Webアプリ開発の基礎を4日間で体系的に学ぶ Django入門 1回目

2026/02/03

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WebアプリケーションフレームワークであるDjangoを利用してDjangoの基礎知識とシステムの作成方法を知る

- 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
- Djangoの深い部分にはあまり触れません（時間的に）

今回のAgenda

- そもそもDjangoとは
- Djangoの役割とMVTとの関係性
- URLルーティング（URLconf）
- 画面表示を行う基本的なViewとTemplate
- 共通レイアウトの使い方（テンプレート継承）

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

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

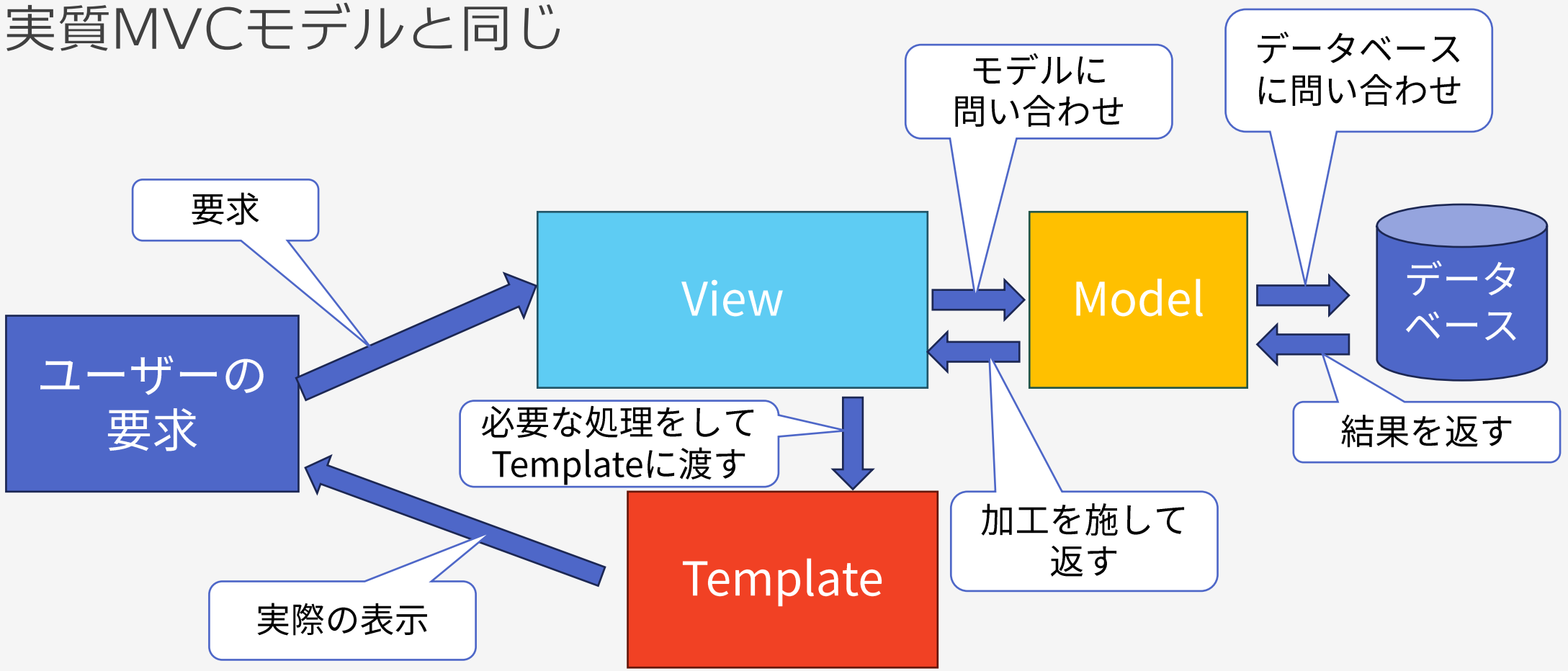
- PythonではWeb用のフレームワークがいくつか存在する
  - Pyramid
    - 柔軟で拡張性の高いフレームワーク。シンプルなアプリケーションから複雑なアプリケーションまで対応可能
    - 中規模以上で使われている傾向がある
  - FastAPI
    - WebAPIを作るときに向いているフレームワーク
    - 画面描画を伴うシステムを作る場合は不向き
  - Bottle
    - ほぼ単一のファイルで構成されてているフレームワーク
    - 軽量でシンプルなWebシステムに向いているが、機能があまりない
  - 他にもいろいろなフレームワークが登場してい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

Django

- Pythonにおける堅牢なフルスタックWebフレームワーク
  - 業務で比較的長く運用するシステム作成に向いている
  - 標準での機能が多くチームで開発しやすい（ブレにくい）
    - 他のPythonフレームワークに比べて機能が多い
  - APIにも対応可能
- 6.0が最新版（2025年12月に登場）
- 業務アプリやCRUD構成が前提のものなどに最適
  - チームで長期的に運用する場合も、設計の型があるのでブレにくい
- 高トラフィックでAPI主導であるなら余り向いていない
  - FastAPIなどの方が向いている

Django

- DjangoはMVTモデルを採用している
  - 実質MVCモデルと同じ



Django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

URLconf

- urls.pyというファイルでURLとViewを紐付ける
  - ルーティング
- URLと処理が一覧化される
  - 処理する単位で整理ができる

ViewとTemplate

- Viewはリクエストを受けてレスポンスを返す関数
  - これ自体は見た目ではない
  - 見た目はTemplate側で対応する
  - データベースからデータを読み出したりすることも可能
- Template
  - DTL（DjangoTemplateLanguage）を用いてテンプレート内に変数を展開したり、制御を行なうことが可能
  - 見た目を制御して、ユーザーに対して実際の表示をおこなう
  - 継承をすることができるため、共通部分などは1つの別ファイルにして共通化させることが可能

Djangoのインストール

- Django6.0.1がリリースされているため、これを利用する
- Python3.12以上が必要
- pipでインストールが可能
  - バージョンを指定しておくと6.0.1が確実に入る
- フォルダーを作成しておき、ターミナルを起動してcdコマンドを利用して移動しておく
  - その上でpipコマンドを実行

```
pip install django==6.0.1
```

macなどではpip3
と指定する

Djangoプロジェクトの作成

- コマンドを利用してプロジェクトを作成する

```
django-admin startproject config .
```

作成するプロジェクト名

現在開いているフォルダーで作成するなら.を付与

- コマンドを実行すると、configというフォルダーが作成されてファイルが自動的に生成される
  - プロジェクト名でconfigは変ではあるが、生成されるファイルが設定に近い物なので、わかりやすいはず
  - manage.pyというファイルが直下に作成されるが、これがDjangoのコマンドファイル

Djangoアプリの作成

- プロジェクトを作成したらpythonのmanage.pyを利用してアプリを作成する

```
python manage.py startapp bbs
```

アプリ名

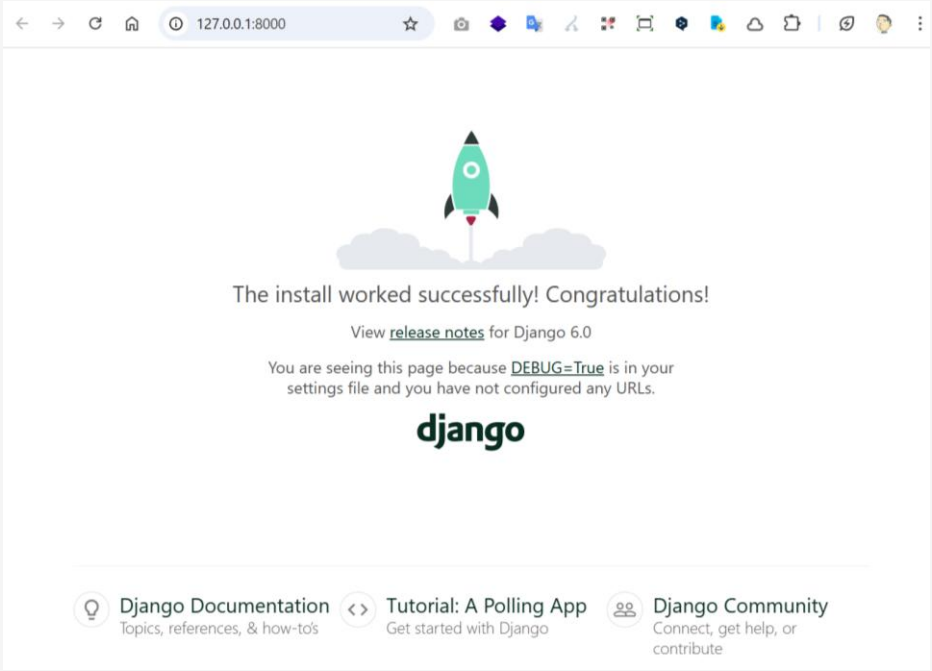
- システムの最低限のファイルが自動的にbbsフォルダー内に作成される

Djangoの実行

- manage.pyに内蔵されているサーバー機能を利用して
  サーバーを立ち上げてブラウザーで表示を確認する
  - migrationを実行しておかないとエラー表示が出るが、
    今のところ気にしなくてOK

```
python manage.py runserver
```

  - 起動するとURLが出るのでそれに
    アクセスするとTop画面が出る
  - 止めるときはCtrl+C（Control+C）



settings.pyファイルの変更

- 基盤となる設定ファイルを変更する
  - 状況に合わせて変更するため、後からも変更することがある
  - デフォルトではデバッグモードがONになっているが、本番運用するときはデバッグモードを必ずOFF（False）設定にすること

処理内容を記述する

- urls.py内にurlpatterns=[]を記述し、中にpath()として処理する先を記述する

```
urlpatterns = [
    path("", views.post_list, name="post_list"),
    path("about/", views.about, name="about"),
]
```

URLとして指定されたもの

処理を行なう関数を指定
この場合は、views.py内にあるabout関数で処理をする

テンプレート内でリンクさせるときに利用する名前を指定

処理内容を記述する

- views.pyに関数を記述する

```
def about(request):
    return render(
        request,
        "bbs/about.html",
        {
            "page_title": "BBS | このサイトについて",
        },
    )
```

関数を定義
引数にはrequestを指定

表示に使うテンプレートを
指定

テンプレートに渡す値を
辞書として定義する

- requestはHttpRequestオブジェクトで、ブラウザから来たリクエスト情報がまとまっている
  - ユーザー情報などを利用するためにそのままテンプレートに渡す
  - 関数内で利用することももちろんできる

テンプレート

- 共通して使うテンプレートを作成しておく

```
{% block content %}{% endblock %}
```

中に記述しておく
他のテンプレートファイルの中身が展開される
contentという名前は自由

```
<link rel="stylesheet"
href="{% static 'css/app.css' %}">
```

CSSファイルの読み込み
staticフォルダーから読み出す場合の記述
先頭に{% load static %}を書いておく

- 中に別のテンプレートを表示するための領域を用意しておく
  - 共通してテンプレートを使うことができる
  - 1つファイルを修正すれば同じテンプレートを使っているところは表示が一気に更新される

各ページのテンプレート

- どのテンプレートを継承して、どこに表示するかのブロックを指定する

```
{% extends "base.html" %}
{% block content %}
  <section class="hero">
    <h1>About</h1>
    <p class="muted">
      BBS は講座用のシンプル掲示板です。第2回で Model/ORM に進みます。
    </p>
  </section>
{% endblock %}
```

base.htmlをベースとなるテンプレートとして使用する

base.htmlにある{% block content %}内に展開する

テンプレート内の制御

- {% %}でいくつかの制御文を利用可能
  - {% for p in posts %}{% endfor %}
    - 辞書の展開、繰り返し
  - {% url ‘about’ %}
    - urls.pyで指定したnameのところへのリンクとして機能する
- 変数は{{ }}内に記述する
  - |default:でデフォルト値を指定することが可能

```
{{ page_title|default:"MiniBoard" }}
```

  - {{ }}内に変数を書くことで自動的にエスケープ処理される

ありがとうございました。

また次回。