Découverte du moteur de Templates Twig

1 À PROPOS

Twig est un moteur de Templates qui a été créé par SensioLabs, les créateurs de Symfony.

On le retrouve nativement dans les Frameworks Symfony et Drupal8, mais il peut être installé sur la majorité des Frameworks ainsi que dans un environnement PHP.

2 AVANTAGES DE TWIG

- Twig permet de séparer la présentation des données du traitement.
  Bien souvent, afficher une donnée brute ne suffit pas. On veut lui appliquer un traitement logique avant de l'afficher (une conversion, un calcul…)
  Twig permet de définir en dehors de la page web des filtres que l'on pourra appliquer à la donnée.
- Twig permet la personnalisation de page web.
  Un block menu, un block recherche, un block contenu… le tout défini dans un Template lui-même héritable

- Twig permet de rendre les pages web plus lisibles, plus claires.
  On a souvent besoin de boucles, de conditions… à la construction d'une page web et pour cela, on utilise PHP. Votre page devient vite illisible, peu maintenable, sujette à de nombreuses erreurs.
  Twig par son langage est moins invasif que PHP et se substitue à celui-ci.

TWIG	PHP

`{% for value in items if value.active %}`
`  <img src="{{ value.url }}"/>`
`{% endfor %}`
	`<?php foreach ($items as $value):`
	`  if ($value.active) {?>`
	`  <img src="<?php echo $value.url;?>"/>`
	`<?php } endforeach;?>`

- Twig est rapide.
  La conversion Twig/PHP ne se fait qu'une seule fois au premier appel, ensuite il est mis en cache pour les autres appels (et donc finalement, c'est aussi rapide qu'en PHP).

- Twig apporte de nouvelles fonctionnalités :
  les macros, les filtres, les blocks, l'héritage de Templates…
- Twig apporte plus de sécurité avec l'échappement des variables qui est activé par défaut.

Mais aussi :

- Twig est facile à apprendre ;
- les messages d'erreur Twig sont clairs et précis ;
- Twig est supporté par la plupart des éditeurs de code.

3 LE LANGAGE TWIG

Syntaxe du langage Twig.
{% %} : pour faire une action : un set de variable, une condition if, une boucle for ;
{{ }} : pour afficher quelque chose.

3.1 AFFICHAGE DES DONNEES

L'affichage des données et l'échappement :

TWIG	Résultats
`<b>{{ 'Coucou' }}</b>`	**Coucou**
`{% set data = '<h1>coucou</h1>' %}` `{{ data }}`	coucou

Pour la sécurité, **l'échappement est activé par défaut**. Les balises ne sont pas interprétées. L'échappement convertit la donnée en code HTML avant l'affichage.

3.1.1 Forcer l'interprétation des balises avec le filtre raw :

```
{% set data = '<h1>coucou</h1>' %}
    {{ data|raw }}
```

coucou

3.1.2 Si l'échappement par défaut a été désactivé. Forcer l'échappement avec le filtre e :

```
{% set data = '<h1>coucou</h1>' %}
    {{ data|e }}
```

<h1>coucou</h1>

3.1.3 L'affichage des données de différents types: objet, tableau…

TWIG	PHP

`{{ produit.nom }}`
`{{ produit.categorie.nom }}`
	`<?php`
	`  echo $produit->getNom();`
	`  echo $produit->getCategorie()->getNom();`
	`?>`

`{{ produit["nom"] }}  ou  {{ produit.nom }}`	`<?php`
	`  echo $produit["nom"];`
	`?>`

`{% set animal = 'oiseaux' %}`
`{{ "il y a des #{animal} sur le toit" }}`
 	
`{{ 14752 / 22 = #{14752 / 22} }}`
	`<?php`
	`  $animal = 'oiseaux';`
	`  echo sprintf('Il y a des %s sur le toit',  $animal);`
	
	`  echo printf('14752 / 22 = %d',  (14752 / 22));`
	`?>`

3.1.4 L'affichage de données avec suppression d'espace entre deux balises à gauche ou à droite ou les deux :

3.1.4.1 Supprimer les espaces entre les balises HTML

TWIG	Conversion en : (avant affichage)
{% spaceless %} foo bar {% endspaceless %}	foo bar

3.1.4.2 Supprimer les espaces à gauche puis des deux côtés avec le caractère '-' :

TWIG	Conversion en : (avant affichage)
{{- 'bonjour' }} {{- 'aurevoir' -}}	- bonjour - aurevoir

3.2 CONCATENATION

TWING	PHP
{{ "Description du produit:" ~ produit.description }}	<?php echo "Description du produit:" . $produit->getDescription(); ?>
{% set texte = bienvenue à ~ nom|lower %} {{ texte }}	<?php $texte = $greeting . strtolower($name); echo $texte; ?>
{{ (bienvenue à ~ nom)|lower }}	<?php echo strtolower($greeting . $name); ?>

3.3 LES VARIABLES

TWIG
{% set pi = 3.14 %} {% set foo = 'foo' %} - tableau simple avec une série de valeurs: {% set tableau=[1, 2, 3] %} - tableau avec des clés: {% set tableau={key1:value1, key2:value2} %} - tableau avec valeur et clé: {% set foo = [3, {"mot": "soleil"}] %} - afficher le contenu de 'mot' donc 'soleil': {{ foo[1]['mot'] }} - déclarer 2 variables en même temps // foo='foo' // bar='bar': {% set foo, bar = 'foo', 'bar' %} - foo contient le texte entre les 2 balises:

```
    {% set foo %}
        <div id="pagination">
            …
        </div>
    {% endset %}
```

[…] représente un tableau avec une série de valeurs.

{ … } représente un tableau avec des couples clé/valeur.

3.4 CONDITION IF : {% IF ... %} ... {% ENDIF %}

and, or, not, is defined, starts with, ends with, matches, in, empty

```
{% if produits is empty %}
       il n'y a plus de produit
    {% endif %}

    {% if ((a==1 and b>0) or not c==0) and d is defined %}
       {% set resultat = (d + a * b) / c %}
       {{ resultat }}
    {% endif %}

    {% if 'Fabien' starts with 'F' %}
       commence par F
    {% endif %}

    {% if 'Fabien' ends with 'n' %}
       Finis par n
    {% endif %}

    - regular expressions:
    {% if phone matches '/^[\\d\\.]+$/' %}
       format telephone ok
    {% endif %}

    {% if 5 not in [1, 2, 3] %}
       5 non présent
    {% endif %}
```

mais aussi :

is null: si est null ;
is constant: comparer si est une constante ;
is divisible by(x): si est divisible par x ;
is even: si est pair ;
is odd: si est impair ;
is iterable: si est du type itérable (comme une liste) ;
is same as: comparer 2 variables (en php correspond ===).

Sous une autre forme :

```
{{ article is null? 'yes': 'no' }}    // affiche yes ou no

    {{ var is odd }}        // Affiche true ou false
                    // pareil que: {% if var is odd %}true{% else %}false{% endif %}

    {{ var is null }}        // Affiche true ou false
```

http://twig.sensiolabs.org/doc/tests/index.html

3.5 LA BOUCLE FOR: {% FOR ... %} {% ENDFOR %}

```
- affiche 0123456789
    {% for i in 0..9 %}  // pareil que {% for i in range(0, 9) %}
       {{ i }}
    {% endfor %}

    {% for produit in produits %}
       {{ produit.nom }}
    {% endfor %}

    - avec une condition:
    {% for produit in produits if produit.etat==1 %}
       {{ produit.nom }}
    {% endfor %}

    - boucle for avec condition vide:
    {% for article in articles %}
        {{ article.nom }}
    {% else %}
        pas d'article trouvé
    {% endfor %}

    - clés et valeurs:
    {% for key, value in table %}
       {{ key }} {{ value }}
    {% endfor %}

    - les 10 premiers users:
    {% for user in users|slice(0, 9) %}
       <li>{{ user.username }}</li>
    {% endfor %}

    - bascule:
    {% for i in 0..10 %}
        <div class="{{ cycle(['fond-noir', 'fond-blanc'], i) }}">       //  fond-noir /
fond-blanc  /  …

        </div>
    {% endfor %}
```

La variable **loop** de la boucle for :

```
{{ loop.index }}        // Numéro de l'itération courante en commençant par 1
        {{ loop.index0 }}         // Numéro de l'itération courante en commençant par 0
        {{ loop.revindex }}        // Nombre itérations restantes avant la fin en
commençant par 1
        {{ loop.revindex0 }}        // Nombre itérations restantes avant la fin en
commençant par 0
        {{ loop.first }}        // La première itération?  True ou false
        {{ loop.last }}            // La dernière itération? True ou false
        {{ loop.length }}        // Nombre total d'itérations
```

Un exemple avec **loop :**

```
{% for produit in produits %}
          <span class="{% if loop.index is even %}fond-noir{% else %}fond-blanc{% endif
%}">
            {{ produit }}
          </span>
        {% endfor %}
```

3.6 COMMENTAIRE: {# ... #}

```
{# mon commentaire #}
```

3.7 LES FILTRES

On peut appliquer des filtres sur une variable à afficher, sur une variable d'une condition IF ou d'une boucle FOR...

Le filtre **length**

TWIG	PHP

{% for i in 0..produits|length %}
 	
{% endfor %}
	<?php
	for ($i=0; $i<count($produits); $i++) {
	
	}
	?>

Appliquer plusieurs filtres : **trim** et **upper**

TWIG	PHP
{{ "nuits de chine"|trim|upper }}	<?php echo strtoupper(trim("nuit de chine")); ?>

Une autre façon d'appliquer un filtre **upper**

TWIG	
{% filter upper %} La nuit était fort noire et la forêt très-sombre. Hermann à mes côtés me paraissait une ombre. Nos chevaux galopaient. A la garde de Dieu ! Les nuages du ciel ressemblaient à des marbres. Les étoiles volaient dans les branches des arbres Comme un essaim d'oiseaux de feu. {% endfilter %}	Avec un long texte, sous cette forme c'est plus pratique et plus clair.

À savoir:

La liste des filtres : http://twig.sensiolabs.org/doc/filters/index.html

```
abs, batch, capitalize, convert_encoding, date, date_modify, default, escape, first,
format, join, json_encode, keys, last, length, lower, merge,
nl2br, number_format, raw, replace, reverse, round, slice, sort, split, striptags,
title, trim, upper, url_encode
```

Plus loin, nous verrons comment créer nos propres filtres.

3.8 LES FONCTIONS

Une fonction effectue un traitement (contrairement à un filtre qui est destiné à faire une conversion) :

- La fonction **dump :** affiche tout le détail d'un objet ou d'un tableau

```
{{ dump(produit) }}
```

- - La fonction **max :** retourne le max d'une série de valeurs

```
{% set tab = [1, 5, 2, 3] %}
{{ max(tab) }}                // affiche  5
```

Liste des fonctions :

`attribute, block, constant, cycle, date, dump, include, max, min, parent, random, range, source, template_from_string`

https://twig.symfony.com/doc/

3.9 LES MACROS

Les macros sont comme des fonctions, mais composées de fragments HTML réutilisables.
Dans un Template, on regroupe toutes les macros que l'on pourra importer dans un autre Template afin d'être utilisées.

Prenons pour exemple un Template forms.html.twig qui va contenir plusieurs macros. Une macro pour la balise input et une qui affiche une liste d'article.

forms.html.twig

```
{% macro input(name, value, type, size) %}
    <input type="{{ type|default('text') }}" name="{{ name }}" value="{{ value|e }}"
size="{{ size|default(20) }}" />
{% endmacro %}

{% macro articleListe(articles, className) %}
    <ul {% if className %}class="{{ className }}"{% endif %}>
    {% for article  in articles %}
        <li class="article"><a href="{{ article.url }}">{{ article.nom }}</a></li>
    {% endfor %}
    </ul>
{% endmacro %}
```

Son utilisation : soit on importe toutes les macros, soit on choisit celle que l'on veut importer :

mapage.html.twig

Avec import de toutes les macros se trouvant dans forms.html.twig

```
{% import "forms.html.twig" as forms %}

<p>{{ forms.input('livre', HUGO') }}</p>
```

Avec import d'une seule macro parmi les macros se trouvant dans forms.html.twig

```
{% from 'forms.html.twig' import input as
input_field %}

<p>{{ input_field('livre', HUGO') }}</p>
```

Une macro dans la page même de son utilisation, pour cela on utilise _self. Mais on perd sa réutilisabilité.

mapage.html.twig : la macro se trouve dans la page même. (sans import)

```
{% macro input(name, value, type, size) %}
    <input type="{{ type|default('text') }}" name="{{ name }}" value="{{ value|e }}"
size="{{ size|default(20) }}" />
{% endmacro %}

<p>{{ _self.input('livre', 'saint-exupery', 'text', 64) }}</p>
```

3.10 NAMED ARGUMENTS

Donner des noms aux arguments pour une meilleure compréhension et de pouvoir cibler un argument.

Quelques exemples :

- au lieu de ... range(1, 10, 2) :

```
{% for i in range(low=1, high=10, step=2) %}
    {{ i }},
{% endfor %}
```

- l'argument ciblé est timezone parmi les autres arguments (format, date...)

```
{{ "now" | date(timezone="Europe/Paris") }}       //

{{ data|convert_encoding(from='iso-2022-jp', to='UTF-8') }}
```

3.11 LES TAGS OU ACTIONS

{% … %} Les commandes de base Twig pour faire des actions comme : for, if, set…

Liste de toutes les commandes :

`autoescape, block, do, embed, extends, filter, flush, for, from, if, import, include, macro, sandbox, set, spaceless, use, verbatim`

https://twig.symfony.com/doc/

3.12 PERSONNALISER LES TEMPLATES : HERITAGE, BLOCK

Pour personnaliser les Templates, Twig propose un système de blocks et d'héritage de Templates.

Prenons cet exemple :
toutes les pages "client" et les pages "produit" ont le même header et footer ;
les pages client ont une recherche par client ;
les pages produit ont une recherche par produit ;
toutes les pages "client" et "produit" ont leur propre contenu respectif.

base.html.twig

```
<html>
<head>...</head>

<body>
   <div class="head">     {% block header %} header {% endblock %}  </div>
   <div class="sidebar">  {% block recherche %}{% endblock %}       </div>
   <div class="content">  {% block contenu %}{% endblock %}         </div>
   <div class="footer">   {% block footer %} footer {% endblock %}  </div>
</body>
</html>
```

layout-client.html.twig

```
{% extends "::base.html.twig" %}

{% block recherche %}
   recherche client
{% endblock %}
```

layout-produit.html.twig

```
{% extends "::base.html.twig" %}

{% block recherche %}
   recherche produit
{% endblock %}
```

Résultat

une page web 'client'	une page web 'produit'
header recherche client contenu client footer	header recherche produit contenu produit footer

Les avantages :

- Finalement, pour créer une page web "client" ou "produit", on hérite de son layout et on ajoute son propre contenu et… c'est tout ;
- Si vous voulez modifier le header (de toutes les pages), il suffit de ne le modifier qu'à un seul endroit c'est-à-dire dans la base.

D'autres possibilités avec les blocks :

Identique	{% extends "AcmeDemoBundle::layout-client.html.twig" %} {% block contenu %} contenu client {% endblock %} *Le fait de ne pas mentionner un block parent a pour résultat que le contenu est tout de même inclus automatiquement par héritage.*	header recherche client contenu client footer
Identique	{% extends "AcmeDemoBundle::layout-client.html.twig" %} {% block header %} {{ parent() }} {% endblock %} {% block recherche %} {{ parent() }} {% endblock%} {% block contenu %} contenu client {% endblock %} {% block footer %} {{ parent() }} {% endblock %} *{{ parent() }}: appelle du contenu parent*	header recherche client contenu client footer
Supprimer un bloc parent Suppression du block recherche	{% extends "AcmeDemoBundle::layout-client.html.twig" %} {% block recherche %}{% endblock%} {% block contenu %} contenu client {% endblock %} *Le fait de mentionner un block et qu'il soit vide a pour résultat que le contenu parent n'est pas appelé*	header contenu client footer
Ajouter du contenu dans un block sur le block recherche	{% extends "AcmeDemoBundle::layout-client.html.twig" %} {% block recherche %} {{parent() }} un texte en dessous {% endblock%} {% block contenu %} contenu client {% endblock %}	header recherche client un texte en dessous contenu client footer

Pour information, dans cette partie "personnalisation, héritage…" seule la façon d'écrire l'extends est liée à Symfony: {% extends "AcmeDemoBundle::layout-client.html.twig" %}

3.13 Les liens

3.14 L'inclusion de feuilles de style et de JavaScripts avec Twig

3.15 Les formats de Template

3.16 Math

Un peu de math pour la route

```
+ - * / : les bases

% : modulo donne le reste de la division
{{ 11 % 7 }} retourne 4.

// : arrondi inférieur ou supérieur sans les décimales
{{ 20 // 7 }} retourne 2
{{ 20 // 6 }} retourne 3

** : puissance
{{ 2 ** 3 }} retourne 8
```

3.17 Coding Standards

- Toujours un et un seul espace entre les balises, les données… (sauf pour les filtres) :

```
{{ foo }}
{# comment #}
{% if foo %}{% endif %}
{{ 1 + 2 }}
{{ foo ~ bar }}
{{ true ? true : false }}
```

- Un seul espace après (mais pas avant) un séparateur :

```
{{ [1, 2, 3] }}
{{ {'foo': 'bar'} }}
```

- Inversement un seul espace avant (mais pas après) une parenthèse :
- Pas d'espace pour les filtres :
- Indenter les blocks :

```
{% block foo %}
   {% if true %}
       true
   {% endif %}
{% endblock %}
```