Création et utilisation de modèles

Un modèle est la meilleure façon d’organiser et de rendre html de l’intérieur de votre application, si vous devez rendre HTML à partir d’un [contrôleur ou](https://symfony.com/doc/current/controller.html) générer le contenu [d’un e-mail](https://symfony.com/doc/current/mailer.html). Les modèles de Symfony sont créés avec Twig : un moteur modèle flexible, rapide et sécurisé.

Langue templating de brindille

Le langage templating [Twig](https://twig.symfony.com/) vous permet d’écrire des modèles concis et lisibles qui sont plus amicaux pour les concepteurs web et, de plusieurs façons, plus puissants que les modèles PHP. Jetez un oeil à l’exemple suivant modèle de brindille. Même si c’est la première fois que vous voyez Twig, vous comprenez probablement la plupart de celui-ci:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!DOCTYPE html>  <html>  <head>  <title>Welcome to Symfony!</title>  </head>  <body>  <h1>{{ page\_title }}</h1>  {% if user.isLoggedIn %}  Hello {{ user.name }}!  {% endif %}  *{# ... #}*  </body>  </html> |

La syntaxe des brindilles est basée sur ces trois constructions :

1. {{ ... }} , utilisé pour afficher le contenu d’une variable ou le résultat de l’évaluation d’une expression;
2. {% ... %} , utilisé pour exécuter une certaine logique, comme un conditionnel ou une boucle;
3. {# ... #} , utilisé pour ajouter des commentaires au modèle (contrairement aux commentaires HTML, ces commentaires ne sont pas inclus dans la page rendue).

Vous ne pouvez pas exécuter le code PHP à l’intérieur des modèles Twig, mais Twig fournit des utilitaires pour exécuter une certaine logique dans les modèles. Par exemple, les **filtres modifient** le contenu avant d’être rendus, comme le filtre supérieur au contenu de la majusécase :

|  |  |
| --- | --- |
| 1 | {{ title|upper }} |

Brindille est livré avec une longue liste de balises, [filtres](https://twig.symfony.com/doc/2.x/filters/index.html) [et fonctions](https://twig.symfony.com/doc/2.x/functions/index.html) qui sont disponibles par défaut. Dans les applications Symfony, vous pouvez également utiliser ces [filtres et fonctions de brindilles définis par Symfony](https://symfony.com/doc/current/reference/twig_reference.html) et vous pouvez créer vos propres [filtres et fonctions de brindilles.](https://symfony.com/doc/current/templating/twig_extension.html)

Brindille est rapide dans l’environnement prod  (parce que les modèles sont compilés en PHP et mis en cache automatiquement), mais pratique à utiliser dans l’environnement de développement (parce que les modèles sont recompilés automatiquement lorsque vous les modifiez).

Configuration des brindilles

Twig a plusieurs options de configuration pour définir des choses comme le format utilisé pour afficher les nombres et les dates, la mise en cache du modèle, etc. Lisez la référence [de configuration Twig pour](https://symfony.com/doc/current/reference/configuration/twig.html) en savoir plus à leur sujet.

Création de modèles

Avant d’expliquer en détail comment créer et rendre des modèles, regardez l’exemple suivant pour un aperçu rapide de l’ensemble du processus. Tout d’abord, vous devez créer un nouveau fichier dans les modèles / répertoire pour stocker le contenu du modèle:

|  |  |
| --- | --- |
| 1  2  3 | *{# templates/user/notifications.html.twig #}*  <h1>Hello {{ user\_first\_name }}!</h1>  <p>You have {{ notifications|length }} new notifications.</p> |

Ensuite, créez un [contrôleur](https://symfony.com/doc/current/controller.html) qui rend ce modèle et lui transmet les variables nécessaires :

*// src/Controller/UserController.php*

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Response;

class UserController extends AbstractController

{

*// ...*

public function notifications(): Response

{

*// get the user information and notifications somehow*

$userFirstName = '...';

$userNotifications = ['...', '...'];

*// the template path is the relative file path from `templates/`*

return $this->render('user/notifications.html.twig', [

*// this array defines the variables passed to the template,*

*// where the key is the variable name and the value is the variable value*

*// (Twig recommends using snake\_case variable names: 'foo\_bar' instead of 'fooBar')*

'user\_first\_name' => $userFirstName,

'notifications' => $userNotifications,

]);

}

}

Nom du modèle

Symfony recommande ce qui suit pour les noms de modèles :

1. Utilisez [un étui](https://en.wikipedia.org/wiki/Snake_case) snake pour les noms de fichiers et les répertoires (p. ex. blog\_posts.html.twig, admin/default\_theme/blog/index.html.twig,etc.);
2. Définissez deux extensions pour les noms de fichiers (p. ex. index.html.twig ou blog\_posts.xml.twig)étant la première extension(html, xml,etc.) le format final que le modèle générera.

Bien que les modèles génèrent généralement du contenu HTML, ils peuvent générer n’importe quel format basé sur le texte. C’est pourquoi la convention à deux extensions simplifie la façon dont les modèles sont créés et rendus pour plusieurs formats.

Emplacement du modèle

Les modèles sont stockés par défaut dans les modèles/répertoires. Lorsqu’un service ou un contrôleur rend le modèle produit/index.html.twig, ils se réfèrent en fait au fichier <your-project>/templates/product/index.html.twig file.

L’annuaire des modèles par défaut est configurable avec [l’option twig.default\_path](https://symfony.com/doc/current/reference/configuration/twig.html#config-twig-default-path) et vous pouvez ajouter plus d’annuaires de modèles comme expliqué plus [loin](https://symfony.com/doc/current/templates.html#templates-namespaces) dans cet article.

Variables de modèle

Un besoin courant de modèles est d’imprimer les valeurs stockées dans les modèles passés à partir du contrôleur ou du service. Les variables stockent généralement des objets et des tableaux au lieu de chaînes, de nombres et de valeurs boolean. C’est pourquoi Twig offre un accès rapide à des variables PHP complexes. Considérez le modèle suivant :

|  |  |
| --- | --- |
| 1 | <p>{{ user.name }} added this comment on {{ comment.publishedAt|date }}</p> |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | *// src/Controller/BlogController.php*  namespace App\Controller;  *// ...*  use Symfony\Component\HttpFoundation\Response;  use Symfony\Component\Routing\Annotation\Route;  class BlogController extends AbstractController  {  */\*\**  *\* @Route("/", name="blog\_index")*  *\*/*  public function index(): Response  {  *// ...*  }  */\*\**  *\* @Route("/article/{slug}", name="blog\_post")*  *\*/*  public function show(string $slug): Response  {  *// ...*  }  } |

Utilisez la fonction path () Brindille pour lier à ces pages et passer le nom de l’itinéraire comme premier argument et les paramètres de l’itinéraire comme deuxième argument facultatif:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <a href="{{ path('blog\_index') }}">Homepage</a>  *{# ... #}*  {% for post in blog\_posts %}  <h1>  <a href="{{ path('blog\_post', {slug: post.slug}) }}">{{ post.title }}</a>  </h1>  <p>{{ post.excerpt }}</p>  {% endfor %} |

La fonction chemin () génère des URL relatives. Si vous avez besoin de générer des URL absolues (par exemple lors du rendu de modèles pour les e-mails ou les flux RSS), utilisez la fonction url(), qui prend les mêmes arguments que path() (par exemple <a href="{{ url ('blog\_index') }}"> ... </a>).

Liens vers CSS, JavaScript et Les actifs d’image[¶](https://symfony.com/doc/current/templates.html#linking-to-css-javascript-and-image-assets)

Si un modèle doit se lier à un actif statique (par exemple une image), Symfony fournit un actif () fonction Brindille pour aider à générer cette URL. Tout d’abord, installez l’ensemble d’actifs :

|  |
| --- |
| composer require symfony/asset |

Vous pouvez maintenant utiliser la fonction asset () :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *{# the image lives at "public/images/logo.png" #}*  <img src="{{ asset('images/logo.png') }}" alt="Symfony!"/>  *{# the CSS file lives at "public/css/blog.css" #}*  <link href="{{ asset('css/blog.css') }}" rel="stylesheet"/>  *{# the JS file lives at "public/bundles/acme/js/loader.js" #}*  <script src="{{ asset('bundles/acme/js/loader.js') }}"></script> |

L’objectif principal de la fonction asset() est de rendre votre application plus portable. Si votre application vit à la racine de votre hôte (par exemple https://example.com), alors le chemin rendu doitêtre /images/logo.png. Mais si votre application vit dans une sous-direction (par exemple https://example.com/my\_app),chaque chemin d’actif doit être rendu avec le sous-directeur (p. ex. /my\_app/images/logo.png). La fonction asset() s’en occupe en déterminant comment votre application est utilisée et en générant les chemins corrects en conséquence.

**pourboire**

La fonction asset () prend en charge diverses techniques de buste de cache via la [version,](https://symfony.com/doc/current/reference/configuration/framework.html#reference-framework-assets-version) [version\_format,](https://symfony.com/doc/current/reference/configuration/framework.html#reference-assets-version-format)et [json\_manifest\_path](https://symfony.com/doc/current/reference/configuration/framework.html#reference-assets-json-manifest-path) options de configuration.

**pourboire**

Si vous souhaitez aider à l’emballage, la version et la minification de vos actifs JavaScript et CSS d’une manière moderne, lisez [sur Webpack Encore de Symfony](https://symfony.com/doc/current/frontend.html).

Si vous avez besoin d’URL absolues pour les actifs, utilisez la fonction absolute\_url () Brindille comme suit :

|  |  |
| --- | --- |
| 1  2  3 | <img src="{{ absolute\_url(asset('images/logo.png')) }}" alt="Symfony!"/>  <link rel="shortcut icon" href="{{ absolute\_url('favicon.png') }}"> |

La variable globale de l’application

Symfony crée un objet contexteur qui est injecté automatiquement dans chaque modèle de brindille comme une variable appelée application. Il donne accès à certaines informations d’application :

|  |  |
| --- | --- |
| 1  2  3  4  5 | <p>Username: {{ app.user.username ?? 'Anonymous user' }}</p>  {% if app.debug %}  <p>Request method: {{ app.request.method }}</p>  <p>Application Environment: {{ app.environment }}</p>  {% endif %} |

La variable d’application (qui est une instance de Symfony\Bridge\Twig\AppVariable)vous donne accès à ces variables :

app.user (en)

[L’objet utilisateur actuel ou](https://symfony.com/doc/current/security.html#create-user-class) nul si l’utilisateur n’est pas authentifié.

app.request (en)

L’objet Symfony\Component\HttpFoundation\Request qui stocke les données de demande [actuelles](https://symfony.com/doc/current/components/http_foundation.html#accessing-request-data) (selon votre application, il peut s’agir d’une [sous-demande ou](https://symfony.com/doc/current/components/http_kernel.html#http-kernel-sub-requests) d’une demande régulière).

app.session (en)

L’objet Symfony\Component\HttpFoundation\Session\Session qui représente la session de [l’utilisateur actuel ou nul s’il](https://symfony.com/doc/current/session.html) n’y en a pas.

app.flashes App

Un tableau de tous les [messages flash stockés](https://symfony.com/doc/current/controller.html#flash-messages) dans la session. Vous pouvez également obtenir uniquement les messages d’un certain type (par exemple app.flashes (« avis »)).

app.environnement (en)

Le nom de [l’environnement de configuration](https://symfony.com/doc/current/configuration.html#configuration-environments) actuel(dev, prod,etc.).

app.debug (en)

Vrai si en [mode débogage](https://symfony.com/doc/current/configuration/front_controllers_and_kernel.html#debug-mode). Faux autrement.

app.token (en)

Un objet Symfony\Component\Security\Core\Authentication\Token\TokenInterface représentant le jeton de sécurité.

En plus de la variable d’application globale injectée par Symfony, vous pouvez également [injecter automatiquement des variables à tous les modèles Twig](https://symfony.com/doc/current/templating/global_variables.html).

Héritage de modèle et mises en page[¶](https://symfony.com/doc/current/templates.html#template-inheritance-and-layouts)

Au fur et à mesure que votre application se développe, vous trouverez de plus en plus d’éléments [répétés](https://symfony.com/doc/current/templates.html#templates-include) entre les pages, tels que les en-têtes, les pieds, les barres latérales, etc. Y compris les modèles [et les contrôleurs d’intégration](https://symfony.com/doc/current/templates.html#templates-embed-controllers) peuvent aider, mais lorsque les pages partagent une structure commune, il est préférable d’utiliser l’héritage .

Le concept de [l’héritage de modèle de](https://twig.symfony.com/doc/2.x/tags/extends.html) brindille est semblable à l’héritage de classe de PHP. Vous définissez un modèle parent que d’autres modèles peuvent étendre à partir et les modèles enfant peuvent remplacer des parties du modèle parent.

Symfony recommande l’héritage de modèle à trois niveaux suivant pour les applications moyennes et complexes :

1. modèles / base.html.twig , définit les éléments communs de tous lesmodèles d’application, tels que <head >, <header >, <footer >,etc;
2. modèles/mise en page.html.twig, s’étend de base.html.twig et définit la structure de contenu utilisée dans toutes ou la plupart des pages, telles qu’un contenu à deux colonnes + mise en page de barre latérale. Certaines sections de l’application peuvent définir leurs propres mises en page (p. ex. modèles/blog/mise en page.html.twig);
3. modèles/\*.html.twig , lespages d’application qui s’étendent de la mise en page principale.html.twig modèle ou toute autre disposition de section.

Dans la pratique, le modèle .html.twig de base ressemblerait à ceci :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | *{# templates/base.html.twig #}*  <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>{% block title %}My Application{% endblock %}</title>  {% block stylesheets %}  <link rel="stylesheet" type="text/css" href="/css/base.css"/>  {% endblock %}  </head>  <body>  {% block body %}  <div id="sidebar">  {% block sidebar %}  <ul>  <li><a href="{{ path('homepage') }}">Home</a></li>  <li><a href="{{ path('blog\_index') }}">Blog</a></li>  </ul>  {% endblock %}  </div>  <div id="content">  {% block content %}{% endblock %}  </div>  {% endblock %}  </body>  </html> |

[L’étiquette de bloc de](https://twig.symfony.com/doc/2.x/tags/block.html) brindille définit les sections de page qui peuvent être remplacées dans les modèles d’enfant. Ils peuvent être vides, comme le bloc de contenu ou définir un contenu par défaut, comme le bloc de titre, qui s’affiche lorsque les modèles enfant ne les remplacent pas.

Le blog / mise en page.html.twig modèle pourrait être comme ceci:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *{# templates/blog/layout.html.twig #}*  {% extends 'base.html.twig' %}  {% block content %}  <h1>Blog</h1>  {% block page\_contents %}{% endblock %}  {% endblock %} |

Le modèle s’étend .html.twig de base et ne définit que le contenu du bloc de contenu. Le reste des blocs de modèle parent affichera leur contenu par défaut. Cependant, ils peuvent être remplacés par le modèle d’héritage de troisième niveau, tels que blog / index.html.twig, qui affiche l’index blog:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | *{# templates/blog/index.html.twig #}*  {% extends 'blog/layout.html.twig' %}  {% block title %}Blog Index{% endblock %}  {% block page\_contents %}  {% for article in articles %}  <h2>{{ article.title }}</h2>  <p>{{ article.body }}</p>  {% endfor %}  {% endblock %} |

Ce modèle s’étend à partir du modèle de deuxième niveau ( blog /mise en page.html.twig) mais remplace les blocs de différents modèles parent: page\_contents de blog / mise en page.html.twig et le titre de base.html.twig.

Lorsque vous rendez le modèle blog/index.html.twig, Symfony utilise trois modèles différents pour créer le contenu final. Ce mécanisme d’héritage augmente votre productivité parce que chaque modèle inclut uniquement son contenu unique et laisse le contenu répété et la structure HTML à certains modèles parent.

**prudence**

Lors de l’utilisations’étend , un modèle enfant est interdit de définir des parties de modèle en dehors d’un bloc. Le code suivant lance un SyntaxError:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *{# app/Resources/views/blog/index.html.twig #}*  {% extends 'base.html.twig' %}  *{# the line below is not captured by a "block" tag #}*  <div class="alert">Some Alert</div>  *{# the following is valid #}*  {% block content %}My cool blog posts{% endblock %} |

Sortie s’échappant

Imaginez que votre modèle inclut le code Hello {{ name }} pour afficher le nom d’utilisateur. Si un utilisateur malveillant définit <script>alert ('hello!') </script> comme leur nom et vous sortie que la valeur inchangée, l’application affichera une fenêtre popup JavaScript.

C’est ce [qu’on appelle une attaque de script cross-site](https://en.wikipedia.org/wiki/Cross-site_scripting) (XSS). Et tandis que l’exemple précédent semble inoffensif, l’attaquant pourrait écrire du code JavaScript plus avancé pour effectuer des actions malveillantes.

Pour éviter cette attaque, utilisez *la « sortie s’échappant »* pour transformer les caractères qui ont une signification particulière (p. ex. remplacer et par le & lt; entité HTML). Les applications Symfony sont sûres par défaut car elles effectuent une sortie automatique s’échappant grâce à [l’option Twig autoescape](https://symfony.com/doc/current/reference/configuration/twig.html#config-twig-autoescape):

|  |  |
| --- | --- |
| 1  2  3 | <p>Hello {{ name }}</p>  *{# if 'name' is '<script>alert('hello!')</script>', Twig will output this:*  *'<p>Hello &lt;script&gt;alert(&#39;hello!&#39;)&lt;/script&gt;</p>' #}* |

Si vous rendez une variable fiable et contient du contenu HTML, utilisez le [filtre brut Twig](https://twig.symfony.com/doc/2.x/filters/raw.html) pour désactiver la sortie s’échappant pour cette variable :

|  |  |
| --- | --- |
| 1  2  3 | <h1>{{ product.title|raw }}</h1>  *{# if 'product.title' is 'Lorem <strong>Ipsum</strong>', Twig will output*  *exactly that instead of 'Lorem &lt;strong&gt;Ipsum&lt;/strong&gt;' #}* |