1-6. テンプレートの継承とinclude

2019年3月18日　コメントをする

今回のテーマは「テンプレートの継承とinclude」です。現時点でtemplate/base/top.htmlはフッダー、ヘッター、サイドバーとコンテンツを全て有する１つのHTMLファイルとして生成されました。しかし、ヘッダー、フッター、サイドバー等は他のページでも使い回すことが想定されるものです。また、ログインページにはサイドバーを出したくないなど、必要に応じて取り込んだり、外したりする部品もあります。このような要望に対してDjangoテンプレートは継承とインクルードという二つの方法でテンプレートを使い回す方法を提供しています。では、実際に見ていきましょう。

※本ページはstaticファイルを扱うまで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

テンプレートの継承

まず、templates/base/base.htmlを用意し、ここに全ページで共通して使う部品を配置します。他のページではこのbase.htmlを継承して変更部分のみをはめ込むことにします。base.htmlはこんな感じになります。

templates/base/base.html

```
{% load static %}
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="content-language" content="ja">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.
    {% block meta_tag %}{% endblock %}
    <link href="{% static 'css/semantic.css' %}" rel="stylesheet">
    {% block css %}{% endblock %}
    <title>
        {% block title %}IT学習ちゃんねる{% endblock %}
    </title>
</head>
<body>
    <div class="ui stackable inverted menu">
        <div class="header item">
            IT学習ちゃんねる
        </div>
        <a class="item">
            このサイトはなに？
        </a>
        <div class="right menu">
            <a class="item">
                Log in
            </a>
            <a class="item">
                Sign up
            </a>
        </div>
    </div>

    <div class="ui container" style="min-height:100vh;">
        {% block content %}
        {% endblock %}
    </div>
    <div class="ui inverted stackable footer segment">
        <div class="ui container center aligned">
            <div class="ui horizontal inverted small divided link list">
                <a class="item">© 2019 Django学習ちゃんねる(仮)</a>
                <a class="item">利用規約</a>
                <a class="item">プライバシーポリシー</a>
            </div>
        </div>
    </div>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script type="text/javascript" src="{% static 'js/semantic.js' %}"></script>
    {% block js %}{% endblock %}
```

```
</body>
```

{% block hogehoge %}{% endblock %}で囲まれた部分の中身をbase.htmlを継承した各テンプレートファイルで作成していきます。templates/base/top.htmlは次のように変更されました。

templates/base/top.html

```
{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
    <div class="eleven wide column">
        <div class="ui breadcrumb">
            <a class="section">TOP</a>
            <i class="right angle icon divider"></i>
            <a class="section">category</a>
            <i class="right angle icon divider"></i>
            <div class="active section">thread</div>
        </div>
        <div class="ui segment">
            <div class="content">
                <div class="header"><h3>新着スレッド</h3></div>
                <div class="ui divided items">
                    <div class="item">
                        <div class="content">
                            <div class="header">
                                <a><h4>dummy thread</h4></a>
                            </div>
                            <div class="meta">
                                <span class="name">投稿者名</span>
                                <span class="date">2019-2-1 00:00</span>
                            </div>
                        </div>
                    </div>
                    <div class="item">
                        <div class="content">
                            <div class="header">
                                <a><h4>dummy thread</h4></a>
                            </div>
                            <div class="meta">
                                <span class="name">投稿者名</span>
                                <span class="date">2019-2-1 00:00</span>
                            </div>
                        </div>
                    </div>
                    <div class="item">
                        <div class="content">
                            <div class="header">
                                <a><h4>dummy thread</h4></a>
                            </div>
                            <div class="meta">
                                <span class="name">投稿者名</span>
                                <span class="date">2019-2-1 00:00</span>
                            </div>
                        </div>
```

```
                    </div>
                    <div class="item">
                        <div class="content">
                            <div class="header">
                                <a><h4>dummy thread</h4></a>
                            </div>
                            <div class="meta">
                                <span class="name">投稿者名</span>
                                <span class="date">2019-2-1 00:00</span>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

この場合は{% block content %}の中身をtop.htmlで作ってはめ込んでいるんですね。尚、今回{% block meta_tag %}や{% block css %}, {% block js %}等を用意したのはページによって特殊に加えたいMETAタグやCSS,JSが出てくることを想定しているためです。またページタイトルに関しては常にbase.htmlのタイトルをハイフンつなぎで表示する目的でblock.superを使ってbase.htmlのタイトルを呼び出しています。

テンプレートのインクルード

今度はサイドバーも別パート化してみましょう。templates/base/sidebar.htmlとします。
templates/base/sidebar.html

```
<div class="five wide column">
    <div class="ui action input" style="width: 100%;">
        <input type="text" placeholder="検索">
        <button class="ui button"><i class="search icon"></i></button>
    </div>
    <div class="ui segment">
        <div class="content">
            <div class="header"><h4>話題のトピック</h4></div>
            <div class="ui relaxed list small divided link">
                <a class="item">dummy</a>
                <a class="item">dummy</a>
                <a class="item">dummy</a>
                <a class="item">dummy</a>
                <a class="item">dummy</a>
            </div>
        </div>
    </div>
</div>
```

これをtop.htmlの下から２行目に追加して読み込んでみましょう。

templates/base/top.html

```
     </div>
+    {% include 'base/sidebar.html' %}
 </div>
```

そうですね。部分的に作ったsidebar.htmlをインクルードするだけです。これで必要な場合のみインクルードすれば使えるように別部品化できました。

見た目に関してはstaticファイルを扱うから変化していません。ただしテンプレートが部品化されたことでより効率的な開発が出来ると思います。

最後に

Djangoのテンプレートにはまだ便利な機能がたくさんありますが、機能を使う際に紹介していければと考えています。クラスベースビューを使ってテンプレートを表示する方法を見ていきたいと思います。

1-5. staticファイルを扱う

2019年3月15日　3件のコメント

今回のテーマは「staticファイルを扱う」です。ここまででテンプレートファイルをviews.pyで表示することができるようになりました。今度はCSSやJavascript、image等のstaticなファイルを読み込めるようにしましょう。

※本ページはビューとテンプレートの基礎まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

Staticファイルの置き場所

まずはプロジェクト直下にstaticディレクトリを作成します。templateと同じ階層になります。

```
(venv)$ mkdir -p static
```

図示すると以下ような感じです。

```
mysite
||└mysite
|└base
└──static
```

ここではstatic以下にcss,jsディレクトリを作成します。

```
(venv)$ cd static
(venv)$ mkdir -p css
(venv)$ mkdir -p js
```

今回はSimantec UIのcssとjsを導入してみます。今回はlassによるカスタマイズ等はしませんのでダウンロードページからzipファイルをダウンロードしてsemantic.cssとsemantic.jsをそれぞれ先程作成した各ディレクトリに設置します。

ただし、この状態ではまだファイルを読み込むことが出来ません。Djangoにどこにstaticファイルがあるのかを教える必要があります。そこでmysite/settings.pyに設定を追加します。

mysite/settings.py

```
STATIC_URL = '/static/'

+ STATICFILES_DIRS = [
+     os.path.join(BASE_DIR, 'static'),
+ ]
```

CSSを適用したトップページのテンプレート

これでstaticディレクトリをDjangoが認識しました。では先程のCSSとJSをテンプレートに読み込んでみましょう。

template/base/top.htmlは以下のようにします。
template/base/top.html

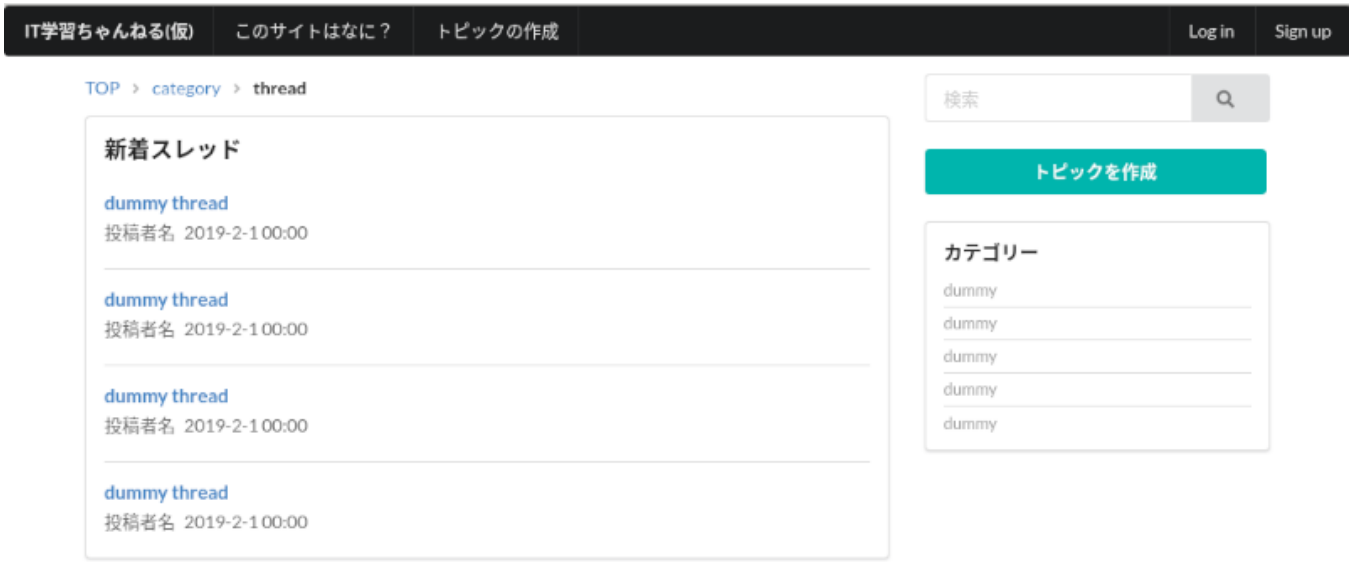
```
{% load static %}
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="content-language" content="ja">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.

    <link href="{% static 'css/semantic.css' %}" rel="stylesheet">
    <title>{{title}}</title>
</head>
<body>
    <div class="ui stackable inverted menu">
        <div class="header item">
            {{title}}
        </div>
        <a class="item">
            このサイトはなに？
        </a>
        <div class="right menu">
            <a class="item">
                Log in
            </a>
            <a class="item">
                Sign up
            </a>
        </div>
    </div>

    <div class="ui container" style="min-height:100vh;">
        <div class="ui grid stackable">
            <div class="eleven wide column">
                <div class="ui breadcrumb">
                    <a class="section">TOP</a>
                    <i class="right angle icon divider"></i>
                    <a class="section">category</a>
                    <i class="right angle icon divider"></i>
                    <div class="active section">thread</div>
                </div>
                <div class="ui segment">
                    <div class="content">
                        <div class="header"><h3>新着スレッド</h3></div>
                        <div class="ui divided items">
                            <div class="item">
                                <div class="content">
                                    <div class="header">
                                        <a><h4>dummy thread</h4></a>
                                    </div>
                                    <div class="meta">
```

```html
                            <span class="name">投稿者名</span>
                            <span class="date">2019-2-1 00:00</span>
                        </div>
                    </div>
                </div>
                <div class="item">
                    <div class="content">
                        <div class="header">
                            <a><h4>dummy thread</h4></a>
                        </div>
                        <div class="meta">
                            <span class="name">投稿者名</span>
                            <span class="date">2019-2-1 00:00</span>
                        </div>
                    </div>
                </div>
                <div class="item">
                    <div class="content">
                        <div class="header">
                            <a><h4>dummy thread</h4></a>
                        </div>
                        <div class="meta">
                            <span class="name">投稿者名</span>
                            <span class="date">2019-2-1 00:00</span>
                        </div>
                    </div>
                </div>
                <div class="item">
                    <div class="content">
                        <div class="header">
                            <a><h4>dummy thread</h4></a>
                        </div>
                        <div class="meta">
                            <span class="name">投稿者名</span>
                            <span class="date">2019-2-1 00:00</span>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
 <div class="five wide column">
    <div class="ui action input" style="width: 100%;">
        <input type="text" placeholder="検索">
        <button class="ui button"><i class="search icon"></i></button>
    </div>
    <div class="ui segment">
        <div class="content">
            <div class="header"><h4>カテゴリー</h4></div>
            <div class="ui relaxed list small divided link">
```

```
                        <a class="item">dummy</a>
                        <a class="item">dummy</a>
                        <a class="item">dummy</a>
                        <a class="item">dummy</a>
                        <a class="item">dummy</a>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="ui inverted stackable footer segment">
        <div class="ui container center aligned">
            <div class="ui horizontal inverted small divided link list">
                <a class="item">© 2019 IT学習ちゃんねる(仮)</a>
                <a class="item">利用規約</a>
                <a class="item">プライバシーポリシー</a>
            </div>
        </div>
    </div>
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script type="text/javascript" src="{% static 'js/semantic.js' %}"></script>
</body>
```

まだ仮のフロントなので仮組みの状態です。キーポイントはファイルの一番最初で行っている{% load static %}です。これによって以降staticをテンプレートで使えるようになります。そしてCSS,JSを読み込む箇所では{% static ‘js/semantic.js’ %}のようにしてファイルを読み込みます。

ブラウザで確認すると下図のように見えるはずです。



Djangoテンプレートは多くのウェブフレームワークのテンプレート同様に条件分岐、繰り返し、フィルター等の機能が使えます。その際には{% %}の間に記述をします。コンテキストでViewから与えられた変数を適用する{{}}と混同しないように注意してくださいね。

最後に

これでCSS、JSがDjangoのテンプレートで読み込めるようになりました。フロントエンドで出来ることも大きく増えると思います。次回はよりテンプレートを効率的に使用する方法を見ていきましょう。

【関連記事】
4-3. DjangoのStaticファイルの扱い

1-4. ビューとテンプレートの基礎

2019年3月13日　5件のコメント

今回のテーマは「ビューとテンプレートの基礎」です。今までは細々と設定作業ばかりでDjangoの面白さを実感出来なかったかも知れません。いよいよ特定のURLに対してHTMLを出力していきますよ。

※本ページはアプリケーションの作成まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

Djangoにおけるビューとテンプレート

DjangoはMVT(Model, View, Template)モデルのフレームワークを自称しています。よく知られているMVCモデルとよく似ていますが、MVCモデルにおけるViewが「情報をどのように見せるのか？」という観点で作られるのに対して、Djangoにおけるviewは「表示すべき情報は何か？」を用意しテンプレートに渡す役割を果たしています。「どのように見せるか？」という出力についてはテンプレートが担います。即ちViewはユーザーのアクションに応じてビジネスロジックを担うモデルに通知し、ユーザーに見せる情報を受け取りテンプレートに渡す仕事をします。詳細は公式ドキュメントを参照下さい。

Hello World

とにかく文字を画面に出しましょう。

base/views.py

```
from django.http import HttpResponse
from django.template import loader

def top(request):
    return HttpResponse('Hello World')
```

このviews.pyに追加したtop関数をURLと結びつけるためbase/urls.pyを修正します。

base/urls.py

```
from django.urls import path
from . import views
name = 'base'

urlpatterns = [
    path('', views.top, name='top'),
]
```

ではブラウザで見てみましょう。

```
(venv)$ ./manage runserver 0.0.0.0:8080
```

ブラウザでlocalhost:8080にアクセスします。Hello Worldが出ましたね。

templatesディレクトリを設定する

テンプレートを使いたいのですが、まずはテンプレートをどこに配置するか決定しましょう。Djangoのデフォルト設定では各アプリケーション内のtemplatesディレクトリから探す設定となっています。
図にすると

```
mysite
|└app1
|  └templates
|    └app1
|      └template.html
|
└app2
  └templates
    └app2
      └template.html
```

というイメージです。一方、テンプレートファイルを１つのtemplatesディレクトリで管理する方法もあります。その場合は

```
mysites
 └templates
   |└app1
   |  └template.html
   └─app2
      └template.html
    ...
```

となります。ディレクトリ構成は好みもありますし、プロジェクトの方針にもよりますが、筆者は後者が好みです。前者の構成はアプリケーションがテンプレートを含んだモジュールとして独立することが念頭にあります。今回は後者の設定で進めます。もし前者のスタイルとする場合は以下の設定は不要です。

mysites/settings.pyのTEMPLATES部分の設定を書き換えます。

```
-        'DIRS': [],
+        'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

templatesディレクトリをプロジェクト直下に作成し、その中にbaseディレクトリを生成しましょう。

```
(venv)$ cd mysite #プロジェクト直下にいない場合は移動する
(venv)$ mkdir -p mysite/templates/base
```

テンプレートを使ってみる

baseアプリケーション用のテンプレートファイルを生成します。
mysite/templates/base/top.html

```
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="content-language" content="ja">
    <title>{{title}}</title>
</head>
<body>
<h1>{{title}}</h1>
<p>テンプレートテスト表示</p>
</body>
```

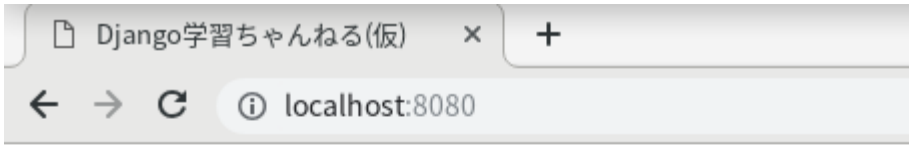
テンプレートを使用するようにbase/views.pyのtop関数も以下のように書き直します。
base/views.py

```
from django.shortcuts import render
from django.http import HttpResponse
from django.template import loader

def top(request):
    template = loader.get_template('base/top.html')
    ctx = {'title': 'Django学習ちゃんねる(仮)'}
    return HttpResponse(template.render(ctx, request))
```

ではブラウザで確認してみましょう。

```
(venv)$ ./manage runserver 0.0.0.0:8080
```

ちなみに毎回runserverを実行することはなく、実行したままソースを書き換えて保存すると自動的に反映されます。(一部settingsに関わるものなどは反映されませんので再起動が必要)



Django学習ちゃんねる(仮)

テンプレートテスト表示

なんとも恥ずかしいタイトルですがスルーしてください。views.pyで渡したtitleパラメータがテンプレートに渡されていることが確認されました。Djangoテンプレートはdict型の変数でパラメータを渡し、テンプレート側では「{{}}」でパラメータ名を囲うことで表示されます。

さて、表示されましたが、テンプレートに表示する度に先程のようにテンプレートを読み込んで表示するのは面倒なのでショートカットが用意されています。先程のviews.pyは以下のように書き換え出来ます。

base/views.py

```
from django.shortcuts import render
# from django.http import HttpResponse
# from django.template import loader

def top(request):
    # template = loader.get_template('base/top.html')
    ctx = {'title': 'Django学習ちゃんねる(仮)'}
    # return HttpResponse(template.render(ctx, request))
    return render(request, 'base/top.html', ctx)
```

loaderもHttpResponseも不要になりスッキリ書けますね。renderを使用すると任意のテンプレートにコンテキスト(テンプレートに変数を渡す入れ物と考えて下さい)を渡して表示することが出来ます。

最後に

現段階ではHTMLを出力するだけでCSSやjavascript等のファイルは読み込んでいません。モダンなWEBアプリにCSS,JSは必須（もはや主役）ですので配置方法を見ていくことにします。

1-3. アプリケーションの作成

2019年3月11日　コメントをする

今回のテーマは「アプリケーションの作成」です。Djangoではプロジェクトの中にアプリケーションという単位のユニットを組み合わせて１つのWEBアプリケーションを作り上げていきます。アプリケーションの作成は何度も行う作業ですので慣れてしまいましょう。

※本ページはデータベースへの接続準備まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

Djangoにおけるアプリケーション

設計に正解はありませんが、Djangoは独立性のある小さなアプリケーションを組み合わせることで１つのWEBアプリケーションを構築することが推奨される風潮があります。この辺りのことはDjango Best Practiceを参考にしてください。独立したアプリケーションを集積するメリットは他のWEBアプリを開発する時に移植性が高い設計ができるという利点があります。どの単位でアプリケーションを区切るかというのは難しい問題ですが、他の機能との相互作用がない独立した機能の場合は１つのアプリケーションとして分けて汎用性の高い設計をしておくと、繰り返す使えるモジュールとして使えるようになります。

Djangoアプリケーションはモデルを有したモジュールであり、配布することも可能です。他のWEBアプリで用いていたアプリケーションをインポートするだけで、そっくり機能を移植することができれば理想的な設計だと思います。

ただ、現実的な問題としてテンプレートが絡む場合には単順な移植が困難なことも多いです。多くのテンプレートはWEBアプリ固有のものが多く、多層構造で依存性が高いためです。残念ながら筆者も抽象度が高く汎用性の高い設計を目指して奮闘中の身であり、多くは語れません。本ブログでも設計が拙い部分は多々あると思いますが、お気づきの点はご指摘、ご指導いただけると幸甚です。

アプリケーションを生成する

今回は掲示板アプリを作成するため”base”という名前のアプリケーションを追加します。このbaseアプリケーションはWEBアプリの基盤としてトップページや利用規約、プライバシーポリシーといった基本的な構成を機能させるためのアプリケーションとなります。

※baseアプリケーションは利用規約等の静的なページ（TOPページは除く）の表示をまとめて行うアプリケーションですが、必ずしもこのようなアプリケーションが必要なわけではありません。

```
$ source venv/bin/activate
(venv)$ cd mysite
(venv)$ ./manage startapp base
```

これでthreadアプリケーションが生成されました。

アプリケーションを組み込む

生成したアプリケーションはプロジェクトに組み込んで使用します。
settings.pyに追記します

```
 INSTALLED_APPS = [
     'django.contrib.admin',
     'django.contrib.auth',
     'django.contrib.contenttypes',
     'django.contrib.sessions',
     'django.contrib.messages',
     'django.contrib.staticfiles',
+    'base',
 ]
```

アプリケーションのURL設定

生成したbaseディレクトリ内に新規ファイルurls.pyを生成しましょう。中身はこうなります。

base/urls.py

```
from django.urls import path

name = 'base'

urlpatterns = []
```

一方でmysiteディレクトリ内のurls.pyにも追記します。

```
- from django.urls import path
+ from django.urls import path, include
+ import base


 urlpatterns = [
     path('admin/', admin.site.urls),
+    path('', include('base.urls'))
 ]
```

これはドメイン直下の部分(http://my.domain.com/ココの部分)がbaseアプリケーションのurls.pyに直結されたことを意味します。この意味は追々分かってくると思います。

最後に

ここまでがアプリケーションの生成と追加でした。次回はこのbaseアプリケーションにビューとテンプレートを追加してDjangoにおける表示について学習します。

1-2. データベースへの接続準備

2019年3月8日　6件のコメント

今回のテーマは「データベースへの接続準備」です。Djangoのデータベースはデフォルトではsqliteです。もしSQLiteを使う想定の場合は本記事の内容は飛ばして結構です。WEBアプリケーションでよく使われるであろうMariaDBもしくはPostgreSQLに接続するための設定について見ていきます。

※本ページはプロジェクトの作成まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

データベースの準備

まずデータベースの準備をします。プロジェクト用のユーザー(postgresqlの場合はROLE)とデータベースを作成します。今回はlocalhostのDBにアクセスする想定で進めます。異なるホストのDBを使用する場合は適宜読み替えて下さい。

MariaDBの場合

```
mysql -u root -p{password}
```

{password}にはrootパスワードを入れて下さい。

```
[MariaDB] CREATE DATABASE forum_data;
[MariaDB] CREATE USER "forum_user"@"localhost" IDENTIFIED BY "{password}";
[MariaDB] GRANT ALL ON forum_data.* TO 'forum_user'@'localhost';
```

{password}にはMariaDBのrootユーザーパスワードを入れて下さい。

Postgresqlの場合

```
$ su postgres
#パスワード入力
postgres$ psql postgres
```

postgresql内で操作します。

```
[postgres] CREATE ROLE forum_user WITH LOGIN CERATEDB PASSWORD '{password}';
[postgres] \connect - forum_user  #ユーザーの切り替え
[postgres] CREATE DATABASE forum_data;
```

標準では認証方式がpeerとなっているためパスワードでログイン出来るようにpg_hba.confファイルを変更します。場所はディストリビューションやバージョンによって異なるので一概に言えないのですが、RedHat系は/usr/local/pgsql/data/以下、Debian系は/etc/{バージョン}/main/以下が多いと思います。Macは…RedHat系と同様だったと思いますが記憶が曖昧です。

pg_hba.conf

```
- local   all         all          peer
+ local   all         postgres     peer
+ local   all         all          md5
```

これでpostgresユーザーのみpeerで他のユーザーはパスワードでログイン可能となりました。

データベースの細かい設定は今回の目的から逸れるので細かく触れません。ご自分の環境に合わせて作成して下さい。尚、ご存知の通りPostgresqlはデフォルト設定でlocalhostからのみアクセス可能な設定となっていますので、外部からアクセスする際は設定を変更して下さい。(postgres.confにて設定)

データベースドライバのインストール

MariaDBの場合

```
(venv)$ pip install mysqlclient
```

Postgresqlの場合

```
(venv)$ pip install psycopg2-binary
```

データベース接続の設定

settigs.pyに以下例のように記述します。
settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'forum_data',
        'USER': 'forum_user',
        'HOST': 'localhost',
        'PASSWORD': '{##your password##}',
    }
}
```

「{##your password##}」の部分はデータベースのパスワードを設定して下さい。Postgresqlの場合はENGINEが'django.db.backends.postgresql'となるだけです。

最後に

これでデータベースに接続する準備が整いました。この後マイグレーションを行い、Django標準のモデルを基にデータベーステーブルを作成するのが一般的ですが、マイグレーションについては別の機会に扱うこととします。

1-1. プロジェクトの作成

2019年3月6日　コメントをする

今回のテーマは「プロジェクトの生成」です。Djangoでは１つのWEBアプリをプロジェクトという単位で管理します。プロジェクトの生成とブラウザでの確認について見ていきましょう。

プロジェクトの生成

まずはプロジェクトを生成しましょう。今回は'mysite'という名前でプロジェクトを作成します。

```
$ source venv/bin/activate
(venv)$ django-admin startproject mysite
```

生成されたmysiteプロジェクトのディレクトリ構成は以下のようになります。

```
.
├── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

言語設定を行う

生成されたばかりのプロジェクトの言語設定は英語ですので、これを日本語にしましょう。myprojectディレクトリのsettings.pyファイルを編集します。

mysite/settings.py

```
- LANG='en_us'
+ LANG='ja'
```

タイムゾーンの設定を行う

タイムゾーンの設定も行っておきましょう。今回は'Asia/Tokyo'を選択します。

mysite/settings.py

```
- TIME_ZONE = 'UTC'
+ TIME_ZONE = 'Asia/Tokyo'
```

ウェブブラウザで確認する

これでmyprojectディレクトリが作成されました。では作成したプロジェクトをウェブブラウザで確認してみましょう。DjangoにはプロジェクトをすぐにWebブラウザで確認できるように開発用サーバーがついています。これはあくまで開発用ですので、決して本番環境では動作させないでくださいね。

```
(venv)$ cd mysite
(venv)$ python manage.py runserver 8080
```

もし8080ポートを他の用途に使用してる場合は8000等、お好みのポートに変更して下さい。では、ウェブブラウザでlocalhost:8080にアクセスして確認してみましょう。図のようなロケットの絵が出ていればOKです。

最後に

次回はデータベースとの接続準備です。一歩ずつ頑張りましょう！

0-2. Django開発環境を整える

2019年3月4日　コメントをする

今回のテーマは「Django開発環境を整える」です。筆者の環境はLinuxですので、Linuxがベースの記事となりますが、Pythonが動く環境であれば大きな変化はないと思いますので適宜読み替えて下さい。ではまずDjangoを使える環境を整えましょう。

python仮想環境の準備

ここでは、Djangoで開発を進めていく際の準備をしていきます。まずpythonですが、3系での利用を想定しています。もし、python2.xしか入っていない状況でしたら3系をインストールして下さい。

次にvenvにて開発用に仮想環境を作ります。本サイトでは基本的にvenv環境で操作をしていきます。

```
$ python3 -m venv venv
```

ディストリビューションによってはpython3へのリンクがpythonだったりpython3だったりと異なりますのでご注意下さい。venvディレクトリが作成されているはずです。コマンドの2つめの'venv'は別のフレーズでもOKです。その場合は指定したフレーズのディレクトリが生成されます。では、作ったvenv環境を有効化しましょう。

```
$ source venv/bin/activate
```

これでvenv環境が有効化されました。

蛇足ですが、以下のコマンドでvenv環境から出ることができます。

```
(venv)$ deactivate
```

Djangoのインストール

ではDjangoをインストールして行きましょう。venv環境のpipを用いるだけなので何も難しいことはありません。

```
(venv)$ pip install django
```

これで準備が整いました。

最後に

次回はDjangoプロジェクトを生成してWebブラウザで確認する作業をしていきます。
本記事がお役に立てば幸いです。