Le format YAML

A

MOTS-CLÉS :

- Types de données
- Tableaux et collections
- Configuration dans Symfony

La plupart des fichiers dans Symfony sont écrits en format YAML. D'après le site officiel de YAML, il s'agit d'un « format standard de sérialisation de données, lisible pour tout être humain, pour tous les langages de programmation… ». La syntaxe de YAML est particulièrement riche et ces quelques pages dévoilent la plupart des concepts du format YAML et ce qu'ils permettent de décrire avec les outils de Symfony.

Bien que le format YAML puisse décrire des structures de données imbriquées complexes, cette annexe décrit seulement le jeu de fonctionnalités nécessaires pour utiliser YAML en guise de simple format de fichier de configuration pour Symfony.

YAML est un langage simple dédié à la description de données. Comme PHP, il dispose d'une syntaxe pour les types de données primitifs tels que les chaînes de caractères, les booléens, les nombres décimaux ou encore les nombres entiers. Néanmoins, contrairement à PHP, il est capable de faire la différence entre les tableaux (séquences) et les tables de hachage (mapping).

Les données scalaires

Les sections suivantes décrivent la syntaxe des données scalaires qui sont finalement très proches de la syntaxe de PHP. Ces données incluent entre autres les entiers, les nombres décimaux, les booléens ou les chaînes de caractères.

Les chaînes de caractères

Tous les bouts de code présentés dans cette section décrivent la manière de déclarer des chaînes de caractères en format YAML.

```
A string in YAML
'A singled-quoted string in YAML'
```

Une chaîne de caractères simple peut contenir des apostrophes. Le format YAML impose de doubler les apostrophes intermédiaires pour les échapper.

```
'A single quote '' in a single-quoted string'
```

La syntaxe avec des guillemets est notamment utile lorsque la chaîne de caractères démarre ou se termine par des caractères d'espacement spéciaux tels que les sauts de ligne.

```
"A double-quoted string in YAML\n"
```

Le style avec les guillemets fournit un moyen d'exprimer des chaînes arbitraires en utilisant les séquences d'échappement `\`. C'est notamment pratique lorsqu'il est besoin d'embarquer un `\n` ou un caractères Unicode dans une chaîne.

Lorsqu'une chaîne de caractères contient des retours à la ligne, il est possible d'utiliser le style littéral, marqué par un caractère *pipe*|, qui indique que la chaîne contiendra plusieurs lignes. En mode littéral, les nouvelles lignes sont préservées.

```
| \/ /| |\/| |
  / / | |  | |__
```

Alternativement, les chaînes de caractères peuvent être écrites avec la syntaxe repliée, marquée par un caractère >, où chaque retour à la ligne est remplacé par une espace.

```
>
   This is a very long sentence
   that spans several lines in the YAML
   but which will be rendered as a string
   without carriage returns.
```

Il est bon de remarquer les deux espaces avant chaque ligne dans les exemples précédents. Ils n'apparaîtront pas dans les chaînes de caractères PHP finales.

Les nombres

Cette section décrit les différentes manières de déclarer des valeurs numéraires en format YAML tels que les entiers, les nombres à virgules flottantes, les nombres octaux ou bien l'infini.

Les entiers

Un entier se déclare simplement en indiquant sa valeur.

```
# Un entier
12
```

Les nombres octaux

Un nombre octal se déclare de la même manière qu'un nombre entier en le préfixant par un zéro.

```
# Un nombre octal
014
```

Les nombres hexadécimaux

Un nombre hexadécimal se déclare en préfixant sa valeur par `0x`.

```
# Un nombre hexadécimal
0xC
```

Les nombres décimaux

Un nombre décimal se déclare de la même manière qu'un entier en indiquant sa partie décimale avec un point.

```
# Un nombre décimal
13.4
```

Les nombres exponentiels

Un nombre exponentiel peut aussi être représenté en format YAML de la manière suivante.

```
# Un nombre exponentiel
1.2e+34
```

Les nombres infinis

L'infini se représente à l'aide de la valeur `.inf`.

```
# L'infini
.inf
```

Les valeurs nulles : les NULL

Une valeur nulle se matérialise en YAML avec la valeur `null` ou le caractère tilde `~`.

Les valeurs booléennes

Les valeurs booléennes sont décrites à l'aide des chaînes de caractères `true` et `false`. Il faut également savoir que l'analyseur syntaxique YAML du framework Symfony reconnaît une valeur booléenne si elle est exprimée avec l'une de ces valeurs : `on`, `off`, `yes` et `no`. Néanmoins, il est fortement déconseillé de les utiliser depuis qu'elles ont été supprimées des spécifications de Symfony 1.2.

Les dates

Le format YAML utilise le standard ISO 8 601 pour exprimer les dates.

```
# Une date complète
2001-12-14t21:59:43.10-05:00

# Une date simple
2002-12-14
```

Les collections

Un fichier YAML est rarement utilisé pour décrire uniquement des simples valeurs scalaires. La plupart du temps, c'est pour décrire une collection de valeurs. Une collection peut être exprimée à l'aide d'une liste de valeurs ou avec une association d'éléments. Les séquences et les associations sont toutes deux converties sous forme de tableaux PHP.

Les séquences d'éléments

Les séquences d'éléments s'expriment à l'aide d'un tiret - suivi d'un espace pour chacun de leurs éléments. Le code ci-dessous présente une séquence simple de valeurs.

```
- PHP
- Perl
- Python
```

Après analyse, ce code YAML est converti sous la forme d'un tableau PHP simple identique à celui ci-dessous.

```
array('PHP', 'Perl', 'Python');
```

Les associations d'éléments

Les associations simples

Les associations se représentent à l'aide d'un caractère : suivi d'une espace pour exprimer chaque couple « clé => valeur ».

```
PHP: 5.2
MySQL: 5.1
Apache: 2.2.20
```

Après analyse, ce code YAML est converti sous la forme d'un tableau PHP associatif identique à celui ci-dessous.

```
array('PHP' => 5.2, 'MySQL' => 5.1, 'Apache' => '2.2.20');
```

Il est bon de savoir que chaque clé d'une association peut être exprimée à l'aide de n'importe quelle valeur scalaire valide. D'autre part, le nombre d'espaces après les deux points est complètement arbitraire. Par conséquent, le code ci-dessous est strictement équivalent au précédent.

```
PHP:    5.2
MySQL: 5.1
Apache: 2.2.20
```

Les associations complexes imbriquées

Le format YAML utilise des indentations avec un ou plusieurs espaces pour exprimer des collections de valeurs imbriquées comme le montre le code ci-après.

```
"symfony 1.0":
  PHP:    5.0
  Propel: 1.2
"symfony 1.2":
  PHP:    5.2
  Propel: 1.3
```

Après analyse, ce code YAML est converti sous la forme d'un tableau PHP associatif à deux niveaux identique à celui ci-dessous.

```
array(
  'symfony 1.0' => array(
    'PHP'    => 5.0,
    'Propel' => 1.2,
  ),
  'symfony 1.2' => array(
    'PHP'    => 5.2,
    'Propel' => 1.3,
  ),
);
```

Il y a tout de même une chose importante à retenir lorsque l'on utilise les indentations dans un fichier YAML : les indentations sont toujours réalisées à l'aide d'un ou plusieurs espaces mais jamais avec des tabulations.

Combinaison de séquences et d'associations

Les séquences et les associations peuvent également être combinées de la manière suivante :

```
'Chapter 1':
  - Introduction
  - Event Types
'Chapter 2':
  - Introduction
  - Helpers
```

Syntaxe alternative pour les séquences et associations

Les séquences et les associations possèdent toutes deux une syntaxe alternative sur une ligne à l'aide de délimiteurs particuliers. Ces syntaxes permettent ainsi de se passer des indentations pour marquer les différents *scopes*.

Cas des séquences

Une séquence de valeurs peut aussi s'exprimer à l'aide d'une liste entourée par des crochets `[]` d'éléments séparés par des virgules.

```
[PHP, Perl, Python]
```

Cas des associations

Une association de valeurs peut aussi s'exprimer à l'aide d'une liste entourée d'accolades `{}` de couples clé/valeur séparés par des virgules.

```
{ PHP: 5.2, MySQL: 5.1, Apache: 2.2.20 }
```

Cas des combinaisons de séquences et d'associations

De même que précédemment, la combinaison des séquences et des associations reste possible avec leur syntaxe alternative respective. Par conséquent, il ne faut pas hésiter à les utiliser pour arriver à une meilleure lisibilité du code.

```
'Chapter 1': [Introduction, Event Types]
'Chapter 2': [Introduction, Helpers]

"symfony 1.0": { PHP: 5.0, Propel: 1.2 }
"symfony 1.2": { PHP: 5.2, Propel: 1.3 }
```

Les commentaires

Le format YAML accepte l'intégration de commentaires pour documenter le code en les préfixant par le caractère dièse #.

```
# Commentaire sur une ligne
"symfony 1.0": { PHP: 5.0, Propel: 1.2 } # Commentaire à la fin
d'une ligne
"symfony 1.2": { PHP: 5.2, Propel: 1.3 }
```

Les commentaires sont simplement ignorés par l'analyseur syntaxique YAML et n'ont pas besoin d'être indentés selon le niveau d'imbrications courant dans une collection.

Les fichiers YAML dynamiques

Dans Symfony, un fichier YAML peut contenir du code PHP qui sera ensuite évalué juste avant que l'analyse du code YAML finale ne se produise.

```
1.0:
  version: <?php echo file_get_contents('1.0/VERSION')."\n" ?>
1.1:
  version: "<?php echo file_get_contents('1.1/VERSION') ?>"
```

Il faut bien garder à l'esprit ces quelques petites astuces lorsque du code PHP est ajouté au fichier YAML afin ne pas risquer de désordonner l'indentation.

- L'instruction `<?php ?>` doit toujours démarrer une ligne ou être embarquée dans une valeur.
- Si une instruction `<?php ?>` termine une ligne, alors il est nécessaire de générer explicitement une nouvelle ligne en sortie à l'aide du marqueur `\n`.

Exemple complet récapitulatif

L'exemple ci-dessous illustre la plupart des notations YAML expliquées tout au long de cette annexe sur le format YAML.

```
"symfony 1.0":
  end_of_maintainance: 2010-01-01
  is_stable:           true
  release_manager:     "Grégoire Hubert"
  description: >
    This stable version is the right choice for projects
    that need to be maintained for a long period of time.
  latest_beta:         ~
    latest_minor:        1.0.20
    supported_orms:      [Propel]
    archives:            { source: [zip, tgz], sandbox: [zip,
tgz] }

"symfony 1.2":
  end_of_maintainance: 2008-11-01
  is_stable:           true
  release_manager:     'Fabian Lange'
  description: >
    This stable version is the right choice
    if you start a new project today.
   latest_beta:         null
   latest_minor:        1.2.5
   supported_orms:
     - Propel
     - Doctrine
   archives:
     source:
       - zip
       - tgz
     sandbox:
       - zip
       - tgz
```

Le fichier de configuration settings.yml

B

MOTS-CLÉS :

- Configuration de Symfony
- Format YAML
- Fichier settings.yml

La plupart des aspects de Symfony peuvent être configurés à travers un fichier de configuration écrit en YAML ou avec du code PHP pur.

Cette annexe est consacrée à la description de tous les paramètres de configuration d'une application qui se trouvent dans le fichier de configuration principal `settings.yml` du répertoire `apps/APPLICATION/config/`.

Comme il l'a été mentionné dans l'introduction, le fichier de configuration `settings.yml` d'une application est paramétrable par environnement, et bénéficie du principe de configuration en cascade. Chaque environnement dispose de deux sous-sections : `.actions` et `.settings`. Toutes les directives de configuration se situent principalement dans la sous-section `.settings` à l'exception des actions par défaut qui sont rendues pour certaines pages communes.

Les paramètres de configuration du fichier settings.yml

Configuration de la section .actions

La sous-section `.actions` du fichier de configuration `settings.yml` contient quatre directives de configuration listées ci-dessous. Chacune d'entre elles sera décrite indépendamment dans la suite de cette annexe.

- `error_404`
- `login`
- `secure`
- `module_disabled`

Configuration de la section .settings

La sous-section `.settings` du fichier de configuration `settings.yml` contient vingt-deux directives de configuration listées ci-dessous. Chacune d'entre elles sera décrite indépendamment dans la suite de cette annexe.

- `cache`
- `charset`
- `check_lock`
- `check_symfony_version`
- `compressed`
- `csrf_secret`
- `default_culture`
- `default_timezone`
- `enabled_modules`
- `error_reporting`
- `escaping_strategy`

- escaping_method
- etag
- i18n
- logging_enabled
- no_script_name
- max_forwards
- standard_helpers
- strip_comments
- use_database
- web_debug
- web_debug_web_dir

La sous-section .actions

Configuration par défaut

Le code ci-dessous donne le détail de la configuration par défaut définie par Symfony pour les directives de cette sous-section.

```
default:
  .actions:
    error_404_module:       default
    error_404_action:       error404

    login_module:           default
    login_action:           login

    secure_module:          default
    secure_action:          secure

    module_disabled_module: default
    module_disabled_action: disabled
```

La sous-section `.actions` définit les actions à exécuter lorsque des pages communes doivent être rendues. Chaque définition se décompose en deux directives de configuration. La première indique le module concerné (elle est suffixée par `_module`), tandis que la seconde indique l'action à exécuter dans ce module (elle est suffixée par `_action`). Les parties qui suivent décrivent l'une après l'autre ces directives de configuration de l'application.

error_404

L'action `error_404` est exécutée à chaque fois qu'une page d'erreur 404 doit être rendue.

login

L'action `login` est exécutée lorsque l'utilisateur tente d'accéder à une page alors qu'il n'est pas authentifié. Généralement, la page affichée par cette directive est celle qui contient un formulaire d'identification.

secure

L'action `secure` est exécutée lorsque l'utilisateur tente d'accéder à une page pour laquelle il n'a pas les droits d'accès nécessaires.

module_disabled

L'action `module_disabled` est exécutée lorsque l'utilisateur demande une page d'un module désactivé pour l'application.

La sous-section .settings

La sous-section `.settings` est l'endroit où toute la configuration du framework est définie. Les parties qui suivent décrivent tous les paramètres possibles qui sont, à cette occasion, grossièrement triés par ordre d'importance. Toutes les valeurs des paramètres de cette sous-section son disponibles depuis n'importe quel endroit du code par le biais de l'objet `sfConfig`, et sont préfixées par `sf_`. Par exemple, pour connaître la valeur de la directive `charset`, il suffit de l'appeler de cette manière :

```
sfConfig::get('sf_charset');
```

escaping_strategy

La valeur par défaut de la directive `escaping_strategy` est `off`.

La directive de configuration `escaping_strategy` est un booléen qui détermine si le sous-framework d'échappement des valeurs de sortie doit être activé ou non. Quand il est activé, toutes les variables rendues disponibles dans les templates sont automatiquement échappées par l'appel au helper défini par le paramètre `escaping_method` décrit plus bas.

La directive `escaping_method` définit le helper par défaut utilisé par Symfony, mais celui-ci peut bien sûr être redéfini au cas par cas lorsqu'il s'agit de rendre une variable dans une balise JavaScript par exemple. Le sous-framework d'échappement utilise la valeur du paramètre `charset` pour l'échappement.

Il est fortement recommandé de changer la valeur par défaut à la valeur `on`.

ASTUCE **Activer automatiquement la stratégie d'échappement**

La valeur de la stratégie d'échappement des données peut être définie automatiquement au moment de la création de l'application en ajoutant l'option `--escaping_strategy` à la commande `generate:app`.

escaping_method

La valeur par défaut de la directive `escaping_method` est `ESC_SPECIALCHARS`.

La directive de configuration `escaping_method` définit la fonction par défaut à utiliser pour automatiser l'échappement des variables dans les templates (voir la directive `escaping_strategy` plus haut). Ce paramètre accepte l'une des valeurs suivantes : `ESC_SPECIALCHARS`, `ESC_RAW`, `ESC_ENTITIES`, `ESC_JS`, `ESC_JS_NO_ENTITIES` et `ESC_SPECIALCHARS` ; sinon il faut créer une fonction spécifique pour l'échappement.

La plupart du temps, la valeur par défaut suffit. Le helper `ESC_ENTITIES` peut aussi être utilisé, particulièrement lorsqu'il s'agit de travailler avec des langues anglaises ou européennes.

csrf_secret

La valeur par défaut de la directive `csrf_secret` est `false`.

La directive de configuration `csrf_secret` permet de définir un jeton unique pour l'application. Lorsqu'elle n'est pas à la valeur false, elle active la protection contre les vulnérabilités CSRF dans tous les formulaires générés à partir du framework de formulaire de Symfony. Ce paramètre est également embarqué dans le helper `link_to()` lorsqu'il a besoin de convertir un lien en formulaire afin de simuler la méthode HTTP `PUT` par exemple.

Il est fortement recommandé de changer la valeur par défaut par un jeton unique.

ASTUCE **Activer automatiquement la protection CSRF**

La valeur du jeton contre les vulnérabilités CSRF peut être définie automatiquement au moment de la création de l'application en ajoutant à la commande `--csrf_secret` l'option : `generate:app`.

charset

La valeur par défaut de la directive `charset` est `utf-8`.

La directive de configuration `charset` définit la valeur de l'encodage qui sera utilisé par défaut dans tout le framework de la réponse (en-tête `Content-Type`) jusqu'aux informations échappées dans les templates. La plupart du temps, la configuration de ce paramètre suffit.

enabled-modules

La valeur par défaut de la directive `enabled_modules` est `[default]`.

La directive de configuration `enabled_modules` est un tableau des noms des modules à activer pour l'application courante. Les modules définis dans les plug-ins ou dans le cœur de Symfony ne sont pas activés par défaut, et doivent absolument être listés dans ce paramètre pour être accessibles.

Ajouter un nouveau module est aussi simple que de l'enregistrer dans la liste. L'ordre des modules activés n'a aucune incidence.

```
enabled_modules: [default, sfGuardAuth]
```

Le module `default` défini par défaut dans le framework contient toutes les actions par défaut qui sont déclarées dans la sous-section `.actions` du fichier de configuration `settings.yml`. Il est vivement recommandé de personnaliser chacune de ces actions, puis de supprimer le module `default` de la liste de ce paramètre.

REMARQUE

Définir un fuseau horaire par défaut

Si aucun fuseau horaire n'a été défini pour cette directive de configuration, il est fortement recommandé de le déclarer dans le fichier de configuration `php.ini` du serveur étant donné que Symfony essaiera de déterminer le fuseau horaire approprié d'après la valeur retournée par la fonction PHP `date_default_timezone_get()`.

default_timezone

La valeur par défaut de la directive `default_timezone` est `none`.

La directive de configuration `default_timezone` définit le fuseau horaire utilisé par PHP. La valeur peut être n'importe quel fuseau horaire reconnu par PHP (voir http://www.php.net/manual/en/class.datetimezone.php).

ASTUCE

Paramétrage du cache des pages HTML

La configuration du système de cache des pages HTML doit être réalisée au niveau des sections `view_cache` et `view_cache_manager` du fichier de configuration `factories.yml` (voir annexe 3). La configuration par niveau de granularité la plus fine est gérée quant à elle par le biais du fichier de configuration `cache.yml`.

cache

La valeur par défaut de la directive `cache` est `off`.

La directive de configuration `cache` active ou non le cache des pages HTML.

etag

La valeur par défaut de la directive `etag` est `on`, à l'exception des environnements `dev` et `test` pour lesquels elle est désactivée.

La directive de configuration `etag` active ou désactive la génération automatique des en-têtes HTTP `ETag`. Les `ETag` générés par Symfony sont de simples calculs MD5 du contenu de la réponse.

i18n

La valeur par défaut de la directive `i18n` est `off`.

La directive de configuration `i18n` est un booléen qui active ou non le sous-framework d'internationalisation de Symfony. Pour les applications internationalisées, la valeur de ce paramètre doit être placée à `on`.

ASTUCE **Paramétrage du système d'I18N**

La configuration générale du système d'internationalisation doit être réalisée au niveau de la section `i18n` du fichier `factories.yml` (voir annexe C).

default_culture

La valeur par défaut de la directive `default_culture` est `en`.

La directive de configuration `default_culture` définit la culture par défaut que le sous-framework d'internationalisation utilisera. Ce paramètre accepte n'importe quelle valeur valide de culture.

standard_helpers

La valeur par défaut de la directive `standard_helpers` est `[Partial, Cache, Form]`.

La directive de configuration `standard_helpers` définit tous les groupes de helpers à charger automatiquement pour tous les templates de l'application. Les valeurs indiquées sont les noms de chaque groupe de helpers sans leur suffixe `Helper`.

no_script_name

La valeur par défaut de la directive `no_script_name` est `on` pour l'environnement de production de la toute première application créée dans le projet, et `off` pour tous les autres.

La directive de configuration `no_script_name` détermine si le nom du contrôleur frontal doit apparaître ou non dans l'URL avant les URLs générées par Symfony. Par défaut, il est fixé à la valeur `on` par la tâche automatique `generate:app` pour l'environnement `prod` de la toute première application créée.

De toute évidence, seulement une application et un environnement peuvent avoir ce paramètre à la valeur `on` si tous les contrôleurs frontaux se trouvent dans le même répertoire (`web/` par défaut). Pour avoir plusieurs applications avec la directive de configuration `no_script_name` à la valeur `on`, il suffit de déplacer le ou les contrôleurs frontaux sous un sous-répertoire du répertoire `web/` racine.

logging_enabled

La valeur par défaut de la directive `logging_enabled` est `on` pour tous les environnements à l'exception de l'environnement de production `prod`.

La directive de configuration `logging_enabled` active le sous-framework d'enregistrement des traces de logs. Fixer la valeur de ce paramètre à `false` permet de détourner complètement le mécanisme d'enregistrement des logs et ainsi d'améliorer légèrement les performances.

ASTUCE **Paramétrage du système d'enregistrement des traces de logs**

La configuration générale du système d'enregistrement des logs doitêtre réalisée dans le fichier de configuration `factories.yml` (voir annexe 3) afin de déterminer son niveau de granularité le plus fin.

web_debug

La valeur par défaut de la directive `web_debug` est `on` pour tous les environnements à l'exception de l'environnement de développement `dev`.

La directive de configuration `web_debug` active la génération de la barre de débogage de Symfony. La barre de débogage est ajoutée à toutes les pages web ayant la valeur HTML dans l'en-tête de type de contenu.

error_reporting

Les valeurs par défaut de la directive `error_reporting` sont les suivantes en fonction de l'environnement d'exécution du script :

- `prod` : `E_PARSE | E_COMPILE_ERROR | E_ERROR | E_CORE_ERROR | E_USER_ERROR`
- `dev` : `E_ALL | E_STRICT`
- `test` : `(E_ALL | E_STRICT) ^ E_NOTICE`
- `default` : `E_PARSE | E_COMPILE_ERROR | E_ERROR | E_CORE_ERROR | E_USER_ERROR`

REMARQUE
En savoir plus sur les niveaux d'erreurs

Le site officiel de PHP donne des informations complémentaires sur comment utiliser les opérateurs bit-à-bit à l'adresse http://www.php.net/language.operators.bitwise.

La directive de configuration `error_reporting` contrôle le niveau d'erreurs à rapporter dans le navigateur et à écrire dans les fichiers de logs. La configuration actuelle est la plus sensible, c'est pourquoi elle ne devrait pas avoir à être modifiée.

REMARQUE **Désactivation des erreurs dans le navigateur**

L'affichage des erreurs dans le navigateur est automatiquement désactivé pour les applications qui ont la directive de configuration `debug` désactivée, ce qui est le cas par défaut pour l'environnement de production.

compressed

La valeur par défaut de la directive `compressed` est `off`.

La directive de configuration `compressed` active ou non la compression automatique du contenu de la réponse. Si la valeur de ce paramètre est à `on`, alors Symfony utilisera la fonction native `ob_gzhandler()` de PHP en guise de fonction de rappel à la fonction `ob_start()`.

Il est néanmoins recommandé de la garder à la valeur `off`, et d'utiliser à la place le mécanisme de compression natif supporté par le navigateur.

use_database

La valeur par défaut de la directive `use_database` est `on`.

La directive de configuration `use_database` indique si oui ou non l'application nécessite l'usage d'une base de données.

check_lock

La valeur par défaut de la directive `check_lock` est `off`.

La directive de configuration `check_lock` active ou non le système de verrouillage de l'application déclenché par certaines tâches automatiques telles que `cache:clear`.

Si la valeur de ce paramètre de configuration est à `on`, alors toutes les requêtes en direction des applications désactivées seront automatiquement redirigées vers la page `lib/exception/data/unavailable.php`.

check_symfony_version

La valeur par défaut de la directive `check_symfony_version` est `off`.

La directive de configuration `check_symfony_version` active ou non le contrôle de la version de Symfony à chaque requête exécutée. Si ce paramètre est activé, Symfony vide le cache automatiquement lorsque le framework est mis à jour.

Il est vivement recommandé de ne pas fixer la valeur de cette directive à `on` étant donné qu'elle ajoute un léger coût supplémentaire sur les performances et qu'il est simple de nettoyer le cache lorsqu'une nouvelle version du projet est déployée. Ce paramètre est seulement utile si plusieurs projets partagent le même code source de Symfony, ce qui n'est véritablement pas recommandé.

web_debug_dir

La valeur par défaut de la directive `web_debug_dir` est `/sf/sf_web_debug`.

La directive de configuration `web_debug_dir` définit le répertoire qui contient toutes les ressources web nécessaires au bon fonctionnement de la barre de débogage de Symfony telles que les feuilles de style CSS, les fichiers JavaScript ou encore les images.

strip_comments

La valeur par défaut de la directive `strip_comments` est `on`.

La directive de configuration `strip_comments` détermine si Symfony devrait supprimer les commentaires lorsqu'il compile les classes du noyau. Ce paramètre est uniquement utilisé si le paramètre de configuration `debug` de l'application est fixé à la valeur `off`.

Si des pages blanches apparaissent uniquement en environnement de production, alors il convient de réessayer en définissant ce paramètre de configuration à la valeur `off`.

max_forwards

La valeur par défaut de la directive `max_forward` est `5`.

La directive de configuration `max_forward` détermine le nombre maximum de redirections internes (`forward()` dans les actions) autorisées avant que Symfony ne lève une exception. Ce paramètre de configuration est une sécurité pour se prémunir des boucles sans fin.

Le fichier de configuration factories.yml

C

MOTS-CLÉS :

- Configuration de Symfony
- Format YAML
- Fichier factories.yml

Tous les objets internes de Symfony nécessaires au bon traitement des requêtes comme `request`, `response` ou `user`, sont générés automatiquement par l'objet `sfContext`. Or, il arrive parfois que les classes utilisées pour construire ces objets ne suffisent pas pour subvenir aux besoins de l'application ; elles doivent alors être remplacées par des classes personnalisées.

Grâce au fichier de configuration `factories.yml` de l'application, Symfony autorise le développeur à définir lui-même son contexte d'exécution.

Introduction à la notion de « factories »

Les *factories* sont les objets du noyau de Symfony et qui sont nécessaires au framework durant tout le cycle de vie du traitement de la requête. Ces objets sont tous définis dans le fichier de configuration `factories.yml` du répertoire `apps/APPLICATION/config/` et sont toujours accessibles depuis n'importe où par l'intermédiaire de l'objet `sfContext`.

```
sfContext::getInstance()->getUser();
```

Le fichier de configuration `factories.yml` d'une application est paramétrable par environnement, et bénéficie du principe de configuration en cascade. Il peut également prendre en compte les constantes globales définies par Symfony.

Lorsque l'objet `sfContext` initialise les factories, il lit le fichier `factories.yml` pour en découvrir les noms des classes (`class`) qu'il doit instancier et la liste des paramètres (`param`) qu'il doit transmettre au constructeur respectif de ces dernières. La description générale en format YAML d'un objet factory est la suivante :

```
FACTORY_NAME:
  class: CLASS_NAME
  param: { ARRAY OF PARAMETERS }
```

REMARQUE Classe de conversion du fichier factories.yml en code PHP

La conversion du fichier de configuration `factories.yml` en PHP est réalisée par la classe PHP `sfFactoryConfigHandler`.

Être capable de personnaliser les factories signifie aussi utiliser des classes personnalisées pour les objets du cœur de Symfony à la place de ceux qui sont créés par défaut. Par conséquent, cette liberté de configuration offre au développeur la possibilité de changer les comportements par défaut de ces objets en leur passant d'autres paramètres.

Présentation du fichier factories.yml

Configuration du service request

L'initialisation de l'objet *requête* par le contexte d'exécution est assurée par quatre paramètres de configuration du fichier `factories.yml`. Chaque paramètre de configuration listé ci-dessous sera décrit indépendamment dans la suite de cette annexe.

- `path_info_array`
- `path_info_key`
- `formats`
- `relative_url_root`

Configuration du service response

L'initialisation de l'objet *réponse* par le contexte d'exécution est assurée par trois paramètres de configuration du fichier `factories.yml`. Chaque paramètre de configuration listé ci-dessous sera décrit indépendamment dans la suite de cette annexe.

- `send_http_headers`
- `charset`
- `http_protocol`

Configuration du service user

L'initialisation de l'objet *utilisateur* par le contexte d'exécution est assurée par trois paramètres de configuration du fichier `factories.yml`. Chaque paramètre de configuration listé ci-dessous sera décrit indépendamment dans la suite de cette annexe.

- `timeout`
- `use_flash`
- `default_culture`

Configuration du service storage

L'initialisation de l'objet de *session* par le contexte d'exécution est assurée par treize paramètres de configuration du fichier `factories.yml`. Chaque paramètre de configuration listé ci-dessous sera décrit indépendamment dans la suite de cette annexe.

- `auto_start`
- `session_name`
- `session_cache_limiter`
- `session_cookie_lifetime`
- `session_cookie_path`
- `session_cookie_domain`
- `session_cookie_secure`
- `session_cookie_httponly`
- `database`
- `db_table`
- `db_id_col`
- `db_data_col`
- `db_time_col`

Configuration du service i18n

L'initialisation de l'objet d'*internationalisation* par le contexte d'exécution est assurée par cinq paramètres de configuration du fichier `factories.yml`. Chaque paramètre de configuration listé ci-dessous sera décrit indépendamment dans la suite de cette annexe.

- `source`
- `debug`
- `untranslated_prefix`
- `untranslated_suffix`
- `cache`

Configuration du service routing

L'initialisation de l'objet de *routage* par le contexte d'exécution est assurée par sept paramètres de configuration du fichier `factories.yml`. Chaque paramètre de configuration listé ci-dessous sera décrit indépendamment dans la suite de cette annexe.

- `variable_prefixes`
- `segment_separators`
- `generate_shortest_url`
- `extra_parameters_as_query_string`
- `cache`
- `suffix`
- `load_configuration`

Configuration du service logger

L'initialisation de l'objet d'*enregistrement des traces de log* par le contexte d'exécution est assurée par deux paramètres de configuration du fichier `factories.yml`. Chaque paramètre de configuration listé ci-dessous sera décrit indépendamment dans la suite de cette annexe.

- `level`
- `loggers`

Le service request

Configuration par défaut

Le service `request` est accessible grâce à l'accesseur `getRequest()` de l'objet `sfContext`.

```
sfContext::getInstance()->getRequest()
```

La configuration par défaut du service `request` est la suivante :

```
request:
  class: sfWebRequest
  param:
    logging:           %SF_LOGGING_ENABLED%
    path_info_array:   SERVER
    path_info_key:     PATH_INFO
    relative_url_root: ~
    formats:
      txt: text/plain
      js:   [application/javascript, application/x-javascript,
text/javascript]
      css: text/css
      json: [application/json, application/x-json]
      xml: [text/xml, application/xml, application/x-xml]
      rdf: application/rdf+xml
      atom: application/atom+xml
```

path_info_array

L'option `path_info_array` définit le tableau PHP superglobal qui doit être utilisé pour retrouver les informations. Sur certaines configurations de serveur web, il arrive parfois que l'on veuille changer la valeur par défaut `SERVER` en `ENV`.

path_info_key

L'option `path_info_key` définit la clé dans le tableau PHP superglobal avec laquelle il est possible de retrouver l'information `PATH_INFO`. Les utilisateurs de serveurs web `IIS` configurés avec un moteur de réécriture d'URLs comme `IIFR` ou `ISAPI` devront certainement changer la valeur de cette directive de configuration par `HTTP_X_REWRITE_URL`.

ASTUCE **Utiliser la méthode setFormat() de l'objet request**

Au lieu de redéfinir cette directive de configuration, il est possible d'avoir recours à la méthode `setFormat()` de la classe de l'objet de la requête.

formats

L'option `formats` définit un tableau associatif d'extensions et leurs valeurs respectives d'en-tête de types de contenu. Ce paramètre de configuration est utilisé par le framework pour gérer automatiquement l'en-tête `Content-Type` de la réponse, en partant de l'extension de l'URL de la requête.

relative_root_url

L'option `relative_root_url` définit la valeur de la partie de l'URL qui se trouve avant le nom du contrôleur frontal. La plupart du temps, cette valeur est déduite automatiquement par le framework, ce qui implique qu'il n'est pas nécessaire de la modifier.

Le service response

Configuration par défaut

Le service `response` est accessible grâce à l'accesseur `getResponse()` de l'objet `sfContext`.

```
sfContext::getInstance()->getResponse()
```

La configuration par défaut du service `response` est la suivante :

```
response:
  class: sfWebResponse
  param:
    logging:           %SF_LOGGING_ENABLED%
    charset:           %SF_CHARSET%
    send_http_headers: true
```

Le code ci-dessous donne la configuration par défaut du service `response` en environnement de test.

```
response:
     class: sfWebResponse
     param:
       send_http_headers: false
```

send_http_headers

L'option `send_http_headers` définit si la réponse a besoin d'envoyer les en-têtes HTTP avec son contenu. Ce paramètre de configuration est essentiellement utilisé en environnement de test étant donné que l'envoi

des en-têtes est réalisé par la fonction native PHP `header()` qui provoque des avertissements lorsque des en-têtes sont envoyés après les premières sorties au navigateur.

charset

L'option `charset` définit l'encodage à utiliser pour la réponse. Par défaut, la valeur de cette directive de configuration est la même que celle définie à l'entée `charset` dans le fichier de configuration `settings.yml` de l'application.

http_protocol

L'option `http_protocol` détermine la version du protocole HTTP à utiliser pour transmettre la réponse. Par défaut, Symfony ira chercher cette valeur dans la variable superglobale `$_SERVER['SERVER_PROTOCOL']` si elle existe, ou utilisera `HTTP/1.0` par défaut.

Le service user

Configuration par défaut

Le service `user` est accessible grâce à l'accesseur `getUser()` de l'objet `sfContext`.

```
sfContext::getInstance()->getUser()
```

La configuration par défaut du service `user` est la suivante :

```
user:
  class: myUser
  param:
    timeout:         1800
    logging:         %SF_LOGGING_ENABLED%
    use_flash:       true
    default_culture: %SF_DEFAULT_CULTURE%
```

Par défaut, la classe `myUser` hérite des propriétés et des méthodes de la classe `sfBasicSecurityUser`, qui peut être configurée dans le fichier de configuration `security.yml`.

timeout

L'option `timeout` définit le temps maximum pendant lequel l'utilisateur dispose de son authentification et de ses droits d'accès. Cette valeur n'est pas relative à celle définie dans la session. Ce paramètre de configuration est seulement utilisé par les classes de gestion de l'utilisateur qui héritent de la classe de base `sfBasicSecurityUser`, ce qui est le cas pour la classe autogénérée `myUser` de chaque application.

use_flash

L'option `use_flash` active ou désactive l'utilisation des messages éphémères stockés dans la session de l'utilisateur entre deux requêtes HTTP.

default_culture

L'option `default_culture` détermine la culture par défaut à assigner à l'utilisateur lorsque celui-ci arrive pour la première fois sur le site Internet. Par défaut, cette valeur est récupérée de la directive de configuration `default_culture` du fichier de configuration `settings.yml`, ce qui est généralement le comportement de la plupart des applications.

REMARQUE **Éviter les effets de bord avec l'objet User**

Pour éviter des effets de bord étranges, la classe de gestion de l'utilisateur force automatiquement la durée de vie maximale du ramasse-miettes (*garbage collector*) de la session (`session.gc_maxlifetime`) à une valeur strictement supérieure à celle du paramètre `timeout`.

Le service storage

Configuration par défaut

Le service `storage` est utilisée par le service `user` pour assurer la persistance des données de session de l'utilisateur entre chaque requête HTTP, et est accessible grâce à l'accesseur `getStorage()` de l'objet `sfContext`.

```
sfContext::getInstance()->getStorage()
```

La configuration par défaut du service `storage` est la suivante :

```
storage:
  class: sfSessionStorage
  param:
    session_name: symfony
```

Une configuration par défaut spéciale pour l'environnement de test est également mise en place. Elle est décrite dans le code ci-dessous.

```
storage:
  class: sfSessionTestStorage
  param:
    session_path: %SF_TEST_CACHE_DIR%/sessions
```

auto_start

L'option `auto_start` active ou désactive le démarrage automatique de la session PHP par le biais de la fonction `session_start()`.

session_name

L'option `session_name` définit le nom du cookie de session utilisé par Symfony pour sauvegarder l'identifiant de session de l'utilisateur. Par défaut, le nom est Symfony, ce qui signifie que toutes les applications du projet partagent le même cookie, et par conséquent les authentifications et droits d'accès correspondants.

Paramètres de la fonction session_set_cookie_params()

Le service `storage` fait appel à la fonction PHP `session_set_cookie_params()` avec les valeurs des options suivantes :

- `session_cookie_lifetime` : durée de vie totale du cookie de session définie en secondes ;
- `session_cookie_path` : domaine de validité du cookie sur le serveur. La valeur `/` indique que le cookie est valable sur tout le domaine ;
- `session_cookie_domain` : domaine du cookie, par exemple, www.php.net. Pour rendre les cookies visibles par tous les sous-domaines, la valeur doit être préfixée d'un point comme `.php.net` ;
- `session_cookie_secure` : si la valeur est `true`, le cookie sera envoyé sur des connexions sécurisées ;
- `session_cookie_httponly` : si la valeur est `true`, PHP attendra d'envoyer le drapeau `httponly` lorsqu'il paramétrera le cookie de session.

session_cache_limiter

Si l'option `session_cache_limiter` est définie, la fonction native de PHP `session_cache_limiter()` sera appelée et la valeur de cette directive de configuration sera passée en argument de la fonction.

REMARQUE Documentation de la fonction session_set_cookie_params()

La description de chaque option de configuration de la fonction `session_set_cookie_params()` provient directement de la documentation officielle en ligne disponible à l'adresse http://fr.php.net/session-set-cookie-params.

Options de stockage des sessions en bases de données

Lorsque l'on utilise un système de stockage des sessions qui hérite de la classe `sfDatabaseSessionStorage`, certaines options spécifiques sont disponibles :

- `database` : le nom de la base de données (obligatoire) ;
- `db_table` : le nom de la table qui stocke les données de session (obligatoire) ;
- `db_id_col` : le nom de la colonne qui contient la clé primaire (`sess_id` par défaut) ;
- `db_data-col` : le nom de la colonne qui contient les données de session sérialisées (`sess_data` par défaut) ;
- `db_time_col` : le nom de la colonne dans laquelle est stockée le date de la session (`sess_time` par défaut).

Le service view_cache_manager

Le service `view_cache_manager` est accessible grâce à l'accesseur `getViewCacheManager()` de l'objet `sfContext`.

```
sfContext::getInstance()->getViewCacheManager()
```

La configuration par défaut du service `view_cache_manager` est la suivante :

```
view_cache_manager:
  class: sfViewCacheManager
```

Cette factory est uniquement initialisée si le paramètre de configuration `cache` est défini à la valeur `on`. La configuration du gestionnaire de cache des pages HTML est réalisée via le service `view_cache` qui définit l'objet cache sous-jacent à utiliser pour le cache des vues.

Le service view_cache

Le service `view_cache` est accessible depuis l'accesseur `getViewCacheManager()` de l'objet `sfContext`. La configuration par défaut du service `view_cache` est la suivante :

```
view_cache:
  class: sfFileCache
  param:
    automatic_cleaning_factor: 0
    cache_dir:                 %SF_TEMPLATE_CACHE_DIR%
    lifetime:                  86400
    prefix:                    %SF_APP_DIR%/template
```

Cette factory est uniquement initialisée si le paramètre de configuration `cache` est défini à la valeur `on`. Le service `view_cache` déclare une classe qui doit absolument hériter de `sfCache`.

Le service i18n

Configuration par défaut

Le service `i18n` est accessible grâce à l'accesseur `getI18N()` de l'objet `sfContext`.

```
sfContext::getInstance()->getI18N()
```

La configuration par défaut du service `i18n` est la suivante :

```
i18n:
  class: sfI18N
  param:
    source:               XLIFF
    debug:                off
    untranslated_prefix: "[T]"
    untranslated_suffix: "[/T]"
    cache:
      class: sfFileCache
      param:
        automatic_cleaning_factor: 0
        cache_dir:                 %SF_I18N_CACHE_DIR%
        lifetime:                  31556926
        prefix:                    %SF_APP_DIR%/i18n
```

Cette factory est uniquement initialisée si le paramètre de configuration `i18n` est défini à la valeur `on`.

source

L'option `source` définit le type de container des traductions au choix parmi `XLIFF`, `MySQL`, `SQLite` ou `gettext`.

debug

L'option `debug` active ou désactive le débogage. Si cette directive de configuration est activée, alors les messages non traduits seront décorés avec un préfixe et un suffixe (voir ci-dessous).

untranslated_prefix

L'option `untranslated_prefix` définit un préfixe à utiliser pour décorer les messages non traduits.

untranslated_sufix

L'option `untranslated_sufix` définit un suffixe à utiliser pour décorer les messages non traduits.

cache

L'option `cache` définit une factory de cache anonyme à utiliser pour mettre en cache les données internationalisées.

Le service routing

Configuration par défaut

Le service `routing` est accessible grâce à l'accesseur `getRouting()` de l'objet `sfContext`.

```
sfContext::getInstance()->getRouting()
```

La configuration par défaut du service `routing` est la suivante :

```
routing:
  class: sfPatternRouting
  param:
    load_configuration:              true
    suffix:                          ''
    default_module:                  default
    default_action:                  index
    debug:                           %SF_DEBUG%
    logging:                         %SF_LOGGING_ENABLED%
    generate_shortest_url:           true
    extra_parameters_as_query_string: true
```

```
cache:
  class: sfFileCache
  param:
    automatic_cleaning_factor: 0
    cache_dir:                 %SF_CONFIG_CACHE_DIR%/routing
    lifetime:                   31556926
    prefix:                     %SF_APP_DIR%/routing
```

variables_prefix

La valeur par défaut de l'option `variables_prefix` est `:`.

L'option `variables_prefix` définit la liste des caractères qui démarre le nom d'une variable dans un motif d'URL d'une route.

segment_separators

La valeur par défaut de l'option `segment_separators` est `/` et `.`.

L'option `segment_separators` définit la liste des séparateurs de segments des routes. La plupart du temps, cette option n'est pas redéfinie pour tout le framework de routage mais uniquement pour une seule route particulière.

generate_shortest_url

La valeur par défaut de l'option `generate_shortest_url` est `true` pour les nouveaux projets et `false` pour les projets mis à jour.

Si la valeur de l'option `generate_shortest_url` est fixée à la valeur `true`, le sous-framework de routage génèrera la route la plus courte possible. Cette directive de configuration doit en revanche rester à la valeur `false` afin de conserver une compatibilité rétrograde avec les versions 1.0 et 1.1 de Symfony.

extra_parameters_as_query_string

La valeur par défaut de l'option `extra_parameters_as_query_string` est `true` pour les nouveaux projets et `false` pour les projets mis à jour.

Lorsque des paramètres ne sont pas utilisés dans la génération de la route, la directive de configuration `extra_parameters_as_query_string` autorise ces arguments supplémentaires à être convertis sous la forme d'une chaîne de requête. Placer la valeur à `false` de ce paramètre dans un projet Symfony 1.2 permet de revenir au comportement des version 1.0 et 1.1 de Symfony. En revanche, pour ces dernières, les paramètres supplémentaires seront simplement ignorés par le système de routage.

cache

L'option `cache` définit une factory de cache anonyme à utiliser pour mettre en cache la configuration et les données du système de routage.

suffix

La valeur par défaut de l'option `suffix` est `none`.

L'option `suffix` définit le suffixe à ajouter au bout de chaque route, mais cette fonctionnalité est désormais dépréciée depuis l'intégration du nouveau framework de routage de Symfony 1.2. Par conséquent, cette directive de configuration devient inutile bien qu'elle soit toujours présente.

load_configuration

La valeur par défaut de l'option `load_configuration` est `true`.

L'option `load_configuration` précise si le fichier de configuration `routing.yml` doit être automatiquement chargé et analysé. La valeur `false` doit être fixée pour utiliser le système de routage à l'extérieur d'un projet Symfony.

Le service logger

Configuration par défaut

Le service `logger` est accessible grâce à l'accesseur `getLogger()` de l'objet `sfContext`.

```
sfContext::getInstance()->getLogger()
```

La configuration par défaut du service `logger` est la suivante :

```
logger:
  class: sfAggregateLogger
  param:
    level: debug
    loggers:
      sf_web_debug:
        class: sfWebDebugLogger
        param:
          level: debug
          condition:       %SF_WEB_DEBUG%
          xdebug_logging: true
          web_debug_class: sfWebDebug
```

```
    sf_file_debug:
      class: sfFileLogger
      param:
        level: debug
        file: %SF_LOG_DIR%/%SF_APP%_%SF_ENVIRONMENT%.log
```

Le service `logger` bénéficie également d'une configuration par défaut pour l'environnement de production `prod`.

```
logger:
  class:   sfNoLogger
  param:
    level:   err
    loggers: ~
```

Cette factory est toujours initialisée, mais la procédure d'enregistrement des traces de logs se produit uniquement si le paramètre `logging_enabled` est défini à la valeur `on`.

level

L'option `level` définit le niveau de gravité des informations de logs à enregistrer et prend une valeur parmi `EMERG`, `ALERT`, `CRIT`, `ERR`, `WARNING`, `NOTICE`, `INFO` ou `DEBUG`.

loggers

L'option `loggers` définit une liste des objets à utiliser pour enregistrer les traces de logs. Cette liste est un tableau de factories anonymes d'objets de log qui figurent parmi les classes suivantes : `sfConsoleLogger`, `sfFileLogger`, `sfNoLogger`, `sfStreamLogger` et `sfVarLogger`.

Le service controller

Configuration par défaut

Le service `controller` est accessible grâce à l'accesseur `getController()` de l'objet `sfContext`.

```
sfContext::getInstance()->getController()
```

La configuration par défaut du service `controller` est la suivante :

```
controller:
  class: sfFrontWebController
```

Les services de cache anonymes

Plusieurs factories (`view_cache`, `i18n` et `routing`) tirent partie des avantages de l'objet de cache s'il est défini dans leur configuration respective. La configuration de l'objet de cache est similaire pour toutes les factories. La clé `cache` définit une factory de cache anonyme. Comme n'importe quelle autre factory, elle accepte deux entrées : `class` et `param`. La section `param` peut prendre n'importe quelle option disponible pour la classe de cache.

L'option `prefix` est la plus importante de toutes étant donné qu'elle permet de partager ou de séparer un cache entre différents environnements/applications/projets.

Les classes natives de cache dans Symfony sont `sfAPCCache`, `sfEAcceleratorCache`, `sfFileCache`, `sfMemcacheCache`, `sfNoCache`, `sfSQLiteCache` et `sfXCacheCache`.