev.php/admin` chargera **routing_dev.yml** (**par défaut il chargera également routing.yml**)

```
# app/config/config.yml
framework:
    # ...
    router: { resource: "%kernel.root_dir%/config/routing.yml" }
```

1 – Routes simple

```
blog:
  pattern:  /blog
  defaults: { _controller: AcmeBlogBundle:Blog:index }
```

La route redirige vers l'action `indexAction` du fichier `src/Acme/BlogBundle/BlogController.php` -> `indexAction()`

```
blog:
  pattern:  /blog/{page}
  defaults: { _controller: AcmeBlogBundle:Blog:index }
```

Nous avons ajouté la variable `{page}`, cependant la route `/blog` n'existe plus.

Les routes `/blog/4` ou `/blog/mon_article` sont fonctionnelles.

```
blog:
  pattern:  /blog/{page}
  defaults: { _controller: AcmeBlogBundle:Blog:index, page: 1 }
```

L'ajout de la clef `page` dans `defaults` rend de nouveau disponible la route `/blog` avec comme valeur `page = 1`

Les routes `/blog/4` ou `/blog/mon_article` sont toujours fonctionnelles.

1 – Confusions de route

```
blog:
  pattern:  /blog/{page}
  defaults: { _controller: AcmeBlogBundle:Blog:index, page: 1 }

blog_show:
  pattern:  /blog/{slug}
  defaults: { _controller: AcmeBlogBundle:Blog:show }
```

Les 2 routes suivantes exécutent des actions différentes.

Cependant la composition des l'url sont identiques est rien ne peut les différencier.

Logiquement Symfony2 s'arrête à la première route correspondante: /blog/{page}

```
blog:
  pattern:  /blog/{page}
  defaults: { _controller: AcmeBlogBundle:Blog:index, page: 1 }
  requirements:
    page: \d+

blog_show:
  pattern:  /blog/{slug}
  defaults: { _controller: AcmeBlogBundle:Blog:show }
```

/blog/1 (route **blog**)

/blog/mon_article (**blog_show**)

Requirements permet d'ajouter des contraintes de validation de la route.

Maintenant la valeur **{page}** doit être numérique pour activer la route **blog**:

1 – Les arguments requirements

requirements et les variables

```
homepage:  
  pattern:  /{culture}  
  defaults: { _controller: AcmeDemoBundle:Main:homepage, culture: en }  
  requirements:  
    culture: en|fr
```

Vous pouvez également définir des valeurs prédéfinies avec la propriété **requirements**

/	route valide
/fr	route valide
/en	route valide
/it	route non valide

requirements: _method

```
contact:  
  pattern: /contact  
  defaults: { _controller: AcmeDemoBundle:Main:contact }  
  requirements:  
    _method: GET  
  
contact_process:  
  pattern: /contact  
  defaults: { _controller: AcmeDemoBundle:Main:contactProcess }  
  requirements:  
    _method: POST
```

Les 2 routes sont identiques, seulement la méthode pour activer une route ne sont pas les mêmes.

Cette méthode est utile si vous souhaitez :

- afficher le formulaire en GET
- enregistrer le formulaire lorsque l'utilisateur envoie en POST son formulaire.

1 – Les arguments requirements

Requirements: `_format`

```
article_show:
  pattern: /articles/{culture}/{year}/{title}.{_format}
  defaults: { _controller: AcmeDemoBundle:Article:show, _format: html }
  requirements:
    culture: en|fr
    _format: html|rss
    year: \d+
```

La variable **culture** doit être soit **en** ou **fr**.

La variable **_format** doit être soit **html**, **rss** ou **null**.

<code>/articles/fr/2012/mon_article</code>	route valide
<code>/articles/fr/2012/mon_article.html</code>	route valide
<code>/articles/en/2009/25.rss</code>	route valide
<code>/articles/it/2012/mon_article.xml</code>	route non valide

1 – Les arguments requirements

Requirements: `_locale`

```
contact:
  pattern: /{_locale}/contact
  defaults: { _controller: AcmeDemoBundle:Contact:index, _locale: en }
  requirements:
    _locale: en|fr|de
```

La variable **`_locale`** est un paramètre spécial.

La locale correspondante sera automatiquement définie dans la session de l'utilisateur.

Vous pourrez différencier la même action du contrôleur avec autant d'adresses url qu'il existe de langues.

[`/fr/contact`](#) et [`/en/contact`](#) pointent sur le même contrôleur. Ils retourneront une vue traduite et différente suivant la valeur locale.

Pour le référencement, il est **obligatoire** d'avoir autant d'adresses que de langue pour une même page.

Les moteurs de recherche ne comprendront pas si 2 url retournent 2 contenus différents. (anglais et français selon l'utilisateur)

1 – Autres arguments

prefix

app/config/routing.yml

```
acme_demo:
  resource: "@AcmeDemoBundle/Resources/config/routing.yml"
  prefix:   /admin
```

src/Acme/DemoBundle/Resources/config/config.yml

```
article_show:
  pattern:  /articles/{culture}/{year}/{title}.{_format}
  defaults: { _controller: AcmeDemoBundle:Article:show, _format: html }
  requirements:
    culture:  en|fr
    _format:  html|rss
    year:     \d+
```

Le premier fichier de routage (ici).

/admin/articles/fr/2012/mon_article