PythonでWebアプリを作ってみよう！
全4回で学ぶFlask入門講座
2回目

2025/10/07

Kazuma Sekiguchi

Jinja2

- Pythonで利用可能なHTMLテンプレートエンジン
  - Flaskにおけるデフォルトのテンプレートエンジンになっている
    - 別にインストールする必要は無い
  - ほとんどをHTMLで記述しておいて、必要なところだけPythonの変数などを指定して、表示させることが可能
  - HTMLとPythonを組み合わせて利用することが可能
- テンプレート内で制御文を利用することが可能
  - if文などを使って、表示、非表示を切り替えたり、表示する内容を変えたりできる

Jinja2を使う

- defでreturnにrender_template('HTMLファイル名')としてJinja2のテンプレートを指定する
  - テンプレートファイルはtemplatesフォルダー内に格納する
  - 拡張子はHTMLにしたファイルを格納しておく
    - 実際には内部でJinja2用の記述に一部置換えたりする
- 必要に応じて、Pythonから変数を渡すことが可能
  - 第二引数以降に渡す
  - 複数渡す場合は、カンマで区切りながらパラメータ名＝値の形式で渡す

Jinja2を使う

- 渡された変数を展開する
  - Jinja2のHTML内で{{変数名}}として記述することで展開される
  - ディクショナリ型やリスト型で渡すことも可能

```
words = {
"word1":"1番目に渡す内容",
"word2":"2番目に渡す内容"
}
return render_template('show.html',key=words)
```

{{key.word1}}と{{key.word2}}が展開される
{{key["word1"]}}でも展開できる

Jinja2

- 継承という機能を使うことができる
  - ベースになるテンプレートを作成しておいて、使う部分だけをそのまま利用し、変更したい場所は上書きすることが可能
  - ヘッダーやフッターなど同じように作成するべきものを1つ作成すれば共有して使い回すことが可能になる

継承元

```
<title>{% block title %}タイトル{%
endblock %}</title>
{% block header %}ヘッダー{%
endblock %}
{% block content %}内容{%
endblock %}
```

継承

上書き

```
{% extends "base.html" %}
{% block title %}TOP{% endblock %}
{% block header %}<h1>トップ画面
</h1>{% endblock %}
```

上書き

url_for関数

- url_for関数はFlaskで利用できる関数でURLを生成する
  - routeデコレータで定義された関数名を引数として渡すことでURLを生成することが可能
- url_forはテンプレート側でも利用可能
  - テンプレート上で{{}}内に記述すればOK
  - リンクとして機能させることが可能

```
<a href="{{url_for('関数名') }}">遷移先名</a>
```

値を与えることも可能

```
<a href="{{url_for('関数名',id=1) }}">遷移先名</a>
```

テンプレートでの条件分岐

- テンプレート内でif、elif、elseが利用可能
  - 最後にendifを記述することが必要

```
@app.route('/color')
@app.route('/color/<target>')
 def selectColor(target="colorless"):
   return render_template('color.html',color=target)
```

ルーティングを重ねて
使用することも可能

```
{% if color == 'red' %}
 <p>赤色が選択されました</p>
{% elif color == 'blue' %}
 <p>青色が選択されました</p>
{% elif color == 'yellow' %}
 <p>黄色が選択されました</p>
{% elif color == 'colorless' %}
 <p>色が選択されていません
</p>
{% else %}
 <p>色は選択されていません
</p>
{% endif %}
```

テンプレートでフィルターを利用する

- テンプレート変数に対して適用できる操作のことをフィルターと呼んでいる
  - フィルターを使うことでテンプレート変数を加工することが可能

```
{% filter フィルター名 %}
        :
{% endfilter %}
```

間に挟まれている部分がフィルターの対象になる

- 特定の変数に対して、フィルターを適用することが可能

```
{{ 変数名 | フィルター名 }}
```

フィルターの種類

- フィルターの種類はかなり多い
  - https://jinja.palletsprojects.com/en/3.1.x/templates/#builtin-filters
  - first: リストの最初の要素を返す
  - last：リストの最後の要素を返す
  - join：リストの要素を区切り文字で連結して1つのテキストにまとめる
  - length：リストの要素数を返す
  - random：リストの要素からランダムに取り出す

フィルターの作成

- フィルターは自分で作成することも可能
  - カスタムフィルター
- @app.template_filter(フィルター名）を記述し、関数で作成を行なう
  - returnでフィルターを適用して返す値を指定する

```
@app.template_filter('truncate')
def str_truncate(value, length=10):
    return value[:length] + '...' if len(value) > length else value
```

利用する時はこの名前で呼び出す

10文字を超えたら後を切って…を付ける。超えていないならそのままにする

ちなみに

```
return value[:length] + '...' if len(value) > length else value
```

は三項演算子（条件式）

- A if 条件 else B
  - 条件がTrueのときは A、FalseのときはBを返す

```
if len(value) > length:
    return value[:length] + '...'
else:
    return value
```

これと同じ

エラーハンドリング

- Webページにアクセスしたときにサーバーは HTTPステータスコードを返す
  - 200:OK
  - 301:リダイレクト
  - 403:アクセス不可
  - 404:ファイルが見つからない
  - 500:サーバーエラー
- Flaskではステータスコードに応じて内容を返すことが可能
  - エラーハンドリング

エラーハンドリング

- @app.errorhandlerを利用してステータスコードに応じた内容を返すことが可能
  - エラー内容を取得してメッセージとして返すことも可能
  - returnの末尾にステータスコードを与えることで、ステータスコードを設定することが可能
    - これをしないとエラーなのに200が返ってしまう

```
@app.errorhandler(ステータスコード)
def not_found(error):
  return render_template('error404.html'),404
```

例外での扱い

```
from werkzeug.exceptions import NotFound, Forbidden, InternalServerError

@app.errorhandler(NotFound)
def handle_not_found(e):
    return render_template("error404.html"), 404

@app.errorhandler(Forbidden)
def handle_forbidden(e):
    return render_template("error403.html"), 403

@app.errorhandler(InternalServerError)
def handle_server_error(e):
    return render_template("error500.html"), 500
```

- として記述することも可能
  - 可読性が良い点と、Pythonの標準的な構造に近い利点

例外での扱い

```
from werkzeug.exceptions import HTTPException

@app.errorhandler(HTTPException)
def handle_http_exception(e):
    """全てのHTTP例外を一括処理"""
    return render_template("error.html", code=e.code, message=e.description), e.code
```

のように全てのエラーを1つのファイルとして扱うことも可能

- 1つ1つのステータスコードに合わせて対応する必要が無い

エラー内容

- エラー内容のメッセージはターミナルに表示される
  - 画面に表示させるのであれば、テンプレートに変数として渡す必要がある
  - ほとんどのケースではユーザーに見せる必要は無いため、一般的なエラーメッセージだけ表示させるのが普通

ありがとうございました。

また次回。