eBook Gratuit

APPRENEZ symfony

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#symfony

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: symfony

It is an unofficial and free symfony ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official symfony.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec symfony

Remarques

Symfony est un ensemble de composants PHP réutilisables, utilisables séparément ou dans le cadre de Symfony Framework.

Comme la plupart des frameworks, Symfony résout les problèmes techniques récurrents pour vous (tels que l'authentification, le routage, etc.) afin que vous puissiez vous concentrer sur les problèmes métier que vous essayez de résoudre.

Contrairement aux autres frameworks, cependant, les composants Symfony sont découplés les uns des autres, vous permettant de sélectionner ceux dont vous avez besoin. Au lieu d'adapter votre application à votre framework, vous pouvez adapter le framework à vos besoins.

C'est ce qui rend Symfony très populaire et permet à d'autres projets et frameworks (y compris Laravel, Drupal, Magento et Composer) d'utiliser les composants sans avoir à utiliser le framework complet.

Open source

Symfony est un projet open-source. Voyez comment vous pouvez contribuer .

Documentation officielle

La documentation officielle de Symfony se trouve sur le site Web de Symfony.

Versions

Symfony 3

Version	Fin de vie	Date de sortie
3.3	07/2018	2017-05-29
3.2	01/2018	2016-11-30
3.1	07/2017	2016-05-30
3.0	01/2017	2015-11-30

Symfony 2

Version	Fin de vie	Date de sortie
2.8	11/2019	2015-11-30
2.7	05/2019	2015-05-30
2.6	01/2016	2014-11-28
2,5	07/2015	2014-05-31
2.4	01/2015	2013-12-03
2.3	05/2017	2013-06-03
2.2	05/2014	2013-03-01
2.1	11/2013	2012-09-06
2.0	09/2013	2011-07-28

Examples

Créer un nouveau projet Symfony à l'aide du programme d'installation de Symfony

Le programme d' installation de Symfony est un outil de ligne de commande qui vous aide à créer de nouvelles applications Symfony. Il nécessite PHP 5.4 ou supérieur.

Téléchargement et installation du programme d'installation de Symfony sous Linux / MacOS

Ouvrez un terminal et exécutez les commandes suivantes:

```
sudo mkdir -p /usr/local/bin
sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
sudo chmod a+x /usr/local/bin/symfony
```

Cela crée un exécutable global `symfony` qui peut être appelé de n'importe où. Vous devez le faire une seule fois: vous pouvez désormais créer autant de projets Symfony que vous le souhaitez.

Créer un nouveau projet avec la dernière version de Symfony

Une fois le programme d'installation installé, vous pouvez l'utiliser pour créer un nouveau projet Symfony. Exécutez la commande suivante:

```
symfony new my_project_name
```

Cette commande créera un nouveau répertoire (appelé `my_project_name`) contenant la version la plus récente de Symfony Standard Edition . Il installera également toutes ses dépendances (y compris les composants Symfony réels) en utilisant Composer.

Créer un nouveau projet en utilisant une version spécifique de Symfony

Si vous souhaitez sélectionner une version Symfony spécifique au lieu de la dernière, vous pouvez utiliser le second argument facultatif de la `new` commande.

Pour sélectionner une version mineure:

```
symfony new my_project_name 3.2
```

Pour sélectionner une version de patch:

```
symfony new my_project_name 3.2.9
```

Pour sélectionner une version bêta ou une version candidate:

```
symfony new my_project 2.7.0-BETA1
symfony new my_project 2.7.0-RC1
```

Pour sélectionner la version la plus récente du support à long terme (LTS):

```
symfony new my_project_name lts
```

Créer un nouveau projet Symfony en utilisant Composer

Si pour une raison quelconque, le programme d' installation de Symfony n'est pas une option, vous pouvez également créer un nouveau projet à l'aide de Composer. Tout d'abord, assurez-vous d'avoir installé Composer .

Ensuite, vous pouvez utiliser la commande `create-project` pour créer un nouveau projet:

```
composer create-project symfony/framework-standard-edition my_project_name
```

Semblable au programme d'installation de Symfony, ceci installera la dernière version de Symfony Standard Edition dans un répertoire appelé `my_project_name` et installera ensuite ses dépendances (y compris les composants Symfony).

Installation d'une version spécifique de Symfony

Comme avec Symfony Installer, vous pouvez sélectionner une version spécifique de Symfony en

fournissant un troisième argument facultatif:

```
composer create-project symfony/framework-standard-edition my_project_name "2.8.*"
```

Notez toutefois que tous les alias de version (tels que `lts` par exemple) ne sont pas disponibles ici.

Exécution de l'application Symfony à l'aide du serveur Web intégré de PHP

Après avoir créé une nouvelle application Symfony , vous pouvez utiliser la commande `server:run` pour démarrer un serveur Web PHP simple, afin que vous puissiez accéder à votre nouvelle application depuis votre navigateur Web:

```
cd my_project_name/
php bin/console server:run
```

Vous pouvez maintenant visiter http: // localhost: 8000 / pour voir la page d'accueil de Symfony.

Important : si vous utilisez le serveur Web intégré pour le développement, vous ne devez **pas l'** utiliser en production. Utilisez plutôt un serveur Web complet tel qu'Apache ou Nginx.

Lire Démarrer avec symfony en ligne: https://riptutorial.com/fr/symfony/topic/9448/demarrer-avec-symfony

Chapitre 2: Conteneur de service

Introduction

Une application Symfony est généralement composée de nombreux objets exécutant différentes tâches, telles que des référentiels, des contrôleurs, des mailers, etc. Dans Symfony, ces objets sont appelés **services** et sont définis dans `app/config/services.yml` ou dans l'un des les bundles installés.

Le **conteneur de service** sait instancier ces services et en garde une référence afin qu'ils n'aient pas à être instanciés deux fois. Si un service a des dépendances, il les instanciera aussi.

Examples

Récupérer un service à partir du conteneur

```
$logger = $container->get('logger');
```

Cela ira chercher le service avec l'ID de service "logger" du conteneur, un objet qui implémente `Psr\Log\LoggerInterface` .

Lire Conteneur de service en ligne: https://riptutorial.com/fr/symfony/topic/10183/conteneur-de-service

Chapitre 3: Contrôleurs

Introduction

Un contrôleur dans Symfony est un appel PHP (une fonction, une méthode sur un objet ou une fermeture) qui reçoit une requête HTTP et renvoie une réponse HTTP. Une réponse HTTP peut contenir n'importe quoi: une page HTML, une chaîne JSON, un téléchargement de fichier, etc.

Pour indiquer à Symfony quel contrôleur doit gérer une requête donnée, vous devez configurer une route .

Syntaxe

- $ this-> generateUrl ('route_name', ['placeholder' => 'value']); // génère une URL pour la route `route_name` route avec un espace réservé
- $ this-> render ('template.html.twig'); // affiche un modèle Twig et renvoie un objet Response
- $ this-> render ('template.html.twig', ['paramètre' => $ value]); // rend un modèle Twig avec un paramètre
- lancer $ this-> createNotFoundException ('Message'); // lance une exception NotFoundHttpException qui fera que Symfony renvoie une réponse 404

Remarques

Les contrôleurs doivent être petits et se concentrer sur le traitement des requêtes HTTP: la logique métier de votre application doit être déléguée à différentes parties de votre application, par exemple votre modèle de domaine.

Examples

Une classe de contrôleur simple

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

use Symfony\Component\HttpFoundation\Response;

class HelloWorldController
{
    public function helloWorldAction()
    {
        return new Response(
            '<html><body>Hello World!</body></html>'
        );
    }
}
```

Rendu d'un modèle Twig

La plupart du temps, vous souhaiterez rendre les réponses HTML à partir d'un modèle au lieu de coder en dur le code HTML de votre contrôleur. En outre, vos modèles ne seront pas statiques mais contiendront des espaces réservés pour les données d'application. Par défaut, Symfony est fourni avec Twig, un puissant langage de modélisation.

Pour utiliser Twig dans votre contrôleur, étendez la classe de base du `Controller` Symfony:

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

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

class HelloWorldController extends Controller
{
    public function helloWorldAction()
    {
        $text = 'Hello World!';

        return $this->render('hello-world.html.twig', ['text' => $text]);
    }
}
```

Créez le modèle Twig (situé dans `app/Resources/views/hello-world.html.twig`):

```
<html><body>{{ text }}</body></html>
```

Twig remplacera automatiquement le paramètre `{{ text }}` par la valeur du paramètre `text` transmis par le contrôleur. Cela rendra la sortie HTML suivante:

```
<html><body>Hello World!</body></html>
```

Retourner une page 404 (non trouvée)

Parfois, vous souhaitez renvoyer une réponse 404 (non trouvée), car la ressource demandée n'existe pas. Symfony vous permet de le faire en lançant une `NotFoundHttpException` .

Le contrôleur de base Symfony expose une méthode `createNotFoundException` qui crée l'exception pour vous:

```
public function indexAction()
{
    // retrieve the object from database
    $product = ...;

    if (!$product) {
        throw $this->createNotFoundException('The product does not exist');
    }

    // continue with the normal flow if no exception is thrown
```

```
    return $this->render(...);
}
```

Utilisation des données de l'objet Request

Si vous devez accéder à l'objet `Request` (par exemple, pour lire les paramètres de la requête, lire un en-tête HTTP ou traiter un fichier téléchargé), vous pouvez recevoir la requête en tant qu'argument de méthode en ajoutant un argument de type:

```
use Symfony\Component\HttpFoundation\Request;

public function indexAction(Request $request)
{
    $queryParam = $request->query->get('param');

    // ...
}
```

Symfony reconnaît l'indicateur de type et ajoute l'argument de requête à l'appel du contrôleur.

Lire Contrôleurs en ligne: https://riptutorial.com/fr/symfony/topic/10085/controleurs

Chapitre 4: La demande

Introduction

La classe Request de Symfony est une représentation orientée objet de la requête HTTP. Il contient des informations telles que l'URL, la chaîne de requête, les fichiers téléchargés, les cookies et autres en-têtes provenant du navigateur.

Syntaxe

- $ request-> getPathInfo (); // renvoie le chemin d'accès (partie locale de l'URL) demandé (mais sans la chaîne de requête). C'est-à-dire lorsque vous visitez https://example.com/foo/bar?key=value , cela contiendra / foo / bar
- $ request-> query-> get ('id'); // retourne un paramètre de chaîne de requête ($ _GET)
- $ request-> query-> get ('id', 1); // retourne un paramètre de chaîne de requête avec une valeur par défaut
- $ request-> request-> get ('name'); // retourne une variable de corps de requête ($ _POST)
- $ request-> files-> get ('pièce jointe'); // renvoie une instance de UploadedFile identifiée par "pièce jointe"
- $ request-> cookies-> get ('PHPSESSID'); // retourne la valeur d'un cookie ($ _COOKIE)
- $ request-> headers-> get ('content_type'); // retourne un en-tête de requête HTTP
- $ request-> getMethod (); // renvoie la méthode de requête HTTP (GET, POST, PUT, DELETE, etc.)
- $ request-> getLanguages (); // retourne un tableau de langues que le client accepte

Examples

Obtenir un paramètre de chaîne de requête

Disons que nous voulons créer une liste paginée de produits, où le numéro de la page est transmis en tant que paramètre de chaîne de requête. Par exemple, pour récupérer la 3ème page, vous devez aller à:

```
http://example.com/products?page=3
```

La requête HTTP brute ressemblerait à ceci:

```
GET /products?page=3 HTTP/1.1
Host: example.com
Accept: text/html
User-Agent: Mozilla/5.0 (Macintosh)
```

Pour obtenir le numéro de page de l'objet de requête, vous pouvez accéder à la propriété `query` :

```
$page = $request->query->get('page'); // 3
```

Dans le cas d'un paramètre de `page` , vous souhaiterez probablement passer une valeur par défaut si le paramètre de chaîne de requête n'est pas défini:

```
$page = $request->query->get('page', 1);
```

Cela signifie que lorsque quelqu'un visite http://example.com/products (notez l'absence de la chaîne de requête), la variable `$page` contiendra la valeur par défaut `1` .

Création d'un objet Request à partir de variables globales

PHP expose un certain nombre de *variables* dites *globales* contenant des informations sur la requête HTTP, telles que `$_POST` , `$_GET` , `$_FILES` , `$_SESSION` , etc. La classe `Request` contient une `createFromGlobals()` static `createFromGlobals()` pour instancier une requête objet basé sur ces variables:

```
use Symfony\Component\HttpFoundation\Request;

$request = Request::createFromGlobals();
```

Lorsque vous utilisez le framework Symfony, vous ne devez pas instancier l'objet de requête vous-même. Au lieu de cela, vous devez utiliser l'objet instancié lorsque le framework est démarré dans `app.php` / `app_dev.php` . Par exemple, par type, en indiquant l'objet de requête dans votre contrôleur .

Accéder à une variable POST

Pour obtenir le contenu d'un formulaire soumis avec `method="post"` , utilisez la propriété `post` :

```
$name = $request->request->get('name');
```

Accéder au contenu d'un cookie

```
$id = $request->cookies->get('PHPSESSID');
```

Cela retournera la valeur du cookie 'PHPSESSID' envoyé par le navigateur.

Lire La demande en ligne: https://riptutorial.com/fr/symfony/topic/10097/la-demande

Chapitre 5: Le routage

Introduction

Le routage est le processus de mappage d'une URL vers un contrôleur . Symfony possède un puissant composant Routing qui vous permet de définir des routes.

Le composant Routing prend en charge un certain nombre de formats de configuration: annotations, YAML, XML et PHP brut.

Paramètres

Paramètre	Détails
prénom	Le nom de l'itinéraire. Exemple: `book_show`
chemin	Le chemin (peut contenir des caractères génériques). Exemple: `/book/{isbn}`
les défauts	Valeurs par défaut des paramètres

Examples

Itinéraires simples

Utiliser YAML:

```
# app/config/routing.yml
blog_list:
    path:     /blog
    defaults: { _controller: AppBundle:Blog:list }
```

Utilisation des annotations:

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

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

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

```
}
```

Une demande pour l'URL `/blog` sera gérée par la méthode `listAction()` du `BlogController` dans `AppBundle` .

Routes avec des espaces réservés

Utiliser YAML:

```
# app/config/routing.yml
blog_show:
    path:     /blog/{slug}
    defaults: { _controller: AppBundle:Blog:show }
```

Utilisation des annotations:

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

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

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

Toute requête avec une URL correspondant à `/blog/*` sera traitée par la méthode `showAction()` du `BlogController` dans `AppBundle` . L'action du contrôleur recevra la valeur de l'espace réservé comme argument de méthode.

Par exemple, une demande pour `/blog/my-post` déclenchera un appel à `showAction()` avec un argument `$slug` contenant la valeur `my-post` . En utilisant cet argument, l'action du contrôleur peut modifier la réponse en fonction de la valeur de l'espace réservé, par exemple en récupérant la publication de blog avec le slug `my-post` de la base de données.

Valeurs par défaut pour les espaces réservés

Si vous voulez avoir un espace réservé qui peut être omis, vous pouvez lui donner une valeur par défaut:

Utiliser YAML:

```
# app/config/routing.yml
blog_list:
    path:      /blog/{page}
```

```
    defaults:  { _controller: AppBundle:Blog:list, page: 1 }
    requirements:
        page: '\d+'
```

Utilisation des annotations:

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

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

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

Dans cet exemple, les URL `/blog` et `/blog/1` correspondent à l'itinéraire `blog_list` et seront gérées par la méthode `listAction()` . Dans le cas de `/blog` , `listAction()` recevra toujours l'argument `$page` , avec la valeur par défaut `1` .

Lire Le routage en ligne: https://riptutorial.com/fr/symfony/topic/10084/le-routage

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec symfony	Braiam, Code Lover, Community, Nic Wortel, Paul Crovella
2	Conteneur de service	Nic Wortel
3	Contrôleurs	Nic Wortel
4	La demande	Nic Wortel
5	Le routage	Nic Wortel