Symfony 5 : TWIG

Nizar Rouatbi

Technologue en informatique à Iset Sousse

nizar.rouatbi@gmail.com

Plan

Symfony

Twig

- moteur de templates pour **PHP**
- apparu en 2009
- syntaxe inspirée par Jinja (moteur de template du framework Django de Python)
- issu et utilisé par **Symfony**
- supporté par plusieurs IDE : NetBeans, PhpStorm, Eclipse, Visual Studio Code...
- supporté par plusieurs éditeurs de texte : Sublime text, notepad++, vim...
- **Symfony 5** utilise la version 3 de **Twig**

Symfony

Twig, Pourquoi ?

- permet de séparer le code **PHP** du code html (lisibilité, maintenabilité)
- offre la possibilité de modifier un fichier sans influencer le deuxième
- facilite le travail d'équipe

Inconvénients

- ralentir le chargement de page
- Un langage (de template) de plus à étudier
- La gestion d'erreurs est plus compliquée

Symfony

Autres moteurs de template

- Smarty
- Liquid
- Mustache
- Plates
- Talus'TPL
- ...

Symfony

Trois types de balises

- {% ... %} : pour exécuter une action
- {# ... #} : pour définir un commentaire
- {{ ... }} : pour afficher

Avant de commencer, considérons le contenu suivant pour le contrôleur HomeController

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\
   AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class HomeController extends AbstractController
{
    /**
     * @Route("/home",name="home_route")
     */
    public function index()
    {
        return $this->render('home/index.html.twig', [
            'controller_name' => 'HomeController',
        ]);
    }
}
```

Symfony

Et le contenu suivant pour `index.html.twig`

```
{% extends 'base.html.twig' %}

{% block title %}Home{% endblock %}

{% block body %}

<div>
    <h1>Hello {{ controller_name }}!</h1>

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

Symfony

{# commentaire #}

- n'affiche rien et ne fait rien
- s'utilise seulement pour les commentaires multi-lignes

Symfony

{{ var }}

- permet de récupérer et afficher la valeur d'une variable `var` envoyée par le contrôleur
- est l'équivalent de `<?php echo $var; ?>`

Symfony

Le contrôleur `HomeController` envoie à la vue `index.html.twig` un tableau contenant une variable nommée `controller name` ayant comme valeur la chaîne de caractères `HomeController`

```
return $this->render('home/index.html.twig', [
        'controller_name' => 'HomeController',
]);
```

Pour l'afficher, dans `index.html.twig` on utilise l'interpolation

```
<div>
    <h1>Hello {{ controller_name }}!</h1>
</div>
{% endblock %}
```

Symfony

Exercice

- Créez un contrôleur `TwigController`
- La route de ce contrôleur doit permettre de récupérer deux paramètres de requête `nom` et `prenom` et les envoyer à la vue
- La vue affiche les valeurs envoyées par le contrôleur

Symfony

{{ tableau['idColonne'] }}

- affiche le contenu d'un élément du tableau
- est l'équivalent de `<?php echo $tableau['idColonne']; ?>`

Symfony

Exemple : contenu de la méthode `index` de `HomeController`

```
$tab = [2, 3, 8];
return $this->render('home/index.html.twig', [
    'controller_name' => 'HomeController',
    'tableau' => $tab
]);
```

Pour afficher le tableau dans `index.html.twig` : trois écritures correctes

```
<ul>
        <li>
                {{ tableau[0] }}
        </li>
        <li>
                {{ tableau['1'] }}
        </li>
        <li>
                {{ tableau["2"] }}
        </li>
</ul>
```

{{ objet.attribut }}

- affiche, logiquement, la valeur de `$_attribut` de `$objet`
- est l'équivalent de `<?php echo $objet->attribut(); ?>`

Réellement {{ objet.attribut }}

- affiche `$objet['attribut']` si `$objet` est un tableau
- affiche `$objet->_attribut` si `$objet` est un objet et `$_attribut` est public
- affiche `$objet->attribut()` si `$objet` est un objet et `attribut()` est une méthode public
- affiche `$objet->getAttribut()` si `$objet` est un objet et `getAttribut()` est une méthode public
- affiche `$objet->isAttribut()` si `$objet` est un objet et `isAttribut()` est une méthode public
- n'affiche rien et retourne `null` sinon.

Pour l'exemple, commençons par créer une entité `Personne` dans `src/Entity`

```php
namespace App\Entity;

class Personne
{
    private $_num;
    private $_nom;
    private $_prenom;
    public function _construct(int $_num, string $_nom, string
       $_prenom)
    {
        $this->_num = $_num;
        $this->_nom = $_nom;
        $this->_prenom = $_prenom;
    }

    // + getters + setters

    public function_toString(): string
    {
        return $this->_num . " " . $this->_nom . " " . $this->_prenom;
    }
}
```

Dans `HomeController`**, créons une instance de** `Personne`

```
namespace App\Controller;

use App\Entity\Personne;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class HomeController extends AbstractController
{
    /**
     * @Route("/home",name="home_route")
     */
    public function index()
    {
        $personne = new Personne(100, "wick", "john");
        $tab = [2, 3, 8];
        return $this->render('home/index.html.twig', [
            'controller_name' => 'HomeController',
            'tableau' => $tab,
            'personne' => $personne
        ]);
    }
}
```

Symfony

Pour afficher les attributs de l'objet `Personne` dans la vue

```
<ul>
        <li>
                {{ personne.num }}
        </li>

        <li>
                {{ personne.nom() }}
        </li>

        <li>
                {{ personne.getPrenom() }}
        </li>
</ul>
```

Symfony

{{ variable1 ~ " " ~ variable2 }}

- affiche le résultat de la concaténation de `variable1` et `variable2`
- est l'équivalent de
  `<?php echo $variable1 . ' ' . $variable2 ; ?>`

Exemple de concaténation

```
<p>
    {{ personne.prenom ~" " ~personne.nom }}
</p>
```

Symfony

Pour le débogage, vous pouvez utiliser la fonction `dump`

```
{{ dump(personne) }}
```

Symfony

Utilisez le mot-clé `set` pour déclarer une variable `str` dans un bloc `{% ... %}`

```
{% set str = 'bonjour' %}
```

L'équivalent PHP

```
<?php
    $str = 'bonjour';
?>
```

Pour afficher la variable `str`

```
{{ str }}
```

Le mot-clé `set` permet aussi de modifier le contenu de la variable si elle existe

```
{% set str = 'bonjour' %}
```

Symfony

On peut utiliser le mot-clé `with` pour donner une portée locale à une variable

```
{% with  %}
{% set x = 2 %}
  {{ x }}
  {# affiche 2 #}
{% endwith %}

{# la variable x n'est plus visible ici #}
```

Une deuxième écriture sans `set`

```
{% with { x: 2 } %}
  {{ x }}
  {# affiche 2 #}
{% endwith %}

{# la variable x n'est plus visible ici #}
```

Symfony

Dans un bloc `with`, par défaut, on a accès aux variables définies dans le contexte global

```
{% set y = 5 %}
{% with { x: 2 } %}
  {{ x }} {{ y }}
  {# affiche 2 5 #}
{% endwith %}

{# la variable x n'est plus visible ici #}
```

Une deuxième écriture sans `set`

```
{% set y = 5 %}
{% with { x: 2 } only %}
  {{ x }}
  {# affiche 2 et la variable y n'est plus accessible #}
{% endwith %}

{# la variable x n'est plus visible ici #}
```

Symfony

Dans un bloc `with`, par défaut, on a accès aux variables définies dans le contexte global

```
{% set x = 2 %}
{% set y = 5 %}
{{ x + y }}

{# affiche 7 #}
```

Symfony

Autres opérations

- Arithmétiques : `+`, `-`, `*`, `/`, `%`, `**` (puissance) et `//` (division entière)
- Logiques : `and`, `or` et `not`
- Comparaisons : `==`, `!=`, `<`, `>`, `>=`, `<=`, `===`, `starts with`, `ends with`, `matches`
- Autres : `is`, `in`, `[]`, `.`, `..`, `??`, `?:`

Symfony

Exemple avec `in`

```
{{ 8 in tableau }}
{# affiche 1 #}
```

Exemple avec `not in`

```
{{ 1 not in tableau }}
{# affiche 1 #}
```

Exemple avec `starts with`

```
{{ personne.prenom starts with 'j' }}
{# affiche 1 #}
```

Symfony

Exemple avec `matches` permet de déterminer si une variable respecte un motif donné par une expression régulière

```
{{ personne.prenom matches '/^j.*n$/' }}
{# affiche 1 #}
```

Symfony

Exemple avec `if ... endif`

```
{% if tableau[0] > 0 %}
        {{ tableau[0] }} est positif
{% endif %}
```

Exemple avec `if ... else ... endif`

```
{{ tableau[0] }} est
{% if tableau[0] > 0 %}
        positif
{% else %}
        négatif
{% endif %}
```

Symfony

Exemple avec `if ... elsif ... else ... endif`

```
{{ tableau[0] }}
{% if tableau[0] > 0 %}
        positif
{% elseif tableau[0] < 0 %}
        négatif
{% else %}
        nul
{% endif %}
```

Symfony

Tester l'existence d'une variable

```
{% if nom is defined %}
        {{ nom }}
{% else %}
        doe
{% endif %}
```

Tester la parité d'une variable

```
{% if tableau[0] is even %}
    {{ tableau[0] }} est pair
{% endif %}
```

Symfony

Tester si une variable est divisible par 2

```
{% if  tableau[0] is divisible by(2) %}
        {{ tableau[0] }} est pair
{% endif %}
```

Autres fonctions de test prédéfinies

https://twig.symfony.com/doc/3.x/tests/index.html

Symfony

Structure itérative : `for`

```
{% for i in tableau %}
    {{ i }} <br>
{% endfor %}
```

Le résultat

```
2
3
8
```

Symfony

On peut utiliser l'opérateur . . pour définir un intervalle

```
{% for i in 0..3 %}
    {{ i }} <br>
{% endfor %}
```

Le résultat

```
0
1
2
3
```

Symfony

On peut aussi utiliser la fonction `range`

```
{% for i in range(0, 3) %}
    {{ i }} <br>
{% endfor %}
```

Le résultat

```
0
1
2
3
```

Symfony

On peut aussi modifier le pas

```
{% for i in range(low=0, high=7, step=2) %}
    {{ i }} <br>
{% endfor %}
```

Le résultat

```
0
2
4
6
```

Symfony

On peut utiliser l'opérateur . . pour définir un intervalle de caractères

```
{% for i in 'a'..'f' %}
    {{ i }} <br>
{% endfor %}
```

Le résultat

```
a
b
c
d
e
f
```

Symfony

Structure itérative : `for` (clé, valeur)

```
{% for cle, valeur  in tableau %}
        {{ cle  ~ ' : ' ~ valeur }} <br>
{% endfor %}
```

Le résultat

```
0 : 2
1 : 3
2 : 8
```

Symfony

La boucle `for` génère un objet `loop` contenant les attributs suivants :

- `loop.index` : numéro de l'itération courante (commence de 1)
- `loop.index0` : numéro de l'itération courante (commence de 0)
- `loop.length` : le nombre total d'itérations
- `loop.first` : contient `true` s'il s'agit de la première itération
- `loop.last` : contient `true` s'il s'agit de la dernière itération
- `loop.revindex` et `loop.revindex0` : contiennent le nombre d'itérations restantes avant la fin de la boucle

Symfony

Exemple

```
{% for  valeur  in tableau %}
        {{ loop.index0 }} :      {{  valeur }}
        <br>
{% endfor %}
```

Le résultat

```
0 : 2
1 : 3
2 : 8
```

Symfony

Filtre

- Permettant de formater et modifier l'affichage d'une donnée
- Pouvant prendre un ou plusieurs paramètres
- Synatxe : {{ `variable |fonction filtre[paramètres]` }}
- Possibilité d'appliquer un filtre sur le résultat d'un autre filtre
- Liste complète : `https://twig.symfony.com/doc/3.x/filters/index.html`

Symfony

Quelques exemples

- `upper` : convertit les lettres en majuscules comme `strtupper()` en PHP (lower est la réciproque)
- `length` : calcule le nombre d'éléments d'un tableau ou le nombre de caractères d'une chaîne
- `sort` : trie les éléments d'un tableau
- `trim` : supprime les caractères spéciaux indiqués du début et de la fin d'une chaîne de caractères
- `striptags` : supprime les balises HTML
- ...

symfony

Exemples avec les chaînes de caractères

```
{{ personne.prenom | capitalize ~ " " ~ personne.nom | upper }}
{# affiche John WICK #}

{{ personne.prenom | length }}
{# affiche 4 #}

{{ '  john wick!  ' | trim }}
{# affiche 'john wick!' #}

{{ '  john wick!' | trim('!') }}
{# affiche '  john wick' #}

{{ '  john wick!  ' | trim(side='left') }}
{# affiche 'john wick!  ' #}

{{ '  john wick!  ' | trim(' ', 'right') }}
{# affiche '  john wick!' #}
```

symfony

Pour appliquer un filtre à une portion du code

```
{% apply upper %}
    Bonjour {{ personne.prenom }}
{% endapply %}
{# affiche BONJOUR JOHN #}
```

symfony

Exemples avec les tableaux (`slice`)

```
{% for elt in tableau | slice(0, 2) %}
        {{ elt }}
{% endfor %}
{# affiche 2 3 #}
```

Un raccourci pour `slice`

```
{% for elt in tableau [0:2] %}
        {{ elt }}
{% endfor %}
{# affiche 2 3 #}
```

symfony

Au moins un des deux paramètres doit être présent dans [:]

```
{% for elt in tableau [:1] %}
        {{ elt }}
{% endfor %}
{# affiche 2 #}
```

Au moins un des deux paramètres doit être présent dans [:]

```
{% for elt in tableau [1:] %}
        {{ elt }}
{% endfor %}
{# affiche 3 8 #}
```

symfony

Exemples avec les tableaux (`reduce`)

```
{{ tableau | reduce((somme, valeur) => somme + valeur) }}
{# affiche 13 #}
```

`reduce` accepte aussi une valeur initiale

```
{{ tableau | reduce((somme, valeur) => somme + valeur, 5) }}
{# affiche 18 #}
```

Exemple avec `map` et `reduce`

```
{{ tableau | map(elt => elt + 2) | reduce((somme, valeur) =>
   somme + valeur) }}
{# affiche 19 #}
```

Symfony

Remarques

- Par défaut, **Twig** protège les variables en appliquant un filtre pour les protéger de balises **HTML**
- Pour désactiver le filtre, on peut utiliser le filtre `raw`
  Par exemple {{ `variable|raw` }}
- Pour les chaînes de caractères non-définies dans une variable, on utilise `e` pour échapper les balises **HTML**
  Par exemple {{ `'texte <br>'|e` }}

symfony

Exemple (`chaine` est une variable définie dans le contrôleur contenant la valeur `bonjour`)

```
{{ 'hello <br>'  }}
{# affiche hello #}

{{ 'hello <br>' | e }}
{# affiche hello <br> #}

{{ chaine }}
{# affiche bonjour<br>  #}

{{ chaine | raw }}
{# affiche bonjour #}
```

Pour interpréter les différentes balises définies dans la variable `test`

```
{% set test = "<strong>Texte en gras</strong>
<script>
        alert('Ceci est un avertissement')
</script>" %}

{{ test | raw }}
```

On peut aussi le faire avec `autoescape`

```
{% autoescape false %}
        {{ test  }}
 {% endautoescape %}
```

Avec `autoescape`**, on peut choisir la stratégie d'échappement (html, js, css...)**

```
{% autoescape 'html' %}
        {{ test  }}
 {% endautoescape %}
```

Symfony

Caractéristiques

- Elle permet de faire référence au répertoire `web` du projet **Symfony** depuis les vues
- Elle permet donc de référencer des fichiers de ressource (CSS, JavaScript, images ...) définis dans `public`

Exemples

```
<link href="{{ asset('css/style.css') }}" rel="stylesheet" />
<script src="{{ asset('js/jquery-1.11.3.js') }}"></script>
<script src="{{ asset('js/bootstrap.js') }}"></script>
<script src="{{ asset('js/script.js') }}"></script>
```

Symfony

`path` et `url`

- Elles permettent de référencer une route enregistrée dans notre routeur
- `path` génère une URL relative.
- `url` génère une URL absolue.

Exemple

```
<a href="{{ path('vehicule_route') }}">Véhicule</a>
<a href="{{ url('vehicule_route') }}">Véhicule</a>
```

Code HTML équivalent

```
<a href="/vehicule">Accueil</a>
<a href="http://localhost:8000/vehicule">Véhicule</a>
```

Symfony

On peut aussi construire une route avec paramètres

```
<a href="{{ path('vehicule_route',{'id': 'value'})
   }}">Véhicule</a>
<a href="{{ url('vehicule_route') }}">Véhicule</a>
```

Symfony

Les variables globales

- `app.request` : la requête d'un contrôleur
- `app.session` : service session
- `app.user` : pour récupérer l'utilisateur courant
- `app.debug` : `True` si le mode debug est activé, `False` sinon.
- `app.environment` : l'environnement courant `dev` ou `prod`

Exemple

```
{% set id = app.request.get('personne').nom %}
```

Symfony

Considérons la page `menu.twig` à définir directement dans `templates`

```
<ul>
  <li>
    <a href="{{ url('home_route') }}">Home</a>
  </li>
  <li>
    <a href="{{ url('vehicule_route') }}">Vehicule</
       a>
  </li>
</ul>
```

Symfony

Pour inclure le menu dans la vue associée à `HomeController`

```
{% include 'menu.twig' %}
```

Inclusion avec ignorance d'erreur si page inexistante

```
{% include 'page.twig' ignore missing %}
```

Notion d'inclusion conditionnelle

```
{% include condition ? 'menu-admin.twig' : 'menu-
   user.twig' ignore missing  %}}
```

Symfony

Pour inclure le menu, on peut créer une méthode dans `HomeController` et lui associer une route

```
/**
* @Route("/menu",name="menu_route")
*/
public function menu()
{
    return $this->render('menu.twig', []);
}
```

Pour exécuter cette méthode dans la vue, on utilise la fonction `render`

```
{{ render("menu") }}
```

Attention à la performance de la fonction `render`

Symfony

Notion de block : zone réservée

```
{% block nom_block %}
    ...
{% endblock %}
```

Symfony

Exemple

```
{# base.html.twig #}
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>
      {% block title %}Welcome!{% endblock %}
    </title>
    {% block stylesheets %}{% endblock %}
    <link href="{{ asset('css/style.css') }}" rel="stylesheet"/
      >
  </head>
  <body>
    {% block body %}{% endblock %}
    {% block javascripts %}{% endblock %}
  </body>
</html>
```

Symfony

Héritage entre block

```
{% extends 'base.html.twig' %}

{% block title %}
    Home
    {# ce contenu sera inséré dans le bloc title de base.
       html.twig #}
{% endblock %}
```

```
{% block body %}
    {# contenu précédent #}
    {# ce contenu sera inséré dans le bloc body de base.
       html.twig #}
{% endblock %}
```

Symfony

Remarques

- L'héritage sert à créer un template parent (avec un ou plusieurs blocks) qui contient le design de base de notre site pour que les templates enfants puissent l'utiliser
- Si le template enfant ne redéfinit pas un block hérité, il aura la valeur définie par le père pour ce block
- {{ `parent()` }} permet de récupérer le contenu du block côté père
- On peut faire des `include` pour ajouter entièrement un template