
- Introduction : Symfony -

Qu'est-ce que Symfony ?

C'est un ensemble de composants PHP. Chaque composant solutionne un problème de développement comme d'abstraire les messages HTTP, créer des outils ligne de commande, etc. Symfony tente de rendre ces composants flexibles et extensibles.

C'est un ensemble de classes basé sur l'approche OO qui utilise PHP 5.3. Ces classes offrent des outils pour les développeurs PHP. Il y a 21 composants présentement.

Certains projets l'utilisent déjà : (Behat, Doctrine, Propel, PHPUnit, Jackalope, easybook, Midgard CMS, Zikula, phpBB, et bientôt Drupal.

- Installation

Nécessite Composer (package manager PHP)
PHP 5.3.3 et +

You are not allowed to access this file. Check app_dev.php for more information
Ouvrir le fichier et enlever le bout de code.

- View

Twig est l'engin de template utilisé par Symfony. A été créé par la même équipe.
Par contre, on peut prendre des templates PHP classique.
Template inheritance
Tags, Filters, Functions
Including another template
Embedding another controller
Links between pages (utilise le Routing avec la fonction « path ») url aussi, mais pour des urls absolues
Assets (Js, images, css) avec la fonction « asset »

- Controller

Permet de générer plus que juste du HTML
Avec format, on peut spécifier différents templates Twig selon le format demandé (xml, html, json).
Redirect & forward (utilise le Routing aussi)
Pages d'erreur (shortcut createNotFoundException pour 404, sinon une exception Symfony va faire une 500)
On reçoit la requête automatiquement (request)
Symfony ont « wrapper » la session (utilise les sessions PHP classique)
Ils ont intégrés des « flash messages » qui sont temproaires
Caching resource (utilisé Cache) en utilisant les headers (maxage, no-cache, etc)

- Architecture

Directory structure :

- app/: the application configuration; (register bundles + load config)
- src/: the project's PHP code;
- vendor/: the third-party dependencies;
- web/: the web root directory (static resources + front controller).

Tout est un "bundle" dans Symfony
Chaque « bundle » est dans un dossier différent
‘’ est configurable a l'aide de fichiers YAML, XML ou PHP
Chaque environnement peuvent changer la configuration
On peut hériter d'un bundle et redéfinir les controllers, templates, etc.

Logical filename :
`@BUNDLE_NAME/path/to/file;`

Logical Controller:
`BUNDLE_NAME:CONTROLLER_NAME:ACTION_NAME.`

Logical Template:
`BUNDLE_NAME:CONTROLLER_NAME:TEMPLATE_NAME`

Tous les fichiers de configurations sont lus qu'une seule fois et mis en cache automatiquement ensuite par Symfony. Cette cache est changée quand un fichier est changé en DEV, mais en prod, c'est manuellement que ce doit etre fait.
Log Files sont sous app/logs

Chaque application vient avec un command-line (app/console)

- Introduction : Symfony avec Drupal -

Comment Symfony est utilisé avec Drupal 8 ?

Dans Drupal8, le but était d'offrir un « framework .Core », mais pourquoi se concentrer sur cela alors que Symfony l'est déjà. Drupal est un CMS et devrait se concentrer sur le CMS et non sur les éléments de bas niveau. De plus, cela facilitera l'intégration avec d'autres applications.

- Composants « core »

Ils sont intégrés automatiquement avec Drupal 8. Ceux qui utilisent le Framework .NET, c'est une version PHP de .NET.

ClassLoader
Charger sur demande les classes PHP. Cependant, il faut suivre les conventions pour le nom. (TODO)

HttpFoundation
Parmi le plus important composant.
Remplace toutes les obtentions a partir de HTTP en PHP pour du code plus simple à tester et à utiliser. Normalement ce sont des variables globales, on doit faire pleins de vérifications et devient complexe ou facile d'y insérer des failles
Représente les messages http selon la spécification.

HttpKernel
Le plus important composant.
Implémente tout ce qui est dynamique dans la specification HTTP. Gère la conversion d'une requête à une réponse.
Représente le workflow par défaut suivant :

```
Request      -> Request event   -> if listener response -> Response
                                -> if controller -> Response
                                -> if view -> Response
```

Lorsque la réponse est envoyée, il est possible de faire un autre traitement qui n'affecte pas la réponse (ex. : envoyer un courriel)
S'il y a une exception, elle est attrapée et un événement d'exception est levé et on peut générer une réponse pour avertir le client de l'erreur.

Routing

Associer les requêtes avec quel « controller » qui va gérer la requête. On se créer une collection de routes qui est ensuite analysé lorsque la requête est reçue.
Permet aussi de gérer lorsqu'on veut rediriger vers une autre page.

Event Dispatcher
Semblable au système de « hooks », mais orienté-objet. Le code envoie un événement et le « dispatcher » est averti de cet événement et chaque « listener » peut réagir sur l'événement.
Il se poiurrait que Drupal 8 ne l'utilisera pas et garder le système de « hooks ».
Par défaut, il y a beaucoup de « listeners » de disponible et c'est très simple de se créer son propre ExceptionHandler.

DependencyInjection
Ninject, Unity
"Don't' call us, we'll call you"
Permet de faire un "mock" de tout pour tester le plus modulairement possible.
Cependant, il faut faire les associations (il n'y a pas de magie) pour lui dire quel objet instancier lorsqu'on le demande.

- Pourquoi passer à Symfony ?

Avantages
- Problématiques au niveau de la performance pour un site utilisant Drupal avec beaucoup d'interaction (avec plusieurs « custom fields »)
- Configuration dans la BD vient complexifier le déploiement de modules entre différents environnements. Features est venu régler ce problème cependant.
- DoctrineORM est plus robuste que le « custom » de Drupal

Inconvénients
- Plus de code
- Déploiement en production est plus difficile avec Symfony, du moins dépendre des « git » des « third-party »

- Conclusion

Symfony est une approche plus professionnel, performance et facilement maintenant pour une équipe de développeur.
Drupal est plus plaisant, moins de code et plus de développement et de déploiement (par contre, parfois quick&dirty).