

Symfony 6 : TWIG

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

`elmouelhi.achref@gmail.com`



Symfony

Plan

- 1 Introduction
- 2 Commentaire
- 3 Interpolation
- 4 Déclaration de variable
- 5 Opérations sur les variables
- 6 Structure conditionnelle
- 7 Structure itérative

Plan

- 8 Filtre
- 9 Référencement d'un fichier
- 10 Lien hypertexte
- 11 Variables globales
- 12 Inclusion
- 13 Block et héritage
- 14 Bonnes pratiques

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

Twig

- Moteur de templates pour **PHP**.
- Apparue 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**...

Remarque

Symfony 6 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

© Achref EL M...

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

Extensions **VSC** pour **TWIG**

- Twig
- Twig Language 2

Symfony

Trois types de balises

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

Symfony

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

```
namespace App\Controller;

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

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

Et le contenu suivant pour `templates/home/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 en mono-ligne et 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',  
]);
```

© Achref EL MOU

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 %}
```


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']; ?>`

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

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

© Achref EL MOUELHI ©

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(); ?>`

© Achref EL MOUELHI ©

```
{{ 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`

```
namespace App\Entity;

class Personne
{
    private $id;
    private $nom;
    private $prenom;
    public function getId(): ?int
    {
        return $this->id;
    }
    public function setId(int $id): void
    {
        $this->id = $id;
    }
    public function getNom(): ?string
    {
        return $this->nom;
    }
    public function setNom(string $nom): void
    {
        $this->nom = $nom;
    }
    public function getPrenom(): ?string
    {
        return $this->prenom;
    }
    public function setPrenom(string $prenom): void
    {
        $this->prenom = $prenom;
    }
}
```

Symfony

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

```
use App\Entity\Personne;

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

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


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

```
<ul>
    <li>
        {{ personne.id }}
    </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 ; ?>`

© Achref EL

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>
```

Pour le débogage, vous pouvez utiliser la fonction `dump` dans les pages Twig

```
<p>  
    {{ dump(personne) }}  
</p>
```

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

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

© Achref EL MOUELHI ©

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

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

L'équivalent PHP

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

© Achref EL MOU

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 }}
```

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 (impossible de modifier une variable sans `set`)

```
{% 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 %}
```

© Achref EL MOUL

Symfony

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

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

`x` n'existe pas en dehors du bloc

```
{% with %}
  {% set x = 2 %}
  {{ x }}
  {# affiche 2 #}
{% endwith %}
{{ x }}
{# Variable "x" does not exist. #}
```

Une deuxième écriture sans `set`

```
{% with { x: 2 } %}  
  {{ x }}  
  {# affiche 2 #}  
{% endwith %}  
  
{# Variable "x" does not exist. #}
```

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 %}
```

© Achref EL MC

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 %}
```

Pour interdire l'accès aux variables définies dans le contexte global, on utilise `only`

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

Dans une interpolation, on peut effectuer des opérations arithmétiques comme l'addition par exemple

```
{% 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 #}
```

© Achref EL MOUELHI ©

Symfony

Exemple avec `in`

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

Exemple avec `not in`

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

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 #}
```

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 %}
```

© Achref EL MOUETRI

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 ... elseif ... 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 %}
```

© Achref EL

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 %}
```


Tester si une variable est divisible par 2

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

© Achref EL M...

Symfony

Tester si une variable est divisible par 2

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

Autres fonctions prédéfinies de test

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

Symfony

Structure itérative : `for`

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

© Achref EL MOU

Symfony

Structure itérative : for

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

Le résultat

```
2  
3  
8
```

Symfony

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

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

© Achref EL MOUL

Symfony

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

```
{% for elt in 0..3 %}  
    {{ elt }} <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 %}
```

© Achref EL MOUL

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 %}
```

© Achref EL MOUL

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 %}
```

© Achref EL MOUËL

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

Ou aussi

```
{% for i in range('a', 'f') %}  
    {{ i }} <br>  
{% endfor %}
```

© Achref EL MOUETRI

Symfony

Ou aussi

```
{% for i in range('a', 'f') %}  
    {{ i }} <br>  
{% endfor %}
```

Le résultat

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

Structure itérative : for (clé, valeur)

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

© Achref EL MOU

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 %}
```

© Achref EL ME

Symfony

Exemple

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

Le résultat

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

Filtre

- Permettant de formater et modifier l'affichage d'une donnée
- Pouvant prendre un ou plusieurs paramètres
- Syntaxe : `{{ 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>

Quelques exemples

- `upper` : convertit les lettres en majuscules comme `strtoupper()` 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 | length }}  
{# affiche 4 #}
```

```
{{ ' john wick! ' | trim | length }}  
{# affiche 10 #}
```

```
{{ ' john wick!' | trim('!') | length }}  
{# affiche 11 #}
```

```
{{ ' john wick! ' | trim(side='left') | length }}  
{# affiche 12 #}
```

```
{{ ' john wick! ' | trim(' ', 'right') | length }}  
{# affiche 12 #}
```

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

Pour appliquer un filtre à une portion du code

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

Exemples avec les tableaux (slice)

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

© Achref EL MOU

symfony

Exemples avec les tableaux (slice)

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

Un raccourci de 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 #}
```

© Achref EL MOU

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 #}
```

© Achref EL MOUELHI ©

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 #}
```

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 #}
```

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
' | e }}`

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 }}
```

© Achref EL

Symfony

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 %}
```

Symfony

```
asset()
```

Permet de référencer des fichiers de ressource (**CSS**, **JavaScript**, images ...) définis dans `public`

© Achref EL MOUËZ

Symfony

`asset()`

Permet 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.

© Achref EL MOUL

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>
```

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>
```

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`

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 %}
```

Considérons la page `_menu.twig` à définir dans `shared` : un répertoire à créer dans `templates`

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

Pour inclure le menu dans la vue associée à HomeController

```
{% include 'shared/_menu.twig' %}
```

© Achref EL MOUELHI ©

Pour inclure le menu dans la vue associée à HomeController

```
{% include 'shared/_menu.twig' %}
```

Inclusion avec ignorance d'erreur si page inexistante

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

Symfony

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

```
{% include 'shared/_menu.twig' %}
```

Inclusion avec ignorance d'erreur si page inexistante

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

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('shared/_menu.twig', []);
}
```

© Achrel

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('shared/_menu.twig', []);
}
```

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

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


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('shared/_menu.twig', []);
}
```

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

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

Attention à la performance de la fonction `render`

Notion de block : zone réservée

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

Symfony

Le fichier `base.html.twig` situé dans `templates`

```
{# 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>
```

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 %}
```

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

Bonnes pratiques

- Utiliser le **Snake Case** pour nommer les vues (Exemple : `page_admin.html.twig`).
- Pour les fragments de vue, préfixer le nom d'une vue par un underscore `_` (Exemple : `_menu.twig`).