Twig

Version	V0
Type	Technique
Date de création	@16 mai 2023 16:18
Dernière modification	@14 décembre 2023 15:54

👋 Présentation

🔗 Documentation Symfony template avec Twig

🔗 Documentation Twig template

Twig est un moteur de template moderne et puissant utilisé dans le framework Symfony. Il est conçu pour faciliter la création de vues et la gestion de la logique de présentation dans une application web. Twig offre une syntaxe concise, intuitive et lisible, ce qui rend la création de templates plus facile et plus agréable.

Twig facilite la réutilisation des templates. Les portions de code fréquemment utilisées peuvent être extraites dans des fragments réutilisables appelés "partials". Cela permet de maintenir un code DRY (Don't Repeat Yourself) et d'éviter la duplication inutile.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Welcome to Symfony!</title>
    </head>
    <body>
        <h1>{{ page_title }}</h1>

        {% if user.isLoggedIn %}
            Hello {{ user.name }}!
        {% endif %}
```

La syntaxe Twig est basée sur ces trois constructions :

`{{ ... }}` utilisé pour afficher le contenu d'une variable ou le résultat de l'évaluation d'une expression.

`{% ... %}` utilisé pour exécuter une logique, telle qu'une condition ou une boucle.

`{# ... #}` utilisé pour ajouter des commentaires au modèle.

Vous ne pouvez pas exécuter de code PHP dans les modèles Twig, mais Twig fournit des utilitaires pour exécuter une logique dans les modèles. Par exemple, les filtres modifient le contenu avant d'être rendus, comme le filtre supérieur pour le contenu en majuscule : `{{ title|upper }}`

🔗 Les tags 🔗 les filtres 🔗 les fonctions

```
        {# ... #}
    </body>
</html>
```

📄 Le fichier base HTML

Ce fichier nous servira de squelette pour le développement de nos différents templates (“vue” dans le modèle MVC).

Ici je souhaite exploiter un fichier CSS/SCSS sur toutes mes pages, je préférerais donc l’importer dans mon fichier `base.html.twig`, par la suite j’étendrais ce template de base pour bénéficier de mon fichier CSS/SCSS sans avoir besoin de l’importer à nouveau.

`base.html.twig`

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>

                    {% block title %}
                        Welcome!
```

```
{% block stylesheets %}
  {{ encore_entry_link_tags('app') }}
{% endblock %}
```

J’ai fais de même pour mon header.

```
{% block header %}
  {% include 'components/header.html.twig' %}
{% endblock %}
```

J’ai fais de même pour le titre de ma page. À l’avenir j’aimerais adapter ce titre pour chaque page et ainsi améliorer le référencement naturel de mon site, les balises de block mises à disposition par Twig me le permettrons (voir chapitre suivant)

```
{% block title %}
        Welcome!
{% endblock %}
```

📦 Blocks

Un bloc permet de modifier le rendu d'une certaine partie d'un template, mais il n'interfère en aucune façon avec la logique qui l'entoure. Vous êtes libres de donner le nom que vous souhaitez à votre block (veillez à respecter les conventions de nommages).

```
                    {% endblock %}

                </title>

        {% block stylesheets %}
            {{ encore_entry_link_tags('app') }}
        {% endblock %}

    </head>
    <body>

        {% block header %}
            {% include 'components/header.html.twig' %}
        {% endblock %}

                {% block body %}
                {% endblock %}

    </body>
</html>
```

`base.html.twig`

```
<!DOCTYPE html>
<html>
    <head>
```

Reprenons l’exemple du block title de ma page `base.html.twig`. Dans une page tiers, par exemple `products.html.twig`, je souhaite modifier le titre de ma page pour améliorer le référencement naturel de mon site.

Je commence par étendre la page `base.html.twig` avec le code `{% extends 'base.html.twig' %}` dans ma nouvelle page. Cela permettra à ma page `products.html.twig` d’hériter du contenu de la page `base.html.twig`, super pratique pour ne pas avoir à ré-importer mes feuilles de styles globales, mon header, etc.

Ma page `products.html.twig` hérite donc bien du contenu de ma page `base.html.twig`, cela inclus qu’elle hérite aussi des block de la page `base.html.twig`, à ce titre je peux faire appel à ses blocks dans ma nouvelle page et cela aura pour effet d’override le block présent initialement dans la page `base.html.twig`.

Dans notre exemple la page `base.html.twig` à un block “title” avec la valeur “Welcome!”. La page `products.html.twig` étend la page `base.html.twig`, si l’on ne modifie rien elle hérite donc de la valeur “Welcome!” comme titre de page. Cependant je peux faire appel au block “title” depuis ma nouvelle page et définir un nouveau titre de page.

Pourquoi utiliser les blocs ? La philosophie DRY

La philosophie DRY, “Don’t Repeat Yourself” en anglais, est une philosophie en programmation informatique consistant à éviter la redondance de code au sein d'une application afin de faciliter la maintenance, le test, le débogage et les évolutions de cette dernière.

```
        <meta charset="UTF-8">
        <title>

                    {% block title %}
                        Welcome!
                    {% endblock %}

                </title>
                ...
    </head>
    <body>
            ...
    </body>
</html>
```

`products.html.twig`

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

{% block title %}
    Page de mes produits
{% endblock %}
```

Grâce au système de block de Twig vous pouvez définir des pages de bases, composants, etc et exploiter les blocks pour adapter vos pages sans jamais avoir à répéter votre code.

Extend

La méthode `extends` permet d'étendre le contenu d'un template dans un autre template.

Dans l'exemple ci-contre, le fichier `base.html.twig` importe un fichier Javascript un fichier CSS, imaginons que ces fichiers soient utilises à toutes l'application, plutôt que de déclarer les imports dans chaqu'un des fichiers il serait plus pertinent d'étendre cette base pour bénéficier des imports sans les re-declarer.

`random.html.twig`

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

Le fichier `random.html.twig` bénéficiera du contenu dans la page étendue, ici `base.html.twig`.

La méthode `extends` permet d'exploiter les blocks mis à dispositions par le template étendu, ici notre `random.html.twig` étend le fichier `base.html.twig` et bénéficie donc du block body notamment.

`base.html.twig`

```
<!DOCTYPE html>
<html lang="fr">
    <head>
                ...

        {% block stylesheets %}
            {{ encore_entry_link_tags('app') }}
        {% endblock %}

        {% block javascripts %}
            {{ encore_entry_script_tags('app') }}
        {% endblock %}

    </head>
    <body>
        {% include 'components/header.html.twig' %}
        {% block body %}{% endblock %}
    </body>
</html>
```

`random.html.twig`

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

{% block body %}

    ... blablabla

{% endblock %}
```

🔍 Include

La méthode `include` permet d'inclure le contenu d'un fichier Twig dans un autre.

Dans l'exemple ci-contre, le fichier `header.html.twig` contient le code nécessaire à mon header, celui-ci sera présent à plusieurs endroits de mon site, plutôt que d'écrire mon header dans chacun des fichiers il serait plus pertinent d'inclure ce fichier-là où je risque d'en avoir besoin.

`random.html.twig`

```
<!DOCTYPE html>
<html lang="fr">
        ...
    <body>

        {% include 'components/header.html.twig' %}
```

`header.html.twig`

```
<!DOCTYPE html>
<html lang="fr">
        <h1>Mon titre</h1>
    <nav>
            <ul>
                    <li>
                        <a href="">Lien 1</a>
                    </li>
                    <li>
                        <a href="">Lien 2</a>
                    </li>
                    <li>
                        <a href="">Lien 3</a>
                    </li>
            </ul>
```

```
        {% block body %}{% endblock %}

    </body>
</html>
```

Embed

La méthode `embed` rempli le rôle des méthodes `include` et `extend`.

À partir de la méthode `embed` vous pouvez inclure le contenu d'un fichier (comme pour le `header.html.twig` du sous-chapitre `include`) et bénéficier des blocks mis à disposition par le fichier (contrairement à la méthode `include`).

`random.html.twig`

```
<!DOCTYPE html>
<html lang="fr">
        ...
    <body>

        {% embed 'components/header.html.twig' %}

                    {% block firstLink %}
                            Mon super 1er lien
                    {% endblock %}

                    {% block secondLink %}
                            Mon super 2ème lien
```

```
    </nav>
</html>
```

`header.html.twig`

```
<!DOCTYPE html>
<html lang="fr">
        <h1>Mon titre</h1>
    <nav>
            <ul>
                    <li>
                        <a href="">
                            {% block firstLink %}
                            {% endblock %}
                        </a>
                    </li>
                    <li>
                        <a href="">
                            {% block secondLink %}
                            {% endblock %}
                        </a>
                    </li>
                    <li>
                        <a href="">
                            {% block thirdLink %}
```

```
                    {% endblock %}

                    {% block thirdLink %}
                            Mon super 3ème lien
                    {% endblock %}

                {% endembed %}

        {% block body %}{% endblock %}

    </body>
</html>
```

```
                            {% endblock %}
                        </a>
                    </li>
            </ul>
    </nav>
</html>
```

🔁 For

La méthode `for` permet de faire des boucles dans un fichier Twig.

```
<ul>

    {% for product in products %}

        <div>
            <p>{{ product.title }}</p>
        </div>

    {% endfor %}
```

```
</ul>
```

Équivalent PHP

```
<?php foreach($products as $key => $product): ?>
    <div>
        <p><?php $product.title ?></p>
    </div>
<?php endforeach; ?>
```

❓ If

La méthode `if` permet de faire des conditions dans un fichier Twig.

```
<div>

    {% if user %}

        Hello {{ user.username }}

    {% elseif %}

        Hello l'inconnu

    {% endif %}
```

```
</div>
```

🔗 Les liens de pages

Au lieu d'écrire les URL de lien à la main, utilisez la fonction `path()` pour générer des URL basées sur la configuration de routage. Plus tard, si vous souhaitez modifier l'URL d'une page en particulier, il vous suffira de modifier la configuration du routage : les templates généreront automatiquement la nouvelle URL.

```
// src/Controller/BlogController.php
namespace App\Controller;

// ...
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class BlogController extends AbstractController
{
        # Le nom de cette route est "blog_index"
    #[Route('/', name: 'blog_index')]
    public function index(): Response
    {
        // ...
    }

    # Ici "blog_post", elle doit recevoir un paramètre "id
    #[Route('/article/{id}', name: 'blog_post')]
    public function show(int $id): Response
    {
        // ...
```

```
{# "blog_index" correspond au nom d'une route, ici défini
<a href="{{ path('blog_index') }}">Homepage</a>

{# ... #}

{% for post in blog_posts %}
    <h1>
                {# pareil ici avec "blog_post" où l'on inc
        <a href="{{ path('blog_post', {id: post.id}) }}">
    </h1>

    <p>{{ post.excerpt }}</p>
{% endfor %}
```

```
    }
}
```

🖼️ Image

Pour lier vos assets (CSS, JS, image …) Twig met à disposition la fonction `asset()`, celle-ci map le dossier `public`.

⚠️ Attention, lorsqu'il s'agit des fichiers CSS et Javascript une nuance est à connaître :

- si vous utilisez du Typescript ou du SCSS et que Webpack est chargé de les compiler pour que ceux-ci puissent être exploité par le navigateur vous ne pourrez pas utiliser `asset()` mais vous devrez définir un point d'entrée dans le `webpack.config`.

```
{# l'image que je cherche à afficher ce situe dans public/images/logo.png #}
<img src="{{ asset('images/logo.png') }}" alt="Symfony!"/>

{# le fichier de style que je cherche à importer ce situe dans public/css/blog.css #}
<link href="{{ asset('css/blog.css') }}" rel="stylesheet"/>

{# le fichier de script que je cherche à importer ce situe dans public/bundles/acme/js/loader.js #}
<script src="{{ asset('bundles/acme/js/loader.js') }}"></script>
```

👤 L'utilisateur

Grâce à la variable globale de Twig : `app.user` vous pouvez accéder à l'utilisateur connecté depuis vos templates ("vue" en MCD).

```
{% if is_granted('IS_AUTHENTICATED_FULLY') %}
    <p>Email: {{ app.user.email }}</p>
{% endif %}
```

```
{% if app.user %}
    <p>Vous êtes connecté</p>
{% endif %}
```