Introduction à Jinja2

Dans cet article, nous allons apprendre à passer des données dynamiques depuis une application Flask vers des templates HTML en utilisant Jinja2. Nous commencerons par voir comment transmettre et afficher des chaînes de caractères simples, puis nous explorerons la façon de passer des listes et de les afficher de manière itérative dans nos pages web. Ces techniques vous permettront de créer des applications web interactives et dynamiques, enrichissant ainsi l'expérience utilisateur.

5TTR 6TTR Intérmédiaire

Passer des Données au Template Jinja2 avec Flask

Introduction

Lorsque vous développez des applications web avec Flask, il est essentiel de savoir comment passer des données dynamiques à vos pages HTML. Flask utilise Jinja2, un moteur de templates, pour faciliter ce processus. Dans cet article, nous allons voir comment passer des chaînes de caractères et des listes aux templates Jinja2 et les afficher dynamiquement.

Étape 1 : Passer des Chaînes de Caractères au Template

Configuration de l'Application

Commencez par créer une application Flask de base. Si vous avez déjà une application, vous pouvez l'utiliser. Sinon, créez un fichier `app.py` et ajoutez le code suivant :

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    title = "Accueil"
    message = "Bienvenue sur notre site web !"
    return render_template('home.html', title=title, message=message)

```

```python
if __name__ == '__main__':
    app.run(debug=True)
```

Dans ce code :

- Nous importons Flask et la fonction `render_template`.
- Nous créons une instance de l'application Flask.
- Nous définissons une route pour la page d'accueil `/`.
- Nous passons deux variables (`title` et `message`) au template `home.html` via `render_template`.

Création du Template

Ensuite, créez un dossier `templates` dans le répertoire de votre projet, puis créez un fichier `home.html` dans ce dossier avec le contenu suivant :

```html
<!-- templates/home.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ title }}</title>
</head>
<body>
    <h1>{{ title }}</h1>
    <p>{{ message }}</p>
</body>
</html>
```

Dans ce template :

- Les doubles accolades `{{ }}` sont utilisées pour insérer les valeurs des variables `title` et `message` dans le HTML.

Résultat

Lorsque vous lancez l'application en exécutant `python app.py` et que vous accédez à http://127.0.0.1:5000/, vous verrez les valeurs des variables `title` et `message` affichées sur la page web.

Étape 2 : Conditions

Dans Jinja2, le système de conditions permet d'ajuster dynamiquement le rendu des templates en fonction des valeurs de données. Ce mécanisme fonctionne de manière similaire aux structures conditionnelles dans les langages de programmation, en utilisant les blocs `{% if %}`, `{% elif %}`, et `{% else %}`. Ces blocs permettent d'exécuter des portions de code HTML spécifiques selon que des conditions soient remplies ou non. Par exemple, il est possible d'afficher un message différent selon le rôle d'un utilisateur (`admin`, `utilisateur`, etc.) ou de ne montrer certains éléments qu'en cas de disponibilité de données.

Les conditions sont particulièrement utiles pour adapter l'affichage sans avoir besoin de créer plusieurs templates. Jinja2 évalue chaque condition de haut en bas, et dès qu'une condition est vraie, le bloc correspondant

est exécuté tandis que le reste est ignoré. Cette flexibilité permet de créer des interfaces plus dynamiques et personnalisées.

Modifier l'application

Ajoutez un booléen "danger":

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    title = "Accueil"
    message = "Bienvenue sur notre site web !"
    return render_template('home.html', title=title, message=message, danger=True)

if __name__ == '__main__':
    app.run(debug=True)
```

Modifier le template

```jinja
{% if danger %}
    <p class="danger">Attention, le worm a été libéré!</p>
{% else %}
    <p class="success">Tout va bien, aucune anomalie détectée.</p>
{% endif %}
```

Étape 3 : Passer et Afficher des Listes

Mise à Jour de l'Application

Modifions maintenant l'application pour passer une liste de données au template. Mettez à jour le fichier `app.py` comme suit :

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    title = "Accueil"
    message = "Bienvenue sur notre site web !"
    items = ["Élément 1", "Élément 2", "Élément 3"]
    danger = True
    return render_template('home.html', title=title, message=message, items=items, danger=dang

```

```python
if __name__ == '__main__':
    app.run(debug=True)
```

Dans ce code :

- Nous avons ajouté une liste `items` contenant trois chaînes de caractères.
- Nous passons cette liste au template `home.html` avec les autres variables.

Mise à Jour du Template

Mettez à jour le fichier `home.html` pour afficher la liste. Ajoutez le code suivant dans le fichier :

```html
<!-- templates/home.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ title }}</title>
</head>
<body>
    <h1>{{ title }}</h1>
    <p>{{ message }}</p>

    <ul>
        {% for item in items %}
            <li>{{ item }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Dans ce template :

- Nous utilisons une boucle `for` de Jinja2 pour itérer sur chaque élément de la liste `items` et l'afficher dans une balise `<li>` à l'intérieur d'une liste non ordonnée `<ul>`.

Résultat

Lorsque vous relancez l'application en exécutant à nouveau `python app.py` et que vous accédez à http://127.0.0.1:5000/, vous verrez les éléments de la liste affichés dans une liste HTML.

Conclusion

Dans cet article, nous avons appris comment passer des données dynamiques aux templates Jinja2 dans une application Flask. Nous avons vu comment passer et afficher des chaînes de caractères ainsi que des listes en utilisant Jinja2. Ces techniques sont essentielles pour rendre vos applications web interactives et dynamiques.

Continuez à explorer Jinja2 et Flask pour découvrir des fonctionnalités plus avancées, telles que les filtres, les macros et l'héritage de templates, afin de créer des applications web encore plus puissantes et flexibles.