PythonでWebアプリを作ってみよう！
全4回で学ぶFlask入門講座
1回目

2025/09/30

Kazuma Sekiguchi

自己紹介

関口和真

株式会社コムセントCTO
Webシステム開発、スマートフォンアプリ制作、
サーバー構築、運用など

スマートフォンを使った動画配信アプリの制作
サーバーサイドシステムの作成
フロントエンド部分の作成
AI周りのいろいろ

目標

- PythonのFlaskを利用してある程度のWebシステムを作成できる

- Pythonの初期文法などは触れません
- Pythonによる統計解析や自然言語処理、AIとの連携などは行ないません

今回のAgenda

- PythonでWebシステムを作成する
- Flaskについて
- Flaskの導入と表示確認
- ルーティングの扱い

PythonでWebシステムを作成する

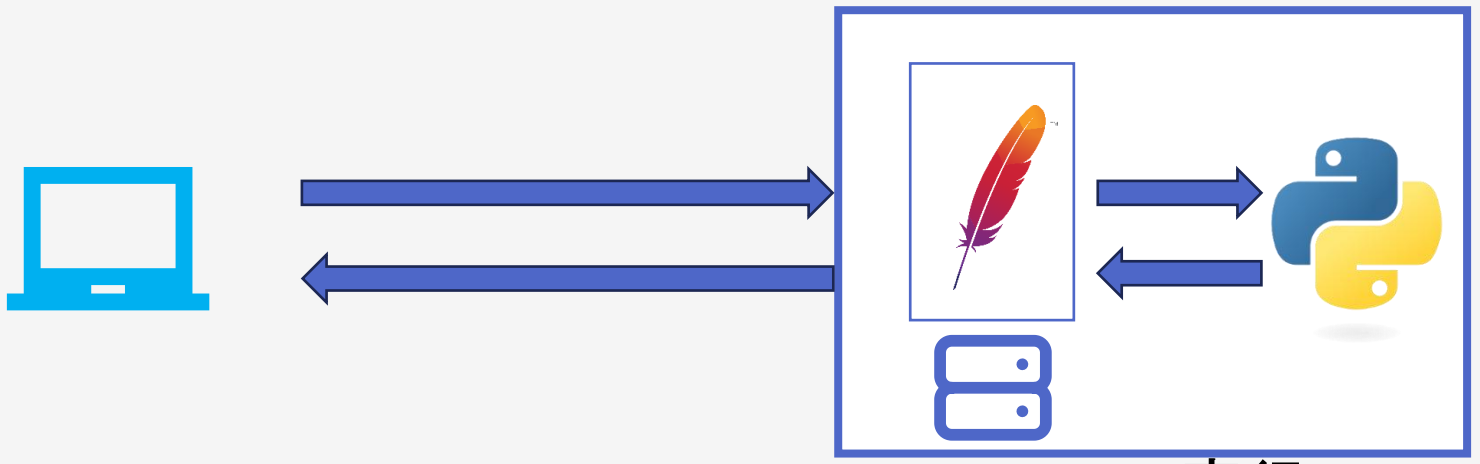
- Webシステムを作成する言語はさまざま存在
  - PHP、Ruby、Java、Go、Rust・・・など
- 他のシステムと連携して動作させたり、特殊な用途のものを作成するのであれば、それに適した言語を使った方が良い
  - 速度重視であればC言語やRustなど
- AIなどのシステムと連携するのであればPythonなどは向いている

PythonでWebシステムを作成する

- Pythonの欠点
  - 実のところ、Pythonは実効速度が遅い
  - 改善されつつあるし、他の仕組みでPythonを高速に動作させるものも存在する
  - そのまま利用した場合、少し速度が他の言語に比べて落ちる
- Pythonの利点
  - 書き方が比較的容易である
  - ライブラリーが豊富に存在するため、上手く活用することで開発コストを抑えることができる

CGIのサポート終了

- CGIの廃止
  - 旧来PythonでWebシステムを作成する場合は、Webサーバーに備わっているCGIという仕組みを利用していた
  - CGIを通じて、Pythonを起動し、Webサーバーが受け取ったデータを渡す
    - Pythonで処理をして貰い、結果を受け取ったらWebサーバーがブラウザーに返す



CGIのサポート終了

- Python3.13からCGIのサポートが無くなる
  - Webシステムを作成する場合、CGI以外の方法でWebサーバーとのやりとりをする必要が生じる
  - CGIは元々、構造的に負荷が高くなる傾向があり、セキュリティリスクが高いため、サポートが終了される次第
- WSGIという仕組みを用いて、Webサーバーとの連携を行なうように変更されている
  - WSGIだとサーバーの負荷が高くなりづらい構造になっている
  - セキュリティの高い環境で動作するようになっている

CGIのサポート終了

- 結果として、パフォーマンスとセキュリティの向上を手に入れることができているが、CGIほど簡易に記述することができなくなった
  - WSGIだと、HTTPレスポンスなどの部分も記述する必要がある

CGIのサポート終了

CGI版

```
#!/usr/bin/env python3

import cgi

print("Content-Type: text/html")
print()
print("<html>")
print("<head>")
print("<title>Hello CGI</title>")
print("</head>")
print("<body>")
print("<h1>Hello, World!</h1>")
print("</body>")
print("</html>")
```

WSGI版

```
def application(environ, start_response):
    status = '200 OK'
    headers = [('Content-Type', 'text/html')]
    start_response(status, headers)

    response_body = """
    <html>
    <head>
    <title>Hello WSGI</title>
    </head>
    <body>
    <h1>Hello, World!</h1>
    </body>
    </html>
    """
    return [response_body.encode('utf-8')]
```

Web用のフレームワークを使う

- PythonでWebシステムを作る場合、フレームワークを利用した方が楽に記述が可能
  - 元々PythonがWebに特化しているわけではないため、フレームワークを利用することで不足しているところを補うことも可能
- フレームワークはアプリケーション開発を簡単にしてくれる仕組み
  - ライブラリーは機能単体として提供されている
  - フレームワークはシステム全体の機能として提供されている
- フレームワークを使う時はそのフレームワークの記述や方法に則って記述する必要がある

Pythonのフレームワーク

- PythonではWeb用のフレームワークがいくつか存在する
  - Django
    - 高機能でさまざまな機能が標準で組み込まれている
    - 高機能ゆえに最初が難しいところがある
    - 大規模サイトで利用されている
    - YoutubeやInstagramで使われているとされる
  - Flask
    - 軽量でシンプルなフレームワーク。必要な機能をプラグインや拡張で追加できる柔軟性がある
    - 機能はあまりない分、比較的簡単に使うことができる
    - Pinterestの初期、Netflix、AirBnBなどで利用されている

Pythonのフレームワーク

- PythonではWeb用のフレームワークがいくつか存在する
  - Pyramid
    - 柔軟で拡張性の高いフレームワーク。シンプルなアプリケーションから複雑なアプリケーションまで対応可能
    - 中規模以上で使われている傾向がある
  - FastAPI
    - WebAPIを作るときに向いているフレームワーク
    - 画面描画を伴うシステムを作る場合は不向き
  - Bottle
    - ほぼ単一のファイルで構成されてているフレームワーク
    - 軽量でシンプルなWebシステムに向いているが、機能があまりない
  - 他にもいろいろなフレームワークが登場している
  - DjangoとFlaskが比較的良く使われている

Flask

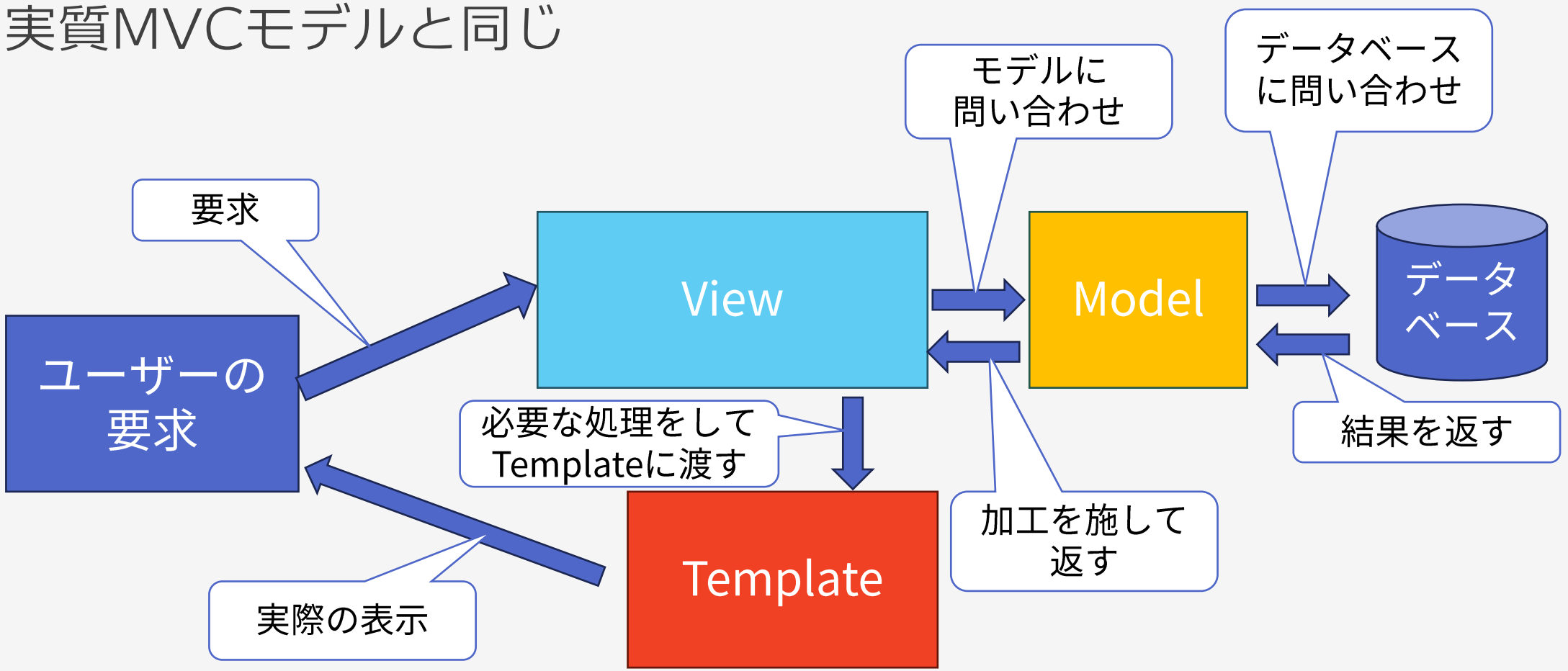
- マイクロフレームワークとも呼ばれていて、標準的に備わっている機能は最小限度に留まる
  - 必要に応じて別途パッケージやライブラリーを追加する必要がある
  - 拡張機能ライブラリーとしてFlask互換のものが提供されている
    - 必要に応じて導入すれば動作する
    - 要らないものは入れる必要が無いため、動作が軽量のままで済む
- 最小限度しか提供されていないため、フレームワークの学習はしやすい

Flask

- 作者はオーストリア人のArmin Ronacher
  - エイプリルフールのジョークとして作ったのだけれど、有名になって、いつのまにか本格的なアプリケーションになってしまった、らしい（wikipediaより）
  - 2010年4月に登場
- 今はFastAPIが人気でもある
  - もっともREST APIに特化しているところがあるため、通常のWebシステムを作るには不適

Flask

- FlaskはMVTモデルを採用している
  - 実質MVCモデルと同じ



FlaskのMVTモデル

- View：ユーザーからのリクエスト（要求）を受け取ってModelやTemplateに処理を渡す
- Model：データベースにアクセスをし、データを保存したり取り出したりする
- Template：HTMLなどの表示用ファイルを作成する実際にデータとして表示される部分を担う

- それぞれのファイルが別々の役割を担うように処理を分離して記述していく

Flaskのインストール

- Flask3.1.2がリリースされているため、これを利用する
- Python3.8以上が必要
- pipでインストールが可能
  - バージョンを指定する点に注意

```
> pip install flask==3.1.2
```

macなどではpip3
と指定する

Flaskの利用

- 先頭でFlaskをimportする
  - 他にライブラリーなどを使う場合もimportを行なう
- 拡張子はpyでOK
  - 別のWebサーバーなどと連携して動作させるのであれば、wsgiなどの拡張子を指定する

```
from flask import Flask
```

Flaskのプログラムを実行する

- Flaskを使ったPythonプログラムを実行する場合は、ターミナルなどからコマンドで実行する

```
> cd 記述したプログラムを保存しているフォルダー
```

- cdコマンドを入力してスペースを入れて、プログラムを保存しているフォルダーを指定してEnterキーで移動する

```
> flask --app app run --port=8080
```

- 上記を入力してEnterキーを押せば、画面に127.0.0.1のURLが表示されるので、そこにブラウザーでアクセスする

Flaskのプログラムを実行する

- 終了するときは、Flaskを実行しているターミナルでCtrl+Cを押す（Windowsの場合。macの場合は、control+cを押す）

- ターミナルを閉じても終了するので、それでもOK

Flaskのプログラムを実行する

```
回目> flask --app app run --port=8080
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deplo
yment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8080
Press CTRL+C to quit
```

- --port=8080は省略可能
  - 記述しない場合5000番として起動する
    - その場合はhttp://127.0.0.1:5000としてアクセスする
  - マレに5000番を利用しているケースがあり、その場合はプログラムが起動しないので、--port=で番号を変更して起動させる
  - --debugを付与すると開発時に便利

ルーティング

- プロウグラム内でdefの上に@app.route('/')を記述している
  - Flaskにおけるルーティングデコレーターであり、特定のURLパスに対して関数を実行するように指示している
  - /へのアクセスが来たら下の関数を実行する

```
@app.route('/')
def main:
#省略

@app.route('/list')
def list:
#省略
```

URLで/listとしてアクセスが来たらdef listを実行する

ルーティング

- 動的ルーティングという機能で、変数を受け取って処理を変えることも可能
  - 変数をURLに付与して、付与された変数に応じて処理内容を変える
  - URL内に変数名または「コンバーター：変数名」を指定することで設定が可能

コンバーター	内容
string	/以外のすべてを受け取る（デフォルトの設定）
int	正の整数を受け取る
float	正の小数を受け取る
path	stringの内容に加えて、/を受け取る

ルーティング

- ルーティングで受け取った値は、関数の引数として受け取ることが可能

```
@app.route('/dynamic/<value>')
def main(value):
  print(f'{value})
#省略

@app.route('/list/<int:number>')
def list(number):
  print(f'{number})
#省略
```

ルーティング

- 与えられた値が指定されたコンバーターと異なっていた場合は、値を受付けずに404表示を返す
  - 特定の形式の値が入ってくる場合は、コンバーターをきちんと設定しておくことで、想定外の値が来ることを防ぐことができる

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

ありがとうございました。

また次回。