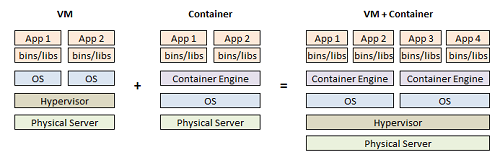
# 仮想環境

## 仮想化基本

コンテナ型

ホストOS状に「コンテナ」という独立空間を形成し，そこに必要なアプリなどをまとめる方式。

ホストOS上のアプリケーションと同レベルの扱い。オーバーヘッドが少なく，リソース・構築・管理の面で非常にシンプル。



ハイパーバイザー

仮想マシンを作成するソフトウェア。物理的なマシン（ホスト）の中に「仮想的なコンピューター（ゲスト）」を作り出す。

ベアメタル型：Xen，Hyper-V，VMware ESX

ホスト型：KVM，VMware Server，VirtualPC

コンテナ型仮想化では，Linuxカーネルの機能によってコンテナごとのユーザ管理やリソース制限を行います。

## VirtualBox

### 概要

仮想マシン１台に付き，

C:\Users\user\_name\VirtualBox VMs\*group\_name*\*user\_name*\*machine\_name*

の設定値フォルダが作成される。

### 仮想マシンの作成

|  |  |  |  |
| --- | --- | --- | --- |
| OS | タイプ | バージョン | 詳細 |
| Ubuntu | Linux | Ubuntu |  |
| Linux Mint | Linux | Ubuntu |  |
|  |  |  |  |

### インストール

以下の２つのどちらかを選択：

・ISOイメージなどをマウントしておき，そのまま仮想マシンを起動 → ブートディスクを選択

・仮想マシン起動 → デバイス → ディスクイメージを選択 → 仮想マシン再起動

Extention Pack(ExtPack)

ライセンスの関係などで，VirtualBox本体に入れられないらしい。

注意点！：VirtualBoxのバージョンと全く同じものをダウンロードする。

ファイル → 環境設定 →　機能拡張 →  → Oracle\_VM\_VirtualBox\_Extension\_Pack-\*.\*.\*.vbox-extpack を選択。

#### 仮想ハードディスク

VDI VirtualBoxネイティブ形式。大体の場合はこれを選べばよい。

VHD Virtual Hard Disk　　Micro SoftのHyper-Vなどで使われる。

VMDK Virtual Machine Disk

#### 共有フォルダ

先にホストOSでディレクトリを作っておく。

Virtual\_Box 設定→共有フォルダ→追加

|  |  |
| --- | --- |
|  |  |
| フォルダのパス | ホストOS上のディレクトリのパス。 |
| フォルダ名 | ゲストOS上での表示名を指定する。 |
| 自動マウント | ｗ |
| 永続化 | Virtual Box を再起動しても，この設定が生きる。 |

20.04.16　マウントが必要？

mount -t vboxsf {共有フォルダ名} {マウント先のディレクトリ名}

<https://qiita.com/SUZUKI_Masaya/items/3444a010cf9903a088b3>

⇒ 但しネットワーク共有フォルダを作成した方が圧倒的に楽。

#### 共有クリップボード

クリップボードを共有したいゲストOSを選択 → 設定 → 高度

クリップボードの共有

⇒ ただし，うまく行った試しがない 20.10.30

### 各種設定

#### USBドライブ

対象の仮想マシン選択 → 設定 → USB → 　でホストOSに接続されたドライブを選択 → ホストOSでは対象デバイスを物理的に取り外す → 仮想マシンを起動 → 仮想マシンが起動完了したら対象デバイスを再度接続

※USB3.0に対応したデバイスを用いる時は，仮想マシンを選択して右クリック → 設定→ USBでUSB3.0(xHCI)コントローラーを選択する必要がある。

#### 画面サイズ変更

「Guest Additionsのインストール」が必要

デバイス → Guest Additions のCDイメージを挿入

#### ネットワーク設定

##### ポートフォワーディング

設定 → ネットワーク → 高度 → ポートフォワーディング

例）httpサーバーのテストをするとき等　80（HTTP），443（HTTPS），22（SSH）を追加。

※特別にポートを変えていない限り，ホストボートとゲストポートは同じで良い

設定アドレスの一例

ローカルアドレス 10.0.2.15/24

ホストオンリーアドレス 192.168.56.1/24

##### ファイルのネットワーク共有

windowsにネットワーク共有フォルダを作成する

ubuntuの場合：ファイル → 他の場所 → サーバーへ接続

アクセス方法はsmb:://192.168.1.1 といった形。

※その際にwindowsアカウント名と，ドメインがあるならドメイン名，パスワードが必要になる。

#### シリアル（comポート）

※Extension Packはインストールが前提。

設定 →  → シリアルポートを有効化

ポートモード

Host Device ホストOS上で使えるシリアルポートをそのまま使わせる。

Disconnected 何も接続されていないポートに見せかける方法

Raw File ホストOS上のファイルをシリアルポートに見せかける方法？？

Host Pile ホストOS上で名前付きパイプをシリアルポートに見せかける方法。高度で，普通使わない。

TCP

パス／アドレス

ゲストOSでどのように表示するか。

（このページの情報元）<https://sgry.jp/blog/2011/04/24/429/>

### その他

#### ラズパイ

TypeをLinux, VersionをDebianにする。

### アンインストール

C:\Users\[*username*] \.VirtualBox

C:\Program Files\Oracle\VirtualBox

ExtentionPackなど

## VMware

仮想化環境のシェアNo1らしいが。

非営利目的ならば無料。ただし機能制限がある。

ソフトウェア名が分かり難い：VMware Workstation Player

使ってみて

・OSの.iso イメージを選択するだけで仮想マシンを作れるのは楽。

・窓サイズ変えるだけで画面解像度も変化する。

・仮想マシンの情報が１フォルダにまとまっていて，分かりやすい。

・USBメモリを差し込んだ時にホスト／ゲストどちらでマウントするか聞いてくれる。気が利いている。

⇒ 全体的にVirtualBox より軽く，使いやすい印象を受けた。

・意味の分からないネットワークの不具合が突発する。再起動で復帰。

各種設定

 → 管理 → 仮想マシンの設定

ホストとゲスト時間の同期

「オプション」タブ → VMwareTools

## Sandbox

Sandboxは一時的な環境として仮想環境を提供する。

ゲストOSはWindowsだけ。

## HyperV

Windows 10エディションに組み込まれているハイパーバイザー。

## Vagrant

Virtual Boxと共に使い、目的ごとにOSやアプリが事前設定されたイメージファイルを扱えるようにしたソフト。

Dockerとともに仮想化2大ソフトとして知られる（2023年現在）

# コンテナ

## Bitnami

ビットナミ

WordPressやRedmineなど、webアプリケーションに必要なphpやmysqlなどのミドルウェアと一緒にパッケージ化してくれるツール。

## Parallels Desktop for Mac

Mac環境での仮想環境ソフト

但しVirtualBox はMacでも動く。

## Docker

### 概要

コンテナ化で最も有名なパッケージ。

コンテナ化：同カーネルで別のディストリビューション環境を提供する。

コンテナはLXCとも呼ばれる。※LXCはLinuxでしか動作しない。

補足

・ホストOSからは１つのプロセスとして認識される。

・1コンテナ１プロセスが原則。

### チュートリアル

#### インストール

先に古いパッケージがあれば削除

sudo apt-get remove docker docker-engine docker.io containerd runc

sudo apt install docker-compose # dockerというのも存在するが，旧バージョンからの移行用らしい

sudo usermod -aG docker username # 自身をdockerグループに入れる。入れない場合は常にsudo dockerが必要

sudo systemctl restart docker # dockerグループに入れた場合はdockerデーモンの再起動が必要らしい

以下の方法が良いらしい

curl -sSL https://get.docker.com | sh #

curl -sSL https://get.docker.com -o hoge.sh # 上のコマンドを補足するとこうなる（hoge.shを実行する）

curl -sSL https://get.docker.com rootless | sh # root権限不要モード

DRY\_RUN=1 sh ./get-docker.sh # 実行内容を確認したい場合(dry run)

公式：<https://matsuand.github.io/docs.docker.jp.onthefly/engine/install/ubuntu/#install-using-the-convenience-script>（Ubuntuの場合）

#### 最初の一歩

UbuntuにCentOSを入れる　という想定のチュートリアル

[ terminal ]

docker pull centos # centosのイメージを取得（デフォルトではDocker Hubから）

docker image ls # イメージのリスト表示　コンテナ名確認の為（任意）

docker run -it -d --name mycentos centos # コンテナの作成

# -it: ターミナルでコンテナを実行できる

# -d: デーモンの意。バックグラウンドでコンテナを実行する

# --name: コンテナに名前をつける

# centos 対象のイメージ名。先程pullした名前。

docker ps # 起動中のコンテナの確認

docker ps -a # 起動していないコンテナも含め表示

docker exec -it mycentos /bin/bash # 実行

#> [root@d7ccbaac3a94 /] となり，centOSのコマンド入力モードとなる。

# exit # CentOSを終了

docker stop mycentos # 停止

docker start mycentos # 起動

docker rm b65d31ac0c3b # コンテナの削除

docker image rm centos # イメージの削除

参考：<https://www.sejuku.net/blog/86137>

#### Dockerfile

Dockerイメージを作成する為のスクリプトのようなもの。

##### 最もシンプルなdockerfile

[dockerfile]

# これだけだとdocker pull python:3.8.2 とほぼ同等

FROM python:3.8.2

注意：dockerfileの中では行の途中からコメントには出来ない

Terminal

docker image build -t django . # 同一ディレクトリにDockerfileを置いて実行

docker image ls # django イメージが確認できる

##### 少し複雑に[python]

[ dockerfile ]

FROM python:3.8.2

ENV PYTHONUNBUFFERED 1

# コンテナ内にcodeディレクトリを作る

RUN mkdir /code

# ワークディレクトリとする

WORKDIR /code

RUN python -m pip install --upgrade pip

ADD requirements.txt /code/

RUN python -m pip install -r requirements.txt

# カレントディレクトリのファイルを/usr/codeにコピー

ADD . /code/

[ terminal ]

docker image build -f ./dockerfile -t mypython:1.0 . # ファイルパスを指定して実行。（-fオプションなしでも可）

docker container run -it mypython:1.0

# もしくは

docker container run -itd --name python-test mypython:1.0 # --nameオプションを省略するとランダムで名前が着く

docker container exec -it python-test python # 最後のpy

docker stop python-test

補足

FROM 元となるイメージを指定。

LABEL メタデータを提供します。

RUN コマンドを実行してパッケージをインストールするために使用。

CMD 実行中のコンテナにコマンドと引数をなげる。CMDは1つだけ。

（参考）[dockerfile](#_dockerfile)

参考：<https://engineer-ninaritai.com/what-dockerfile/>

#### docker-compose(Django)

docker-composeを使ったチュートリアル。Django用のコンテナを作る。

##### ディレクトリ構成

django\

├ db\ # sqlite以外を使う場合

├ sql\ # sqlite以外を使う場合

├ nginx\

│ ├ conf

│ │ └ app\_nginx.conf

│ └ uwsgi\_params

├ python\

│ ├ Dockerfile

│ └ requirements.txt

├ app\

├docker-compose.yml

python\requirements.txt

Django==3.2.12

docker-compose.yml（全体像のみ；詳細は順に後述）

version: ‘3’

services:

nginx\_srv: # あくまでチュートリアルなのでこんな名前だが、nginx、proxy、serverなど名前は変える

db:

python\_srv: # あくまでチュートリアルなのでこんな名前だが、python、web、djangoなど名前は変える

##### Django

注意！実際のdockerfile、docker-compose.ymlは行の途中からのコメント（#）は使えない。

[ python\dockerfile ]

FROM python:3.7-alpine

ENV PYTHONUNBUFFERED 1

RUN mkdir /usr/app # コンテナ内にcodeディレクトリを作る

WORKDIR /usr/app # ワークディレクトリとする

RUN python -m pip install --upgrade pip

ADD requirements.txt /usr/app

RUN python -m pip install -r requirements.txt

ADD . /usr/app

[ docker-compose.yml ]：djangoの部分

python\_srv:

build: ./python # イメージは指定されたパスにあるdockerfileで作成される

container\_name: my\_python # 任意：コンテナに名前を付けてわかりやすくする

restart: always # 任意

depends\_on: # sqlite以外を使う場合、dbにdepends\_on:

- db

volumes:

- "./app:/usr/app/" # 補足：後でchown -R user ./app しておく（root所有になる為）

ports: # ホストport：コンテナport

- "8001:8000"

command: python manage.py runserver 0.0.0.0:8000

コマンドの実行

[ terminal ]

docker-compose run --rm python\_srv django-admin startproject config . # 初回はbuildもする為時間がかかる

docker-compose run --rm python\_srv django-admin startapp my\_app

docker-compose run --rm python\_srv python3 manage.py makemigrations my\_app

docker-compose run --rm python\_srv python3 manage.py migrate

ここまで来たら、下のコマンドでコンテナ上のDjangoが動く

[ terminal ]

docker-compose up

ttp://localhost:8001 などとするとDjangoのWelcomeページが表示される。

##### Nginx

（編集中）

docker-compose.yml：nginxの部分

nginx:

image: nginx

container\_name: my\_nginx # 任意：コンテナに名前を付けてわかりやすくする

restart: always

depends\_on:

- python\_srv

ports:

- "8080:8000"

volumes:

- "./nginx/:/etc/nginx/"

- "staticdata:/opt/apps/static/"

参考）<https://djangobrothers.com/blogs/django_docker/>

##### トラブルシューティング

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

CLI インターフェイス向きではない

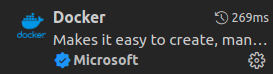
('42000', '[42000] [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]GROUP BY 句の GROUP BY リストに使用される式内では、集計またはサブクエリは使用できません。 (144) (SQLExecDirectW); [42000] [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]GROUP BY 句の GROUP BY リストに使用される式内では、集計またはサブクエリは使用できません。 (144); [42000] [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]ステートメントを準備できませんでした。 (8180)')

#### 空のdockerを基にする

#### 複数の同一サービスの起動

ポート番号を変える事で実現可能？

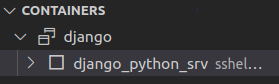
#### VSCodeで使う

インストール



を右クリック → Compose Up

をクリック



上のCONTAINERSのところで作られたコンテナを右クリック

スタート→ ViewLogsで標準出力が見られる。

スタート→ Attach Shell でログインできる。

補足：コンテナ名は、「基底ディレクトリ名\_サービス名」となるらしい。

#### トラブルシューティング

##### no such file or directory

### dockerコマンド

#### 概要

構文：docker サブコマンド [オプション] コンテナ名 コマンド [arg…]

補足：

docker1.13（docker CE 17.03) 以降，docker container，docker images コマンドなどが追加され，ほとんどのサブコマンドはそちらに移行された

（参考；dockerコマンドの新旧）：<https://qiita.com/zembutsu/items/6e1ad18f0d548ce6c266>

#### サブコマンド

##### search

Docker Hubに上がっているイメージの検索を行う

構文：docker search [オプション]

docker search debian # debian（系）のイメージのリストが表示される

オプション

--filter=[] フィルターを使用して結果を出力

docker search --filter "stars=3" --filter "is-official=true" mysql

--no-trunc 出力内容を省略しない

ここでもイメージの情報は得られる：<https://hub.docker.com/search?type=image>

注意：

タグに関してはhtmlで取得するしか無いらしい。

例：https://registry.hub.docker.com/v1/repositories/mysql/tags

##### rename

コンテナの名称を変更

##### info

Dockerの状態の取得

#### 共通オプション

-d コマンドをバックグラウンドで実行（デタッチモード）。 --detach

--detach-keys コンテナーデタッチ時のキーシーケンスをオーバーライド？？？意味不明2022-01-21

-e 環境変数を設定します。--env（ver1.25 以上）

--env-file 環境変数ファイルを読み込みます。（ver1.25 以上）

-i 標準入力を開き続ける。 --interactive

--privileged コマンドに拡張権限を付与。

-t 疑似的なtty（標準入出力）を割り当てる。 --tty

-u ユーザー名または UID。--user

（記述書式t: <name|uid>[:<group|gid>]）

-w コンテナー内部のワーキングディレクトリ。--workdir（ver1.35 以上）

--net

--runtime 実行されるランタイムを指定する

#### ヘルプ

ヘルプはdocker サブコマンド --helpでも出せる

docker exec --help

### docker containerコマンド

#### 概要

dockerコマンドのうちコンテナを管理するサブコマンド群の再編成の為に生まれたコマンド。

#### オプション

#### サブコマンド

##### attach

バックグラウンドで実行中のコンテナにログインする。

##### commit

現在のコンテナの状態からdockerイメージを作成する。

構文：docker container commit [オプション] コンテナ [リポジトリ[:タグ]]

オプション

-a, --author="" 作者 (例 "John Hannibal Smith <hannibal@a-team.com>")

-c, --change=[] イメージをコミット時の Dockerfile 命令を追加指定

--help 使い方を表示

-m, --message="" コミット・メッセージ

-p, --pause=true コンテナをコミット時に一時停止（pause)する

##### cp

コンテナとローカル間でファイルとディレクトリのコピー

##### create

##### diff

##### exec

実行中のコンテナ内で新たなコマンドを実行する。

コンテナにログインする時などに使用する。

docker container exec –it コンテナ名 /bin/bash

##### export

コンテナのファイルシステムをtarアーカイブに保存する

##### inspect

コンテナの詳細情報を表示。IPアドレスを調べたりする。

docker container inspect --format ‘[[.NetworkSettings.Networks.wordpress\_mynet.IPAddress]]’ mycontainer

##### kill

コンテナのプロセスを停止

##### logs

コンテナのログを表示

##### ls

コンテナ一覧の表示

旧：docker ps

##### port

コンテナが使用しているポート情報の表示

##### pause

コンテナのプロセスを一時停止

（反）unpause

docker-compose

概要

docker-compose.yml関連のコマンド。

構文

オプション

サブコマンド

build

up コンテナ（アプリ）の起動

config docker-compose.ymlで書かれてる内容が表示される

関連

dockerfile

##### ps

オプション

-a 停止しているコンテナも含め，全て表示

結果

STATUS

Up 起動中

Exited 休止中

##### rm

コンテナを削除する。

構文：docker rm [コンテナID|コンテナ名]...

コンテナIDはdocker container ps -a などで確認できる。

##### run

コンテナを起動する

構文：docker run [オプション] イメージ [起動時コマンド] [arg1…]

オプション

--name 省略すると適当な名前が付けられる

-v --volume

構文：docker run -v ホスト側の絶対パス[:コンテナ側の絶対パス]:ro [イメージ名] [コマンド]

docker container run -it -d -v /home/user1/dir: /home/user1/dir --name mycentos centos # フルパスで指定

docker container run -it -d -v ~/dir:~/dir --name mycentos centos # コンテナの作成

-p ホスト側[:コンテナ側] ポート番号を割り当てる。（特定のwebサービスとして使う）

--rm コマンド終了時にコンテナが削除される

--tmpfs tmpfsディレクトリをマウントする

docer container run –itd –tmpfs /run

参考：<https://matsuand.github.io/docs.docker.jp.onthefly/storage/tmpfs/>

##### start

停止中のコンテナを起動

（反）stop

docker container stop my\_python

##### stats

コンテナのリソース使用状況を表示

##### top

コンテナが実行しているプロセスを表示

##### prune

docker container prune # docker image pruneコマンドも存在する事に注意

Docker1.13よりまえのdocker rm $(docker ps -a -q)と同等

##### pause

（反）unpause

##### update

コンテナの（リソース）設定を変更

##### wait

コンテナが停止するまで待機

### docker imageコマンド

#### 概要

#### サブコマンド

##### build

又はdocker image build

パスにあるDockerfileなどに従って新しいイメージを構築

構文：docker build [オプション] [パス | URL | - ]

オプション

-f ファイル名 Dockerfileを指定（def.=./Dockerfile’） --file=ファイル名

-t 名前[:タグ] 名前とオプションのタグを指定

docker image build -t mysite:1.0

##### history

Dockerイメージの「レイヤー」の履歴を確認する

##### import

exportしたtarファイルからイメージ作成

（関連）export

##### inspect

イメージの詳細情報を表示

##### ls

旧：docker images

ローカル環境のイメージを一覧表示する。

docker image ls| grep api

##### prune

停止中の全コンテナを一括削除

##### pull

Dockerイメージの取得

構文：docker image pull [オプション] [イメージ名[:タグ] ]

構文：docker image pull [オプション] [ レジストリホスト/Dockerイメージ名 ]

-a 指定したリポジトリ名のDockerイメージを全て入手する（タグ名は任意）

--all-tags=[ true | false ]

--disable-content-trust=[ true | false ]

Dockerイメージ入手時の認証を行わない（true指定時：デフォルトはtrue）

docker image pull centos # centosのイメージを取得

docker image pull debian:jessie # バージョンを指定する場合

docker image pull debian:latest # 最新バージョンを指定

docker image pull myregistry.local:5000/testing/test-image

（関連）push

##### push

Docker Hubへアップロード

（関連）push

##### save

Dockerイメージを tar 形式または標準出力にストリーム

##### load

saveしたtar形式ファイル，または標準入力からイメージを読み込む

##### rm

旧： docker rmi

イメージを削除する。

イメージIDはdocker images で確認できる。

補足：docker rm コンテナを削除　のコマンドも存在するので注意

トラブルシューティング

image has dependent child images

docker image rm $(docker images --filter "dangling=true" -q --no-trunc) # これで消してから　-f を付けても良い

トラブルシューティング

image is being used by stopped containet

docker rm $(docker ps -q -a)

##### tag

Dockerイメージを参照する新しいタグを付ける

### dockerfile

Dockerイメージの作成手順を記す設定ファイル。

これを用意しておくと、以下のコマンドでDockerイメージを作成してくれる。

sudo docker build . # 実行ディレクトリにDockerfileがある場合？

docker build -f /path/to/a/Dockerfile . # ファイルパスを指定して実行する場合。

ディレクトリ構成

test

├ dockerfile

例

FROM ubuntu:18.10

LABEL version=”1.0”

LABEL description=”Dockerのテスト”

RUN apt update

RUN apt install -y apache2

CMD [“apachectl”, “-D”, “FOREGROUND”]

注意：dockerfileの中では行の途中からコメントには出来ない

#### Dockerfileコマンド

##### FROM

##### COPY

##### CMD

イメージ実行時に起動させる主プロセスを設定

##### ENTRYPOINT

##### EXPOSE

##### RUN

##### WORKDIR

詳細

FROM

元となるイメージを指定する。ディストリビューションだけでなく、Dockerイメージも指定できる。

空のイメージである「scratch」を指定する事もできる。

LABEL メタデータを提供。 通常、コンテナ情報を含める。

ENV 永続的な環境変数を設定します。

RUN

コマンドを実行してイメージレイヤを作成。 パッケージをコンテナにインストールするために使用される。

COPY ファイルとディレクトリをコンテナにコピー。

リモートからのファイル追加は出来ない

ADD ファイルとディレクトリをコンテナにコピ。

ローカルの.tarファイルをアンパックできます。

リモートからもファイル追加できる

圧縮ファイルが自動解凍される

CMD

実行中のコンテナにコマンドと引数を提供。 パラメータは上書きできます。 CMDは1つだけ。コマンドを空白で区切ったTuppleのような書き方をする。

WORKDIR 指定ディレクトリを作業ディレクトリに設定。

ARG ビルド時にDockerに渡す変数を定義します。

ENTRYPOINT 実行中のコンテナにコマンドと引数を提供します。 引数は存続します。

EXPOSE 公開用のポート番号を指定する。例 EXPOSE 8080

VOLUME 永続データにアクセスして保存するためのディレクトリマウントポイントを作成。

NODE.jsの情報（割と分かりやすい）：<https://nodejs.org/ja/docs/guides/nodejs-docker-webapp/>

公式情報（英語）<https://docs.docker.com/engine/reference/builder/>

#### .dockerignore

Dockerfileからのイメージのビルド時に対象外とするファイルなどを羅列する。

これにより，実際はイメージのビルドに不要な大容量ファイルなどを省く事ができる。

#### package.json

npm向きの使用ライブラリ一覧。

python（pip）で言えばrequirement.txtみたいなもの。

npm init # 現在のパッケージ状態をpackage.jsonに書き出す？

|  |  |
| --- | --- |
| 項目名 | 内容 |
| name | モジュールの名前。必須項目。 |
| version | モジュールのバージョン。必須項目。nameとversionで一意とする。 |
| private | =trueで非公開に。 |
| description | モジュールの説明。npm searchで表示される。 |
| main | 最初に呼ばれるスクリプトファイルを指定する。 |
| scripts | 任意のスクリプトへのエイリアスを定義できる。 |
| repository | ソースコードが管理されている場所。 |
| homepage | プロジェクトのホームページURL |
| author | 一人だけ指定する |
| contributors | 複数指定する |
| license | ライセンス情報　例：MIT |
| bugs | 問題の情報へのURL。GitHubのIssueのURLを載せるのが一般的らしい。 |

### docker-compose

#### 概要

docker-compose.ymlがあるディレクトリでコンテナ作成の一連のコマンド郡を実行する。

dockerコマンドとは別にインストールする必要がある。

sudo curl -L https://github.com/docker/compose/releases/download/1.16.1/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

#### コマンド

構文：docker-compose [-f <arg>...] [オプション] [サブコマンド] [ARGS...]

オプション

-f ファイル名 docker-compose.ymlのファイルを指定する

-rm

#### サブコマンド

##### run

docker container runと同等。「service」でコマンドを実行する。

docker-compose run --rm python\_srv django-admin startproject config .

##### exec

docker container execと同等。「service」でコンテナを実行する。

docker-compose run --rm python\_srv django-admin startproject config .

##### up

docker-compose.ymlに全servicesでコマンドを実行する。

docker-compose up

オプション

--scale

同じコンテナを複数用意する。（これをスケールと言うらしい）

docker-compose up --scale myservice=3

ポイントは、以下のようにportなどに範囲を持たせる事。

[ dockercompose.yml ]

myservice:

image: nginx:alpine

ports:

- 10080-10089:80

##### down

コンテナを停止。そのコンテナとネットワークを削除。開発環境を一旦クリーンにする。

docker-compose down --rmi all # イメージも削除する場合の例

#### チュートリアル

何度もdocker-composeしてテストする事になる

# upなしの場合

docker-compose up # テスト用

docker container run -it --rm --name hoge-test hoge-test # 起動

docker container stop hoge-test # 停止（バックグラウンドで起動した場合）

# コマンド１つだけ実行して終了する

docker-compose run --rm python django-admin startproject config .

docker-compose log # ログの確認

docker container prune # 停止中のコンテナの全削除

docker-compose down –rmi all # ゼロから作り直す

ディレクトリ構成

├ app

│

├ docker-compose.yml

├ .dockerignore

.dockerignoreの例

# Git

.git

.gitignore

# Docker

.docker

# Python

app/\_\_pycache\_\_/

app/\*/\_\_pycache\_\_/

app/\*/\*/\_\_pycache\_\_/

app/\*/\*/\*/\_\_pycache\_\_/

.env/

.venv/

venv/

# Local PostgreSQL data

data/

#### docker-compose.yml

##### 概要

docker-composeの定義ファイルで、OS環境の為のbatファイルやcmakeといった感じ。

Dockerコンテナ、Dockerイメージ、Dockerボリューム、Dockerネットワークなどを制御できる。

ンテナをサービス化する為のファイルとも言える。

docker-compose.ymlの例

version: "3.0"

services:

my\_app: # 自己のサービスに名前を付け、その下層にその設定を記述

build ./myapp # Dockerfileを用いてイメージ作成。この場合、カレント直下のmyappディレクトリ内

ports:

- 8000:80 # ホストの 8000番とコンテナーの 80番を対応させる

db:

image: mysql

version: Docker Composeのバージョン。マイナーバージョンまで付けるようにする。

2022年2月現在、大きくver3系とver2系が存在し、記述方法が異なる。

なお、dockerのバージョンと対応するcomposeファイルのマイナーバージョンに注意する。

例：

|  |  |
| --- | --- |
| ファイル内version表記 | Docker Engine |
| 3.8 | 19.03.0 以上 |
| 3.7 | 18.06.0 以上 |
| 3.6 | 18.02.0 以上 |

バージョンに関する公式情報：[https://matsuand.github.io/docs.docker.jp.onthefly/compose/compose-file/compose-file-v3/#compose-](https://matsuand.github.io/docs.docker.jp.onthefly/compose/compose-file/compose-file-v3/#compose-と-docker-の互換マトリックス)と-docker-の互換マトリックス

##### build

構文build: パス

Dockerfileがあるディレクトリへのパスを指定する

##### image:

dockerイメージの指定

version: ‘3.7’

services

db:

image: mariadb:latest

##### container\_name:

コンテナ名を明示的に指定

services:

web:

image: nginx:stable-alpine

container\_name: web-container

hostname: web-server

##### environment

環境変数を記述

services:

myapp:

environment:

MODE:debug # MODE = debug と言った形

##### env\_file

ファイルで環境変数を指定

[ docker-compose.env ]

MODE=debug

[ docker-compose.yml ]

services:

myapp:

env\_file: docker-compose.env

##### command:

##### entrypoint

##### depends\_on:

コンテナ内の各サービスの依存関係を示す。

services:

a:

image: busybox

b:

image: busybox

depends\_on:

- a

この場合、起動時はa→bの順に、終了時はb→aの順に終了する。

restart: [always]

tty: コンテナ永続化するか否か [true/false]

##### links:

services:

myapp: # サービス名

links: #

- db:mysql #

？？？

##### networks

##### volumes:

ホスト側のディレクトリと関連付ける？

services:

myapp: # サービス名

volumes: #

- ./app/:/usr/src/ # ホストの./appとコンテナの/srcディレクトリを同期

こんな書き方も見つけた

volumes: # サービス名からのネストでなく、最上層にあるvolumes:

myapp-vol:

driver\_opts:

type:none

device: /data/html # ホスト側のパス？

o: bind

### 詳細

#### ファイルシステムのマウント

以下の３種類がある

|  |  |  |
| --- | --- | --- |
| タイプ | type= |  |
| Volumes | volume | データを保持したいならこれが最良。できるだけこれを使う。 |
| bind mounts | bind | ホストシステムのどこにでも保存できる。 |
| tmpfs mounts | tmpfs | メモリ上にのみ保持。高速。 |

参考：https://qiita.com/y518gaku/items/456f34c317a65a9dae86

##### コマンド

Docker container run -it –mount type=volume,src=vol01,dst=/root/ctdir0002 centos /bin/bash

ホストOSの/var/lib/docker/volumes/vol01/\_data/に保持される

コンテナ内では/root/ctdir0002となる

docker system pruneで削除

readonly 読み込み専用

type=volume,src=vol01,dst=/root/ctdir0003,readonly

#### ラズパイの場合

##### 概要

ラズベリーパイではCPUの違い（armとInterl）から，特に2017年くらいまでは動作させるのにかなり苦労していたらしい。

x\_86のマシンでARM版のイメージを作ってから、イメージをラズパイに転送する方が効率が良い。

buildx？

##### インストール

pip経由ではインストールできないらしい。以下のようにする

curl -sSL https://get.docker.com | sh #

curl -fsSL https://get.docker.com | sh # こんな表記も

curl -fsSL https://get.docker.com rootless | sh # root権限不要モード

# sudo pip3 install docker-compose ？？？なぜpip?

sudo usermod -aG docker pi # ユーザーをdockerグループに追加

sudo gpasswd -a pi docker # これでも良い

参考：https://docs.docker.com/engine/install/ubuntu/

#### 用語

cgroup

### 関連アプリなど

#### Docker Hub

##### 概要

Dockerイメージを管理するオンラインリポジトリ。

<https://hub.docker.com/>

なお、Docker Hub以外にも類似のサービスは存在するらしい

Amazon Elastic Container Registry(ECR)

Google Containerr Registry

Microsoft Azure Container Registry

無料で使うための条件

・非商用で公開

・Open Source Initiativeのオープンソースの定義に合致

・Pullは登録ユーザで 6 時間に 200 回まで，匿名ユーザで 6 時間に 100 回まで。

・6カ月以上使われなかったコンテナイメージは削除する（現在保留中）

##### イメージ

scratch 空っぽのイメージ

alpine Alpine Linux；超軽量Linuxとの事。組み込み系OSをベースにしているらしい。

python 文字通り、pythonの実行環境のイメージ

busybox データ専用コンテナ

##### Portainer

GUIのDockerイメージ、Dockerコンテナの管理ツール。

Portainer自体がDockerイメージとして提供されている為、利用への敷居は低い。

起動したホストのポート9000番にアクセスする。（Webアプリ）

### 具体例

Gatsbyの構築

<https://blog.flan999.com/?p=559>

### その他

Windows

<https://softengineerblog.com/docker-desktop-install/>

RaspberryPi上での起動

以前はarmhfに対応していなかった気がしたが、Bullseye 11，Buster 10では問題なく使えるらしい。

公式doc：<http://docs.docker.jp/v19.03/index.html>

<http://docs.docker.jp/index.html>

環境変数 HTTPS\_PROXY、HTTP\_PROXYプロキシサーバーのURL設定

入門情報：<https://qiita.com/gold-kou/items/44860fbda1a34a001fc1>（あまりまとまってないが、詳しい）

## OpenShift

RedHatが開発するコンテナ。

## Kubernetes

クバーネテス

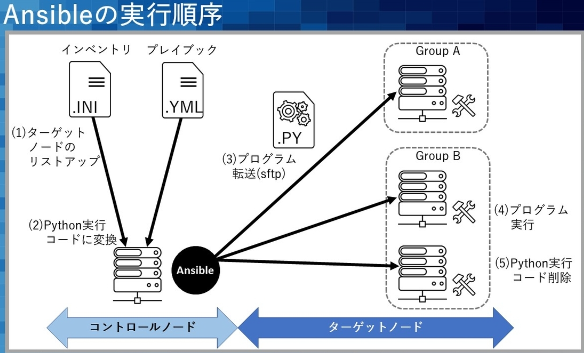
複数のコンテナをまとめた「pod」を、更に複数制御する為のソフトウェア（らしい）

似たようなパッケージにSwarmがある

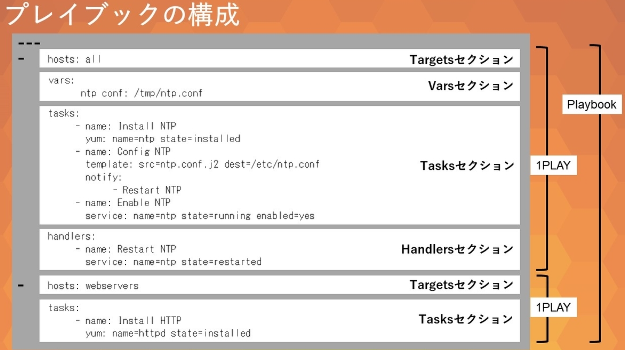
## Ansible

### 概要

IT機器の管理を効率化・自動化するオープンソースの構成管理ツール



### Playbook



### 類似ツール

|  |  |
| --- | --- |
| 構成管理ツール | 概要 |
| Kickstart | 自動インストール（RHEL系） |
| Preseed | 自動インストール（Debian GNU/Linux系） |
| Puppet | Rubyで記述されたサーバ、クライアント型アプリケーション |
| FAI | 自動インストール（Debian GNU/Linux系） |
| Chef | Rubyで記述されたサーバ、クライアント型アプリケーション |

## オーケストレーションその他

weave

docker network

etcd

flannel

# GUI開発

## Glade

### 基本

GladeはGUI部分のみ作成するツールで，イベントハンドラの内容などは別途他のエディタで記述する必用がある。

.gladeファイルからC++やPythonにバインドする（らしい）

.gladeファイルはxml形式のテキストファイルになっている。

### 最初のチュートリアル

#### プロジェクトの作成

（操作）画面左上の  新規作成 → 画面右上の  プロジェクトのプロパティ → ライセンスタブ

→ Name：にプロジェクト名　Description：に説明文などを入れる。

#### コントロールの配置

Gladeのコントロールは，トップレベル，コンテナ，コントロール，ウィジェット　の４種類ある。

特徴として，ウインドウの上には原則1つのパーツしか配置できないので，コンテナを多用する事になる。

まずはトップレベル。

コントロールには一意のIDをつけるようにする。

#### イベント（シグナル）の記述

なお，Gladeではイベントの事を「シグナル」と呼ぶらしい。

（操作）対象のボタンを選択　→　画面の右側のプロパティ画面： 　タブ

試しにclickedシグナルを追加してみる。

（操作）ハンドラーの名前を入力

交換 ？不明 20.10.16

後でよびだす ？不明 20.10.16

なお，シグナルハンドラの記述以降はコンパイル時に -rdynamic オプションが必要になる。

makefileなら

LDFLAGS = `pkg-config --libs gtk+-3.0` -lm -rdynamic

直接コンパイルなら

gcc gtk\_source.c -o my\_app -rdynamic -lm `pkg-config --cflags --libs gtk+-3.0` # まだ良く分かっていない 20.10.06

（参考）<http://utsukemononi.gozaru.jp/gtk/glade001.html>

<https://www.nslabs.jp/glade.rhtml>

#### .cファイルへの展開

簡単な流れ：

.gladeファイルからbuilderオブジェクトを生成 →

#### シグナルハンドラ

#### まとめ

main.c

#include <gtk/gtk.h>

GtkWidget \*mainwindow;

GtkBuilder \*builder;

int main(int argc, char \*argv[])

{

gtk\_init(&argc,&argv);

builder = gtk\_builder\_new\_from\_file("glade\_test.glade");

mainwindow = GTK\_WIDGET(gtk\_builder\_get\_object(builder, "main\_window"));

g\_signal\_connect(mainwindow, "destroy", G\_CALLBACK(gtk\_main\_quit), NULL); // デストラクタ？

gtk\_builder\_connect\_signals(builder, NULL); // シグナルハンドラの関連付け？

gtk\_widget\_show(mainwindow); // ウィジェット（mainwindow）の表示

gtk\_main(); // メインのメッセージループ開始？

return 0;

}

void button1\_clicked\_cb(GtkButton \*button, gpointer user\_data)

{

gtk\_label\_set\_text(GTK\_LABEL(label1), (const gchar\*)"Hello World!");

}

### 補足情報

Glade2だとC言語のソースを出力してくれるらしい。

メモ書き

## Qt SDK

有料版と無料版があり，無料版はLGPLのライセンスによる使用制限がある。

プロジェクトファイルの拡張子は.pro

ウィジェットアプリケーション GUIアプリ。Widgetモジュールが必要。

Qt Quickアプリケーション GUIアプリを作成するもう一つの方法。QtQuickモジュールが必要。

参考

<https://www.kunimiyasoft.com/raspberry_pi_qtcreator/>

<https://qiita.com/soramimi_jp/items/b2b238d1537dbb99abea>

### Windows版

クライアントのWindowsマシンを使って，リモートのLinuxマシン上でのデバッグができる。（らしい）

その情報を残す。

プロジェクトの新規作成

GPIOの使用

シリアル通信の使用

### Raspberry Pi版

ラズパイで開発用SDKを動かす事もできる。

但し，画面サイズやCPU能力から考えるとあまり良い手とは言えない。

#### チュートリアル

インストール

sudo apt-get install qt-sdk qtbase5-dev libgl1-mesa-dev qt5-default –fix-missing

※20.10.06 この中に不要な物があるような気がする。

プロジェクトの新規作成

コンパイラの設定

ビルドディレクトリの修正

一般のビルドディレクトリ。最初は存在しないディレクトリが設定されているので修正。　必要？20.10.06

コンパイラの設定１

Manage Kits ⇒ ビルドと実行 → 〔キット〕タグ → コンパイラ → CとC++共にGCCに。

コンパイラの設定２（Kitを使っている場合必要ない）

Manage Kits ⇒ ビルドと実行 → 〔コンパイラ〕タグ → 追加 → GCC →

C = /usr/bin/gcc C++ = /usr/bin/g++ ※但しこれは最初から設定されている。

GPIOの使用

シリアル通信の使用

### Node-RED

#### 概要

「フロー図」をつなげていくタイプの開発ツール

Node-REDはイベント駆動型・ノンブロッキングモデル

（本家より分かりやすい）<https://blog.secondselection.com/column/how-to-node-red/>

#### 詳細

Node.js（ローカルにWebサーバーを構築）とnpmをインストールする必要がある。

Webブラウザを使って開発を行う。

基本コマンド

node-red-restart # 再起動

#### memo

・htmlサーバー（Webアプリ）も作れるらしい。

#### リファレンス

公式：https://flows.nodered.org/

##### 共通

inject

ソフトウェア上の押しボタンを提供？

##### サードパーティ

Google Home Notifier

トラブルシューティング

Error: get key failed from google

→ package.jsonのgoogle-tts-apiのバージョン情報を書き換える

node-redの場合 ~/.node-red/node\_modules/google-home-notify/package.json に存在

... 略

"license": "MIT",

"dependencies": {

"castv2-client": "^1.2.0",

"google-tts-api": "0.0.6", # ここ

"bluebird": "3.5.1"

}

|  |  |  |
| --- | --- | --- |
| 更新日 | 有効バージョン | 備考 |
| 2020/11/9 | 0.0.5 | 2022-12-18だめだった |
| 2020/12/7 | 0.0.6 |  |

情報元：<https://qiita.com/ktetsuo/items/21f4f85a98145772be3d>

# Web系開発

## Node.js

JavaScript実行環境。htmlサーバー上でなくともJavaScriptが実行できるようになる。

詳細はhtml.docに

## Selenium

Webブラウザの自動化が可能。

対応言語：

Java、Python、C#、Ruby、JavaScript、Kotlin

# Webフレームワーク

## 一覧

|  |  |
| --- | --- |
| 名称 | 概要 |
| React | UIに特化した仕組み。非常によく使われる。 |
| AngularJS | Googleが開発しているJavaScriptのフレームワーク。  フルスタック。学習コストが高め |
| Django | 中・大規模アプリ向け。非常に多くのプラグインがある。 |
| Django REST Framework | 良く分からない |
| Responder | 2021年頃？に登場。  MVCタイプ向き。非同期処理に強い。 |
| FastAPI | 文字通りAPI（RESTful）に特化  pythonでRESTful組むならこれ一択 |
| Flask | pythonでシンプルなWebページを作るのに向く |
| Vue.js |  |
| Ruby on Rails | Webアプリでは圧倒的な人気。  日本で生まれた言語Rubyを使うため，日本語情報が多い。 |
| Zappa |  |
| CakePHP | オープンソースのphpフレームワーク。 |
| BACKBONE.JS |  |
| Laravel |  |
| Swagger |  |

## Django

### 概要

pythonを利用したWebアプリ作成のフレームワークで，比較的大規模なアプリまでサポートする。

バージョンの確認（インストール後）

python.

import django

django.VERSION

manage.pyという管理モジュールでアプリの管理が行える。

例：

python3 manage.py startproject 「アプリ」の開発開始

python3 manage.py shell 専用シェルを起動。

その他情報：

開発時は開発サーバーを利用すれば良いが、１組のリバースプロキシとアプリサーバーを利用する。

例：Nginx とgunicorn

例：Apache2 とmod-wsgi

### チュートリアル

#### インストール

##### インストール(venv)

他のpipパッケージのバージョンがアプリに干渉しない用に、仮想環境を用いる。

基本ファイル構成

この「事前準備」が終わるとファイル構成は以下のようになる

~/myproject # mkdir でディレクトリを作成

+ venv/ # venv コマンドで作成される

+ bin/ # 実際に使用するコマンド群が格納されている

などなど

補足：ドキュメント・ルート（/var/wwwなど）にmyprojectを作ると，Web経由でソースコードが読めてしまう為，ドキュメント・ルート以外の場所に作成する事

仮想環境の作成とアクティベート

sudo apt install python3-venv # 仮想環境を作るpipパッケージのインストール

cd myproject # 作業ディレクトリに移動

~/myproject $ python3 -m venv env # env という名前の仮想環境を作成。

~/myproject $ source env/bin/activate # 仮想環境作成スクリプト実行。

補足

・仮想環境名に空白文字は使えない

・仮想環境：独自のpip環境を構築。プロジェクトごとにpipもジュールを管理し，衝突を防ぐ。

・requirements.txt

複数のpipパッケージのバージョンを完全に一致させる時に使うファイル。

pip freeze > ./requirements.txt # 現在pipで管理しているパッケージのリストを作成

pip install -r ./requirements.txt # テキストファイルからインストール

次ページに続く

Django のインストール

# Django インストール

(myvenv) pip3 install --upgrade pip # 仮想環境内のpipを最新にしておく。

# python3 -m pip install --upgrade pip # 更新はこれでも良い

(myvenv) python3 -m pip install django # Django インストール

(myvenv) pip install django-crispy-forms django-filter # 拡張機能などのインストール（不要かも？）

(myvenv) python3 -m pip install psycopg2 # その他Pythonの必要モジュールをインストール

(myvenv) python3 -m django --version # これがエラーにならなければインストール成功

>>> print(django.get\_version()) # python3でこれを実行しても良い（import djangoが必要）

プロジェクト作成

# プロジェクトの作成

# ※名前は自由だが，config とした方がこのディレクトリの役割と一致する。

(myvenv) django-admin startproject config ． # 最後の「 . (ドット) 」はカレントディレクトリの意。

# manage.py などが作成される。

##### インストール(docker)

より厳密に実行環境を管理する為、dockerを使う場合

ディレクトリ構成

~/myproject

├ app/ Dockerfileで作成する

├ Dockerfile

├ requirements.txt

├ docker-compose.yml

requirements.txt

Django==3.2.12 # バージョン指定は無くても良い

Dockerfile

FROM python:3.8-alpine

ENV PYTHONUNBUFFERED 1 # 環境変数

# 作業ディレクトリの作成；/usrには一般的に使用される共有アプリなどを入れる

RUN mkdir /usr/app

WORKDIR /usr/app # /usr/srcを 作業ディレクトリとする

RUN python -m pip install --upgrade pip

ADD requirements.txt /usr/app/

RUN python -m pip install -r requirements.txt

ADD . /usr/app/ # カレントディレクトリのファイルを/usr/codeにコピー

注意！　実際のDockerfileでは、行の途中からのコメント（#）は使えない。

docker-compose.yml

version: ‘3.7’ # Docker Composeのバージョン

services:

python\_srv: # サービス名。（名前は何でも良い）

build: . # Dockerfileのパスを指定する。この場合カレントディレクトリ。

volumes: # データを永続化させる場合。`host:container`でパスを記載

- ./app/:/usr/app/ # ホストの./appとコンテナの/srcディレクトリを同期

ports:

- 8001:8000 # 開放するポートを指定。（host側：container側）

environment: # 環境変数を指定

- DEBUG=1 # =1ならデバックモード

- SECRET\_KEY=hoge # project作成後、setting.pyのSECRET\_KEYを記入

command: python3 manage.py runserver 0.0.0.0:8000 # サービス起動後に入力されるコマンドを設定

.ymlファイルはpythonと同じく、インデント位置が重要になるので注意

terminal

docker-compose run --rm python\_srv django-admin startproject config . # 初回はbuildを行うため時間がかかる

docker-compose run --rm python\_srv django-admin startapp my\_app

（公式）<http://docs.docker.jp/compose/django.html>

#### データベースの変更

標準ではSQLiteだが，postgresql，MySQL，Oracleにも変更可能。

補足：非公式でSQLServerにも対応

##### sqliteの場合

初期設定でsqliteとなっているが、備忘録的に残す。

[ config/setting.py ]

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'), # sqliteの場合はファイルの絶対パスを指定

},

##### PostgreSQLの場合

先にpostgresqlをインストール。

sudo apt install postgresql # Debian系

必要パッケージのインストール

注意：仮想環境側（terminalの左端に(myenv):~/myproject $と表示している状態）でインストールする。

terminal:

sudo apt install libpq-dev # C言語でつくられたPostgreSQL用のインターフェイス

python3 -m pip install psycopg2 psycopg2-binary # python を使う時に必要。

・設定ファイルの変更

[ config/setting.py ]

DATABASES = {

‘default’: {

‘ENGINE’: ‘django.db.backends.postgresql’,

‘NAME’: ‘postgre-user’, # データベース名

‘USER’: ‘postgre-user’, # ユーザー名

‘PASSWORD’: ‘abcdabcd’, # パスワード認証する場合（任意）

‘HOST’: ‘localhost’, # postgresql のユーザーパスワード（任意）

‘PORT’: ‘‘, # 通信に使うポート番号を指定する。

}

}

##### SQLSeverの場合

SQLServerの場合は少し厄介。

まず、Djangoのバージョンを2.1系に落とさなければならない。（2022年02月現在）

ODBCドライバのインストール

公式資料：<https://docs.microsoft.com/ja-jp/sql/connect/python/pyodbc/python-sql-driver-pyodbc?view=sql-server-ver15>

次に必要なpipパッケージのインストール

python -m pip install pyodbc

python -m pip install django\_pyodbc\_azure

python -m pip install django\_pyodbc\_azure\_2019 # 2022-02-04現在 SQLServer2019に対応した唯一のバージョン

先にSQLServer Mangement Studio

settings.py

DATABASES = {

‘default’: {

‘ENGINE’: ‘sql\_server.pyodbc’,

‘HOST’: ‘localhost\SQLEXPRESS’, # 接続先

‘NAME’: ‘my\_db’, #

‘USER’: ‘username’,

‘PASSWORD’: ‘password’,

‘OPTIONS’: {

‘driver’: ‘ODBC Driver 13 for SQL Server’, # django\_pyodbc\_azure　試してないので詳細不明

‘driver’: ‘ODBC Driver 17 for SQL Server’, # django\_pyodbc\_azure\_2019の場合

},

}

}

python

補足

Microsoftの公式情報ではdjango-mssqlを使えと書いてある。

但しこちらはSQLServer2012，Django1.8，python2.7or3.4のみに対応という、なかなか厳しいもの

参考：<https://pypi.org/project/django-mssql/>

MS：<https://docs.microsoft.com/en-us/samples/azure-samples/mssql-django-samples/mssql-django-samples/>

次ページに続く

～データベースの変更：SQLSeverの場合　つづき

トラブルシューティング

データ ソース名および指定された既定のドライバーが見つかりません

settings.pyを以下のように変更したら接続できた

DATABASES = {

‘default’: {

…略

‘OPTIONS’: {

‘driver’: ‘ODBC Driver 13 for SQL Server’,

↓

‘driver’: ‘ODBC Driver 17 for SQL Server’,

},

}

}

the backend doesn’t support altering from/to

⇒ Identity field（つまりプライマリーキー；pk）のAlter Table(データ型の変更)は不可

Djangoのマイグレーションはpkであるidを先にAutoField（int32[4byte]自動インクリ）で作成し、後でBigAutoField（int64[8byte]）に変換しているらしい

その為、myapp/migrations/\*\*\*.0006\_auto\_\*\*\*\*.py など最後のファイルの実行時にエラーが発生

参考資料：<https://qiita.com/dokura/items/fa330343660661426d3a>

参考：<https://github.com/jazzband/django-oauth-toolkit/pull/615>

解決方法

[ config/settings.py ]

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

解決方法（過去）

・migrationsの.pyファイルを直接書き換える。

0001\_initial.pyで最初からBigAutoFieldを使う。さらに最後の000n\_auto\_\*\*\*\*.pyでの型変換処理を削除（または同ファイルを削除）してmigrate。⇒ 型変換しない形なら何でもよい

⇒ これはmigration毎に行うことになるので**非常にめんどくさい**

・django-oauth-toolkit2を使う

⇒ 時間が惜しいのでまだ調べていない2022-02-04

公式資料？にチュートリアルがある様子

<https://django-oauth-toolkit.readthedocs.io/en/latest/>

##### 共通

変更が終わったらデータベースの変更を適用する（これをマイグレーションと呼ぶ）

python3 manage.py makemigrations myapp #

python3 manage.py migrate #

補足

・Django側でmodelを作ってなくとも（後述），runserverによるサーバー起動の為にmigrateが必要。

サーバーの起動

現時点での起動にはあまり意味がないが，設定が正しいか確認する意味で一度起動する。

(myvenv) python manage.py runserver 8000 # サーバーを起動

あとはWebブラウザでlocalhost:8000 などとすると，djangoが動作している事を確認できる。

Ctrl+Cでサーバーの起動を終了できる。

#### アプリの開始と初期設定

設定ファイルの修正

[ myproject/config/settings.py ]

LANGUAGE\_CODE = 'ja' # 言語の変更（任意）

TIME\_ZONE = 'Asia/Tokyo' # タイムゾーンの変更（任意）

##### アプリを作成

# アプリケーションの作成

(myvenv) python3 manage.py startapp myapp # アプリを作成（ディレクトリが作成される）

[ myproject/config/settings.py ]

INSTALLED\_APPS = [ #

… #

‘django.contrib.messages’, # この項目は最初から存在する。

‘myapp.apps.MyAppConfig’, # 【ここに追加】　新しいアプリの情報を追加※

]

（補足）AppConfig

このファイルはstartappコマンドで自動的に作成される。

また、ここを ‘myapp’ とする情報をよく見かけるが、古いやり方。（但しあまり問題ともされていない）2022-03-09

##### カスタムユーザーclassの作成（任意）

補足

【任意ではあるが、強く推奨】

Django学習の一番最初の時だけ、この工程をスキップしても良い。

あとでマイグレーションできるように、簡単なクラス定義だけしておく

[ myapp/models.py]

from django.contrib.auth.models import AbstractBaseUser, PermissionsMixin

class MyAppUser (AbstractBaseUser , PermissionsMixin):

username = models.CharField(max\_length=150, unique=True)

USERNAME\_FIELD = ‘username’

REQUIRED\_FIELDS = []

raise\_exception # 403エラーを発生させる？

[ config/setting.py]

# どこでも良いが、AUTH\_PASSWORD\_VALIDATORSの上にしておく

AUTH\_USER\_MODEL = ‘myapp.CustomUser’ # myapp.models.CustomUserでは無い点に注意！

※情報の一元化の為に、ここではこの程度とする。

（参照）フレームワーク情報 - カスタムユーザー

ユーザー専用でアプリを作成する場合

[ terminal ]

(myvenv) python3 manage.py startapp users

[ config/settings.py ]

INSTALLED\_APPS = [

‘users.apps.UsersConfig’,

]

（参考）<https://office54.net/python/django/model-custom-user#section3>

#### モデルを作成

##### モデルの作成

[ myapp/models.py ]

from django.db import models

class BookCategory(models.Model): #

class Meta: # 本modelの情報を管理するMetaクラス

db\_talbe = ‘book\_category’ # テーブル名を規定。省略すると appName\_className という名前になる

# フィールド（データベースで言う「列」）の規定

book\_name = models.TextField() # データベースの列名と，その型

price = models. IntegerField(null=True) # 値段を格納する。（null許容）

memo：自動でできるテーブル名におけるアプリ名は大文字小文字区別する？

補足

・フィールド「id」はDjangoにより自動で（**強制的に**）作成される為models.pyでは記述不要。

（なお，idのデータ型はbigserial）

補足：settings.pyのDEFAULT\_AUTO\_FIELDで変更可能。

・作成したモデルの集合インスタンス（リスト）がDjango内で使われる際，変数名の最後にsが付く為，「status」などは避けた方が良い。

⇒ もしくはMetaクラス内でverbose\_name\_plural=‘statuses’ などとする。

・Windows Serverではdb\_table が使えない？？

##### マイグレーション

マイグレーション（Django側からのデータベースの内容を変更）を行う。

今作成したModelをデータベースへ適用する

python manage.py makemigrations myapp # マイグレーション用のファイルが作成される（初回）

python manage.py makemigrations # ファイルが既に存在する場合はこちらでもよい（２回目以降）

python manage.py migrate # データベースに適用する

補足

・makemigrationsが実行されるとmyapp/migrationsに0001\_initial.pyというファイルが作成される。

・作成されるテーブル名はこの場合myapp\_categoryとなる。

・モデルを一切しようしないアプリでも最低一度はマイグレーションが必要

⇒ ValueError("Dependency on app with no migrations") がraiseする

#### ルーティングとビューの作成

##### ルーティング

アプリケーションmyapp専用のルーティング情報ファイルを作成する

terminal

touch myapp/urls.py # デフォルトでは存在しない為，新規作成

ルーティング情報の編集

[ myproject/myapp/urls.py ]（新規作成）

from django.urls import path

from . import views

app\_name= ‘ myapp ‘ # これの追加を忘れないように注意！！

urlpatterns = [

path(‘‘, views.index, name=‘myapp’), ‘

]

プロジェクト全体のルーティング情報config/urls.pyにmyapp/urls.pyを追加する。

[ myproject/config/urls.py ]

from django.contrib import admin

from django.urls import include, path # import対象に「include」を追加

urlpatterns = [

path(‘admin/’, admin.site.urls), # これは最初からある行

path(‘myapp/’, include(‘myapp.urls’)), # 各アプリケーションのルーティングテーブル

]

views.pyを編集して，上記ルーティング情報に対応したビューを，簡易的に，仮で作成。

[ myproject/myapp/views.py ]

from django.http import HttpResponse

def index(request): # urls.pyのviews.index に対応したメソッド

return HttpResponse("Hello, World.")

この状態でrunserverしてWebブラウザでttp://sample.com/myapp/ にアクセスすると，Hello Worldが表示される。（URLはttp://sample.com/myapp/indexでも良い）

##### ビュー（html）の作成

まずはデフォルトでは存在していない必要ファイルを作成

terminal:

mkdir -p templates/myapp # .htmlファイルを格納するディレクトリを作成

touch templates/base.html # 本アプリの共通部分を記述する「親」テンプレートhtml

touch templates/myapp/list.html # 機能の一つを提供する「子」htmlファイル

設定ファイルの修正が必要。（各アプリのディレクトリごとに置く場合は不要）

[ myproject/config/settings.py ]

import os # 一番上の方に追加

（省略）

TEMPLATES = [

‘DIRS’: [os.path.join(BASE\_DIR, ‘templates’], # templatesをmyproject直下に置く場合

# 各アプリのディレクトリごとに置く場合は ‘DIRS’: [], でOK

]

基底ビュー（参考）

[ templates/base.html ]

{% load static %} <!--この行はstaticを使う時のみ -->

<html>

<head>

<title>

{% block page\_title %}{% endblock %}

</title>

<link rel="stylesheet" href="{% static ‘css/styles.css’ %}"> <!--この行はstaticを使う時のみ -->

</head>

<body>

<h1>my project</1>

{% block page\_content %}

{% endblock %}

</body>

</html>

個別ビュー（参考）

[templates/myapp/list.html ]

{% extends ‘base.html’ %} <!-- ルートディレクトリはtemplatesなのでtemplates/base.thml となる -->

{% block page\_title %}My App: List{% endblock %}

{% block page\_content %}

<h2>test</h2>

{% endblock %}

##### ルーティングの再編集

上で作った.htmlファイルのビューへのルーティングを作成

[ myapp/urls.py ]

from django.shortcuts import render, path # renderを追加

from . import views

urlpatterns = [

path(‘‘, views.index, name=‘myapp’), ‘

path(‘list’, views.book\_list, name=‘book\_list’), ‘

]

[ myapp/views.py ]

from django.shortcuts import render # 【追加】

from django.http import HttpResponse

def index(request):

return HttpResponse("Hello, World.")

def book\_list(request): # 【追加】

return render(request, ‘book/list.html’)

補足：この書き方はメソッド式だが，一般的には**クラス式が推奨**されている

参照：<https://docs.djangoproject.com/ja/4.0/topics/class-based-views/generic-display/>

補足：config/urls.pyでテンプレートを使う方法

例として、myproject/templates/index.html を使う場合

[ myproject/config/urls.py ]

from django.contrib import admin

from django.views.generic import TemplateView #【追加】

from django.urls import include, path

urlpatterns = [

path(‘‘, TemplateView.as\_view(template\_name=‘index.html’), name=‘project\_top’), #【追加】

path(‘admin/’, admin.site.urls),

path(‘myapp/’, include(‘myapp.urls’)),

]

##### 静的ファイルの追加（任意）

css（スタイルシート）やjs（Java Script）などの静的ファイルを使用できるように設定する。

必要ディレクトリ，ファイルを作成して設定ファイルを更新する。

terminal

mkdir -p myapp/static/css # スタイルシートを格納するディレクトリ

mkdir myapp/static/js # JavaScriptのファイルを格納するディレクトリ（任意）

touch myapp/static/css/styles.css # メインのスタイルシートファイル（任意）

（補足）アプリディレクトリ（この場合myapp）の下に静的ファイルを格納するのは非推奨。

（詳細）[静的ファイル](#_静的ファイル)

ベースのテンプレート（.htmlファイル）の編集

[ templates/base.html ]

{% load static %} <!-- staticを使う場合は.htmlファイルの上の方に追加 -->

（略）

<head>

<link rel="stylesheet" href="{% static ‘css/styles.css’ %}"> <!--スタイルシートを読み込む -->

</head>

#### サーバーの起動

この時点で一度起動して確認してみる

サーバーの起動に先立ち，設定ファイルを変更

[ setting.py ]

ALLOWED\_HOSTS = [ # サーバー側でリッスンするアドレスを指定

‘127.0.0.1’, # 自局からの接続。

‘.localhost’, # ‘.localhost’ と書くと，127.でなくlocalhostで接続できる

‘192.168.1.1’, # 外からアクセスするアドレスでも記載しておく

]

起動

(myvenv) python manage.py runserver 192.168.1.1:8080 # サーバーを起動

あとはWebブラウザでlocalhost:8080 などとすると，djangoが動作している事を確認できる。

Ctrl+Cでサーバーの起動を終了できる。

#### 実用ページの作成

##### モデルを用いたビュー

[ myapp/views.py ]

from .models import Book # 【追加】

def book\_list(request): # 【追加】

bk\_list = Book.objects.all()

context = { # htmlに投げる引数。慣例的にcontextという名前を使う。

‘context\_msg’ : "message from book\_list", # dict 型　’key’ : value となる。（この行は任意）

‘book\_list’ : bk\_list, # クエリセットAPIから取得した変数を格納

}

return render(request, ‘accounts/top.html’, context) # render の第3引数でDTL（≒html）に渡す

個別ビューの再編集

[ myapp/templates/book/list.html ]

{% extends ‘base.html’ %} <!-- ルートディレクトリはtemplates -->

{% block page\_title %}My App: List{% endblock %}

{% block page\_ontent %}

<h2>{{ context\_msg }} </h2> <!-- views.pyでcontext\_msgを追加した場合はこの様にできる -->

<table>

<tr>

<th>名前</th><th>値段</th>

</tr>

{% for bk in book\_list %}

<tr>

<td>{{ bk.book\_name }}</td>

<td>{{ bk.price }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

補足：

・「フラッシュメッセージ」というものを使う時の変数名が「messages」なので注意する。

・formとform.as\_tableを使うと一発でテーブルにできる。（参考）[テンプレート - フォーム関連](#_フォーム関連)

##### レコードの編集（任意）

forms.py

from django import forms

from .models import AccountRecord

class BookEditForm(forms.ModelForm):

class Meta:

model = Book

fields=["book\_name", " price "]

labels = {

‘book\_name’ : ‘名前’,

‘ price’ : ‘値段’,

}

def \_\_init\_\_(self, \*args, \*\*kwargs): # コンストラクタ

super().\_\_init\_\_(\*args, \*\*kwargs)

views.py

from django.shortcuts import render, redirect

from .forms import BookEditForm

class BookAddView(TemplateView):

def get(self, request, \*args, \*\*kwargs):

context = {

‘form’ : BookEditForm (request.POST)

}

return render(request, ‘gyoumu/project-request/request\_add.html’, context )

def post(self, request, \*args, \*\*kwargs):

name = request.POST["name"]

print(name)

return redirect(reverse('myapp:top')) # 補足：urls.pyからの逆引き（reverse）を基本とすると良い

[ urls.py ]

from django.urls import path

from . import views

app\_name=‘myapp’

urlpatterns = [

path(‘book/add/’, views. BookAddView.as\_view(), name=‘book \_add’),

path(‘book/list/’, views. BookAddView.as\_view(), name=‘book \_list’), # このようなviewがあるとする

}

[ templates/book\_add.html ]

{% extends ‘./base.html’ %}

{% block app %}

<h1>本の登録</h1>

<form action=‘{% url "showUsers" %}’ method=‘POST’ class=‘form’>

{% csrf\_token %}

{{ userForm }}

<button type=‘submit’>登録</button>

<a href=‘{% url "book\_list" %}’>戻る</a>

</form>

{% endblock %}

#### テンプレート（.html）の書き方Tips

##### パンくず

[ base.html ]

{% load static %} <!-- staticを使う場合，この行を追加 -->

<html lang="jp">

<head>

<!-- 省略 -->

</head>

<body>

<ol class="breadcrumb">

{% block breadcrumb\_block %}

{% endblock %}

</ol>

</body>

</html>

対象となるhtml

{% block breadcrumb\_block %}

<li class="breadcrumb-item"><a class="nav-link" href="/">トップ</a></li>

<li class="breadcrumb-item"><a class="nav-link" href="{% url ‘shop:book\_top’%}">ショップ</a></li>

<li class="breadcrumb-item active">本</li>

{% endblock %}

#### ユーザーの追加とログインページ（任意）

##### 概要

##### ログインページの作成

myproject\_url/adminにログインする為の管理者ユーザーの作成

terminal:

python manage.py createsuperuser # 管理者ユーザーの作成。

# ユーザー名，e-mail，パスワードが必要

# > このパスワードは短すぎです　の以下のメッセージ

# > Bypass password validation and create user anyway? 【パスワード認証を無視して強制的にこのパスワードを使う？】

開発用サーバーの起動

terminal:

python manage.py runserver # 管理者ユーザーの作成。

⇒ Webブラウザで プロジェクトのURL/adminにログイン

ここでユーザーとグループが作成できるが、ここで作成できるのは一般ユーザーであり、管理者ユーザーではない。

資料

メイン情報：<https://tutorial.djangogirls.org/ja/how_the_internet_works/>（読み物としても面白い）

サブ情報：<https://qiita.com/okoppe8/items/54eb105c9c94c0960f14>

#### チュートリアルのデータの削除

プロジェクト自体はディレクトリごと削除すればそれでよい

Terminal

rm -rf myproject # ディレクトリごと強制削除

sqllite以外のデータベースを使っている場合は対象のデータベースを削除する。

Terminal:

sudo -u postgres dropdb db\_name # potgrseSQLの場合

補足：この事から考えるに，Djangoは専用のデータベースを使った方が良いかも

#### トラブルシューティング

##### SyntaxError: invalid syntax 　ALLOWED\_HOSTS = [127.0.0.1, localhost]

**‘**127.0.0.1**’**といった形でシングルクォーテーションで囲う必要がある

##### Error loading psycopg2 module: No module named ‘psycopg2’

pipにpsycopg2がインストールされていない。

##### ’static’. Did you forget to register or load this tag?

setting.pyにSTATIC\_URL = ‘/static/’ が無い

##### WARNINGS Auto-created primary key used …, by default 'django.db.models.AutoField'.

概要

解決方法

[ config/settings.py ]

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

#### CSS

staticファイルの置き場所。ファイル名はstyle.cssでなくても可

**myproject**/**myapp**/static/css/style.css

templete の一例

{% load static %}

<html>

<head>

<title>Django Girls blog</title>

<link rel="stylesheet" href="{% static ‘css/style.css’ %}">

</head>

<body>

</body>

</html>

block

**myproject**/**myapp**/templates/**myapp**/base.html

<html>

<head>

<title>Title</title>

<link rel="stylesheet" href="{% static ‘css/style.css’ %}">

</head>

<body>

<div class="content container">

{% block content %}

{% endblock %}

</div>

</body>

</html

templeteの一例

{% extends ‘blog/base.html’ %} <--! これは，どのタグよりも早く書く必要がある -->

{% block content %}

．．．

{% endblock %}

### フレームワーク情報

#### ディレクトリ構成

##### 基本ファイル構成

黄色マーカーは新たに作る必要のあるファイル／ディレクトリ

myproject # ├　などは「けいせん」で出てくる

├─myvenve # 上のステップで作成した仮想環境

├─config #

│ ├ \_\_init\_\_.py #

│ ├ settings.py #

│ ├ urls.py # プロジェクト全体のルート情報。低メンテの為，各appのurlsへの参照としておく。

│ ├ views.py # アプリ間で共有するviewがある場合、このファイルを追加して記述するのが良い。

│ └ wsgi.py #

├─ static / # css やjavaScprit などのファイルを格納。プロジェクト直下が良い。

│ ├ css/ #

│ │ └styles.css # チュートリアルでは myapp.css となっていた。アプリごとに分けた方が良いかも

│ └ js/ # JavaScriptを使う場合

├─ myapp/ #

│ │ \_\_init\_\_.py # フォルダをモジュールとして認識してもらう為のファイル。(Pythonの仕様)

│ │ admin.py # Django管理サイトの設定などが記述されている。

│ │ apps.py #

│ │ filters.py #

│ │ forms.py # 本アプリで使用するフォームを記述。

│ │ models.py # アプリ内データベースで用いるクラスを記述する。

│ │ tests.py # ディレクトリにする事もある

│ │ urls.py # 独自のルート情報を記載する。主にルートをここに記載するとメンテが楽。

│ │ views.py # 実際のメソッドを記述。≒他のMVCのController

│ ├─migrations/ # makemigrationsを実行した結果のファイルが格納される

├ templates # プロジェクト直下（推奨）ここにtemplatesを置く場合はsetting.pyの変更が必要。

│ └myapp # 各app ごとにディレクトリを用意する。

│ ├ base.html #

│ └ などなど #

│ ├ templatetags #

│ │ item\_.py #

│ └ などなど #

├ manage.py # Django での各種操作を行うメソッドの集まり

##### 静的ファイル

場所：myproject/myapp/static

概要

JavaScriptなどの静的ファイルを格納するディレクトリ。Webアプリでは一般にassetsと呼ばれるらしい。

利用する場合はmyapp/setting.pyを以下のように変更する必要がある

[ myapp/settings.py ]

INSTALLED\_APPS = [

‘django.contrib.staticfiles’, # 念のためこれがあるかどうか確認

‘myapp’,

]

TEMPLATES = [

{

‘OPTIONS’ : [

‘context\_processors’ : [

…（略）

‘django.template.context\_processors.static’ # これを追加

],

},

},

]

STATIC\_URL = ‘/static/’ # これは基本的には変更しない

STATICFILES\_DIRS = [ os.path.join(BASE\_DIR, ‘static’) ] # DEBUG=Trueならこれを追加

STATIC\_ROOT = ‘var/www/myproject/ ‘ # DEBUG=Falseならこれを追加

[ テンプレート（.htmlファイル）]

{% load static %} # ベーステンプレートで読込んでいようが，読込みが必要

<img src=”{% static ‘photo\_image.png’ %}” /> # 具体的なファイル選択方法

<img src={% static "photo\_image.png" %} /> # これでも読み込む。（書式は適当？？）

仕様

manage.py runserverを用いて開発用サーバーを使う時は，アプリディレクトリ直下のstaticディレクトリとSTATICFILES\_DIRSに記載のあるディレクトリが静的ファイルの提供元とされる。

本番時（DEBUG=False）にはSTATIC\_ROOTで指定したディレクトリ**のみ**を提供元とする。

補足

STATIC\_ROOTは本番で使うリバースプロキシのフォルダを指定することになる。

STATIC\_ROOT = ‘var/www/{}/static’.format(PROJECT\_NAME)

PROJECT\_NAME = os.path.basename(BASE\_DIR) # 参考 PROJECT\_NAMEの内容

更に補足：一般にWebアプリケーションにおいては以下のような指針がある。

静的ファイルはHtmlサーバーのドキュメントルート下に配置する。

ソースコード、実行ファイルはホームディレクトリに配置する。

##### 本番環境における静的ファイルの配信

<https://docs.djangoproject.com/ja/4.0/howto/static-files/deployment/>

##### myapp/apps.py

以下のようにする事で、設定ファイルとしても使える（らしい）

from django.apps import apps

date\_format = apps.get\_app\_config(‘myapp’).date\_format

デフォルトでの状態

[ apps.py ]

from django.apps import AppConfig

class MyappConfig(AppConfig):

default\_auto\_field = ‘django.db.models.BigAutoField’

name = ‘myapp’

AppConfigクラスのフィールド

name モデルのapp\_labelなどで参照されるアプリ文字列

verbose\_name アプリの表示名。Django管理サイトなどで表示される文字列

#### 設定ファイル

##### settings.py：基本

# hostとして許容するアドレス。自身のアドレス，デプロイ先のアドレス等

ALLOWED\_HOSTS = [ # サーバー側でリッスンするアドレスを指定

‘127.0.0.1’, ‘192.168.1.30’, # ‘.localhost’ と書くと，127.でなくlocalhostで接続できる

‘.pythonanywhere.com’ # DNS名でもアクセスできる

] # ポートを含まない、大文字小文字を無視した比較

TIME\_ZONE = ‘Asia/Tokyo’

LANGUAGE\_CODE = ‘ja’

STATIC\_URL = ‘/static/’ # 静的ファイルにアクセスする際のURL文字列

STATIC\_ROOT = os.path.join(BASE\_DIR, ‘static’) # 本番で使う静的ファイルの配信元。（リスト）

STATICFILES\_DIRS = [ os.path.join(BASE\_DIR, ‘static’) ] # DEBUG時の静的ファイル置き場。「DIR**S**」なので注意

STATICFILES\_STORAGE

STATICFILES\_FINDERS

MEDIA\_URL

MEDIA\_ROOT = os.path.join(BASE\_DIR, ‘media\_root’)

TEMPLATES = [

{

DIRS : [ os.path.join(BASE\_DIR, ‘templates’ ], # 優先テンプレートディレクトリ

‘DIRS’: [os.path.join(BASE\_DIR, ‘templates’], # templatesをmyprojectに直接置く場合

APP\_DIRS : True # 各アプリのディレクトリも探すかどうか

},

]

SECRET\_KEY = # 個別の秘密鍵；startprojectで自動生成される。

AUTH\_USER\_MODEL = ‘myapp.CustomUser’ # カスタムユーザーModelで認証を行う場合

注意点

STATICFILESにはSTATIC\_ROOT以下のディレクトリは選択できない。（らしい）

補足

STATIC\_ROOTは本番で使うリバースプロキシのフォルダを指定することになる。

STATIC\_ROOT = ‘var/www/{}/static’.format(PROJECT\_NAME)

PROJECT\_NAME = os.path.basename(BASE\_DIR) # 参考 PROJECT\_NAMEの内容

##### settings.py：応用

###### 概要

DATABASE = [

‘default’ : {

‘ATOMIC\_REQUESTS’: True, # トランザクションの有効範囲をリクエストの開始から終了までにする

}

]

LOGGING = [ # ロギング設定。

# 書き方は (env\)Lib\site-packages\django\utils\log.pyのDEFAULT\_LOGGINGを参考にする。

# ここに記述すると上記log.pyの各項目を上書きする。（？log.pyは無効化かもしれない 2022-05-09）

# 補足：python の loggingモジュールと全く同じなので、使い方はそちらの資料を参考にする。

# 補足：’loggers’の’level’がINFO までだとSQLクエリまで表示されて正直うっとうしい。

# Django特有の設定があれば記述する予定

]

STATICFILES\_DIRS = [ # 独立した素材があり，それを含めたい場合はこうする

os.path.join(BASE\_DIR, "static"), # BASE\_DIR = /myproject/　つまり /myproject/static を含める

‘/var/www/static/’,

]

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media') # ダウン／アップロードするファイルを格納

MEDIA\_URL = '/media/'

EMAIL\_BACKEND = "django.core.mail.backends.filebased.EmailBackend"

EMAIL\_FILE\_PATH = os.path.join(BASE\_DIR, "sent\_emails")

LOCALE\_PATHS = [ # 他言語化用ファイルのベースディレクトリの指定

BASE\_DIR

]

SITE\_ID = 1 # 複数siteフレームワークを利用する際の識別ID？

DEFAULT\_AUTO\_FIELD = ‘django.db.models.BigAutoField’

補足

\_\_init\_\_.py でディレクトリをモジュール化する事で，設定ファイルを分割する事も可能。

公式：<https://docs.djangoproject.com/ja/4.0/ref/settings/>

参考:本ドキュメント内のライブラリリファレンス　mail.backends

settings.py：

###### AUTHENTICATION\_BACKENDS

##### settings.py：その他

###### settings.pyのAUTH\_PASSWORD\_VALIDATORS

パスワードのバリデーションルールを指定

以下はデフォルトの値

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

公式：<https://docs.djangoproject.com/ja/4.0/topics/auth/passwords/>

##### settings.py：INSTALLED\_APPS

django.contrib.admin 管理（admin）サイト。

django.contrib.auth 認証システム

django.contrib.contenttypes コンテンツタイプフレームワーク

django.contrib.sessions セッションフレームワーク

django.contrib.messages メッセージフレームワーク

django.contrib.staticfiles 静的ファイルの管理フレームワーク

よく使われるのでデフォルトで付属

##### settings.py：ロギング

2022-08-30 書かないつもりだったが、具体例を一つ残す

# ロギング設定

LOGGING = {

'version': 1,

'disable\_existing\_loggers': False,

'formatters': {

'production': {

'format': '%(asctime)s [%(levelname)s] %(process)d %(thread)d '

'%(pathname)s:%(lineno)d %(message)s'

},

'local': {

'format': '%(asctime)s [%(levelname)s] %(process)d %(thread)d '

'%(pathname)s:%(lineno)d %(message)s'

},

},

'handlers': {

'file': {

'level': 'INFO',

'class': 'logging.FileHandler',

'filename': r'C:\python\gyoumu\_mng\log\app.gyoumu.log',

'formatter': 'production',

'encoding': 'utf-8',

},

'stdout': {

'level': 'INFO',

'class': 'logging.StreamHandler',

'formatter': 'local',

},

},

'loggers': {

# 自作アプリケーション全般のログを拾うロガー

'': {

'handlers': ['file'],

'level': 'INFO',

'propagate': False,

},

# Django本体が出すログ全般を拾うロガー

'django': {

'handlers': ['file'],

'level': 'INFO',

'propagate': False,

},

},

}

##### settings.pyの分解

###### 概要

最終的なディレクトリ構成

config/settings

├ base.py 開発時／本番で変わる事のない共通の設定項目

├ local.py 開発時ローカル設定。（チーム開発時の各自ローカル設定）

├ production.py 本番で利用する設定

公式：<https://docs.djangoproject.com/ja/4.0/topics/settings/>

参考：<https://allneko.club/django/separate-settings/>

###### 作業の流れ

・必要ファイルを作成

- configディレクトリ内にsettingsディレクトリを作成

- 今まで使っていたsettings.pyをbase.pyに名前変更してsettingsディレクトリへ移動

- local.py, production.py 本番を作成

・base.pyの内容変更

[base.py]

# BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# ↓変更：settings.pyのディレクトリ化により、階層が1つ増えた為

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))))

# Django ver3.1以降は以下のようにも出来る

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent.parent

補足：ディレクトリ方式に移行する時は、まずsettings.pyをbase.pyに名前変更すると楽

[ local.py ]

from config.settings.base import \* # これでbase.pyに書かれた内容を読み込む

if DEBUG: # こういった方法もある

pass # ダミー。実際にはDEBUG用の設定項目を記述。

[ production.py]

from config.settings.base import \* # local.pyと同じく、base.pyの読み込み。

・manage.py，wsgi.pyの変更

[ manage.py ] 開発用サーバーの設定

# os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", "mysite.settings "

# ↓ 変更

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", "config.settings**.local**"

[ config.wsgi.py ] 本番サーバー用の設定も同じく

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", " config.settings "

# ↓ 変更

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", " config.settings**.production**"

補足 近年のDjangoにはasgi.pyという

補足

以下のようにrunserverコマンドで開く設定ファイルを変更する事もできる

python .\manage.py runserver 0.0.0.0:8080 --settings=config.settings.local

次ページへ続く

～settings.pyの分離　つづき

###### 分離項目の参考例

base.pyに記述すべき項目の参考例

|  |  |
| --- | --- |
| 項目 | 備考など |
| SECRET\_KEY | 本番ではdjango-environを用いた環境変数化が望ましい |
| DEBUG | 開発時= True、本番時=False |
| INSTALLED\_APPS |  |
| MIDDLEWARE |  |
| ROOT\_URLCONF |  |
| TEMPLATES |  |
| WSGI\_APPLICATION |  |
| AUTH\_USER\_MODEL |  |
| AUTH\_PASSWORD\_VALIDATORS |  |
| STATIC\_URL |  |

localまたはproduction.pyに記述すべき項目の参考例

|  |  |
| --- | --- |
| 項目 | 備考など |
| ALLOWED\_HOSTS | DEBUG：['\*'] |
| DEBUG | 開発時= True、本番時=False |
| SECRET\_KEY | 本番ではdjango-environを用いた環境変数化が望ましい |
| DATABASES | ローカル環境ではsqlite3を使った方が楽 |
| LOGGING | ログ設定 |
| EMAIL\_BACKEND | eメール送信機能。開発時はコンソール表示にしたい場合など。 |
| STATICFILES\_DIRS | local.pyのみ |
| STATIC\_ROOT，MEDIA\_ROOT | production.pyのみ  本番環境では/var/www/などに格納される |

static関連はこのようにする方法もある

if DEBUG:

STATICFILES\_DIRS = [ os.path.join(BASE\_DIR, 'static') ]

else:

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

次ページへ続く

##### 非公開が推奨される設定値

|  |  |  |
| --- | --- | --- |
|  | 推奨度 | 推奨される理由 |
| SECRET\_KEY | ◎ |  |
| ALLOWED\_HOSTS | ○ | 関連する公開ドメインの情報を知られてしまう為 |
|  |  |  |

##### 設定の詳細説明

###### SECRET\_KEY

django内部で生成・利用される暗号鍵や署名をさらに安全に扱う為の仕組み

以下の用途に用いられる

・セッションの署名

・パスワードリセットのトークン

・django.core.signingによる暗号化や署名のキー

・CSRFトークン

公式：<https://docs.djangoproject.com/ja/4.2/ref/settings/#secret-key>

#### urls.py

##### 基本

Djangoにおけるルーティング処理の設定を記載するファイル。

（公式情報）<https://docs.djangoproject.com/ja/3.2/ref/templates/builtins/>

URLパターンからリクエストを割り出し，それにマッチしたビュー関数を呼び出す仕組み。

これを「URLディスパッチャ」と呼び，パターンを記述するURL設定ファイルを「URLconf」と呼ぶ。

例えばttp://project\_url/shop/123 というURLではshopアプリのid=123の情報を対応させる，という感じ。

URLconf

[ /config/urls.py ] プロジェクト全体のルーティング情報を記述。

[ /myapp/urls.py ] 特定のアプリに限定したルーティング情報を記述（djangl.urls.includeメソッドを使った場合）。必須では無いものの，config/urls.pyの肥大化を防ぐためにもアプリごとにurls.pyを作成した方が良い。

##### 静的ファイル

from django.contrib.staticfiles.urls import staticfiles\_urlpatterns

#

urlpatterns += staticfiles\_urlpatterns()

DEBUG=False の時でもDjangoフォルダから静的ファイルを提供したい場合

from django.conf.urls.static impoprt static

#

urlpatterns += static(settings.STATIC\_URL, document?root=settings.STATIC\_ROOT

##### 名前空間

urls.pyに記述するapp\_nameの事。

[ myapp/urls.py ]

from django.urls import path

app\_name=‘polls’ # 名前空間の設定。推奨。

urlpatterns = [

path(‘‘, views.index, name=‘index’)

path(‘<int:question\_id>‘, views.detail, name=‘detail’)

]

このような例だと，URLの逆引きの際にpolls:indexやpolls:detailと指定する必要がある。

（app\_nameを付けると**nameの文字列だけでは，このディスパッチ情報には引っかからなくなる**点に注意）

（補足）

URLの逆引きはテンプレート（.htmlファイル）での {% url %} や，views.pyでdjango.urls.reverseなどを使用する際に使う。

##### サンプル

ディレクトリ構成（抜粋）

myproject

├ config

│ ├ urls.py プロジェクト全体のURLConf

├ myapp アプリごとのディレクトリ

│ ├ urls.py アプリ単体のURLConf

│ ├ views.py

├ templates ここにtemplatesを置く場合はsetting.pyの変更が必要。

│ ├ index.html プロジェクトのトップページ

│ ├ myapp 各app ごとにディレクトリを用意する。

│ │ ├ detail.htm l 詳細画面用の.htmlファイル

[ config/urls.py ]

from django.contrib import admin

from django.urls import path, include

from django.views.generic import TemplateView

# トップのurls.pyには名前空間を付けない

urlpatterns = [

path(‘admin/’, admin.site.urls), # 最初からある管理用ページ

path(‘‘, TemplateView.as\_view(template\_name=‘index.html’), name=‘home’) # トップページ

path(‘myapp/’, include(‘myapp.urls’)), # アプリケーションのURLConfを追加

]

[ myapp/urls.py ]

from django.urls import path

app\_name=‘schedule’ # URLディスパッチャに付ける名前空間

path(‘schedule/’, views.schedule\_top, name=‘top’), # myapp/scheduleに対応する。

path(‘schedule/<int:day>‘, views.schedule\_d, name=‘detail’), # 引数付き。myapp/schedule/1などに対応。

]

対応するビューの例

[ view.py ]　（抜粋）

class ScheduleView(View):

def get(self, request, \*args, \*\*kwargs):

return render(request, ‘myapp/detail.html’, {} )

[ myapp/index] （抜粋）

{% url ‘shedule:top’ %} <!-- accountsアプリのurl.pyのname=‘resister’ を取得 -->

<a href="{% url ‘schedule:detail’ day %}">指定日</a> <!-- パラメータ付きのURL　変数 -->

<a href="{% url ‘schedule:detail’ day=1 %}">初日</a> <!-- パラメータ付きのURL　変数＋初期化？ -->

<a href="{% url ‘homel’ %}">トップページ</a> <!-- 名前空間（app\_name）なしの場合 -->

（詳細）[DTL：テンプレートタグ](#_DTL：テンプレートタグ)

リダイレクト

<https://www.codeflow.site/ja/article/django-redirects>

##### URLconf内のパターン

###### 基本

^パターン$ パターンとの完全一致

[^/] スラッシュ以外の任意の1文字

(?P<グループ名>パターン) 正規表現グループ名を付けて参照する（具体例？）

###### 特殊シーケンス

\d 任意の数字。[0-9]に等しい。

\s 任意の英数字およびアンダーバー。[a-zA-Z0-9\_]に等しい

\w 空白文字。[\t\n\r\f\v] と同じ。

###### データ型

以下の型は明示的に指定可能

str

int

slug

uuid

path

（公式）<https://docs.djangoproject.com/ja/4.0/topics/http/urls/#path-converters>

注意点

・URLパターンの左端には / を付けず，右端にはつける。

・クエリ文字に関する記述は不要 ※?keyword=django などurl文字列で記述した引数の事

・path()メソッドなどのname引数の値はURL逆引きにも使われる為，アプリケーションを通して一意の値が望ましい。（重複した場合は後ろのものが優先される）

・app\_name という引数も同様に逆引きなどで使われる。（reverse関数というものがある）

・URLディスパッチャは404エラーなどのルーティングも行う

～urls.py　続き

##### 正規表現

正規表現を扱うためのdjango.urlsのre\_pathを用いる事もできる。

from django.urls import re\_path

urlpatterns = [

re\_path(r’^admin/’, admin.site.urls),

re\_path(r’^$/’, views.index, name=‘index’),

re\_path(r’^accounts/login/’, accounts\_views.login, name=‘accounts\_login’),

re\_path(r’^shop/$’, shop\_views.index, name=‘shop\_index’),

re\_path(r’^shop/(?P<book\_id>\d+)/$’, shop\_views.detail, name=‘shop\_detail’),

]

※rはraw文字列である事を示す。’\\’とr’\’は等しい。

正規表現の特殊文字の例（一部）

|  |  |
| --- | --- |
| 特殊文字 | 意味 |
| ^ | 行頭を表す。前方一致のパターンを構成。 |
| $ | 行頭を表す。前方一致のパターンを構成。 |
| . | 改行以外の任意の文字を表す。 |
| [] | 文字集合。[0-9][abc][a-Z]など |
| \* | 直前にあるパターンを0回以上、できるだけ多く繰り返す。 |
| + | 直前にあるパターンを1回以上繰り返す。 |
| ? | 直前にあるパターンを0回か1回繰り返す。 |
| {} | 直前にあるパターンの繰り返し回数を指定。[0-9]{4} など |
| () | グループを表す |
| \ | 特殊文字をエスケープする。 |

##### 資料

<https://docs.djangoproject.com/ja/4.0/ref/urls/>

#### views.py

##### 概要

urlディスパッチャからリクエストオブジェクトを受け取って，対応するオブジェクトを返す。

##### method式／class式

以下の２通りの方法がある。（当然結果も同じになる）

[ myapp/views.py ] method式

from django.shortcuts import render

def login(request):

if request.method == ‘GET’:

context = {

‘message’ : "Hello World!",

}

return render(request, ‘acounts/login.html’, context)

[ myapp/views.py ] class式

from django.shortcuts import render

from django.views import View

class LoginView(View):

def get(self, request, \*args, \*\*kwargs):

context = {

‘message’ : "Hello World!",

}

return render(request, ‘accounts/login.html’, context )

login = LoginView.as\_view()

# loginメンバを作らずに，urls.pyで path(‘login/’, views.LoginView.**as\_view**(), name=‘login\_view’) などとしても良い

class式は記述量が少し多いが，継承などの恩恵を受けやすく，応用が利きやすい。（推奨）

特にdjango.views.generic.TemplateViewなどの恩恵が受けられるのは大きい。

method式は簡単に書けるが，継承などに対応していない。

##### get()／post()

ビューには複雑なロジックなどを書かず，見通しを良くする

postメソッドの例

def post(self, request, \*args, \*\*kwargs):

context=self.get\_context\_data(\*\*kwargs)

form = BookForm(request.POST)

context['form'] = form

if not form.is\_valid():

return render(request, self.template\_name, context)

title = form.cleaned\_data['title']

form.save()

return redirect(reverse('myapp:project-request\_list'))

##### ２つ以上のsubmitボタンに対応する

def post(self, request, \*args, \*\*kwargs) -> HttpResponse:

# 2つ以上のsubmit ボタンに対応した処理

if "copy\_and\_insert" in self.request.POST:

pass

else:

pass

return redirect(reverse('myapp:top'))

##### Modelインスタンスの取得方法

リクエストからの値の取得

INSERT（追加）を行いたい場合

def post(self, request, \*args, \*\*kwargs):

form = BookEditForm(request.POST) # ModelForm継承オブジェクトを使う場合。

book = form.save(commit=False) # メソッド名が紛らわしいが，CommitせずにModelインスタンスを取得

UPDATE（更新）を行いたい場合

def post(self, request, id, \*args, \*\*kwargs):

book = get\_object\_or\_404(Book, pk=id) # レコードを特定する為のpk; primary keyが必要

# モデルフォームの場合

form = BookEditForm(request.POST, instance=record) # 注意：instanceはModelFormの時のみ有効

（公式資料）<https://docs.djangoproject.com/ja/3.2/ref/models/instances/#how-django-knows-to-update-vs-insert>

補足

get\_object\_or\_404はレコードを確実に特定する必要がある場合／できる場合に使う。

対象レコードがあるかどうか、少しでも疑いがある場合はfilterを使った方が良い。

# この方法だと、存在しない場合は Noneになる

book = Book.objects.filter(pk = id).first()

##### messages

概要

・画面を更新（F5）すると消えるメッセージ

・逆に言うとAjaxでも使わない限り？時間経過で消える事は無い。

[ settings.py ]

MIDDLEWARE = [

‘django.contrib.sessions.middleware.SessionMiddleware’,

‘django.contrib.messages.middleware.MessageMiddleware’

]

MESSAGE\_STORAGE = ‘django.contrib.messages.storage.cookie.CookieStorage’

MESSAGE\_STORAGE = ‘django.contrib.messages.storage.cookie.CookieStorage’

[ views.py ]

from django.contrib import messages

def post (self, request, id, \*args, \*\*kwargs):

form = BookForm(request.POST)

if not form.is\_valid():

messages.info(request, ‘データに誤りがあります。’)

messages.add\_message(request, messages.INFO, "INFO") # こういった情報もみられるが？

[　テンプレート（.htmlファイル）]

{% if messages %}

{% for message in messages %}

<li {% if message.tags%} class="{{ message.tags }}" {% endif%}>{{ message }}</li>

{% endfor %}

{% endif %}

2022-02-25 contextのmessageとはどう違う？

##### views.pyの分割

概要

pythonの通常のモジュールディレクトリ化と同じ方法でよい（らしい）

ディレクトリ構成

myproject #

├ myapp/ #

│ ├ views/ # これを新規作成

│ │ ├ \_\_init\_\_.py ここで同階層の \*.py をすべてimport

│ │ ├ mymodel\_views.py # 分割モジュール１

│ │ ├ others.py # 分割モジュール２

│ ├ urls.py # ルーティングテーブル

myapp/models/mymodel\_view.py

from django.views.generic import TemplateView

from ..forms import MyModelForm # ..（ドット２つ）は非推奨だったような？要確認 2022-02-18

from ..models import MyModel

class MyModelEditView(TemplateView):

def get(self, request, \*args, \*\*kwargs):

pass # あとは普通に記述

myapp/views/\_\_init\_\_.py

from .mymodel\_view import MyModelEditView

from myapp.views.others import BookEditView, ShopAddView # 完全なnamespaceで記述する方法（めんどくさいので非推奨）

補足：

python3.3以降は\_\_init\_\_.pyが無くともディレクトリを１つのパッケージとして見なしてはくれる。

但し、今回のように同階層のモジュールの読込みなどを行いたい場合は必要。

myapp/urls.py

from django.urls import path

from . import views

app\_name=‘myapp’

urlpatterns = [

path(‘‘, views.others.TopView.as\_view(), name=‘index’), # views.others となっている点に注目

path(‘mymodel/’, views.mymodel\_views.MyModelEditView.as\_view(), name=‘project-request\_top’),

]

ちょっと不満がある。views.others.TopView とするところとか

##### Viewのメソッド

※発生順

|  |  |  |
| --- | --- | --- |
|  |  | 備考 |
| get\_form |  | ModelFormを使う際にしか発生しない。 |
| get\_context\_data |  |  |

#### カスタムユーザー

##### 概要

Djangoでログイン機能などを実装する際、カスタムユーザーclassの作成が強く推奨されている。

そのチュートリアル。

通常、以下のどちらかの抽象クラスを継承する

AbstractBaseUser 思いっきりカスタマイズしたい場合

AbstractUser 少しだけ拡張したい場合

補足：AbstractUserはPermissionsMixin継承済み

注意点

管理サイトにログインできるユーザーを作成したい場合には初回のマイグレーションまでにカスタムユーザーclassの作成が必要

2022-02-21普通の継承userならマイグレーション後でも作れる。

AbstractBaseUser例

from django.contrib.auth.models import AbstractUser, PermissionsMixin

from django.db. import models

class CustomUser(AbstractUser, PermissionsMixin): # カスタムユーザーModel

login\_count = models.IntegerField(verbose\_name=‘ログイン回数’, default=0)

補足

PermissionsMixinを後から付けたら「Models aren’t loaded yet.」のエラーが発生。

2022-03-07現在ではいまいち意味が分かってないが、つけておいた方が無難

システムが既に運用開始している場合などの苦肉の策として「別モデルを作ってOneToOneFieldで関連させる」という方法もある。

OneToOneの例

from django.contrib.auth.models import User

class Employee(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

department = models.CharField(max\_length=100)

##### チュートリアル

ポイント

・myapp/models.pyにAbstractBaseUser、またはAbstractUserを継承したユーザーclassを作成

・myapp/models.pyにBaseUserManagerを継承したユーザー管理classを作成

・カスタムclassにUSERNAME\_FIELD とREQUIRED\_FIELDS を定義する

・setting.pyにAUTH\_USER\_MODELを追加

[ myapp/models.py ]（Userの参考）

from django.db import models

from django.contrib.auth.models import AbstractBaseUser

class MyAppUser(AbstractBaseUser, PermissionsMixin):

username = models.CharField(max\_length=150)

email = models.EmailField(verbose\_name=‘メールアドレス’, max\_length=255, unique=True)

is\_active = models.BooleanField(default=True) # DBのフィールドが必要なのでoverrideする

is\_staff = True # Django管理サイトにログインする為に必要

USERNAME\_FIELD = 'email' # 値が一意のフィールドの指定。レコードの判別に用いる。

REQUIRED\_FIELDS = ['username',] # manage.py createuserで必須となる引数名

objects = MyAppUserManager()

補足：Managerは下記「Manager」を参考

補足：USERNAME\_FIELDのフィールドをREQUIRED\_FIELDSに含める必要はない。むしろ入れるとエラーとなる。

[ config/setting.py]

# どこでも良いが、AUTH\_PASSWORD\_VALIDATORSの上にしておく

AUTH\_USER\_MODEL = ‘myapp. MyAppUser’

注意点：

Django管理サイトにログインする為には、is\_staffとis\_active両方が=TrueであるUserレコードでログインし、なおかつパスワードが正しい必要がある。

補足：

利用する際は、get\_user\_modelメソッドを用いる。

from django.contrib.auth import get\_user\_model

get\_user\_model().objects.all()

資料

参考：<https://noumenon-th.net/programming/2019/12/13/abstractbaseuser/>

（公式）<https://docs.djangoproject.com/ja/4.0/topics/auth/customizing/#substituting-a-custom-user-model>

次ページに続く

##### Manager

モデルに対するクエリを作成する為に定義するクラス。（ユーザーに限らない）

また、カスタムユーザーでcreatesuperuserする際には，そのマネージャーも実装する必要がある。

from django.db import models

from django.contrib.auth.models import BaseUserManager

class MyAppUserManager(BaseUserManager): # create\_modelname という形式

def create\_user(self, username, password=None): # manage.pyと管理サイトで使われる

if not username: # 必須項目がある場合

raise ValueError(‘Users must have an username’)

user = self.model(

username = username,

)

user.set\_password(password)

user.save(using=self.\_db)

return user

def create\_superuser(self, username, password): # manage.pyで用いられる

user = self.create\_user( username, password=password )

user.is\_stafff = True # Django管理サイトにログインする為に必要

user.save(using=self.\_db)

return user

補足

ここで使っているuser 変数は、self.modelというクラス？を用いてこそいるが、.NETなどのobject型に近いような形になっていて、どんなフィールドでも追加できるような形らしい。

（最終的にバリデーションみたいな事は行われると思うが）

（公式）<https://docs.djangoproject.com/ja/4.0/topics/db/managers/>

##### AbstractUser とAbstractBaseUserの違い

おおまかには，AbstractBaseUserの方が初めから定義されているフィールドなどが少ない為，自由度が高い。

迷ったらAbstractBaseUserにしておく。

参考：本ドキュメント内のライブラリリファレンス – django.db.models

##### PermissionMixin

is\_superuser

user\_permissions

has\_perms

補足

AbstractUser を継承した場合は PermissionsMixin を継承済な為，この対応をする必要が無い。

資料

公式：<https://docs.djangoproject.com/ja/4.0/topics/auth/customizing/>

参考：<https://thinkami.hatenablog.com/entry/2020/04/18/215826>

公式（英語）：<https://github.com/dfunckt/django-rules>

##### LoginViewを用いたログイン

概要

ほとんどLoginViewというクラスを継承するだけで、アプリにログイン機能を持たせられる。

ディレクトリ構成

myproject

├ config\

│ ├ settings.py

├ myapp\

│ ├ forms.py

│ ├ views.py

│ ├ urls. py

├ templates\

│ ├ login.html

│ └ index.html

[ config\settings.py ]

LOGIN\_URL = ‘myapp:login’ # 初期値 /account/login

LOGIN\_REDIRECT\_URL = ‘myapp:index’ # 初期値 /account/profile

LOGOUT\_REDIRECT\_URL = ‘myapp:index’ # 初期値 None

[ myapp\urls.py ]

from django.urls import path

from django.contrib.auth.decorators import login\_required, permission\_required

from . import views

app\_name=‘myapp’

urlpatterns = [

path(‘‘, views.views.IndexView.as\_view(), name=‘index’),

path(‘login/’, views.user\_views.MyAppLoginView.as\_view(), name=‘login’),

[ myapp\views.py ]

from django.contrib.auth.views import LoginView, LogoutView

class MyAppLoginView(LoginView):

#form\_class = LoginForm # formを使う場合。（validationとか）

template\_name = ‘login.html’

class MyAppLogoutView(LogoutView):

template\_name = ‘index.html’

[ templates/login.html ]（例）

<form method="POST">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Login</button>

</form>

（参考）<https://houdoukyokucho.com/2021/02/03/post-2952/>

##### login\_requiredアトリビュート

対象のview(=url)だけをログイン必要にする

method式の場合

from django.contrib.auth.decorators import login\_required, permission\_required

@**login\_required**

def report(request):

if request.method == ‘POST’:

pass

class式の場合

from django.contrib.auth.decorators import login\_required, permission\_required

from . import views

urlpatterns = [

path(‘report/’, **login\_required**(views. ManagerView.as\_view()), name=‘data\_report’),

]

補足

django.contrib.auth.middleware.AuthenticationMiddlewareが必要（らしい）

関連：LoginRequiredMixin、django-glrm

##### LoginRequiredMixin

from django.contrib.auth.mixins import LoginRequiredMixin

class View(TemplateView,LoginRequiredMixin):

template\_name = "template/index.html"

関連：login\_requiredアトリビュート、django-glrm

##### ミドルウェアを使ったログイン

いずれも「プロジェクト全体」に完全なログインシステムを適用する。

MiddlewareMixinを利用した自作

[ myapp/middlewares.py ]

class LoginRequiredMiddleware(MiddlewareMixin):

def process\_response(self, request, response):

if request.path != ‘/myapp/login/’ and not request.user.is\_authenticated:

return HttpResponseRedirect(‘/gyoumu/login/’)

return response

[ config/settings.py ]

MIDDLEWARE = [

‘gyoumu.middlewares.LoginRequiredMiddleware’,

]

情報元：<https://tsukasa-blog.com/programming/django-loginrequiredmiddleware/>

GlobalLoginRequiredMiddleware

pip install django-glrm

[ config/settings.py ]

MIDDLEWARE = [

‘global\_login\_required.GlobalLoginRequiredMiddleware’, # 追加

]

PUBLIC\_PATHS = [

r'^/accounts/.\*', # 対象外とするURLを正規表現で指定

'', # 【注意】こうするとすべてのViewがpublicになってしまう

]

PUBLIC\_VIEWS = [

'django.contrib.auth.views.PasswordResetView', # 特定のviewを指定する（未検証）

'myapp.views.MyAppTopView',

]

ログイン制御の対象から外す（以下のどちらでも良い）

[ views.py ]

from global\_login\_required import login\_not\_required

class BookView():

LOGIN\_NOT\_REQUIRE = True

@login\_not\_required

class BookView():

pass

公式：<https://django-glrm.readthedocs.io/en/latest/readme.html>

##### Eメール送信

[ users/models.py ] ※usersというアプリを作成した場合

from django.core.mail import send\_mail

class MyCustomUser(AbstractBaseUser, PermissionsMixin):

EMAIL\_FIELD = "email"

def email\_user(self, subject, message, from\_email=None, \*\*kwargs):

send\_mail(subject, message, from\_email, [self.email], \*\*kwargs)

※この情報の真偽は不明 2022-03-09

[ config/settings.py ]

#

EMAIL\_BACKEND = ‘django.core.mail.backends.smtp.EmailBackend’ # 実際に送信（本番）

# Gmailの場合の例

EMAIL\_HOST = ‘smtp.gmail.com’

EMAIL\_PORT = 587

EMAIL\_HOST\_USER = ‘gmailアカウント名’

EMAIL\_HOST\_PASSWORD = ‘パスワード’

EMAIL\_USE\_TLS = True

（補足）pythonにもsmtplibというメール用モジュールがある。Django内機能はそのラッパーとなる。

（参考）本ドキュメント内のライブラリリファレンス　mail.backends

（参考）<https://blog.narito.ninja/detail/64/>

（公式資料）<https://docs.djangoproject.com/ja/4.0/topics/email/>

##### パスワードポリシー（validation）

###### 標準のvalidatorを微調整する

・パスワードのvalidationルールはsettings.pyで定められている

AUTH\_PASSWORD\_VALIDATORS = [

]

デフォルトで定義されているのは以下の４つ

|  |  |  |  |
| --- | --- | --- | --- |
|  | 概要 | OPTIONS | 備考 |
| UserAttributeSimilarityValidator | パスワードとユーザの属性との類似性 | max\_similarity  など |  |
| MinimumLengthValidator | パスワードに必要な最低文字数 | min\_length | def. min\_length=8 |
| CommonPasswordValidator | 単純なパスワードでないか | password\_list\_path | デフォルトでは約2万種類のパスワード辞書と比較する |
| NumericPasswordValidator | 数字のみのパスワードになっていないか | 特になし |  |

・以下のようにして微調整も可能

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

'OPTIONS': {

'min\_length': 10,

}

},

]

公式：<https://docs.djangoproject.com/ja/4.2/topics/auth/passwords/#password-validation>

###### カスタムvalidatorを作成する

[ settings.py ]

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'myapp.validators.MyCustomValidator',

},

]

[ myapp/validators.py ]

import re

from rest\_framework.exceptions import ValidationError

class MyCustomValidator:

def validate(self, password, user=None):

result = re.match() # 正規表現で判定するものとする

if not result:

raise ValidationError("")

資料

公式：<https://docs.djangoproject.com/ja/4.2/topics/auth/passwords/>

##### パスワード再設定

PasswordChangeViewなどを用いる事で実現できる。

以下のいずれかの方法で記述。

###### 方法１

Django内部のviewに記述してあるPasswordResetViewなどを利用する。

基本的に以下の記述を加えるだけで良い。

[ config/urls.py ]

from django.urls import path, include

urlpatterns = [

…

path('', include(‘django.contrib.auth.urls’)), # 追加

]

manage.py runserver などでサーバーを起動し、Webブラウザにlocalhost:8000/password\_resetと入力するとパスワードリセット画面が表示される。

【重要　パスワード送信メッセージが表示される条件】

・is\_active が TrueかつemailフィールドがNullでは無いユーザーが存在し、リセット画面で入力したemailアドレスと一致する事。（is\_staffは無関係；多分）

※エラーメッセージが一切表示されずにメール送信されない、といった事象もあるので、要注意。

補足

・例えばmyvenvという仮想環境を利用している場合は

「myvenv/Lib/django/contrib/auth/views.py」に格納されていて、そのソースコードを見る事でPasswordResetViewなどの使い方は大体検討がつく。

・複数のユーザーで１つのメールアドレスを共有する場合にパスワードリセットを行うと、ユーザー一人に付き１回同じメールアドレスにメールが送信される。

注意

この方法は、後述するようなmyapp/urls.pyに記述する事が出来ず、config/urls.pyに記述する必要がある。

次ページに続く。

###### 方法２

[ users/urls.py ] ※usersというアプリを作成した場合

from django.contrib import admin

from django.contrib.auth import views as auth\_views # 追加

from django.urls import path, include, reverse\_lazy

urlpatterns = [

…

path(‘password\_change/’, auth\_views.PasswordChangeView.as\_view(), name=‘password\_change’),

path(‘password\_change/done/’, auth\_views.PasswordChangeDoneView.as\_view(), name=‘password\_change\_done’),

path(‘password\_reset/’, auth\_views.PasswordResetView.as\_view()

success\_url = reverse\_lazy(‘users:password\_reset\_done’)),

, name=‘password\_reset’),

path(‘password\_reset/done/’, views.PasswordResetDoneView.as\_view(), name=‘password\_reset\_done’),

path(‘password\_reset/confirm/<uidb64>/<token>/’, \

views.PasswordResetConfirmView.as\_view(),name=‘password\_reset\_confirm’),

path(‘password\_reset/complete/’, views.PasswordResetCompleteView.as\_view(), name=‘password\_reset\_complete’),

]

補足：success\_url以外もカスタマイズしたい場合はmyapp/views.pyに継承したカスタムviewを作成する。

（後述）

メールでリセットする例

[ config/settings.py ]

EMAIL\_BACKEND = ‘django.core.mail.backends.console.EmailBackend’ #コンソール表示

EMAIL\_BACKEND = ‘django.core.mail.backends.smtp.EmailBackend’ # 本番用

# Gmailの場合の例

EMAIL\_HOST = ‘smtp.gmail.com’

EMAIL\_PORT = 587

EMAIL\_HOST\_USER = ‘メールのユーザー名’

EMAIL\_HOST\_PASSWORD = ‘パスワード’

EMAIL\_USE\_TLS = True

ローカル環境の場合（ファイル保存）

[ config/settings.py ]

EMAIL\_BACKEND = "django.core.mail.backends.filebased.EmailBackend"

EMAIL\_FILE\_PATH = os.path.join(BASE\_DIR, "sent\_emails")

（参考）本ドキュメント内のライブラリリファレンス　mail.backends

次ページに続く

カスタムviewの例

class PasswordReset(PasswordResetView):

"""パスワード変更用URLの送付ページ"""

subject\_template\_name = ‘users/mail\_template/password\_reset/subject.txt’

email\_template\_name = ‘users/mail\_template/password\_reset/message.txt’

template\_name = ‘users/password\_reset\_form.html’

form\_class = MyPasswordResetForm

success\_url = reverse\_lazy(‘users:password\_reset\_done’)

class PasswordResetDone(PasswordResetDoneView):

"""パスワード変更用URLを送りましたページ"""

template\_name = ‘users/password\_reset\_done.html’

class PasswordResetConfirm(PasswordResetConfirmView):

"""新パスワード入力ページ"""

form\_class = MySetPasswordForm

success\_url = reverse\_lazy(‘users:password\_reset\_complete’)

template\_name = ‘users/password\_reset\_confirm.html’

class PasswordResetComplete(PasswordResetCompleteView):

"""新パスワード設定しましたページ"""

template\_name = ‘users/password\_reset\_complete.html’

django/contrib/auth/views.pyでの定義（抜粋）

class PasswordResetView(PasswordContextMixin, FormView):

email\_template\_name = ‘registration/password\_reset\_email.html’

extra\_email\_context = None

form\_class = PasswordResetForm

from\_email = None

html\_email\_template\_name = None

subject\_template\_name = ‘registration/password\_reset\_subject.txt’

success\_url = reverse\_lazy(‘password\_reset\_done’)

template\_name = ‘registration/password\_reset\_form.html’

title = \_(‘Password reset’)

token\_generator = default\_token\_generator

class PasswordResetDoneView(PasswordContextMixin, TemplateView):

template\_name = ‘registration/password\_reset\_done.html’

title = \_(‘Password reset sent’)

補足

メールの送信にはSMTPポートの開放が必要

（参考）<https://akiyoko.hatenablog.jp/entry/2021/12/01/080000>

（参考）<https://office54.net/python/django/django-password-reset>

（公式）<https://docs.djangoproject.com/ja/4.0/topics/auth/default/#django.contrib.auth.views.password_reset>

###### AUTH\_PASSWORD\_VALIDATORS

##### primarykeyの変更

##### メモ

to\_field でUSERNAME\_FIELD or REQUIRED\_FIELDS, を外部キーかできる？

admin関連？

[ models.py ]

from django.contrib import admin

from django.contrib.auth.admin import UserAdmin

class CustomUser(UserAdmin): # UserAdminを継承したカスタムユーザーModel

login\_count = models.IntegerField(verbose\_name=‘ログイン回数’, default=0)

管理者ユーザーの作成

terminal

python manage.py createsuperuser

python manage.py runserverしてWebブラウザで 192.168.1.1/adminと入力

⇒ Django管理サイトが表示される。

・ここでユーザーとグループが作成できる

・ここで作成できるのは一般ユーザーであり、管理者ユーザーではない。

##### 資料

Django標準のUserクラスを使う場合：

<https://docs.djangoproject.com/ja/4.0/topics/auth/default/>

公式１：https://docs.djangoproject.com/ja/4.0/topics/auth/customizing/#auth-custom-user

（参考）https://docs.djangoproject.com/ja/2.2/ref/models/fields/

https://qiita.com/nachashin/items/f768f0d437e0042dd4b3

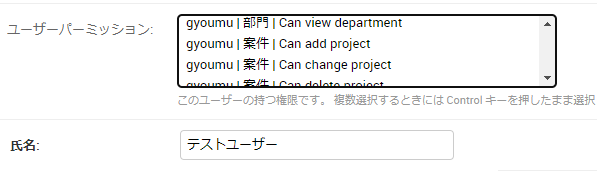
#### Django管理サイト

##### ユーザーパーミッション

ユーザー毎に各レコードのadd, change, delete, viewの権限を与える事ができる。

この権限は**管理サイトでのみ有効**。

また、is\_superuser=Trueのユーザーはここで設定した権限は無視される。（全てのアクセスが許可）



（補足）このような権限の管理は「グループ機能」で実装すべき。

（補足）admin.pyでadmin.ModelAdminを継承したクラスでhas\_\*\*\*\_permissionをオーバーライドする事で、ここでの設定を無視して強制的に権限を変更する事も可能。

class BookAdmin(admin.ModelAdmin)

def has\_delete\_permisson(self, request, obj=None)]

return False

##### グループ

複数のユーザーに同じのアクセス権限を適用する為に用いる。

Django管理サイト

##### get\_ordering

orderingフィールドよりも詳細な並べ替え条件を指定できる。

##### formfield\_for\_foreignkey

管理サイトの編集でドロップダウンにて外部キーを選択する時の条件を記述できる。

class MyModelAdmin(admin.ModelAdmin):

def formfield\_for\_foreignkey(self, db\_field, request, \*\*kwargs):

if db\_field.name == "car": # 実際にはModelのfield名にする必要がある

kwargs["queryset"] = Car.objects.filter(owner=request.user)

return super().formfield\_for\_foreignkey(db\_field, request, \*\*kwargs)

注意：変数名はdb\_field.nameだが、実際にはModelのfield名と合わせる必要がある。DBのフィールド名ではない

（公式）<https://docs.djangoproject.com/ja/4.0/ref/contrib/admin/#django.contrib.admin.ModelAdmin.formfield_for_foreignkey>

##### その他メモ

・必須項目は太文字表示される。

・nullが

list\_editable

#### models

##### 基本

基本的なModelクラスの例

from django.db import models

class Publisher(models.Model):

class Meta:

db\_talbe = ‘publisher’

name = models.CharField(verbose\_name=‘出版社名’, max\_length=255)

def \_\_str\_\_(self): # この値がModelChoiceFieldなどでの表示名となる為，ほぼ必須

return self.name

そのほか

フィールドidは以下のように自動的に作成される為，model.pyでは記述する必要がない。

CREATE TABLE "polls\_choice" (

"id" serial NOT NULL PRIMARY KEY,

・デフォルトのUserモデルを使用するのは避ける

参考）オンラインドキュメント：カスタムのユーザーモデルを使用する

##### Metaクラス

class MyPost(models.Model):

name = models.CharField(max\_length=128)

class Meta:

db\_talbe = ‘my\_post’ # テーブル名を規定。省略すると AppName\_mypost という名前になる

app\_label = ‘account’ # 通常不要。必要となる場合がある

managed = False # makemigrations, migrate の影響下に置くかどうか

verbose\_name = ‘mypost’ # 管理サイトで用いるモデル名（単数形）

verbose\_name\_plural = ‘myposts’ # 管理サイトで用いるモデル名（複数形）

ordering = [‘no’] # 並び替え順序。「-」をつけると降順になる。

補足

・pythonの\_\_name\_\_ などのprivate変数もメタクラスと呼ばれるが，このMetaとは関係が無い。

・verbose\_name\_pluralが無いと「ブログs」みたいな表記になり得る。

・列の名前はfield継承クラスのフィールド「db\_column」で指定。

その他のMetaクラスフィールド：

managed migrationsの管理対象外とするかどうか（def.=True）

##### 抽象モデル

継承専用のモデルの作成の仕方（データテーブルが作成されない）

class BookBase(models.Model):

class Meta:

app\_label = 'myapp' # models ディレクトリに入れて分割する場合必要

abstract = True # こうする事で抽象クラス化され、データテーブルが作成されない

class Manga(BookBase):

##### models.pyの分割方法

概要

ポイントは以下の3点

・modelsディレクトリを作成

・\_\_init\_\_.pyを作成して自作モデルのclassをimportする

・モデルのclassのMeta: にapp\_labelを追加。

ディレクトリ構成

myproject #

├─ myapp/ #

│ ├ models/ # これを新規作成

│ │ ├ \_\_init\_\_.py # ここで同階層の \*.py をすべてimport

│ │ ├ mymodel.py # 分割モジュール１

│ │ ├ others.py # 分割モジュール２

myapp/models/mymodel.py

from django.db import models

class MyModel(models.Model):

name = models.CharField(verbose\_name=‘名前’, max\_length=255)

class Meta:

app\_label = ‘myapp’ #分割する場合に必要となる

※others.pyも同様に記述

myapp/models/\_\_init\_\_.py

from .mymodel import MyModel

from myapp.models.others import Book, Shop # 完全なnamespaceで記述する方法（めんどくさいので非推奨）

補足：

python3.3以降は\_\_init\_\_.pyが無くともディレクトリを１つのパッケージとして見なしてはくれる。

但し、今回のように同階層のモジュールの読込みなどを行いたい場合は必要。

##### 循環参照対策

models.pyをディレクトリ化する事により循環参照が発生する事がある。

補足：循環参照のエラー【 most likely due to a circular import 】

具体例：

ディレクトリ構成（models.pyの分割方法のものとほぼ同じ）

myproject

├ myapp/

│ ├ models/

│ │ ├ \_\_init\_\_.py

│ │ ├ users.py

│ │ ├ models.py

[ users.py ]

from django.db import models # これがある為に、一行下のを import models とできない

from myapp.models import Department # 別ファイルのクラスをimport

# import myapp.models as myapp\_models # myapp\_models.Departmentで良い！という情報も⇒失敗

class MyCustomUser(AbstractBaseUser):

username = models.CharField(verbose\_name=‘氏名’, max\_length=150)

department\_id = models.ForeignKey(Department, verbose\_name=‘部門’)

[ models.py ]

from django.db import models

import myapp.users as myapp\_users # これにより循環参照が発生する

from myapp.models import Department # 別ファイルのクラスをimport

class Department(models.Model):

name = models.CharField(verbose\_name=‘部門名’, max\_length=100)

class Task(models.Model):

person\_done\_by = models.ForeignKey(MyCustomUser, verbose\_name=‘担当者’)

解決策

ForeignKeyのto: 引数には下のように文字列で指定する方法もある。これを用いれば解決。

[ users.py ]

# import myapp.models as myapp\_models # 不要となる

class MyCustomUser(AbstractBaseUser):

department\_id = models.ForeignKey(**‘myapp.Department’**, verbose\_name=‘部門’)

補足：

そもそもmodels.pyの分解はviews.py分解より恩恵が少ないので分解しなくても良い。

##### モデル側で入力規制を行う

form のバリデーションではなく、model側で入力規制を行いたい事がある。

その場合models.pyの各クラスの\_\_init\_\_メソッドのオーバーライドが最初に思いつくが、公式によるとその方法は非推奨らしい。

代案として推奨される方法（少し修正版）：

class BookManager(models.Manager):

def create\_book(self, title, \*args, \*\*kwargs):

book = self.create(

title=title,

price=int(kwargs["price"], # 引数が多い場合はkwargsで対応した方が良い

)

return book

class Book(models.Model):

title = models.CharField(max\_length=100)

objects = BookManager() # ここでManagerクラスを指定

# 実際の処理（実際にはviews.pyで行われる）

book = Book.objects.create\_book(

title="Pride and Prejudice",

price=1000,

)

公式：<https://docs.djangoproject.com/ja/4.0/ref/models/instances/#django.db.models.Model>

##### 画像やファイルを扱う

以下のようにする事で「http://<サーバ名>/media/xx/<ファイル名>」でファイルにアクセスできるようになる

[ settings.py ]

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

MEDIA\_URL = '/media/'

[ urls.py ]

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [ ～省略～

]

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

（公式）<https://docs.djangoproject.com/ja/4.0/faq/usage/#how-do-i-use-image-and-file-fields>

#### forms.py

##### 概要

最もシンプルな「ModelFormの例」

ファイルパス：/**myapp**/forms.py 慣例的にこのファイル名（自分で作る）

from django import forms

from .models import Book

class BookEditForm(forms.ModelForm):

class Meta:

model = Book

fields=["name, price"] # ‘\_\_all\_\_’ でも良い

def \_\_init\_\_(self, \*args, \*\*kwargs): # 任意；def \_\_init\_\_(self): ×

super().\_\_init\_\_(\*args, \*\*kwargs)

実際に使うには，viewでrenderを返す時のdict型引数，contextに含める。

[ myapp/views.py ]

from .forms import LoginForm

from django.views import View

from django.shortcuts import render

# メソッド式

def login():

if request.method == ‘POST’: # html メソッドの判別

form = LoginForm(request.POST)

else:

form = LoginForm() # コンストラクタの引数は必須では無いらしい

return render(request, ‘myapp/login.html’.html’, {‘form’ : form }

# クラス式

class LoginView(TemplateView):

template\_name = 'login.html'

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['form'] = LoginForm(request.POST)

return context

[ templates/login.html ]

<h2>ログインフォーム</h2> # イメージしやすいように付けただけ。特に不要。

{{ login\_form }} # これで自動的に <input> を用いた html が作成される。

自動的にhtmlが作成されるのは楽で良いが，自身でCSSなどのデザインを触りたいときにこれでは困る。

（詳細）[django.forms](#_django.forms)

##### Formクラスのメソッド

|  |  |  |
| --- | --- | --- |
|  |  | 使用場所／備考 |
| form(request.POST) | Formのコンストラクタ。 | view |
| form.is\_valid() | 入力チェックを行う。OKだとcleaned\_data という変数が使えるようになる。 | view |
| form.add\_error() |  |  |
| form.save() | データベースに変更を適用する。  なお，.save(commit=Fals) でインスタンスにのみ適用（データベースにはコミットしない） |  |
| form.non\_field\_errors | フィールドに関係のないエラーをlistで返す | html |
| form.ValidationError | 入力チェック中に問題があった場合に，意図的にraiseさせる例外 |  |
| form.get\_user() | 指定のユーザーオブジェクトを取得する  引数：user\_id　省略時は現在のユーザー |  |
|  |  |  |
| field.errors | 各フィールドのエラーをlistで返す |  |
|  |  |  |
| clean\_[フィールド名]() | フィールド単体の入力チェック。  また，このメソッド内では対象のフィールドの  self.cleaned\_data が使える。 | form  cleaned\_dataはdict型 |
| clean() | form 全体の入力チェック  また，このメソッド内では各フィールドの  self.cleaned\_data が使える。 | viewでform.is\_validを実行すると呼び出される |

関連

View.get\_form()

##### Formとhtmlタグの関係

###### 一番簡単に表示する方法

[ forms.py ]

from django import forms

class BookForm(forms.Form):

title = forms.CharField(label=‘タイトル’)

[ テンプレート（.htmlファイル）]

{% for field in form %}

{{ field }}

{% endfor %}

実際に出力されるhtml

<input type="text" name="title" required id="id\_title">

###### Formクラスを用い、html内でループさせる場合

[ テンプレート（.htmlファイル）]

{% for field in form %}

{{ field.label }}

{{ field }}

{% endfor %}

こういった事も可能

<label for="{{field.id\_for\_label}}">{{ field.label }}</label>

{{ field }}

実際に出力されるhtml

<input type="text" name="" required id="">

<input type="text" name="title " list="items" class="label" required= id="id\_title">

基本的なフィールドごとのエラーメッセージの表示

[ テンプレート（.htmlファイル）]

{% for field in form %}

<p>{{ field.label }} {{ field }}</p> <!-- ラベルの横に値を表示 -->

{% if field.errors %}

<p>{{ field.errors.0 }}</p>

{% endif %}

{% endfor %}

・要素を追加する場合

[ forms.py ]

from django import forms

class BookForm(forms.Form):

title = forms.CharField(label=‘タイトル’)

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields[‘title’].widget.attrs[‘list’] = 'book\_list' # html5以降のdatalistに対応する項目

self.fields[‘title’.widget.attrs[‘class’] = 'label' # カスタムでhtmlのclassを適用する場合

次ページに続く

###### Formクラスを使い、htmlでループを使わない場合

class BookForm(forms.Form):

asset\_number= forms.CharField(

label=‘固定資産番号’,

required=False,

)

publish\_date = forms.DateTimeField(

label=‘出版日’,

widget=forms.DateInput(attrs={"type": "date"}),

input\_formats=[‘%Y-%m-%d’],

required=False,

)

[ views.py ]

class BookListView(ListView):

model = Book

template\_name = ‘myapp/shop/book\_list.html’

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

dt\_publish = datetime.now() # 初期値

query\_dt\_publish = self.request.GET.get(‘ publish\_date ‘, None) # クエリ文字列からの値の取得

if query\_dt\_publish: # クエリ文字列があれば、値を修正

dt\_publish = datetime.strptime(query\_dt\_publish, ‘%Y-%m-%d’).date()

context[‘form’] = BookForm(initial = # context内のフォーム変数の初期値を設定

{

' dt\_field ' : dt\_publish

}

[ テンプレート（.htmlファイル）]

<form class="form" method="post">

{% csrf\_token %}

<div class="row"> <!-- class=form-rowも使われる -->

<div class="col">

<label for=" dt\_field ">{{ field.label }}</label>

<label for="{{ form.dt\_field.id\_for\_label}}">{{ dt\_field.label }}</label>

{% if form.dt\_field.required %} <!-- form.dt\_field.field.required でも良いらしい-->

<span class="asteriskField">\*</span>

{% endif %}

{{ form.publish\_date }}

</div>

</div>

</form>

注意点

・form.fields.publish\_date とすると上手く表示できないので注意

・レイアウトは<table class="table table-borderless">タグを使った方が楽な事が多い。

choiceフィールドなどはそれぞれループを回す事が可能

例：book\_type= forms.ChoiceField() の時

{% for radio in form.book\_type %}

<div"> {{ radio }} </div>

{% endfor %}

###### Formクラスを使わず、htmlを完全カスタマイズする

（編集中）

[ views.py ]

class BookView(TemplateView):

template\_name=‘gyoumu/project/taskhours\_estimate.html’

def post(self, request, \*args, \*\*kwargs):

id = kwargs[‘project\_id’] # URL文字列の引数を取得

project = get\_object\_or\_404(Project, pk=id)

val = request.POST(‘title’) # この方法だと、request（dict型）にキーが存在しないと例外がraise

val = request.POST.get(‘title’, None) # こちらの方法が推奨される

return redirect(reverse(‘myapp:book\_detail’, kwargs={"book\_id" : id}))

[ html ]

<form class="form" method="post">

{% csrf\_token %}

<div class="row">

<div class="col">

<div id="div\_id\_title" class="form-group">

<label class="form-label" for="title">タイトル</label>

<span class="asteriskField">\*</span>

<input>

</div>

</form>

利用可能なフィールド

name

errors エラーメッセージの集まり

field フォームクラスのプロパティ

help\_text 説明文言

html\_name インプットタグのname要素の値

id\_for\_label <label>タグのfor要素のid

<label class="form-label" for="{{ field.id\_for\_label }}">{{ field.label }}</label>

label ラベル用のテキスト

label\_tag ラベルのHTMLタグ

value インプットタグ用のvalue要素の値

（参考１）<https://blog.narito.ninja/detail/98/>

（参考２）<https://django.kurodigi.com/2019/03/>

プロパティ

.tag

.label

使用例

{% if messages %}

{% for message in messages %}

<li{% if message.tags %} class="{{ message.tags }}"{% endif %}>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

旧情報

・以下のように１つずつ指定する方法もあるようだが、手間。

<input

type="{{item.field.widget.input\_type}}"

class="form-control"

{% if item.field.required %}required{% endif %}

name="{{item.name}}"

id="{{item.id\_for\_label}}">

・完全にカスタマイズする場合。（{% for %} を使わず自分で書く）

<form action="" method="POST">

<div class="form-row">

{{ form.email.label\_tag }}

{{ form.email }}

<span class="helptext">{{ form.email.help\_text }}</span>

{{ form.email.errors }}

</div>

</form>

##### 日付の入力

###### 概要

おおむね、下のいずれかでやる

forms.DateInput() おすすめ

input type=”date”

jQuery UI Datepicker

AdminDateWidget

bootstrap-datetimepicker

forms.DateInput() の場合

[ forms.py]

class BookForm(forms.Form)

date = forms.DateTimeField(

label=‘日付’,

widget=forms.DateInput(attrs={"type": "date"}),

input\_formats=[‘%Y-%m-%d’],

)

ModelFormの場合？

class BookForm(forms.ModelForm):

class Meta:

model = Book

widgets = {

‘date’ : forms.DateInput(attrs={‘type’:’date’})

}

次ページに続く

～日付の入力　つづき

input（htmlタグ）

IE(internet explorer)とSafariでは使用できない。

ちょっとめんどくさい。

jQuery Datepicker

インストール

pip install django-bootstrap-datepicker-plus

INSTALLED\_APPS = [

…

‘bootstrap\_datepicker\_plus’, # 追加

]

forms.py

from django import forms

from bootstrap\_datepicker\_plus.widgets import DateTimePickerInput

class BookForm(forms.ModelForm):

class Meta:

widgets = {

publish\_date’: DateTimePickerInput(),

公式：<https://pypi.org/project/django-bootstrap-datepicker-plus/>

bootstrap

forms.py

import bootstrap\_datepicker\_plus as datetimepicker

from django import forms

from .models import Item

class ItemForm(forms.ModelForm):

class Meta:

model = Item

fields = (‘sample\_1’, ‘sample\_2’, ‘sample\_3’, ‘sample\_4\_start’, ‘sample\_4\_end’)

widgets = {

‘sample\_1’: datetimepicker.DatePickerInput(

format=‘%Y-%m-%d’,

options={

‘locale’: ‘ja’,

‘dayViewHeaderFormat’: ‘YYYY年 MMMM’,

}

),

参考：<https://qiita.com/okoppe8/items/999b8e3c86708fbb3926>

##### バリデーション

###### forms.pyで行う方法（推奨）

formで使う場合には以下の２つのメソッドが各Modelクラスに必要。

clean\_<フィールド名>() フィールド単体のバリデーション

clean() １レコード内での相関チェックやデータベースとの整合性確認

フィールド用バリデーションの例

[ forms.py ]

def clean\_username(self):

password = self.cleaned\_data.get('password') # 他のフィールドの値を取得(cleaned\_dataはdict型)

password = self.data.get('password', None) # 敢えてバリデーション前のデータを取得

username = self.cleaned\_data['username'] # 本命データを取得

if len(username) < 3:

raise forms.ValidationError('3文字以上で入力して下さい’, params={‘min\_length : 3;’} )

return username # 最後に検証済みのデータを返す

レコード用バリデーションの例

views.pyでform.is\_validを実行すると呼び出される。

こちらの場合は，NGの場合に例外をraiseすればよい。（return する値は無くて良い）

def clean(self):

password = self.cleaned\_data.get(‘password’,None) # getメソッドを利用。初期値を決められる。

username = self.cleaned\_data[‘username’] # dictで直接取得。取得できない場合はNoneが返る。

try:

user = User.objects.get(username=username)

except ObjectDoesNotExist:

raise forms.ValidationError("正しいユーザー名を入力してください")

if not user.check\_password(password):

raise forms.ValidationError("正しいパスワードを入力して下さい")

補足

・unique\_togetherは親クラスのclean（upper().clean()）では処理されない。

次ページに続く

～forms.py　バリデーション　続き

###### 正規表現を使ったバリデーション例

import re

class MyForm(forms.Form):

my\_field = forms.CharField(max\_length=100)

def clean\_my\_field(self):

cleaned\_my\_field = self.cleand\_data['my\_field']

if not re.match(r'^[a-z]', cleaned\_my\_field):

raise form.ValidationError("英数字小文字のみ使用可能です")

return cleaned\_my\_field

views.py での使用例

if not form.is\_valid():

for fld, err in form.errors.items(): # invalidになった原因を追究する必要がある場合

print('field:%s error:%s' & (fld, err))

messages.info(request, ‘データに誤りがあります。’) # htmlに表示するメッセージ

return render(request, self.template\_name, context)

###### views.pyでバリデーションを行う方法

def form\_valid(self):

data = form.cleaned\_data

if data['title'] == 'invalid':

form.add\_error('title', 'タイトルが無効です。')

return self.form\_invalid(form)

return super().form\_valid(form)

###### models.pyでバリデーションを行う方法（非推奨）

クライアントサイドバリデーションはセキュリティの観点からのみ使用すべきであり、サーバーサイドでもバリデーションを行う必要がある。

[model.py]

from django.db import models

from django.core.validators import RegexValidator

class MyModel(models.Model):

alphanumeric\_validator = RegexValidator(r'^[a-zA-Z0-9]\*$', '半角英数字のみ入力してください。')

my\_field = models.CharField(max\_length=100, validators=[alphanumeric\_validator])

html（テンプレート）

<input type="text" pattern="[a-zA-Z0-9]\*" title="半角英数字のみ入力してください。">

from django import forms

class MyForm(forms.Form):

my\_field = forms.CharField(max\_length=100)

def clean\_my\_field(self):

data = self.cleaned\_data['my\_field']

if not data.isalnum():

raise forms.ValidationError('半角英数字のみ入力してください。')

return data

##### 値の復元（initial）

おおきく以下の方法がある

・views.py でForm生成時に代入

・views.py でModelFormを使う場合（CreateViewなどを使う場合を含む）

・

views.py でForm生成時に代入する場合

form = BookForm(initial ={

'items' : ('46926','46927',),

})

form.fields['items'].initial=('46926','46927',)

form.fields[‘pricet’] = forms.IntegerField(

initial=2000,

)

views.pyでModelFormを使う場合

def get\_initial(self):

initial = super().get\_initial() #

initial['user'] = test.user # testモデルのuserを初期値としてセット

def get\_form(self): # get\_formを使って同様の事をできそうな

pass

forms.pyで代入する場合

[ forms.py ]

class BookForm(forms.Form):

title =forms.CharField(label=‘タイトル’,)

def \_\_init\_\_(self, \*args, \*\*kwargs): # ‘initial’ を受け取れるようにkwargs

super().\_\_init\_\_(\*args, \*\*kwargs) # 基底クラスにkwargsを渡す

self.fields['price'].initial = 2000

注意点

MultipleChoiceFieldの値はtupleかlistで指定する必要がある。

（lQuerySetのlistではダメ?）

##### ModelFormの継承

from django import forms

from .models import item

class BookForm(forms.ModelForm):

class Meta:

model = Book # このフォームで使用するモデルを選択する

fields = (‘title’, ‘price’,) # 利用するフィールドを指定；ArrayLikeなのでリスト[] でも良い

補足：ModelFormでは\_\_init\_\_時の引数としてinstance が利用可能

##### 動的なform作成

[ views.py ]

from django import forms

class BookView(TemplateView):

template\_name=‘myapp/book/detail.html’ # この例におけるテンプレート名は適当

def get\_context\_data(self, \*\*kwargs: Any) -> Dict[str, Any]:

form = forms.Form() # ModelFormの場合はself.get\_form()を使う

for name in [‘a’,’b’]: # 実際にはデータベースなどから作成したリスト

form.fields[name] = forms.CharField(

label=name,

required=False,

)

context = {

‘form’ : form

}

return context

但しpostが課題 2022-05-10

・cleaned\_dataで値が取得できない

・is\_validメソッドを自分で書く（オーバーライド）する事ができない

⇒ htmlのinputで直接行う方が良い？

forms.py でやる場合

class DailyReportForm(forms.Form):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs) # 基底クラスのコンストラクタを実行

for name in [‘a’,’b’]: # 実際にはデータベースなどから作成したリスト

self.fields[name] = forms.CharField(

label=name,

required=False,

)

self.fields[name].widget.attrs = {‘autocomplete’ : ‘off’}

##### formset\_factory

複数レコードを同時に編集する際に用いる。

概要

フォームを配列状に並べた感じ。テーブル形式に編集。

例

[ forms.py ]

class SingleBookForm(forms.Form):

titile = forms.CharField(label='題名')

price = forms.IntegerField(label='単価',)

[ views.py ]

from django.forms import formset\_factory

from .forms import SingleBookForm

BookFormSet = formset\_factory(SingleBookForm, extra=0) # extraはdef.=0なので省略可能

class FormSetTestView(TemplateView):

template\_name = 'myapp/index.html'

def get\_context\_data(self, \*\*kwargs):

context['formset']= BookFormSet(

initial = queryset.valuse(), # dict型でないといけない

)

return context

<form class="form" method="post" onsubmit="return false;">

{{ formset.management\_form }} <!-- これを忘れないように！ -->

{% for form in formset %}

<tr>

<td>{{ form.recieved\_date }}</td>

</tr>

{% endfor %}

</form>

次ページに続く

from django.forms import formset\_factory

from .forms import SingleBookForm

class FormSetTestView(TemplateView):

template\_name = 'myapp/index.html'

BookFormSet = formset\_factory(SingleBookForm)

def get\_context\_data(self, \*\*kwargs):

formset = BookFormSet(

data={

'form-TOTAL\_FORMS' : 2, # extraとほぼ同等？

'form-INITIAL\_FORMS': '0', # ？？？

'form-0-price': '123', # 初期値の指定

},

)

context['formset'] = formset

return context

def post(self, request, \*args, \*\*kwargs):

formset = BookFormSet (request.POST)

# form の入力内容が適切か確認（form の clean メソッドが実行される）

if not formset.is\_valid():

# 基本的にはこれで良い

formset.save()

補足

id\_form-1-field\_name という書式を変更する手はないらしい2022-08-17

公式：<https://docs.djangoproject.com/ja/4.0/topics/forms/formsets/>

エラーを確認する方法

for form in formset:

print(form.errors)

print(formset.errors)

[ views.py ]

formset = forms.modelformset\_factory(

model=MyModel,

fields=('text',), # 文字列のtupleで指定

form=BookForm, # 自作フォームを使う事も可能。カスタマイズ性があり推奨

formset=BookFormSet, # updateなど、初期値の投入が必要な場合

extra=3,

)

※fieldsかformはどちらか片方のみを指定する

class BookForm (forms.ModelForm):

class Meta:model = Book

fields=['author', 'price']

exclude = (,) # もしくは、使わないフィールドの指定（どちらか片方が必要）

class BookFormSet(BaseModelFormSet):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.queryset = None # 初期値を設定する場合

[ template(.html) ]

{% for form in formset %}

{{ form.author }} # あとは通常のformと同じようにする

{% endfor %}

##### modelformset\_factory

ModelForm+ formset\_factoryといった感じ

[ models.py ]

from django.db import models

class Book(models.Model):

titile = models.CharField(label='題名')

price = models.IntegerField(label='単価',)

[ views.py ]

from django.forms import formset\_factory

from .forms import SingleBookForm

class FormSetTestView(TemplateView):

template\_name = 'myapp/index.html'

def get\_context\_data(self, \*\*kwargs):

# このどちらか

formset = modelformset\_factory(Book, form=BookForm)), # 第一引数はmodel

formset = modelformset\_factory(model=Book, fields=('titile', 'price')),

context['formset'] = formset

return context

補足：fieldsまたはextcludeが必須オプション

[ index.html]

<form calss="form" method="POST">

{% csrf\_token %}

{{ formset.management\_form }}

<table>

<thead>

<tr>

<th class="col-4">タイトル</th>

<th class="col-2">値段</th>

</tr>

</thead>

<tbody>

{% for form in formset %}

<tr>

{% for field in form %}

<td>{{field}}</td>

{% endfor %}

</tr>

{% endfor %}

</tbody>

</table>

</form>

補足：querysetによる項目の絞り込み

class FormSetTestView(TemplateView):

template\_name = 'myapp/index.html'

def get\_context\_data(self, \*\*kwargs):

BookFormSet = modelformset\_factory(Book, fields=('titile', 'price')),

context = {

'formset' : BookFormSet(queryset=Book.objects.filter(price\_\_lt = 3000)),

}

return context

補足：外部キーを持つテーブルの編集には向かない。

資料

前半わかりやすい：<https://qiita.com/sand/items/9b9d5d0b3a189a05711f>

公式： <https://docs.djangoproject.com/en/4.0/topics/forms/modelforms/#model-formsets>

##### 外部キーを扱う

###### 一番簡単な例

以下のようなモデルがあったとする

class Author(models.Model):

name = models.CharField(verbose\_name='名前')

class Book (models.Model):

title = models.CharField(verbose\_name='タイトル', primary\_key=True, max\_length=50)

author = models.ForeignKey(to=Author, verbose\_name='著者', on\_delete=models.DO\_NOTHING)

[ forms.py ]

class BookForm(forms.ModelForm):

class Meta:

model = Book

fields = ['title','author']

注意点としては、この場合はWebフォームでは Author.id（int）を入力する事になる為、少し分かりにくくなる。

反対に、文字列のprimary keyフィールドが用意されている場合には都合が良い。

補足

・db\_columnの値は、この際は関係ない。

<https://docs.djangoproject.com/ja/4.0/ref/forms/validation/#cleaning-and-validating-fields-that-depend-on-each-other>

###### Django3.1以降の場合

class BookForm(forms.ModelForm):

author\_name = forms.CharField ( source= 'author.name' )

###### Django3.1より前の場合

class BookForm(forms.ModelForm):

author\_name = forms.CharField ()

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

if self.instance.author:

self.fields['author\_name'].initial = self.instance. author.name

#### admin.py

Django管理サイトの、編集できるフィールド、モデル、表示ソート　などをカスタマイズできる。

[ myapp.admin.py ]

from django.contrib import admin

from .models import Book, Shop

class BookAdminForm(forms.ModelForm):

pass # formを使ってバリデーションなどが行える。（ここでは省略）

class BookModelAdmin(admin.ModelAdmin):

ordering = (‘name’,) # 昇順、降順。※注意！あくまでtuppleなのでコンマが必要。(‘name’**,** )

fields = (‘name’) # 編集できるフィールド

radio\_fields = {‘category’: admin.VERTICAL} # 編集コントロールも選べる？

form = BookAdminForm # formを利用する場合（任意）

admin.site.register(Shop) # Django管理サイトで編集できる項目を追加

admin.site.register(Book, **BookModelAdmin**) # 編集できる項目をカスタマイズする場合

こんな事ができるらしい 2022-02-28

if not is\_superuser:

form.base\_fields[‘username’].disabled = True

補足

fieldsなどの値はtuppleかlistのどちらかなら良いが、listにする理由が無いのでtuppleが良い。

admin.ModelAdminでreadonly\_fields =('is\_admin',) を無効にするとDjango標準の管理者画面でも管理者権限を編集できる。

#### Migration

class Migration(migrations.Migration):

initial = True

atomic = False # Try adding `atomic = False` to the Migration class.が出た時に試す

#### フィールド（データ型）

##### まとめ（基本フィールド）

db.modelsでもformsでも使えるフィールドをまとめる

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | pythonのtype | RDBでの型名 | html  input type= | formsのwidget | 備考 |
| Field | str | なし | なし | なし | 自作Fieldを使う時くらいしか使わない |
| BooleanField | bool | boolean | checkbox | CheckboxInput |  |
| CharField | str | varchar | text | TextInput |  |
| DateField | datetime.date | date | date | DateInput |  |
| DateTimeField | datetime.datetime | datetime | datetime-local | DateTimeInput |  |
| DecimalField | Decimal | decimal | number | NumberInput |  |
| DurationField | datetime.timedelta | interval | text | TextInput |  |
| FileField | django.core.files.File | varchar | file | FileInput |  |
| EmailField | str | varchar | email | EmailInput | EmailValidatorでvalidation |
| UUIDField | uuid.UUID | uuid | text | TextInput |  |
| TimeField | datetime.time | time | time | TimeInput |  |
|  |  |  |  |  |  |

※当然、使用するデータベースエンジンにより異なるが、目安としてPostgreSQLの場合。

##### 共通引数

verbose\_name

Django管理サイトでModel名として表示。

フィールド名。フォーム自動生成で見出しとして使う。

null NULL制約。True(NULL許可)/False(許可しない; def.)

※あくまでデータベースに対してnullを認めるかどうか。django上で空白文字を認める場合はblankを使う。

owner\_id = models.IntegerField(null=True) # NULL（=None）を許容する

（関連）default, blank

blank False：空白がダメということ。つまり入力必須(def.)　 True：空白許可。

memo = models.TextField(null= True, blank= True)

（関連）null

choices フォーム利用時にセレクトボックスに表示する選択肢

db\_index インデックスを設定するかどうか

db\_column

データベース上の列の名前を指定する。

省略した場合、Django上のフィールド名がそのまま使われる。

default 既定値を設定する。

ForeignKeyなど，NULL値を許容しない列メンバでは必須のオプション。

error\_messages バリデーションNGの場合のエラーメッセージ

help\_text

unique ユニーク制約

validators 文字種チェックなどのバリデーションを指定

###### required

必須項目。既定値=True

self.fields[‘email’].required = True

class BookForm(forms.Form):

title = forms.CharField(label='案件番号', required=False) # こんな方法もある

例

[models.py]

class Writer(models.Model):

name = models.CharField(max\_length=20)

class Article(models.Model):

writer = models.ForeignKey(Writer, on\_delete=models.CASCADE, null=True)

この場合template（.htmlファイル）で以下のようにすると，Writerテーブルのレコードが取得できる。

{% for ar in artible\_list %}

{{ ar.writer.name }} <!-- Writerテーブルのname列を取得 -->

{% %}

次ページに続く

##### CharField

Char型の列を返す。

max\_length ※必須項目。文字列の最大文字数。

strip 前後の空白文字を取り除く。def=True

title = models.CharField(max\_length=200)

model：

form：<https://docs.djangoproject.com/en/4.1/ref/forms/fields/#django.forms.CharField>

##### DateField系

DateField()

公式：<https://docs.djangoproject.com/ja/4.0/ref/models/fields/#datefield>

DateTimeField()

DateとDateTime共通

auto\_now\_add 追加時に現在時間を設定 def.=False

auto\_now 新規作成と更新時に現在時間を設定 def.=False

補足：auto\_now\_add、auto\_now、default は相互に排他的

create\_date = DateField(auto\_now\_add=True)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | データ型 |  | 初期値 |
| auto\_now |  | bool | 自動的に現在値を入力する |  |
|  |  |  |  |  |

補足

テンプレート（.html）で日付の書式を変更する方法。

<td>{{ rq.date|date:"Y年m月d日" }}</td>

月の抽出方法：

<https://stackoverflow.com/questions/21918802/problems-filtering-django-datetime-field-by-month-and-day>

次ページに続く

##### DecimalField

max\_digits 数値全体での桁数。

decimal\_places 小数点の位置を下から1で始まる数値で指定。＝小数の桁数とも言える。

例：(max\_digits=5, decimal\_places=2) なら -999.99 ～ 999.99 が有効値となる

（公式）<https://docs.djangoproject.com/ja/4.0/ref/models/fields/#decimalfield>

（公式）<https://docs.djangoproject.com/ja/4.0/ref/forms/fields/#decimalfield>

##### DurationField

timedeltaを返すField。2つのDateTime型を入力し、その差を格納する時に使う。

1:30（1時間30分）など最初から差の値を入力する際はTimeFieldの方が良い。

また、標準では入力された値を「分(minute)」扱いのtimedelta値で返すだけのTextFieldなので微妙。

最低でも以下のようにする必要がある

class TimeInputWidget(forms.TimeInput):

input\_type='time'

class MyForm(forms.Form):

duration = forms.DurationField(

widget=TimeInputWidget(),

)

2023-06-23 実験

カスタムFieldクラスを作成する

class TimeDeltaField(forms.Field):

widget=TimeInputWidget()

次ページに続く

ライブラリレファレンス：Field継承

##### FloatField

myFloat = models.FloatField(

default=0,

[validators.MinValueValidator(0.001),

validators.MaxValueValidator(0.999)])

)

（参考）[https://qiita.com/okoppe8/items/a1149b2be54441951de1#](https://qiita.com/okoppe8/items/a1149b2be54441951de1#浮動小数点floatfield)浮動小数点floatfield

##### FileField

ファイルを扱う。formでは，ファイル選択ダイアログを使用する。

view側でのファイル名取得方法

csv\_file = request.FILES[‘csvfile’]

reader = csv.reader(csv\_file) # ついでにcsv読み込み　import csvが必要

##### IntegerField

PositiveIntegerFieldも存在する。（正の値のみを扱う）

仕様概要

・1~0, -, .(ドット) 以外は入力できない

そもそも、htmlのinput type=”number” の仕様なのかもしれない

プロパティ

min\_value 入力可能な最低値を指定。

補足：htmlのinputの属性に適用される為、htmlフォームにて入力不可となる。

##### 自作フィールド

###### 概要

###### 具体例

[ models.py]

from django.db import models

class MeasurementField(models.FloatField):

description = “単位付きのfloat値を扱うクラスです”

def \_\_init\_\_(self, \*args, \*\*kwargs)

self.unit = kwargs.pop(‘unit’)

class Book(models.Model:

weight = MeasurementField(unit='g')

###### description

###### to\_python

データベースから取得した値をpythonのオブジェクトに変換する為のメソッド。

テンプレート的な例

def to\_python(self, value):

if not value:

return None

try: # tryを利用した値変換が推奨される

pass # 実際にはここに変換処理を記述

except:

raise ValidationError()

return value

###### from\_db\_value

データベースから値を取得する際に呼び出される。

###### db\_type

db.model.Fieldを継承する際に必要

###### validate

form.Fieldを継承する際に必要

###### clean

form.Fieldを継承する際に必要

#### クエリセットAPI

##### 概要

クエリが発行されるタイミング

・forループなどイテレーションが開始された時

（補足：後述するselect\_relatedに深く関連）

・[]を使ってスライスした時

・オブジェクトを直列化した時

・オブジェクトをREPL（対話型評価環境）やprintで表示した時。

・len()でサイズを取得した時。

・list()で強制的にリストに変換した時。

・bool()で強制的にboolにした時。

モデルのリレーションに気を付けないと想定外にクエリの数が増えてしまう可能性がある為注意。

##### all()

User.objects.all()

User.objects.all()[:10] # 上位10レコードのみ

・結果が0件でも例外は発生しない。

・必要に迫られるまではクエリが発行されない。（後述するfilterも同様）

##### aggregate()

各種集計を**dictで**返す。

Sum（合計）

Product.objects.aggregate(Sum(‘price’))

# > { ‘price\_\_sum’ : 1234 } このような値が返る。keyの名前はmyfiled\_\_sumとなる

Product.objects.aggregate(total=Sum(‘price’))

# > { ‘total’ : 1234 } dict型のkeyの名前を指定する事ができる

他の集計メソッド

from django.db.models import Avg, Count, Max, Min, StdDev

情報元：<https://docs.djangoproject.com/ja/4.0/ref/models/expressions/#aggregate-expressions>

情報元：<https://docs.djangoproject.com/ja/4.0/ref/models/querysets/#aggregation-functions>

##### annotate ()

ForeignKeyなどでリレーションしているモデルを、フィールドを追加する事で返す。

annotate（注釈）の意味通り、補足情報を追加する

注意点：ForeignKeyで**参照されている側のモデル**でメソッドを実行する。

Sum（合計）

Count

book\_count = Book.objects.annotate(authors\_count=Count('authors'))

songs\_count\_by\_singer = Singer.objects.annotate(number\_of\_songs=Count('songs'))

GroupByしたい場合：

task\_hour\_by\_employee = Task.objects.select\_related().values('employee') \

.annotate( Sum('task\_time')).order\_by('total').reverse()

関連オブジェクトの最初のインスタンスをannotateする例

subqueryが結果を返さない場合はNoneになる

subquery = Book.objects.filter(book\_type=OuterRef('pk'))

queryset = BookType.objects.annotate(first\_book\_price=Subquery(sub\_book\_type.values('price')[:1]))

##### 共通項目（集計関数）

参考：django.db.models

###### Func

データベース関数（COALESCE やLOWER）を利用する

あくまで、データベースエンジンで関数を作成している点に注意

PurchaseHistory.objects.annotate(

# 年でフィルタするため、YEAR 関数を使って購入日の年のみを算出する

year=Func(

F('purchase\_date'),

function='YEAR',

template="%(function)s(%(expressions)s)",

output\_field=IntegerField(),

),

関係ありそうなデータベース関数

LOWER

2023-06-27こんなことも出来るらしいが本当だろうか（ChatGPT情報なので怪しい）

def price\_include\_tax(price):

return price \* 1.10

price\_list = Book.objects.annotate(price\_incl\_tax=Func(

F('price'), function=price\_include\_tax, output\_field=models.FloatField()

))

###### Case

条件に応じてannotateする値を変える

>>> Client.objects.annotate(

... discount=Case(

... When(account\_type=Client.GOLD, then=Value('5%')),

... When(account\_type=Client.PLATINUM, then=Value('10%')),

... default=Value('0%'),

... ),

例

queryset = PurchaseItem.objects \

.select\_related('purchase\_request', 'project') \

.filter(project=proj) \

.annotate(quotation\_count=Count('quotations')) \ # 下のWhenで使うための値を生成しておく

.annotate(state=Case(

When(purchase\_info\_\_recieved\_date\_\_isnull=False, then=V('受領済')),

When(purchase\_request\_\_isnull=False, then=V('購入依頼済')),

When(quotation\_count\_\_gt=0, then=V('購入依頼済')),

default=V(None), output\_field=CharField())) \

.order\_by('item\_number')

公式：<https://docs.djangoproject.com/ja/4.0/ref/models/conditional-expressions/#case>

###### Cast

Django3.2以降で導入

from django.db.models import FloatField

from django.db.models.functions import Cast

author = Author.objects.annotate(

age\_as\_float=Cast("age", output\_field=FloatField()),

)

###### Coalesce

レコードの列の値に初期値を与える事でNullの可能性が有る場合に例外を防ぐために使う

hours\_sum = Book.objects

.aggregate(price\_without\_zero=Coalesce('price', 0))

###### Concat

author\_names = Book.objects \

.annotate(author\_full\_name = Concat('first\_name', V('-'), (F('family\_name')))

Fはあっても無くてもよいらしい

###### Count

最もシンプルな例

queryset = Author.objects.annotate(book\_count=Count('book'))

サブクエリを用いた例

subquery = Book.objects.filter(author=OuterRef('pk'))

.filter( #任意の条件# ) \

.annotate(count=Count('pk')).values('count')

authors = Author.objects.annotate(

books\_count=Subquery(subquery, output\_field=models.IntegerField()) # ()まで付ける！

)

補足：SQLServerの場合はorder\_by+valuesによる明示的なGroupByが必要

ManyToManyFieldだとそのまま

authors \_count = Count('author')

###### Exists

最もシンプルな例

authors\_with\_books = Author.objects.annotate(

has\_books=Exists(

Book.objects.filter(author=OuterRef('pk'))

)

)

Subqueryを使った例

subquery = Author.objects \

.filter(book = OuterRef('pk')) \

.filter(book\_published\_date\_\_isnull = False)

queryset = Book.objects \

.annotate(published = Exists(subquery))

補足：SQLServerの場合はorder\_by+valuesによる明示的なGroupByが必要

注意点

・複数の型で計算する場合、output\_field引数の指定が必要。

例：price\_per\_unitがDecimal、quantityがIntegerとする。

from django.db.models import Sum, Value, F, DecimalField

…（略）

.aggregate(total=Sum(F('price\_per\_unit') \* F('quantity'), output\_field= DecimalField()))

補足

逆参照でexistっぽい事をする例

authors\_with\_books\_existence = Author.objects.annotate(

books\_existence=Case(

When(book\_\_isnull=False, then=Value('exists')),

default=Value('none'), output\_field=models.CharField(),

)

)

備考

自作のAggregate関数

<https://docs.djangoproject.com/ja/4.0/ref/models/expressions/#creating-your-own-aggregate-functions>

###### Subqueryクラス

書式： Subquery(queryset, output\_field=None)

subquery = Book.objects.filter(author=OuterRef('pk')) # OuterRef 本クエリで主キーとなっている値を代入する

.filter( #任意の条件# ) \

.annotate(count=Count('pk')).values('count')

authors = Author.objects.annotate(

books\_count=Subquery(subquery, output\_field=models.IntegerField())

)

補足：SQLServerの場合は**明示的なgroupbyが必要**

subquery = Book.objects.filter(author=OuterRef('pk')) # OuterRef 本クエリで主キーとなっている値を代入する

.filter( #任意の条件# ) \

orderby( 'author' ).valuesr( 'author' ) \ # この行で明示的なGroupbyを行っている

.annotate(count=Count('pk')).values('count')

authors = Author.objects.annotate(

books\_count=Subquery(subquery, output\_field=models.IntegerField())

)

|  |  |  |  |
| --- | --- | --- | --- |
|  | １対多 | 多対多 |  |
| subquery | 多の方  ForeignKeyフィールドに対して行う |  |  |
| 本クエリ | １の方  ForeignKeyの被参照Modelのクエリ |  |  |

公式：<https://docs.djangoproject.com/ja/4.0/ref/models/expressions/>

Exist以外でレコードを1件にしたい場合（SQLでいうTOP 1 とかができる）

tsk\_list = Author.objects \

.annotate(booksr=Subquery(sub\_query.values('cost')[:1]))

###### 集計関数共通の注意点

・グループ化して集計

sumup = Books.objects \

.values('author') \ # GROUP BYされる

.annotate(price\_average=Average("price")) \

・複合型の場合はフィールド名（エイリアス名）が必要

例：

.aggregate(Sum(F('book\_\_price') \* Value(3)))

>> Complex aggregates require an aliasがraiseする

##### add()

ManyToManyFieldのデータ追加

詳細：本ドキュメント内のManyToMany

##### set()

ManyToManyFieldのデータ更新

##### bulk\_create

一括追加

##### bulk\_update

Django2.2から採用。

##### create系まとめ

SQLでいうINSERT（一部UPDATEの機能も包括）

|  |  |  |
| --- | --- | --- |
|  | 概要 | 備考 |
| create() | フィールド値を直接指定して新規作成 |  |
| save() | インスタンス経由で新規作成／更新 |  |
| get\_or\_create() | あれば取得、無ければ作成 |  |
| bulk\_create() | 一括INSERT |  |
| bulk\_update() | 一括UPDATE | django2.2からサポート |

補足：ModelFormを使う場合

def post(self, request, \*args, \*\*kwargs):

form = BookForm(request.POST)

instance = form.save(commit=True) # commit = Falseにするとインスタンスだけ取り出せる

補足

誤解しやすいmethod

|  |  |
| --- | --- |
| add() | ManyToManyFieldのデータ追加 |
| set() | ManyToManyFieldのデータ更新 |

##### create()

フィールド値を直接指定する事でレコードを新規作成

book = Book.objects.create(title=“test1”, price=1000)

（比較）save、update\_or\_create

##### save()

modelのインスタンスからレコードを新規作成(CREATE)／更新(UPDATE)

主キーが重複するレコードが存在する場合はUPDATE。それ以外はCREATEとなる。

user = User(username=‘admin’) # 新規作成（コンストラクタ）

user.email = ‘admin@example.com’

user.save()

user = User.objects.get(username=‘admin’) # 対象レコードを限定する

user.username = ‘root’ # フィールドの値を変更

user.save()

user.pk = None # Primary KeyをNoneにすると新規レコードとして扱われる

注意点

・基本的にこれら更新処理は「オートコミットモード」で行われる。1つの更新につき1トランザクションとなる為，基本的にロールバックはできない。

関連：transaction.atomicメソッド

関連：update

##### delete()

対象レコードのDELETE

user = User.objects.get(username=‘admin’)

user.delete()

Sample.objects.all().delete() # 全レコードの削除

##### exclude()

filter の逆。条件に一致するものを省く。

##### exists()

Book.objects.filter(title\_\_icontains=‘Django’).exists()

Post.objects.filter(like\_count\_\_gte=10).exists()

・True/Falseが返る

##### count()

レコードの件数を返す

Book.objects.all().count()

Book.objects.filter(title\_\_icontains=‘Django’).count()

##### filter()

SQLのwhereを実現させる為のもの

User.objects.filter(is\_active=True)

User.objects.filter(~Q(id=3)) # not

User.objects.filter(title=‘Django’, price=1000) # 2つ以上のAND条件の指定。, = ANDとも言える。

User.objects.filter(Q(title=‘Django’) | Q(price=1000)) # 2つ以上のOR条件の指定。Q(条件)｜ ※ | はパイプ

User.objects.filter(Q(title=‘Django’), Q(price=1000) | Q(price=2000)) # ANDとORの混合

q\_list = Q()

q\_list.add(Q(title=‘Django), Q.OR) # Qにはこんな使い方もある

queryset = User.objects.filter(q\_list) #

[(u.pk, u.name) for u in User.objects.filter().all()] # tupleのdict型を作成する

[[u.pk, u.name] for u in User.objects.filter().all()] # listのdict型を作成する

Product.objects.filter(size\_\_isnull=True) # null

Book.objects.filter(author\_\_name=‘sample’) # 外部キー

Book.objects.filter(price\_\_gt=1000) # 「 > 」Greater Than

Book.objects.filter(price\_\_gte=1000) # 「 >= 」Greater Than or Equal

Book.objects.filter(price\_\_lt=1000) # 「 < 」Less Than

Book.objects.filter(price\_\_lte=1000) # 「 <= 」Less Than or Equal

Book.objects.filter(price\_\_in=[900,1000]) # IN

Book.objects.filter(date\_start\_\_range=[date\_min, date\_max]) # 範囲

Book.objects.aggregate(total\_price=Sum("price")) # 集計関数を利用する（後述）

・特にprice\_\_gtという名前のメソッドなどを用意する必要はない。field\_name\_\_gtで良い。

・結果が0件でも例外は発生しない。

（参考）<https://qiita.com/uenosy/items/54136aff0f6373957d22>

LIKE（検索）系

Book.objects.filter(title\_exact= 'Django') # 完全一致（大文字小文字**区別**）

Book.objects.filter(title\_iexact='Django') # 完全一致（大文字小文字区別**しない**）

Book.objects.filter(title\_contains='Django') # 部分一致（大文字小文字を**区別**） =LIKE UPPER(%値%)

Book.objects.filter(title\_icontains='Django') # 部分一致（大文字小文字を区別**しない**）=LIKE %値%

Book.objects.filter(title\_startswith='Django') # 前方一致（大文字小文字を**区別**）

Book.objects.filter(title\_istartswith='Django') # 前方一致（大文字小文字を区別**しない**）

Book.objects.filter(title\_endwith='Django') # 後方一致（大文字小文字を**区別**）

Book.objects.filter(title\_iendwith='Django') # 後方一致（大文字小文字を区別**しない**）

次ページに続く

～filter　つづき

応用編

正規表現

Book.objects.filter(title\_\_regex=r '[0-9]')

dictを使ったfilter

condition = {

'age': 20,

'gender': 'M',

}

Person.objects.filter(\*\*condition)

##### get()

Userオブジェクトを1件取得する場合。⇒通常、プライマリーキー（pk）を用いる。

User.objects.get(pk=2)

> User: tarou

・pkはPrimary Keyの略。

・1件も見つからない場合はmodel.DoesNotExist例外が発生する。

・複数行が返ると、model.MultipleObjectsReturned例外が発生

・filterと異なり、検索に使えるフィールドは１つ。

⇒ レコード有無が不明な場合、複数フィールドによる検索がしたい場合はfilterを用いる。

get\_or\_none

カスタムマネージャーを使う事で実現可能。

（補足）Django運営が実装していないと言う事は、この方法は**非推奨**、という事。

⇒ 有無が不明な場合はfilterを使う。

from django.db import models

class BaseManager(models.Manager):

def get\_or\_none(self, \*\*kwargs):

try:

return self.get\_queryset().get(\*\*kwargs)

except self.model.DoesNotExist:

return None

class Foo(models.Model):

objects = BaseManager()

情報元：<https://remotestance.com/blog/1496/>

##### get\_or\_create

データを取得、なければ追加

##### none

queryset = Book.objects.none() # 空のクエリセットを用意する

##### order\_by()

フィールド名のHash値で並び替えを行う。

Book.objects.all.order\_by (‘price’) # 昇順（小→大）

Book.objects.all.order\_by (‘-price’) # 降順（大→小） -を付ける

Book.objects.all.order\_by (‘price’,’publish\_date’) # 複数フィールドでのソート

##### query

クエリセットで発行されるSQL文を確認する

注意！以下の場合は例外エラーが発生

queryset = Project.objects.none()

print(queryset.query)

##### select\_related()

複数回SQLが発行されるような場面での**クエリの最適化**を行う。

・対一（OneToOneField，ForeignKey）の外部キーが対象。

・外部キーフィールドが存在するクエリセットオブジェクトから2回以上レコードを取り出す場合に有効。

例

e = Entry.objects.get(id=5) # DBへのアクセスが発生

b = e.blog # 再度アクセスが発生

こういうコードを以下の様に書ける。

# DBを叩く

e = Entry.objects.select\_related(‘blog’).get(id=5)

# 予め取得したオブジェクトを参照しているのでDBを叩いてない

b = e.blog

ちなみにselect\_relatedを繋ぎたい場合には以下のようにする

select\_related(‘foo’, ‘bar’)

補足：具体的にはINNER JOINを用いて参照先レコードを取得するクエリ文が発行される。

（参考；非常にわかりやすい）<https://mkai.hateblo.jp/entry/2018/11/05/100000>

（公式）<https://docs.djangoproject.com/ja/4.0/ref/models/querysets/#select-related>

補足

外部キーフィールドが存在すればselect\_relatedというわけでは無い。

以下の場合はクエリが1回しか発行されない。

for blog in Entry.objects.filter(title\_\_startswith=(‘hoge’)).all():

二重参照の場合のselect\_relatedも可能。

select\_related(‘book’)

# ↓

select\_related(‘book\_\_author’)

django\_debug\_toolbarなどでSQLを確認し、以下のようなクエリが複数発生していなかを確認する

SELECT \* FROM "book" WHERE "isbn" = ''12345'''

（関連）prefetch\_related many-to-many、many-to-oneでつかう

Author.objects.prefetch\_related('book\_set').get(pk=4)

formを利用したテンプレートで使う場合

結構苦労したので、備忘録として残す。2022-03-04

[ forms.py ]

from django import forms

class BlogChoiceField(forms.ModelChoiceField): # ModelChoiceFieldを継承

def label\_from\_instance(self, obj): # label\_from\_instanceをオーバーライド

return "%s" % obj[‘blog \_\_title’] # 任意の形式を指定

class BlogSearchForm(forms.Form):

blog\_choice = BlogChoiceField( # 上で定義したカスタムクラス

label=‘ブログ’,

queryset = Entry.objects.**select\_related**(‘blog’) \ # 厳密にはdict型になる

.values(‘blog\_\_id’, ‘blog\_\_title’) # 外部キーの要素なのでtable\_\_fieldnameとなる

)

ただ、下のように、２回クエリを発行するのが無難だと思われる

[ forms.py ]

inner\_query = Entry.objects.filter(blog \_\_isnull=False).values(‘blog \_\_id’)

blog\_list = Blog.objects.filter(id\_\_in=inner\_query)

blog\_choice = forms.ModelChoiceField(

queryset = blog\_list,

)

（参考）<https://just-python.com/document/django/orm-query/select_related-prefetch_related>

##### prefetch\_related()

対多の外部キーでクエリの最適化を行う場合に用いる。

例えば下の例の場合、ループ１回につき1回クエリが発行される

for p in Post.objects.all():

print(p.author.name)

下のようにすると、クエリは２回になる

for p in Post.objects.prefetch\_related("author").all():

print(p.author.name)

こんな感じでつかう（らしい）

queryset = Book.objects.select\_related(‘publisher’).prefetch\_related(‘authors’).order\_by(‘publish\_date’)

##### update

基本的にはfilterの後ろに付けて一括更新を行う

Entry.objects.filter(pub\_date\_\_year=2007).update(headline='Everything is the same')

関連：save（pkが一致するレコードが存在する場合はUPDATEになる）

##### update\_or\_create()

データを更新、なければ追加

構文：update\_or\_create(defaults=None, \*\*kwargs)

defaults 更新（update）情報を { field: value } の形式のdict型で指定する。

kwargs 対象となるレコードを特定する為の引数（群）

[ views.py ]

author = get\_object\_or\_404(Author, pk=1)

obj, created = Book.objects.update\_or\_create(

author=author, # 対象のレコードを絞り込む条件（\*kwargs；つまり、複数条件が可能）

defaults={ # 更新する情報

‘price’: 1000,

}

)

検索結果やインスタンスをそのまま使いたい場合はdict型に変えてくれるvaluesメソッドを用いる。

query\_set.first().values()になりそうな気がするが、query\_set().values().first()が正解。

Book.objects.update\_or\_create(

author=author, # 条件

defaults = query\_set.values().first() # 検索結果やインスタンスを使う場合はvaluesメソッドを用いる

（公式）<https://docs.djangoproject.com/ja/4.0/ref/models/querysets/#update-or-create>

##### values() / values\_list()

いずれも複数のフィールド名とその値を選択して取得する

比較

|  |  |  |
| --- | --- | --- |
| メソッド | 戻り値 | 備考 |
| values | dict[str, 選択したフィールド] のlist | primarykeyがdict型のkeyになる |
| values\_list(flat=False) | 選択したフィールドのタプルのlist |  |
| values\_list(flat=True) | 選択したフィールドのリスト | フィールドは１つしか選べない |

補足：戻り値は厳密には ValueQuerySet[MyModel, フィールド]となる

values

自身のテーブル（=class）のフィールドはfiledname，外部キーのフィールドはtable\_\_fieldnameといった形。

queryset = Book.objects.select\_related(‘author’).values('price', 'author\_\_description')

book\_dict = list(Book.objects.select\_related('author').values('price', 'author\_\_description'))

補足：values()だと、primarykeyがdict型のkeyになる（らしい）

自分好みのdict型を生成したい場合は以下の様にする

input\_dict = { book.pk : {

'title' : book.title

'author' : book.author.name # 外部キーのフィールドにアクセス。

} for book in Book.select\_related(‘author’).all() } # 外部キーのフィールドを使わない場合はBook.object.all() で良い

values\_list()

引数

flat:bool

True：値（通常pk）を１つ指定し、[pk, dict]

False：値を**tupleで返す**。

#### ロギング

#### Djangoテンプレートエンジン

別名：DTL；Django Template Language

##### 概要

Djangoのhtmlテンプレートエンジン

概要

|  |  |  |
| --- | --- | --- |
| 項目 | 概要 | 備考 |
| タグ | {% %} で区切って特殊な処理をする |  |
| フィルタ | パイプ（|）で渡して値を変更する | 自作のカスタムフィルタが作成可能 |
|  |  |  |

補足

Jinja2に類似しているが別のテンプレートエンジン。

むしろJinja2はDjangoのテンプレートエンジンに影響を受けて開発されたらしい

比較

|  |  |  |
| --- | --- | --- |
|  | Django | Jinja2 |
| 設計思想 | 安全性を重視 | Expressiveness？ |
| 対象テキスト | html | html，JSON，CSV，YAML，XML |
|  |  | パフォーマンスが高速 |

公式：<https://docs.djangoproject.com/ja/4.2/ref/templates/language/>

Djangoテンプレートエンジン

##### 変数

{{ arg\_name }} # 変数を使う

{{ user}} # userモデルのインスタンス。\_\_str\_\_メソッドの結果が表示される

クラス内のメンバの扱いは，以下のように書くと以下のような優先順位で行われる。

{{ arg.field }} # インスタンスのメンバ変数を使う

arg[ 'field' ] dictとして照合

arg.field 属性値（メンバ）として照合

arg.field() メソッドを呼び出して照合

arg[field] list 又はtupleとして照合

例

arg1.0.field # list の0番目の field を取得（同名のメンバが無ければ）

arg.table.field 外部キー（ForeignKey）テーブルの列を照合

他に使える変数

request リクエストオブジェクト

user サイトにアクセスしているユーザー

perms サイトにアクセスしているユーザーのパーミッション

messages フラッシュメッセージ

args contextのkeyに追加した変数名

注意点

・() や [] は使えない。

with

DTL内で変数を定義する

以下のどちらでも良いらしい

{% with arg="hoge" %}

{% with "hoge" as arg %} 古い書き方らしいが、まだ有効

add

数値の加算を行う

Djangoテンプレートエンジン

##### タグ

###### 基本

{% %} で囲み，様々なプログラム的処理を行う。

{% if %}

{% if user.is\_superuser %}

システム管理者です。

{% elif user.is\_staff %} # {% if user.username|length < 3 %} など，フィルタと併用も可能

スタッフです。

{% else %}

一般ユーザーです。

{% endif %}

具体例

{% if user.is\_authenticated %}

{% if my\_list | length > 0 %}

公式：<https://docs.djangoproject.com/en/4.0/ref/templates/builtins/#if>

and

注意点

% if ではカッコ()が使えない

orと同時に使う際にはandが優先される

{% if a == b and c == d or e %} # 下の行のように解釈される

(a == b and c == d) or e)

その為andとorを複合する場合は、2行以上に分けるのが無難。（可視性の観点で）

{% if a == b %}

{% if c == d or e %}

in, not in

is not

is not None はどう書く？

（公式）<https://docs.djangoproject.com/ja/4.0/ref/templates/builtins/#if>

（参考）<https://arakan-pgm-ai.hatenablog.com/entry/2020/01/22/000000>

Djangoテンプレートエンジン

{% ifequal %}

{% ifequal val ‘true’ %}

val is true.

{% endifequal %}

Djangoテンプレートエンジン

{% for %}

{% for book in book\_list %}

{{ book.title }}

{% empty %} # 値の定義がされていないメンバがあった場合（任意）

ありません。

{% endfor %}

forループ内で使える変数

forloop.counter 1から始まるループカウンタ

forloop.counter0 0から始まるループカウンタ

{{ forloop.counter|divisibleby:2|yesno:'even,odd' }} {# 奇数偶数の判別 #}

<tr class="{{ forloop.counter|divisibleby:2|yesno:'table-light,table-dark' }}"> {# テーブル行を交互に #}

revcounter 残りのループ回数（そのループを含む）

revcounter0 残りのループ回数（そのループを除く）

forloop.first 最初のループか否か（真偽値）

forloop.last 最後のループか否か（真偽値）

parentloop 親ループ（入れ子のループの場合にのみ使用可能）

reversed 逆順ループ

{% for value in values reversed %}

<p>{{ value }}</p>

{% endfor %}

{% empty %} リストの中身が空だった場合

{% for member in members %}

<div>{{ member.name }}</div>

{% empty %}

<div>登録されているメンバーはいません</div>

{% endfor %}

{% ifchanged %}

前のループと違う値であった場合は出力

{% ifchanged %}

{{ book.isbn }}

{% endifchanged %}

regroup/grouper

regroup リストから特定の属性をキーとして新しいグループを作る。

{% cycle %} / {% recycle %}

指定した値を順番に出力するためのもの

{% for book in books %}

<div class="text-{% cycle 'danger' 'primary' 'success' %}">{{ book.title }}</div>

{% endfor %}

{% url %}

URLconf(url.py) の逆引き。

構文：{% url *reverse\_ulr\_name* [ *param*=*value*…] %}

構文：{% url *reverse\_ulr\_name* [ *arg1*…] %}

{% url ‘accounts:register’ %} <!-- accountsアプリのurl.pyのname=‘resister’ を取得 -->

<a href="{% url 'book\_detail' id=999 %}"> <!-- パラメータ付きのURLの取得　直接指定 -->

<a href="{% url 'book\_detail' id\_from\_query %}"> <!-- パラメータ付きのURLの取得　変数を使用 -->

<a href="{% url 'book\_detail' %}"?title='test'> <!-- クエリ文字列を追加-->

<!-- ボタンの場合 -->

<button onclick="location.href='{% url 'book\_detail' %}?title ={{ title }}'">戻る</button>

{% include %} 外部テンプレート（ヘッダ，フッタなど）の読み込み

{% include "header.html" %}

{% include 'header.html' %} {# クオーテーションはシングル／ダブルどちらでも良い #}

{% include template\_name %} {# 変数を用いた動的な処理も可能 #}

{% include "header.html" with val1=1 val2=2 %} {# テンプレートに変数を渡す #}

[ header.html ]（例）

<header class="sticky-top">

<ul>

</ul>

</header>

公式：<https://docs.djangoproject.com/en/4.2/ref/templates/builtins/#include>

{% autoescape %}　エスケープ制御。

{% endautoescape %} と共に使用する。

{% static %}　静的ファイルの配信URL取得

{% load static %} と共に使用する。（こちらが先）

{% static “images/logo.png” %}

Djangoテンプレートエンジン

{% extends %}

テンプレートとしているhtmlファイルのオーバーライド。{% block %} と共に使う。

この{% extends %} タグは.htmlファイルの最初に記述する必要がある。

基本

[ base.html ]

Hello <!-- ここまでが親htmlで記述する範囲 -->

{% block hoge %} <!-- ここからが子htmlで記述する範囲 -->

{% endblock %}

[ child.html ]

{% extends “base.html %} <!-- /templateas/base.htmlを使う -->

{% block hoge %} World {% endblock %}

⇒ Hello World となる。

また，block.superという特別な変数を用いる事で，以下のようにもできる。

[ base.html ]

{% block base%} Hello {% endblock %} <!-- 親htmlの同名ブロックを用意しておく -->

[ child.html ]

{% extends “base.html %}

{% block base%}{{ block.super }} World {% endblock %} <!-- 変数なので {%%}では無く{{ }}

⇒ 同じくHello World となる。

[Django公式：tags & filters](https://docs.djangoproject.com/ja/1.9/ref/templates/builtins/)

情報）<https://qiita.com/nachashin/items/d3f9cd637a9cecbda72c>

Djangoテンプレートエンジン

###### ループ、list系

dictやlist

[ views.py ]

from django.shortcuts import render

def sample\_page(request):

context={

‘sample1’: [1, 2, 3],

‘sample2’: [ [4, 5, 6], [7, 8, 9] ],

‘sample3’: {‘a’:10 , ‘b’: [11, 12, [13, 14, 15]]},

}

return render(request, ‘sample\_page.html’, context)

[ テンプレート（.htmlファイル）]

{{sample1.0}}

{{sample2.0.2}}

{{sample3.a}}

{{sample3.b.2.2}}

参考<https://kuma-server.com/one-element/>

first

falseでない最初の値を取得

{% firstof var1 var2 %}

これと同等

{% if var1 %}

{{ var1 }}

{% elif var2 %}

{{ var2 }}

{% endif %}

firstof

与えられた引数の中から False でない最初の 1 つを出力

全てfalseの場合は指定した文字列を出力

{% firstof var1 var2 ‘文字列’%}

公式： <https://docs.djangoproject.com/ja/4.2/ref/templates/builtins/#firstof>

##### フィルタ

{{ }} 内にはロジックは書けない。代わりに「フィルタ」を用いる

{{ 変数名 | フィルタ名 }} 又は {{ 変数名 | フィルタ名:”引数” }}

パイプで結合すると{{ 変数名 | フィルタ名1：”引数” | フィルタ名2：”引数” }} 又は {{ }}

{{ user.first\_name | default:"" }} # 未定義（値がNone, ‘‘, 0 など）の場合に既定値として適用する文字列

{{ user.first\_name | default\_if\_none:""}} # None（=NULL）の時だけ指定した文字列に変換

{{ user.username | length }} # 文字列長

{{ book.price | floatformat:2 }} # 小数点の桁数。

{{ book.description | safe }} # < > ‘ “ & などのエスケープを無効にする

{{ user.last\_login | date:"Y-m-d H:i:s" }} # 2019-04-01 12:00:00　などになる

{{ book.description | linebreaksbr }} # \nを<br>に変換する。

{{ book.description | urlize }} # URLやメールアドレスでリンク表現に変換される

{{ book.description | truncatechars\_html:5}} # 決められた文字列まで切り詰め，残りは…に変換。

注意点

・「 | 」の前後にはスペースを入れない方が良い。（上の例は見やすさの為入れている）

・フィルタへの引数を示す：の前後にはスペースを入れられない。

公式：<https://docs.djangoproject.com/ja/4.2/ref/templates/builtins/>

公式：<https://docs.djangoproject.com/ja/4.2/ref/templates/builtins/#built-in-filter-reference>

###### 自作フィルターを作成する

ディレクトリ構成

myproject\

├ myapp\

│ ├ templatetags \ 新規作成

│ │ ├ \_\_init\_\_.py 中身は空で良い

│ │ ├ filters.py このファイル名は何でも良い

│

[ templatetags/filters.py ]

from django import template

register = template.Library()

@register.filter("lower")

def test(value, arg):

return value.replace(arg, ‘‘)

@register.filter("category\_field")

def category\_field(value, arg):

return value[(‘category\_%s’ % arg)]

[ テンプレート（.htmlファイル）]

{% load filters %} <--! フィルターの読み込み -->

<td>{{ form.fields.category\_3.label | test:"0" }}</td>

value arg

argは任意。（valueのみでも良い）

補足

filterかsimple\_tagかの違いは大きくの２のポイントがある。

・関数の引数が2つ以下か。（filterは引数を1つか2つにする必要がある）

・template変数を引数などで使いたいか（simple\_tagの方ではtemplate変数の値を利用する事ができる）

（公式）<https://docs.djangoproject.com/ja/4.0/howto/custom-template-tags/>

##### フォーム関連

便利なメソッド

as\_table

formの内容を<table>タグで囲い，一気にテーブルに変換してくれる。

テンプレート（.htmlファイル）

<table>

{{ form.as\_table }}

</table>

as\_p

すべて表示。エラーメッセージもすべて表示される。

as\_ul

formの内容を<ul>タグ（リスト）で囲います。

問題点

個別にhtml属性を設定したい場合などに困る。

⇒ 結果的に{%for%} でformのfieldをループさせるのが一番良い

formのactionタグ

htmlのformタグのaction属性でurlを指定する事もできる

action="{% url ‘app:delete\_memo’ memo.pk %}">

add\_class

CSSクラスを追加

構文：{{ form.some\_field | add\_class: "クラス名" }}

id\_for\_label

label\_tag

errors

フィールドに関連したエラーメッセージ

widget\_type

以下のような事ができる

{% for field in form %}

{% if field.widget\_type == ‘checkbox’ %}

# render one way

{% else %}

# render another way

{% endif %}

{% endfor %}

as\_hidden

非表示にする。

例えば、以下の様に値は見せたいが、変更を許可したくない場合などに用いる

<table class="table table-borderless">

<tbody>

<tr>

<td>{{ form.book\_title.label }}</td>

<td>{{ form.book\_title.value }}</td>

{{ form.book\_title.as\_hidden }}

</tr>

</tbody>

<table>

公式情報：<https://docs.djangoproject.com/ja/4.0/ref/forms/api/>

関連：forms.HiddenInput()

is\_hidden hiddenのフィールドかどうか

##### その他

コメント

{# --- 入力フィールドごとのエラーメッセージ（最初のエラーのみを表示） --- #}

divisibleby(3) nで割り切れるか

資料

<https://docs.djangoproject.com/ja/4.0/ref/templates/builtins/>

リストの数(len)でhtmlの内容を変えたいとき

{% if object\_list | length > 1 %}

breadcrumb\_block

bool値に対応

{{ is\_expired|yesno:"期限切れ,有効,None" }}

#### テンプレート(html)

##### 概要

htmlの文字列に変換する前に，modelなどからの変数を加工したり，繰り返したりしてメンテナンス性を良くするシステム。DTL；Django Template Languageというpython**風**の記法で { } % | : # を使う。

注意

inputで同名のnameが２つ以上ある場合は、後ろ側が優先される（Htmlの仕様？）

補足

jinja2を利用する事もできる（らしい）　2022-03-25

##### ファイル構成

実はtemplatesディレクトリはどこに作成しても良い。実際には以下の２通りが多い。

方法１

myproject

├ templates # プロジェクト直下に配置する

│ └ myapp # アプリごとにディレクトリを作成する（推奨）

[ setting.py ]

TEMPLATES = [

{

‘DIRS’: [], # 各appごとに置く場合

‘DIRS’: [os.path.join(BASE\_DIR, ‘templates’], # templatesをmyprojectに直接置く場合

…

}

方法２

myproject

├ myapp

│ └ templates # 各

[setting.py]（必要部分のみ抜粋）

DIR = [BASE\_DIR, ‘templates’]

Web上ではどちらかと言うと方法１を推奨する意見が多い。が，結局のところ状況によると思われる。

また，Djangoのチュートリアルでは1,2両方の方法を使っている。

##### ajaxを使う

部分的な画面の更新を非同期で行うもの。

必要な基礎知識

|  |  |  |  |
| --- | --- | --- | --- |
| 区分 | 名前 | 概要 | 例 |
| hml - form | action | 送信先URLを指定 | <form action="https://example.com"> |
| hml - input | datalist | 入力中に動的に候補を表示  html5以上 | <datalist id="items">  <option value="MacBook">  <option value="MacBook">  </datalist> |
|  |  |  |  |

次ページに続く

[ テンプレート（.htmlファイル）]

{% block extrajs %}

<script src="{% static ‘js/author\_book\_ajax.js’ %}"></script> <!-- 別ファイルのJavaScriptを読み込む -->

{% endblock %}

{% block content %}

<!-- 資料として分かりやすくする為にinputを直接利用。普通はformとFieldを継承したフィールドを使うと思われる -->

<input type="text" id="id\_author" autocomplete=off placeholder="著者名">

<input type="text" id="id\_title" list="id\_book\_list" autocomplete=off placeholder="本のタイトル">

<datalist id="id\_author\_list">

{% for book in author\_list\_from\_view %} <!-- context変数を通じてviewから取得 -->

<option value="{{ book.title }}">

{% endfor %}

</datalist>

<datalist id="id\_book\_list"></datalist> <!-- html読み込みの段階では空にしておく -->

{% endblock %}

補足：formを使う場合はwidget.attrs = { ‘list’ : ‘item\_name\_list’} といった形で指定する

[ static/js/author\_book\_ajax.js ]

$(document).ready(function() {

$(‘#id\_author).blur(() => { // blur は「フォーカスを失った時」のイベント

$.ajax({

type: ‘GET’,

url: ‘api/author\_book/get/’, // .html（テンプレート）以外では逆引きが使えない点に注意

data:{ // クエリ文字列の設定

‘author’ : $(‘#id\_author).val(), // htmlのinputの入力値

},

dataType: ‘json’

}).done( response => {

$('#id\_book\_list').empty(); // リスト内を初期化

for (var bk of response.book\_list\_from\_json) { // for で回しつつ、<datalist>タグに要素を追加して行く。

$("#id\_book\_list").append(`<option value=${bk.author}>`); // 文字列内で変数を使いたいときは ` を使うらしい

}

}).fail(function(jqXHR, textStatus, errorThrown) { // このパラメータの意味は分からない 2022-04-28

console.log(‘fail’); // console画面への文字列出力

});

});

});

次ページに続く

[ urls.py ]

from django.urls import path

from . import views

app\_name=‘my\_app’

urlpatterns = [

path(‘api/author\_book/get/’, views.api\_author\_book, name=‘api\_author\_book \_get’),

]

補足：webページ表示ごに動的に変わる値ではない場合にのみ、URLパラメータも使用可能

[ views.py ]

def ajax\_book\_list\_with author\_get(request): # jsonを返すメソッド

author\_name\_from\_ajax = request.GET[‘author’] #

book\_list = Book.objects.filter(author\_\_name=author\_name\_from\_ajax)]

.values('pk', 'author', 'title') # dict型を返すクエリ（ValueQuerySet）を生成

return JsonResponse({ ‘booklist’: list(book\_list) }) # json変換可能な型list(dict) で返す

class BookSearchView (ListView):

model = Book

template\_name = ‘gyoumu/component/component\_in-project\_add.html’

def get\_context\_data(self, \*\*kwargs: Any) -> Dict[str, Any]:

context = super().get\_context\_data(\*\*kwargs) # 現在のcontextを取得

author\_list=Author.objects.all()

context[‘author\_list\_from\_view’] = author\_list

def get\_form(self, form\_class=None): # formクラスで対応しても良い

form = super().get\_form(form\_class)

form.fields['author'].widget.attrs={ 'list' : 'id\_author\_list', }

return form

…以下略

シンプル：<https://chuna.tech/detail/73/>

参考：<https://qiita.com/skokado/items/a25d64cafa3db791b283>　クラス式

##### 具体例

テーブルでそのまま編集する（未完成）

{% for rec in record\_list %}

<tr>

<td><input type=”text” value=”{{ rec.value }}”></td>

</tr>

{% end for %}

リンク

アンカーリンク

<a href=”/myapp/book/{{ bk.id }}”></a> <!-- 最初の / を忘れないように -->

⇒ project\_base\_url.com/myapp/book/1 などに対応する

ボタンを使う場合

##### エラー関連

templatesディレクトリ直下に403.htmlなどを作るだけ。

DEBUG=Falseの時に指定されたhtmlテンプレートが読み込まれる。

401 認証が必要

403 ページが見つからない

404 アクセス禁止

500 サーバー内エラー

##### シンプルタグ

ディレクトリ構成

myapp/

├ models.py

├ templatetags/

│ ├ \_\_init\_\_.py 中身は空で良い

│ ├ my\_simple\_tags.py

├ views.py

[ templatetags/my\_simple\_tags.py ]

from django import template

register = template.Library()

@register.filter

def multiply(value1, value2):

return value1 \* value2

[ html（テンプレート）]

{% load my\_simple\_tags %}

<span>掛け算:{{ num1 | multiply:num2 }}</span>

##### 特殊変数など

request.get\_full\_path 現在のフルurl

request.path 現在のurl

##### メモ

こんな方法もある

<form action="{% url 'gyoumu:component\_update' object.id %}?next={{next\_url}}" method="post">

ただし、UpdateViewなどではget\_success\_urlメソッドがある為に、あまり意味がない。

#### ミドルウェア

##### 主なミドルウェア

‘django.middleware.security.SecurityMiddleware’,

‘django.contrib.sessions.middleware.SessionMiddleware’,

‘django.middleware.common.CommonMiddleware’,

‘django.middleware.csrf.CsrfViewMiddleware’,

‘django.contrib.auth.middleware.AuthenticationMiddleware’,

‘django.contrib.messages.middleware.MessageMiddleware’,

処理の成否などを画面上に表示する「フラッシュメッセージ」を管理する

from django.contrib import messages

公式：<https://docs.djangoproject.com/ja/2.0/ref/contrib/messages/#enabling-messages>

参考１：<https://office54.net/python/django/display-message-framework>

参考２：<https://runebook.dev/ja/docs/django/ref/contrib/messages>

‘django.middleware.clickjacking.XFrameOptionsMiddleware’,

##### 自作ミドルウェア

\_\_init\_\_と\_\_call\_\_メソッドを実装させる事でミドルウェアは自作できる。

シンプルなミドルウェア

class SimpleMiddleware:

def \_\_init\_\_(self, get\_response): # サーバー起動時に実行される

self.get\_response = get\_sesponse

def \_\_call\_\_(self, request): # viewの関数が実行される度に呼び出される

response = self.getresponse(request)

return response

※django.utils.deprecation.MiddlewareMixinを継承する方法もあったが現在は非推奨。

views

補足：Djangoのviewsは他のMVC で言うところのController。情報を処理して，テンプレートに渡す。

ファイルパス：/**myapp**/views.py

def post\_list(request):

items = Item.objects.all() # objects なので注意

return render(request, ‘blog/post\_list.html’, { ‘args’: items})

render関数について

request クライアントPCからDjango（Htmlサーバー）へのリクエスト情報

以下のような取得方法がある。（但しformを利用した方が有益）

views.py

request.POST.get(‘username’)

request.POST.get(‘password’)

templateName 使用するテンプレート(.html) のファイル名。templateディレクトリからの相対。

context 省略可能。テンプレートに送る引数のlist

#### 自動テスト（ユニットテスト）

##### 概要

RequestFactoryはrequestを返す

Clientはresponseを返す。

補足：以下のようなソースをよく見かけるが、初期化の必要はない（下記コードは不要）

def setUp(self):

self.factory = RequestFactory()

self.client = Client()

##### チュートリアル

ディレクトリ構成（ファイル名、クラス名、メソッド名に命名規則がある；後述）

myproject

├ myapp\

│ ├ test\_minimal.py 単一ファイルの場合。通常tests.pyだが、命名規則に従う限り何でも良い。

│ ├ test\_fuga\ ディレクトリにする場合。通常tests/ だが、同上

│ │ ├ \_\_init\_\_.py モジュールが2つ以上になる場合は、ここでimportが必要

│ │ ├ test\_foo.py ここのファイル名も命名規則に準じる必要がある。

│ …

├ manage.py

最小限テストの例

[ myapp/test\_minimal.py ]

from django.test import TestCase

class MinimalTestHoge(TestCase):

def test\_minimal (self):

print(‘this is a minimal test!’)

[ test\_fuga.py（またはtest\_fugaディレクトリ直下に配置したtest\_foo.py）]

from django.test import TestCase

from myapp.models import Book

class Some**Test**AAA(TestCase):

def setUp (self): # 本クラスが呼び出された時に最初に実行されるメソッド

print(‘this is setup for a test!’) # クラス内のすべてのテストの実行前準備

def **test**\_method (self):

print(‘this is a test!’)

[ test\_fuga/\_\_init\_\_.py ]

from .test\_fuga import \* # test\_fugaディレクトリを使用する場合

実行

manage.py test myapp # アプリ単位

manage.py test myapp.test\_minimal # モジュール単位 モジュール名に関係なしにtests

manage.py test myapp.test\_fuga.SomeTest # テストクラス単位

manage.py test myapp.test\_fuga.SomeTest.test\_method # メソッド単位

次ページに続く

##### 詳細

命名規則

各モジュール、クラス、メソッドは命名規則に則った名前を付けないとテストとして認識してくれない。

|  |  |  |
| --- | --- | --- |
| 対象 | 命名規則 | 備考 |
| モジュール | test\*.py | ディレクトリの場合はtest\*/ |
| クラス | \*Test\* | django.test.TestCaseを継承する必要がある  unittest.case.Testcaseでも良いらしい |
| メソッド | test\* |  |

参考

テストを実行するとtest\_myproject（test\_ディレクトリ名） という名前で新しくデータベースが作成される。

その為、例えばtest\_db.pyで100件のレコードを追加しても、本番環境ではそのレコードは存在しない。

また、デフォルトではGot an error creating the test database: ‘text\_myproject’は既に存在しますというエラーメッセージが表示され、初期化するかどうかの対話がある。

yes：テスト用データベースを初期化 no：テストのキャンセル

タグ

@tag(‘integration’)

python manage.py test --tag=integration

python manage.py test --exclude-tag=integration

##### formのテスト

class FormTests(TestCase)

def test\_valid(self):

params = {

‘title’: ‘0123456’,

‘code’: ‘print(“hello”)’,

‘description’: ‘”説明”‘,

}

book = BookModel()

from = BookForm(params, instance=BookModel) # （補足）instanceはModelFormの時のみ有効

self.assetTrue(form.is\_valid())

##### viewのテスト

最もシンプルな例

Client式

from django.test import TestCase

from django.urls import reverse

from .views import BookView

class BookViewTest (TestCase):

def test\_edit\_book\_view(self):

url = reverse('book:edit\_book')

response = self.client.get(url)

self.assertEqual(response.status\_code, 200)

Factory式

from django.test import TestCase, RequestFactory

from .views import MyClassBasedView, my\_method\_based\_view

class BookViewTest(TestCase):

fixtures = [‘dump\_book.json’]

def setUp(self):

self.factory = RequestFactory()

def test\_edit\_book\_view(self):

url = reverse('book:edit\_book')

request = self.factory.get(url) # postの場合はself.factory.post(url)となる

# method式viewの場合

response = my\_method\_based\_view(requset)

# class式viewの場合

view = MyClassBasedView.as\_view()

response = view(request)

self.assertEqual(response.status\_code, 200)

違いのまとめ

|  |  |  |
| --- | --- | --- |
|  | Client式 | Factory式 |
| 概要 | 実際のhttpリクエストを想定したシミュレート。総合的にテスト。 | オブジェクト生成に焦点をあてる。  ユニットテストの観点に近い。 |
| ミドルウェアやデコレータ | 適用される | 適用されない |
| テスト速度 | やや遅め | 高速 |
| 用途 | 総合的にビューの振る舞いをテストする | ロジックの確認 |
| その他 | URLルーティングなども考慮される | context\_dataを取得できる |
|  |  |  |

次ページへ続く

～viewのテスト　つづき

contextを取得する方法 注意：Factory式じゃないと取得できないらしい

context = response.context\_data

dataを指定する方法

request = RequestFactory().get('shop/edit\_book/', data={

'book' : book.pk,

})

factory\_boyを用いた方法

表示文字列のテスト？

class TemplateTests(TestCase)

def test\_template(self):

template = Template(“{{ message | wcswidth}}”)

context = Context ({

“message” : “hello”,

})

actual = template.render(context)

self.assetEqual(actual, “5”) # wcswidthの結果の確認

##### TransactionTestCase（トランザクションのテスト）

（公式）<https://docs.djangoproject.com/ja/4.0/topics/testing/tools/#transactiontestcase>

##### 資料

公式：<https://docs.djangoproject.com/ja/4.0/topics/testing/tools/>

#### カスタムコマンド

##### 概要

以下のような事ができる

python manage.py mycommand

ディレクトリ構成

myproject\

├ myapp\

│ ├ management \

│ │ ├ commands\

│ │ │ ├ my\_command.py

[ myapp/management/commands/my\_command.py ]

from typing import Optional, Any

from django.core.management.base import BaseCommand

class Command(BaseCommand):

help=‘テストコマンド’

def handle(self, \*args: Any, \*\*options: Any) -> Optional[str]:

print(‘hello world’)

補足

class Command(BaseCommand):

def add\_arguments(self, parser): # 引数をセットする

parser.add\_argument('parameter', nargs='+', type=str)

（参考）<https://qiita.com/ekzemplaro/items/7349d43388abfd8ca2db>

（公式）<https://docs.djangoproject.com/ja/4.0/howto/custom-management-commands/>

#### Django内の変数の種類

##### 概要

|  |  |  |
| --- | --- | --- |
| 項目名 | 概要 | 説明 |
| context | htmlに渡す変数 | views.pyからhtmlにデータを渡す。 |
| request | htmlメソッド | getやpostなど、htmlメソッドに関連する値を保持。  クエリ文字列はここに含まれる |
| args |  | タプル型(tupple)の引数。 |
| kwargs |  | 辞書型(dict)の引数。  パスパラメータ（URL変数）はここに含まれる。 |
| session | セッション引数 | 一般的には（半）永続化するデータを扱う  settins.pyで指定した長さ保持。  保存先はデータベース、キャッシュ、ファイルなどを選択。  デフォルトはデータベースで保存の為、半永久的に保存。 |
| cache | アプリケーションキャッシュ | django.core.cacheモジュール。  一般的には一時的なデータを保存する。Redisを指定する事もできる。  TTL；Time to Liveの設定がある |
| クッキー |  |  |

次ページに続く

##### context

以下の方法で取得できる

def post(self, request, \*args, \*\*kwargs): # get でcontextを取得する事は少ない。

context = self.get\_context\_data(kwargs) #

self.context = super().get\_context\_data( kwargs ) # あまり使われない

context[‘value’] = 1 # 値の指定

テンプレート（.html）に渡す

（補足）テンプレート先ではTDLを用いる

contextを使う

context={

‘book\_list’ : Book.objects.all() # これでレンダリング先で{{ book\_list }} とすれば使える

]

予約語

'user' ログインユーザーとして使われる？

##### request（htmlメソッド関連；クエリ文字列）

取得方法は以下の２通り

get(self)

request = self.requet

get(self, request)

クエリ文字列

補足：クエリ文字列 ?val=123などの部分の事。URLパラメータとも呼ぶ

補足：htmlのinput でname=””指定した場合の入力値はこれになる。（idは入力値には関係ない）

[ views.py ]

url\_str\_val = request.GET.get('val' , None) # HtmlメソッドがGETの時にhtml文書内でid=valのinput値

url\_str\_val = request.POST.get('val', None) # HtmlメソッドがPOSTの時に（同上）。第二引数は既定値

url\_str\_val = request.GET.get('val') # 第二引数は省略可能。def.=None

url\_str\_val = request.GET['val'] # dict型なのでこれも可能。指定keyが無いと**例外をraise**。

補足

・**htmlフォーム**の入力データもこの方法で取得できる。

（補足）バリデーションなどの事を考えると、htmlフォームのデータはForm.cleaned\_dataで取得する方が良い。

##### kwargs（パスパラメータ）

keyword argumentsの意

例えばpath(‘edit/{int:id}’ の引数idを受け取る場合

方法１：kwargsから受け取り

def get(self, request, \*args, \*\*kwargs):

if “id” in kwargs:

id\_int = int(kwargs[“id”]) # 補足：kwargsはdict型

else:

方法２：同名の引数を用意

def get(self, request, **id**, \*args, \*\*kwargs):

print(id) # kwargs[“id”] と変換しなくてもそのまま使える。

引数のバリエーションが少ない時はこちらでも良い。

なお、特定のパラメータの受け取り方は以下の３通りある

kwargs['title'] # 指定したパラメータが存在しない場合はエラー

kwargs.get('title') # 指定したパラメータが存在しない場合はNone

kwargs.get('title', None) # def.値の指定

##### session（セッション変数）

特徴：他のviewに渡す事ができる

[ settings.py ]

INSTALLED\_APPS = [

‘django.contrib.sessions’,

]

MIDDLEWARE = [

‘django.contrib.session.middleware.SessionMiddleware’,

]

SESSION\_ENGINE = ‘django.contrib.sessions.backends.cache’ # 任意：永続化されないが高速になる。

[ views.py ]

from django.shortcuts import render, redirect

from django.urls import reverse

def passing\_from():

if request.method == ‘GET’:

request.session['my\_key']= json\_value # json にシリアライズ出来ないとダメらしい

return redirect(reverse(‘myapp:passing\_to’)) # passing\_toにリダイレクト

def passing\_to():

if request.method == ‘GET’:

my\_value = request.session.get('my\_key', None) # 値の取得方法。

my\_value = request.pop('my\_key', None) # 値を取得＋削除

return render(‘myapp/passing\_to.html’))

注意：dict.pop() は指定keyが存在しない時に例外をraise。その為、第二引数がほぼ必須。

次ページに続く

～session（セッション変数）　つづき

注意

Sessionのデータは、デフォルトではデータベースのdjango\_sessionというテーブルで管理している。その為、ブラウザを再起動しても、ホストマシンを再起動しても保持され続ける。

その性質を考慮して、必要なくなったkeyと値はできるだけ早く削除するように努める。

manage.pyで行う場合

python manage.py clearsessions # 期限切れのセッションのみ削除してくれる。cronに組込むのが良い。

# cached\_dbの場合、データベースに直接アクセスしてSQL(DELETE)で削除しても良い。

Django内のviewsなどで行う場合

del request.session['my\_key'] # 単一の値を削除

val = request.pop('my\_key', None) # 取得＋削除。

# 補足１：del だと指定したkeyが存在しない場合に例外がraiseするので、popで削除した方が無難。

# 補足２：第二引数無しで無効keyを指定すると例外をraise。

request.session.clear() # セッションの初期化（ログイン状態も初期化される）

request.session.clear\_expired() # 期限切れのものだけを削除

request.session.flush() # セッションとクッキーを初期化

[ settings.py ] （関連設定）

SESSION\_ENGINE = ‘django.contrib.sessions.backends.db’ # データベースで保持(def.)

SESSION\_ENGINE = ‘django.contrib.sessions.backends.cache’ # 永続化されないが高速。

SESSION\_ENGINE = ‘django.contrib.sessions.backends.cached\_db’ # キャッシュ＋データベース。

SESSION\_ENGINE = ‘django.contrib.sessions.backends.file’ # ファイルで保持

SESSION\_ENGINE = ‘django.contrib.sessions.backends.signed\_cookies’ # クッキーで保持（SECRET\_KEY を使う）

SESSION\_FILE\_PATH # backends.file の時の保存場所。def. = /tmp

SESSION\_COOKIE\_AGE = 3600 # session cookieの期間　単位[sec]　def.=2週間

SESSION\_SAVE\_EVERY\_REQUEST=True

SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE=False

（公式）<https://docs.djangoproject.com/ja/4.0/topics/http/sessions/>

（参考）<https://www.monotalk.xyz/blog/django-session_data-を定期的に削除する/>

##### cookies（クッキー）

requestから取得？？

request.COOKIES[‘key’]

request.get\_signed\_cookie(key, default=RAISE\_ERROR, salt=‘‘, max\_age=None)

def sample\_view(request):

# 値の取得

count = request.COOKIES[‘count’]

count = request.COOKIES.get(‘count’, 0) # このように書いてもよいらしい

# 値の設定

request.COOKIES[‘count’] = 1 # 2022-05-17 できなかった。max\_age=1とかになっている可能性はある

sessionから取得？

responseからの取得

webにはこの情報が一番多い。（但しスマートではない気がする）

HttpResponseのインスタンスは以下の様に取得する

class BookView(TemplateView):

def get(self, request, \*args, \*\*kwargs):

context = {

}

response = render(request, ‘myapp/shop\_top.html’, context )

return response

def post

response =

（公式）<https://docs.djangoproject.com/ja/4.0/ref/request-response/#methods-2>

##### cleaned\_data

htmlフォーム（form）からの入力データを受け取る

def post(self, request, \*args, \*\*kwargs):

context=self.get\_context\_data(\*\*kwargs)

form = BookForm(request.POST)

if not form.is\_valid(): # is\_validメソッドを実行するまではcleaned\_dataが無い

return render(request, ‘myapp/shop.html’, context) # というエラーが発生する点に注意！

date = form.cleaned\_data.get(‘date’, None) # ここで取得

注意

is\_validメソッド実行前には取得できない。

このcleaned\_dataでは外部キーの値は取得できない（らしい）。

##### args

キーワードなしで引数が投げられた場合にこちらが使用される。

2022-05-23具体的な使用用途が思い浮かばないし、利用されているところも見たことが無い

##### render/redirectの場合

redirectの例

return redirect(reverse('myapp:shop', kwargs={ ‘book\_id’: 1 }))

return redirect(‘myapp:shop’, book\_id = 1) # これではできないっぽい

補足：redirectの場合はcontextが使用できない。

補足：reverseを使う場合、kwargsが必要な場合はすべて指定しないと例外がraiseされる。

renderの場合

context = context = self.get\_context\_data(\*\*kwargs)

return render(request, 'myapp/top.html', context) # contextは省略可能

renderingする.htmlファイルを変えるだけ。画面遷移目的でrenderを使わない方が良い。

### 具体例

#### 命名規則

（編集中）

View

#### データベース関連

データベース操作（SQL）から見た際のDjango操作のメモ書き

（補足）詳細情報は本ドキュメント内【 django.modelの [Field継承クラス](#_型（Field継承）) 】を参照。

外部キー系（詳細は後述）

|  |  |  |
| --- | --- | --- |
| リレーション | フィールド | 備考 |
| 一対一 | OneToOne |  |
| 多対一 | ForeignKey |  |
| 一対多 | 無い | SQL99から配列がサポートされ、それも用いればできる。  が、**通常タブー**とされる。 |
| 多対多 | ManyToManyField | 中継テーブル（intersection table）が作成される |

##### 複合主キー

Djangoでは複合主キーをサポートしていない。

代案：複合ユニーク

from django.db import models

class Reservation(models.Model):

room = models.CharField(verbose\_name="部屋名", max\_length=20)

date = models.DateField(verbose\_name="予約日")

class Meta:

constraints = [

models.UniqueConstraint(fields=[‘room’, ‘date’], name=‘unique\_booking’),

]

但し、バリデーションの時にめんどくさくなる。

⇒ forms.pyのcleanメソッドで既に２つの列の重複値があった場合にValidationErrorをraiseする。

（参考）<https://mizzsugar.hatenablog.com/?page=1561192506>

（参考）<https://runebook.dev/ja/docs/django/ref/models/constraints>

##### INSERT

ModelFormを使う場合

new\_book = form.save(commit=False)

views.py で行う場合

#オブジェクト生成とレコードINSERTを同時に行う方法

book = Book.objects.create(title = 'Dr Jekyll and Mr Hyde ')

#オブジェクト生成後にINSERTを同時に行う方法

book = Book (title = 'Dr Jekyll and Mr Hyde ')

book.save() # ModelFormと異なり、save()メソッドはNoneを返す

##### UPDATE

book = Book.objects.filter(title = 'Dr Jekyll and Mr Hyde')

book.title = 'hoge'

book.save()

# 以下のようにすると、UPDATEでは無く、INSERTになるので注意

book.id = None # pkをNoneにする。自動でpkを割り振る（AutoIncrementになっている場合）

book.id = 2 # pkに別の値を代入する（複製される）

##### DELETE

##### FOREIGNKEY

class Book (models.Model):

team = models.ForeignKey(‘Publisher’, to\_field=‘id\_team’, on\_delete=models.SET\_NULL)

##### IN

Book.objects.filter(price\_\_in=[900,1000]) # IN

外部キーテーブルが関連する場合

inner\_qs = Blog.objects.filter(name\_\_contains=‘Ch’).values(‘‘name’) # 検索（IN）に使用する内部クエリ

entries = Entry.objects.filter(blog\_\_name\_\_in=inner\_qs) # Entry内のBlog外部キーを扱う場合

情報元：<https://docs.djangoproject.com/ja/4.0/ref/models/querysets/#in>

～データベース関連　続き

##### INNER JOIN

queryset = Book.objects.filter(author\_\_name=‘hoge’) # これでINNER JOIN句が生成される

# 複数行で分けたい場合は select\_relatedを使うと良い（らしい）

orders = Order.objects.select\_related("order\_detail").all() # INNER JOIN

# リレーションが1対1の場合は prefetch\_related

##### GROUP BY

queryset = Book.objects.values('author').annotate(book\_count=Count('pk'))

##### DISTINCT

MyModel.objects.distinct('author', 'pub\_date') # 但し、PostgreSQL以外には非対応

PostgreSQL以外での対応方法

MyModel.objects.all().values\_list('name', flat=True) \ # あらかじめ必要フィールドのみ抽出

.order\_by('name').distinct() # 先にorder\_byをしないと上手く機能しない

（情報元）<https://qiita.com/maisuto/items/de5d5ad77b5a502ecd03>

（公式）<https://docs.djangoproject.com/ja/4.0/ref/models/querysets/#pickling-querysets-1>

##### CONSTRAINTS

cleanはformでしか使えない。→代わりにSQLのCHECK制約を使う？

…Mixinがバリデーション？

<https://docs.djangoproject.com/ja/4.0/ref/models/constraints/>

##### SQLの直接実行

with connection.cursor() as cursor:

cursor.execute("DBCC CHECKIDENT ('gyoumu\_purchaseitem', RESEED, 0);")

<https://docs.djangoproject.com/en/4.0/topics/db/sql/#executing-custom-sql-directly>

#### 既存データベースの利用

##### 概要

注意点

authなどの管理用テーブルが必要。対応は以下のいずれか

- DB側に管理用スキーマを作る

- 管理用テーブルだけ別のデータベースを利用

##### 資料

公式：<https://docs.djangoproject.com/ja/4.0/howto/legacy-databases/>

#### 複数のデータベースを使う

##### 概要

##### チュートリアル

ディレクトリ構成

├ config\

│ ├ router.py auth系のrouter

設定ファイルに

[ settings.py ]

# 複数のデータベースの情報を記述

DATABASES = {

'primary': {

# 1つ目のデータベース設定

},

'secondary': {

# 2つ目のデータベース設定

},

}

# ルーターの情報を記述

DATABASE\_ROUTERS = ["config.routers.DatabaseRouter"]

ルーターを追加

[ config.routers.py ]

class SecondaryRouter:

route\_app\_labels = {'myapp1', ' myapp2'}

def db\_for\_read(self, model, \*\*hints):

return "'secondary'"

def db\_for\_write(self, model, \*\*hints):

return "'secondary'"

class DefaultRouter:

def db\_for\_read(self, model, \*\*hints):

return "default"

def db\_for\_write(self, model, \*\*hints):

return "default"

公式：<https://docs.djangoproject.com/ja/4.2/topics/db/multi-db/>

補足：Django2.2よりまえのバージョンの場合以降ならを用いた方法もある

class DatabaseRouter:

route\_app\_labels = # 互換性の為

def db\_for\_read(self, model, \*\*hints):

if model.\_meta.app\_label in self.route\_app\_labels: # 互換性の為

return

else:

return "default"

def db\_for\_write(self, model, \*\*hints):

if model.\_meta.app\_label == 'gyoumu': # シンプルなif文で書く場合

return

else:

return "default"

公式：<https://docs.djangoproject.com/ja/2.1/topics/db/multi-db/>

##### 資料

・メソッド

def db\_for\_read(self, model, \*\*hints):

def db\_for\_write(self, model, \*\*hints):

def allow\_relation(self, obj1, obj2, \*\*hints):

def allow\_migrate(self, db, app\_label, model\_name=None, \*\*hints):

・using

shellでの使用の際にはusingでデータベースを選択する事もできる

[ views.py ]

MyModel.objects.using('secondary').all()

・アプリごとにrouter.py

こんな情報があったが？

+ myappg\

+ router.py アプリに特化したrouter

・参考

<https://qiita.com/thino168/items/1100280a0ba191f5a8d8>

<https://bee3jp.com/django-で複数のｄｂを使用する/>

class SampleRouter:

"""

A router to control all database operations on models in the

auth and contenttypes applications.

"""

route\_app\_labels = {'auth', 'contenttypes'}

def db\_for\_read(self, model, \*\*hints):

"""

Attempts to read auth and contenttypes models go to auth\_db.

"""

if model.\_meta.app\_label in self.route\_app\_labels:

return 'auth\_db'

return None

def db\_for\_write(self, model, \*\*hints):

"""

Attempts to write auth and contenttypes models go to auth\_db.

"""

if model.\_meta.app\_label in self.route\_app\_labels:

return 'auth\_db'

return None

def allow\_relation(self, obj1, obj2, \*\*hints):

"""

Allow relations if a model in the auth or contenttypes apps is

involved.

"""

if (

obj1.\_meta.app\_label in self.route\_app\_labels or

obj2.\_meta.app\_label in self.route\_app\_labels

):

return True

return None

def allow\_migrate(self, db, app\_label, model\_name=None, \*\*hints):

"""

Make sure the auth and contenttypes apps only appear in the

'auth\_db' database.

"""

if app\_label in self.route\_app\_labels:

return db == 'auth\_db'

return None

#### 本番Webサーバー

##### 概要

python manage.py runserverで使うアプリケーションサーバーはあくまで開発用。

本番環境では、もっと動作の早いアプリケーションサーバーを使う。

リバースプロキシ

nginx

アプリケーションサーバー

WSGIに準拠したアプリケーションサーバーを選択する。

uWSGI 多機能でRudyやPHPなどにも使えるが、やや複雑。

gunicorn シンプルかつ必要十分な機能を持つ

mod\_wsgi Apache用。Django標準だったような？

NGINX Unit

※迷ったらuWSGIにするのが良いらしい

##### gunicorn

インストール

(env) $ python -m pip install gunicorn

(env) $ which gunicorn # インストールされたパスの確認。（インストール確認にもなる）

ディレクトリ構成（抜粋）

myproject

├ config django-admin startproject config ．でプロジェクトを始めた事を想定。

│ ├ wsgi.py WSGI設定が記述されいる。startprojectで自動的に作成される。

├ env python -m venv env で作成した仮想環境

│ ├ gunicorn gunicorn の実行モジュール（上のwhich gunicornで確認できるパス）

上の場合の環境の場合、以下のコマンドでgunicornサーバーは起動する

(env) $ gunicorn config.wsgi:application # ディレクトリパス．モジュール名：アプリケーション名

(env) $ gunicorn config.wsgi # アプリケーション名がapplicationの場合は省略可能

おまけ：Djangoのsetting.pyを切り替える方法

gunicorn --env DJANGO\_SETTINGS\_MODULE=config.settings.production myproject.wsgi

おまけ：gunicornサーバーのサービス化

.serviceファイルの一例

[Unit]

Description=gunicorn service for Django app

[Service]

Restart=always

[Install]

WantedBy=multi-user.target

情報もと：<https://www.petitmonte.com/python/django_gunicorn_nginx.html>

etc/nginx/sites-available/mysite

server{

listen 80;

listen [::]:80;

server\_name \_;

location / {

proxy\_pass http://127.0.0.1:8000/;

proxy\_set\_header Host $http\_host;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

.jpgファイルなどに読み取り権限が必要！

（参考）https://qiita.com/lv-kit/items/bf4af2da35a4281f403f

##### uWSGI

インストール

(env) $ python3 -m pip install uwsgi

##### Windows Server(IIS)

概要

wfastcgiを使う （参考）<https://pypi.org/project/wfastcgi/>

情報

参考１：<https://qiita.com/okoppe8/items/905d2c22e750261c62f0>

参考２：<https://ichi.pro/django-apurike-shon-o-windowsiis-sa-ba-ni-tenkaisuru-162379334044737>

参考３：<https://thinkami.hatenablog.com/entry/2015/10/26/070118>

##### waitress

###### 概要

wsgiサーバーはLinuxならuWSGIやgunicornなどがあるが、Windows環境ではこれのみ。

参考：<http://lutetiums.koushijima.com/python/20200823_django>

インストール。

pip install waitress

通常は以下のように使う。

waitress-serve --port=8080 config.wsgi:application

注意点

waitressはpip installを行った時のディレクトリをwaitressの実行ディレクトリとして設定するらしい。

その為、pip installを行う際のディレクトリに注意。また、ディレクトリ名を変更した時には実行ディレクトリの再設定が必要（2023-02-28やりかた不明；pip installし直す方が早い）

###### winswの設定

ディレクトリ構成

myproject

├ myservice.xml 実行ファイルと同名にしておく

├ myservice.exe <https://github.com/winsw/winsw/releases/>よりDLして名前変更

補足：要管理者権限

[ myservice.xml ]

<service>

<id>myservice.service</id>

<name>MyService</name>

<env name="WORKING\_DIR" value="/path/to/workingdir"></env>

<env name="VIRTUAL\_ENV" value="%WORKING\_DIR%/venv"></env>

<env name="PATH" value="%VIRTUAL\_ENV%\Scripts;%PATH%"></env>

<description>Djangoのプロジェクト/description>

<executable>waitress-serve</executable>

<arguments>--port=8080 config.wsgi:application </arguments>

<logmode>rotate</logmode>

</service>

補足

上記ファイルは下記コマンドに対応させる場合

waitress-serve --port=8080 config.wsgi:application

（参考）<http://lutetiums.koushijima.com/python/20200823_django>

###### インストール

[ command prompt ]

cd d:/workspace

myservice.exe install :: サービス登録

myservice.exe uninstall :: サービス削除

2023-10-19 これで良いらしい

[ powershell ]

.\myservice.exe start # 「.\」が無いとエラーになる

myservice.exe stop

以下、旧情報

[ powershell ]

# サービスの登録　補足：cmdのscコマンドの方が成功しやすい

New-Service -Name "MyDjangoService" -BinaryPathName "C:\src\MyService.exe"

# サービスの開始

Start-Service myservice.service

# 今登録したサービスを確認する

Get-Service -Name myservice.service

#Get-WmiObject win32\_service -Filter "name='MyService'" # これでも良いらしい？

# その他サービス関連のコマンド

Stop-Service -Name <サービス名> # 停止

Restart-Service -Name <サービス名> # 再起動

Set-Service -Name <サービス名> -StartupType Manual # 起動方式の変更

###### Django側の設定

静的ファイルのROOTを設定

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

ドキュメントルートの設定

[ urls.py ]

from django.urls import re\_path

from django.views.static import serve

urlpatterns = [

# 下の2 行のどちらでも良い

static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

re\_path(r'^static/(?P<path>.\*)$', serve,{'document\_root': settings.STATIC\_ROOT}),

]

#### キャッシュバックエンド

Rediesを使う？

#### バッチ処理

##### crontab

具体例

この例のポイント：

・仮想環境のpythonを使用

・sessionsと例外をraiseしやすい項目のレコードを省いてdump.jsonを作成

・settings.pyが分離されており、production.pyを指定。

・外付け（マウント）デバイスにdump.jsonを保存

・5分毎に実行

[ crontabのテキスト ]

\*/5 \* \* \* \* ~/my\_proj/env/bin/python3 ~/my\_proj/manage.py dumpdata --settings=config.settings.production --exclude sessions --exclude admin --exclude auth.permission --exclude contenttypes > /mnt/backup/dump.json

補足１

Djangoカスタムコマンドと併用する事で、DjangoのORMなどを使った処理も可能となる。

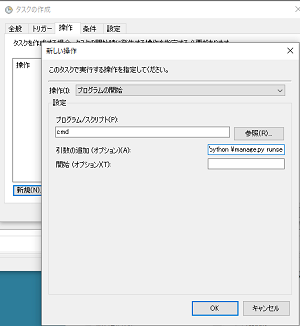
補足２

windowsサーバーの場合はcronが存在しないため、代わりにタスクスケジューラーを利用する。

引数をたくさん記述するのが大変（不可能かも？）なので、カスタムコマンドが必須。

参考：<https://qiita.com/hirockio2206/items/04a509b2aaebd3436a7a>

##### タスクスケジューラ(windows)

概要

Windowsにはcronが無いので代わりにこれを用いる。

スタート > Windows管理ツール > タスクスケジューラ

（ファイル名を指定して実行 > taskschd.mscでも良い）

操作：プログラムの開始

プログラム／スクリプト：cmd

引数の追加(オプション)(A)：python manage.py my\_cmd

（但し、絶対パスで記述！）

例：D:\src\my\_proj\venv\Scripts\python D:\src\my\_proj\manage.py

##### django-background-tasks

pip install django-background-tasks

INSTALLED\_APPS = (

# ...

'background\_task',

# ...

)

<https://django-background-tasks.readthedocs.io/en/latest/>

#### YesNo確認

・JavaScript(Vanilla)でやる方法

<script type="text/javascript">

function delete\_confirm(){

if(window.confirm(‘本当に削除しますか？’)){ // 確認ダイアログを表示

location.href = "{% url ‘myapp:book \_delete’ record.pk %}"; // 指定urlへ移動

}

else{

window.alert(‘キャンセルされました’); // 警告ダイアログを表示

}

}

</script>

<form class="form" method="post">

{% csrf\_token %}

{{ form }}

<button type="button" class="btn btn-secondary col-10" onclick="delete\_confirm()">削除</button>

</form>

DeleteViewを使った方法もある。⇒ ページ移管が発生する為、個人的にはいまいちだと思った。

#### migrationで初期データを与える

メイン情報：<https://qiita.com/d9magai/items/71dfa9ff95f7b70486a7>

#### 入力規制

forms.Formクラスのcleanメソッドを使う

views.pyで規制スクリプトを記述する

#### 複数のsubmitボタン

[ sample.html(template) ]

<form method="post">

{% csrf\_token %}

<input type="text" name="data">

<button type="submit" name="submit\_and\_back ">前の画面に戻る</button>

<button type="submit" name="submit\_and\_redirect">別のページに移動する</button>

</form>

[ views.py]

def post(self, request: HttpRequest, \*args: str, \*\*kwargs: Any) -> HttpResponse:

if "submit\_and\_back" in request.POST:

render(request, 'previous.html')

else:

redirect(reverse('rediret\_to\_view'))

#### Djangoでuuidを使う

インストール

[ terminal ]

pip install django-uuidfield

import uuid

from django.db import models

class Base(models.Model):

class Meta:

abstract = True

id = models.UUIDField(

primary\_key = True,

default = uuid.uuid4, # uuid.uuid4()だとエラー

editable = False

)

#### ポップアップ

javaScriptを使う

[ html（テンプレート）]

<a href="javascript:void(0);" onclick="window.open(‘{% url ‘myapp:book\_popup’ %}’,’subwin’,’width=1000,height=500’);">ポップアップテスト</a>

PopupTagCreate(TagCreate)

CategoryCreate

from bootstrap\_modal\_forms.mixins import PopRequestMixin, CreateUpdateAjaxMixin

<https://blog.narito.ninja/detail/62>

#### submitボタンのキャンセル

JavaScriptやjQueryを用いて以下のように書くのが良い

<script type="text/javascript">

$(document).ready(function() {

$('form').on('keydown', (event)=>{

if(event.key == 'Enter'){

event.preventDefault();

};

})

})

</script>

<form class="form" method="post">

</form>

補足

このような方法もある

<form class="form" method="post" onsubmit="return false;">

<button type="button" onclick="submit();">適用</button>

</form>

ただし、この場合post処理自体がキャンセルされる為、例えばrequiredになっているinputの入力確認処理が実行されないなど不都合が多い為、非推奨

2023-08-08追記；以下のようにするとrequiredなどの入力確認もされる事が確認された。

<script type="text/javascript">

function submit\_confirm(){

if(window.confirm('適用します。よろしいですか？')){

var fm = document.getElementById("mainform");

fm.submit();

}

}

</script>

#### 戻るボタン

##### 直前の画面へ戻る

views.pyの場合

referer\_url = self.request.META['HTTP\_REFERER']

htmlの場合

<a href="{{request.META.HTTP\_REFERER}}">前のページに戻る</a>

JavaScriptでやる場合

<input type="button" value="Back" onClick="javascript:history.go(-1);">

##### back\_url方式

関連：request.get\_full\_path

##### 戻る時に入力内容を保持する

案１

・戻るボタン押下で仮post(submit)を行う。当然、DBへのINSERT等は行わない。

・postイベントの処理時にsession変数で保持する

※post処理を行うメソッドで、本postと仮postを判別して処理する必要がある

案２

・戻るボタンでリダイレクトする。

・リダイレクト処理中にweb formの入力内容をsession変数で保持させる。

#### コンマ区切りのinput

表示だけならこれで問題ない

[ settings.py ]

INSTALLED\_APPS = [

'django.contrib.humanize',

]

NUMBER\_GROUPING = 3

[ テンプレート(.htmlファイル) ]

{% load humanize %}

{{ price| intcomma }}円

入力はJavaScriptのライブラリを用いるのが良い

・AutoNumeric

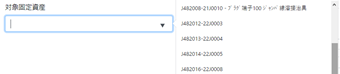
公式：<http://autonumeric.org/>

GitHubのリポジトリをダウンロード

<https://github.com/autoNumeric/autoNumeric>

・Numeral.js

#### datalistで任意の表示名と値を設定する

右図のように自由な文字を表示して、内部ではプライマリキーの値を扱う、といった事ができる。

（結構複雑にはなるが）

[ views.py ]

from django.db.models.functions import Concat

class BookCreateView(TemplateView):

template\_name = 'book\_create.html'

def get\_context\_data(self, \*\*kwargs: Any) -> Dict[str, Any]:

context = super().get\_context\_data(\*\*kwargs)

book\_list = Book.objects.annotate(

display\_name = Concat('book\_title', V(' - '), 'author\_\_author\_name') # html上に表示する任意の文字列

)

context['book\_list'] = book \_list # datalistで使うlist

context['form'] = BookForm

return context

def post(self, request, \*args, \*\*kwargs):

book\_pk = self.request.POST.get('book\_id', None) # このpkの値でfilterを使えば良い

[ forms.py ]

class BookForm(forms.Form):

book\_input=forms.CharField( widget = forms.TextInput( attrs = {

'list' : 'id\_book\_list', # <input list="id\_book\_list"> になる

'autocomplete' : 'off', # これは任意だが、推奨

}))

[ book\_create.html ]

<script type="text/javascript">

$(document).ready(function() { // イベントに対応する為jQueryを使う

function apply\_book\_input ( ) {

id = $("#id\_book\_list option[value='" + $('#id\_book\_input').val() + "']").data('id'); // data-○○ を data(○○)で取得できる

$('#id\_book\_id').val(id); // hiddenのinputに値を代入

}

$('#id\_book\_input').change(() => apply\_book\_input()) // book\_inputの値変更イベントの登録

})

</script>

<form class="form" method="post">

{{ form.book\_input }}

<input id="id\_book\_id" type="hidden" name="book\_id" value=""> <--! pkの値を保持するhidden input -->

</form>

<datalist id="id\_book\_list">

{% for book in book\_list %}

<option value="{{ book.display\_name }}" data-id="{{ book.id }}"\> <--! data-○○の部分が重要-->

{% endfor %}

</datalist>

#### JupyterNotebookでDjangoを使う

必要パッケージのインストール

[ terminal ]

pip install jupyter django-extensions

VSCodeの場合

を押してkernelを再起動する

Webサーバー版Jupyterの場合

[ terminal ]

python manage.py shell\_plus –notebook

Djangoのプロジェクトディレクトリ内のstaticディレクトリにjupyter\_notebookと言うディレクトリを作成

Jupyterでそのディレクトリへ移動し、New > Django Shell-Plusを選択

[ JupyterNotebook ]

コードセルで以下を実行

import os

import django

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", "config.settings") # manage.pyからコピペすると良い

django.setup()

参考：<https://sleepless-se.net/2018/05/17/django%E3%81%A7ipython-jupyter-notebook%E3%82%92%E4%BD%BF%E3%81%86%E6%96%B9%E6%B3%95/>

django-extensionsを使う方法

]

#### DjangoでCeleryを使う

requirements.txt

celery

redis

Celery公式：<https://docs.celeryq.dev/en/stable/django/first-steps-with-django.html>

#### Jinja2を使う

TEMPLATES = [

{

# 'BACKEND': 'django.template.backends.django.DjangoTemplates',

'BACKEND': 'django.template.backends.jinja2.Jinja2',

}

注意点

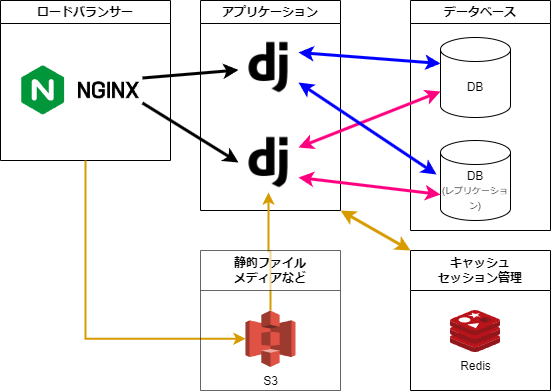
・{% load static %}

Jinja2にはloadというタグは無い。静的ファイルへのリンクを取得する際には以下のようにする

<img src="{{ 'img/image.png' | static }}"

#### 複数のレプリカ＋ロードバランス

構成イメージ



概要

・大規模なアプリケーションでは、ほとんどの場合データベースがボトルネックとなる

その為、レプリケーション機能のあるRDBを選定する

・ロードバランサーはリバースプロキシにその機能があるものを選定する。

・静的ファイルはクラウドストレージの利用がベストだが、ファイルサーバーを中心として同期をとっても良い

・セッション管理、キャッシュ管理はRedis、MemchachedなどのキャッシュベースのDBを用いる

#### フロントエンドにReactを使う

バックエンドにDjango REST framework

[ terminal ]

pip install djangorestframework

フロントエンドのReactのアプリを作成

[ terminal ]

npx create-react-app react-app --template typescript

注意点

・CORS；Cross Origin Resource Sharing問題（同一オリジンポリシー）が発生する可能性がある

django-cors-headersをつかうと良い？

参考：<https://noauto-nolife.com/post/django-react-methodology/>

参考：<https://qiita.com/shota_mci/items/314c1f25d2c8dc47b4e4>

参考（英語）：<https://www.digitalocean.com/community/tutorials/build-a-to-do-application-using-django-and-react>

#### VSCodeの設定

##### Django Templateエンジンのlinter

Django Template

DJLINT

Pretty Diff

#### その他のTips

##### プライマリーキーについて

このようにする事で「id」以外のプライマリーキーを使用する事ができる。

[ models.py ]

class Book(models.Model):

isbn\_code = models.CharField(primary\_key=True, verbose\_name=‘ISBNコード’, max\_length=13)

また、以下のようにする事で「id」のデータ型を変える事ができる

[ config/settings.py ]

DEFAULT\_AUTO\_FIELD = ‘django.db.models.BigAutoField’

但し、外部キーのフィールド名に注意。デフォルト設定の時DB上では\_idというサフィックスが付く。

[models.py]

class Order(models.Model):

book = models.ForeignKey(Book, # ⇒ book\_idというデータ列名になる。

book = models.ForeignKey(Book, db\_column= ‘book\_isbn\_code’ # このように列名を指定すれば良い

##### nullとblankについて

blank htmlフォームからのPOSTで値が空かどうか。

null データベース上のレコードで特定の列のデータがない場合を許可するかどうか。

対応表

|  |  |  |  |
| --- | --- | --- | --- |
| Blank | null | 概要 | 備考／補足 |
| False | False | Htmlフォームの入力もDB上も未入力を認めない。 | Djangoのデフォルト |
| True | False | Htmlフォームの入力のみ未入力を認める |  |
| False | True | DB上のみ未入力を認める。 |  |

⇒ 要するに、未入力を認めたければ大体の場合は**両方とも=True**にする。

参考：<https://qiita.com/yama04070319/items/0495a5209deea3807cfb>

##### Djangoアプリ名の変更方法

以下の手順に従う

・アプリのフォルダ名変更 \old\_app → \new\_app

・settings.pyのINSTALLED\_APPSの文字列を変更

・\project\_name\new\_app \app.pyの

OldappConfig → NewappConfig （ここの名前は何でも良いらしい）

・old\_app\

### Djangoのツール

#### manage.py

##### loaddata

ダンプされたファイルからDBにロードする

python manage.py loaddata dump.json

注意点

dumpdataは文字コードUTF-16 LEでjsonファイルを出力するが、loaddata時はUTF-8でないと読み込めないらしい。この挙動は2022-03-03 WindowsServerだけ？

公式：<https://docs.djangoproject.com/en/4.0/ref/django-admin/#loaddata>

##### collectstatic

複数のディレクトリ下に置かれているstaticファイルをSTATIC\_ROOT（本番サーバーのドキュメント・ルート；setting.pyで指定）に集約する。

（補足）manage.py runserverだと自動でcollectstaticしてくれるが、本番のアプリサーバーの場合は自分でやる必要がある点に注意。

##### createsuperuser

管理者（super user）を作成

python manage.py createsuperuser

# ユーザー名，メールアドレス，パスワードを入力する。⇒ /adminに入れるようになる

補足：厳密にはUserクラスが継承しているAbstractBaseUser のREQUIRED\_FIELDS で指定しているフィールドがcreatesuperuserの必須引数となる。その為、カスタムユーザーを作る事で、必須フィールドは変更できる。

##### dumpdata

データベースのデータをテキストファイルにダンプする。

python manage.py dumpdata > dump.json # デフォルトはjsonで保存

python manage.py dumpdata my\_app.MyClass > dump.json # 特定のmodelだけ

オプション

--format=形式 ダンプファイルの形式を選択。json|yaml（※）

※pip install PyYAMLしている場合

--exclude

# プライマリキーが重複しやすい項目を省く例

python manage.py dumpdata --exclude auth.permission --exclude contenttypes > dump.json

python manage.py dumpdata --exclude admin --exclude auth.permission --exclude contenttypes

関連：loaddata

公式情報：<https://docs.djangoproject.com/ja/3.1/ref/django-admin/#dumpdata>

##### findstatic

python .\manage.py findstatic admin/css/base.css

> home\user\myenv\Lib\site-packages\django\contrib\admin\static\admin\css\base.css などと表示

##### flush

python manage.py flush

・削除されるのはデータのみ。migrateをし直す必要は無い。

##### inspectdb

指定したデータベースからDjango用のmodels.pyを生成する。リバースエンジニアリング。

標準出力に表示される為、以下のようにパイプを使ってファイルに保存する事になる

python3 manage.py inspectdb --database mmydb > myapp/models.py

書式 inspectdb [-h] [options]

オプション

|  |  |
| --- | --- |
| オプション | 概要 |
| --database DATABASE | settings.py で指定したデータベース名を指定  （dictのkeyになっている文字列） |
| --include-views] |  |
| --version |  |
| -v {0,1,2,3} |  |
| --settings SETTINGS] |  |
| --pythonpath PYTHONPATH |  |
| --traceback] |  |
| --no-color] |  |
| table [table ...] |  |

公式：<https://docs.djangoproject.com/ja/4.2/ref/django-admin/#inspectdb>

##### makemigrations

データベースの変更の適用

構文：makemigrations [アプリケーション名]

makemigrations myapp # myapp/migrations が存在しない場合（初回）

makemigrations # 上記ファイルが存在する場合（２回目以降）

オプション

--empty 空のマイグレーションファイルを作成

makemigrations myapp --empty # app名の指定が必要

参考：[https://qiita.com/d9magai/items/71dfa9ff95f7b70486a7#](https://qiita.com/d9magai/items/71dfa9ff95f7b70486a7#空のmigrationファイルを作る)空のmigrationファイルを作る

##### migrate

マイグレーション（データベースの定義を自動的に作成・管理）を行う。

settings.py の設定に従ってすべてのテーブルを作成する。

（参考）<https://docs.djangoproject.com/ja/3.2/topics/migrations/>

特定のマイグレーションに戻す場合

python .\manage.py migrate myapp 0005\_auto\_20200101\_1230 # myapp/migrationのファイルを参考にする

この場合0006以降のファイルを削除すると、それ以降普通にmigrateできるようになる

##### shell

python を即時実行する，pythonシェルを起動する。

例

(env) PS C:\python\gyoumu\_mng> python .\manage.py shell

Python 3.8.10…

Type "help", "copyright", "credits" or "license" for more information.

(InteractiveConsole)

>>> from myapp.models import Book

>>> Book.objects.all().delete()

(3, { ‘myapp.Book’ : 3})

（関連）django-

##### showmigrations

アプリ開発最初から現在までのmigration情報を表示

##### sqlmigrate

作成されたSQLの実行計画を表示する（SQL文を表示）

makemigrationsが実行されるとmyproject/myapp/migrationsに0001\_initial.pyというファイルが作成される。そのファイルをSQLコマンドにしたものを表示する。

python manage.py sqlmigrate myapp 0001 # 0001\_initial.pyが対象になる

オプション

--backwards

ロールバック時に実行されるSQLを表示する

次ページに続く

##### runserver

htmlサーバーの起動

構文：runserver [addrport]

python manage.py runserver 8080 # ポート番号のみ指定。IPは127.0.0.1が選択される。

python manage.py runserver 0.0.0.0:8080 # リッスン対象に全く規制をかけない

python manage.py runserver 192.168.1.1:8080 # 192.168.1.1ポート8080をリッスンする

注意点

・このサーバーはあくまで開発用。本番ではapacheなどのwsgi対応のサーバーアプリを用いる事

・リッスンするIPアドレスがALLOWED\_HOSTSに載っていない場合，接続時にエラーとなる。

--settings

django-admin runserver --settings=mysite.settings.local

公式：<https://docs.djangoproject.com/en/4.0/topics/settings/#designating-the-settings>

（参考）<https://ebi-works.com/django-manage/>

（参考）<https://qiita.com/okoppe8/items/7e3de8a4dd40b48debea>

##### startapp

アプリを作成する

##### test

概要

自動テストを実行できる。

manage.py test myapp # アプリ単位

manage.py test myapp.tests # モジュール単位

manage.py test myapp.tests.SomeTest # テストクラス単位

manage.py test myapp.tests.SomeTest.test\_method # メソッド単位

（参考）自動テスト

##### 資料

manage.py まとめ：<https://qiita.com/okoppe8/items/7e3de8a4dd40b48debea>

公式：<https://docs.djangoproject.com/ja/4.0/ref/django-admin/>

#### django-admin

startproject

$ django-admin startproject project1 #

sendtestemail

django-admin sendtestemail foo@example.com bar@example.com

#### makemessages

オプション

-i virtualenvで作られた仮想環境などを除外する --ignore

#### compilemessages

### 関連するpythonパッケージ

#### architect

テーブルパーティションを簡単に実現してくれる。

PostgreSQL、MySQLのみが対象。

#### behave-django

BDD（Behavior-Driven Development）の恩恵を受ける

（補足）BDD：テスト駆動開発から派生した、要求仕様からテストコードを起こすシステム。

#### cookiecutter

厳密にはdjangoのツールでは無く，Pythonプロジェクトの雛形を作成するツール。

インストール

pip install cookiecutter

構文：cookiecutter [オプション] ひな形のurlなど [EXTRA\_CONTEXT]…

cookiecutter https://github.com/pydanny/cookiecutter-django # 例

オンラインのひな形は以下のところなどで選ぶ

<https://djangopackages.org/grids/g/cookiecutters/>

大体以下のものが選択されている様子。

https://github.com/pydanny/cookiecutter-django

https://github.com/asifpy/django-crudbuilder

様々な質問をされるが，空白エンターで既定値（[]内に表示されたもの）が設定される

（主な質問項目は次ページ）

主要な質問項目

project\_name プロジェクト名

project\_slug プロジェクトのスラッグ スラッグ：URLの末尾の部分の任意の文字列

timezone [UTC] 値：Asia/Tokyoなど

windows Windows環境かどうか。y/n

use\_pycharm Python向きのIDE，PyCharmの使用可否

posetgresql\_version なぜか13.2, 12.6, 11.11, 10.16の中からしか選べない。理由は？

js\_task\_runner 開発作業を自動化するためのツール。Gulp, webpack, npm-scriptsなど。

use\_async ASGIを使うか否か？

use\_drf Django REST Framework

custom\_bootstrap\_compilation

Bootstrap：CSSフレームワーク

use\_compressor ？

use\_celery Python向きCelery：タスクキューの実装？

use\_mailhog 開発用メールサーバー

use\_sentry エラー監視？

use\_whitenoise Webアプリの静的ファイルの提供

use\_heroku PaaSの一種。GCPやAWSみたいなもの。

keep\_local\_envs\_in\_vcs ？

debug ？

#### django-allauth

Twitter、Githubなどのソーシャル認証もサポートしている高機能な認証機能を提供

（比較）django-registration-redux

#### django-bootstrap5

概要

Djangoでbootstrap5を使う為のテンプレートタグを使用する

備考

djangoのバージョン2.2以上

インストール

pip install django-bootstrap5

[ settings.py ]

INSTALLED\_APPS = [

…

‘django\_bootstrap5’,

]

[ html（テンプレート）]

{% load django\_bootstrap5 %}

<head>

…

{% bootstrap\_css %}

{% bootstrap\_javascript %}

</head>

formを使う場合

[ html（テンプレート）]

{% bootstrap\_form form %}

{% bootstrap\_button パラメーター %}

補足

django-bootstrap4 も存在する（django>=2.2,<=4.2）

参考：<https://office54.net/python/django/django-bootstrap5-css>

#### django-background-tasks

簡易的に非同期処理を行う為のパッケージ？

#### django-crispy-forms

概要

Djangoのformをbootstrapで扱えるように整形する

Bootstrap4のスタイルが適用できる

チュートリアル

[ terminal ]

pip install django-crispy-forms

[ settings.py ]

INSTALLED\_APPS = [

‘crispy\_forms’, # 追加

]

CRISPY\_TEMPLATE\_PACK = ‘bootstrap4’ # 追加

[ html（通常base.htmlなどのレイアウトテンプレートになるはず）]

<head>

<meta charset=utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

</head>

[ html（実際の使用）]

{% extends ‘base.html’ %}

{% load crispy\_forms\_tags %}

{% block content %}

<form method="post">

{% csrf\_token %}

{{ form**|crispy** }}

<button type="submit" class="btn btn-primary">Sign in</button>

</form>

{% endblock %}

フィールドにカスタマイズが必要な場合：as\_crispy\_fieldを使う

<div class="container">

<div>

{{ form.title |as\_crispy\_field }}

</div>

</div>

参考：<https://nmomos.com/tips/2019/09/16/django-crispy-forms/>

#### django-celery

概要

非同期処理をする為のパッケージ。

redisも必要？

pip3 install celery django-celery-results redis django-redis

django-celery-email

Celeryを用いた非同期メール送信バックエンド。

（補足）Celery：Pythonベース非同期タスク処理

#### django-extensions

pip install django-extensions Werkzeug

shell\_plusコマンドなどがある

（参考）

django-celery-results

django-celery-beat

#### django-debug-toolbar

GUIによるデバッグ

インストール

pip install django-debug-toolbar

[ config/settings.py ]

if DEBUG:

def show\_toolbar(request):

return True

INSTALLED\_APPS += (

'debug\_toolbar',

)

MIDDLEWARE += (

'debug\_toolbar.middleware.DebugToolbarMiddleware',

)

DEBUG\_TOOLBAR\_CONFIG = {

'SHOW\_TOOLBAR\_CALLBACK' : show\_toolbar,

}

[ config/urls.py ]

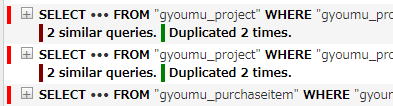
from django.conf import settings

if settings.DEBUG:

import debug\_toolbar

urlpatterns = [path('\_\_debug\_\_/', include(debug\_toolbar.urls)) ] + urlpatterns

（公式）<https://django-debug-toolbar.readthedocs.io/en/latest/installation.html#setting-up-urlconf>

補足

similar queriesやduplicated n times（右図参照）はループなどで同じクエリが発行されている可能性が高い。

#### django-environ

##### 概要

環境変数を扱う為のパッケージ。

システム（OS）で設定された環境変数や.envなどの環境変数設定ファイルを扱う。

特に本番環境下で、SECRET\_KEYなどの見えない方が良い設定項目を環境変数化する為に用いる

目的

・ソースコードから機密情報を分離する

・設定の一元管理、バージョン管理を楽にする。

ディレクトリ構成

myproject

├ config\

│ ├ settings.py

├ .env

登録方法１ export（OSで通常の方法で環境変数を設定する）

[ terminal ]

export SECRET\_KEY='\*\*\*\*\*\*\*\*\*\* '

Set-Item Env:SECRET\_KEY '\*\*\*\*\*\*\*\*\*\* ' # windowsのpowershellの場合

登録方法２

.envファイルを作成して編集

[ .env ]

SECRET\_KEY='\*\*\*\*\*\*\*\*\*\* '

ALLOWED\_HOSTS=127.0.0.1

注意：「=」の前後の空白禁止！

使用方法

[ settings.py ]（例）

import environ

env = environ.Env()

env.read\_env(os.path.join(BASE\_DIR, ‘.env’)) # .env があれば読み込み

SECRET\_KEY = env(‘SECRET\_KEY’)

ALLOWED\_HOSTS = env.list(‘ALLOWED\_HOSTS’) # list

公式：<https://django-environ.readthedocs.io/en/latest/>

pypi：<https://pypi.org/project/django-environ/>

##### リファレンス

###### Path

BASE\_DIR = environ.Path(\_\_file\_\_)

公式：<https://django-environ.readthedocs.io/en/latest/api.html#environ.Path>

#### django-glrm

アプリ全体をログイン必須にする場合に便利なミドルウェア。

インストール

pip install django-glrm

[ settings.py ]

MIDDLEWARE = [

...

'global\_login\_required.GlobalLoginRequiredMiddleware', # ←追加

]

PUBLIC\_PATHS = [

r'^/auth/.\*', # ログイン、パスワード関連のURLを対象外に

]

PUBLIC\_VIEWS = [

'myapp.views.MyAppTopView', # アプリのトップ画面を対象外に

]

ログイン制御の対象から外す（以下のどちらでも良い）

[ views.py ]

from global\_login\_required import login\_not\_required

class BookView():

LOGIN\_NOT\_REQUIRE = True

@login\_not\_required

class BookView():

pass

公式：<https://django-glrm.readthedocs.io/en/latest/>

#### django-guardian

データベースで各レコードに対する権限を管理する。

（関連）django-rules

[ settings.py ]

INSTALLED\_APPS = [

…

‘guardian’,

]

#### django-mqtt

公式：<https://pypi.org/project/django-mqtt/>

#### django-mssql-backend

データベースがSQLServerだと、my\_field\_\_regexでのfilterなど、一般的なRDBで出来る事ができなかったりするが、その一般的なDBバックエンド処理を提供する。

Django2.2以上

参考：django-mssql

django-mssql-backendより昔に提供されていたライブラリ。Django1.8への対応を最後にメンテナンスが停滞していた。

django-mssql-backendはdjango-mssqlのフォークでは無く、ゼロベースでの開発。

#### django-oauth-toolkit

#### django-redis

Django上でAWSのRedisを用いる。

インストール

pip install django-redis

#### django-registration-redux

メールアドレスを用いたユーザー認証機能を提供。

[terminal]

pip install django-registration-redux

[ settings.py ]

INSTALLED\_APPS = (

...

‘django.contrib.sites’,

‘registration’,

)

（比較）django-allauth

#### django-rest-framework

##### 概要

Djangoを用いてバックエンドだけを作成する場合に用いる

##### チュートリアル

インストール

pip install djangorestframework

プロジェクトの作成

[ terminal ]

django-admin startproject config .

設定ファイルの変更

[ settings.py ]

INSTALLED\_APPS = [

'rest\_framework', # 追加

]

serializers

serializers.ModelSerializer

#### django-rest-framework-simplejwt

#### django-rest-swagger

#### django-rules

pythonコードで各レコードに対する権限を管理する。

pip install rules # パッケージ名に注意。django-rulesというパッケージは開発放棄されている。

（関連）django-guardian

[ settings.py ]

INSTALLED\_APPS = [

‘rules.apps.AutodiscoverRulesConfig’,

]

#### django-silk

時間のかかるリクエスト、SQLを計測することができる

[ settings.py ]

MIDDLEWARE = [

...

‘silk.middleware.SilkyMiddleware’,

...

]

INSTALLED\_APPS = (

...

‘silk’

)

#### django-storages

#### django-tinymce

Django上でTinyMCEを使う為のツール。

TinyMCE；Tiny Moxiecode Content Editor

実際の見た目と同じ画面で編集できるエディタ

（参考）<https://oxynotes.com/?p=11177>

#### django-widget-tweaks

#### drm-spectacular

#### django-wkhtmltopdf

#### django\_xhtml2pdf

#### summernote

Django上でWYSIWYGを使うためのツール。

WYSIWYG ；What You See Is What You Get（見たままが得られる）

ウィズウィグと呼ばれる。

Wordなどの様に文字の大きさや色、表組などを入力しながら確認できる機能。

htmlなどの編集が楽になる。

#### wctidth

### ライブラリリファレンス

#### django.core.mail.backends

##### 概要

|  |  |  |  |
| --- | --- | --- | --- |
| バックエンド | 名前空間 | 概要 | 関連する設定項目 |
| SMTP バックエンド | smtp | Eメールで送信 | 色々（後述） |
| コンソールバックエンド | console | コンソール（標準出力）に表示 |  |
| ファイルバックエンド | filebased | テキストファイルに出力  EMAIL\_FILE\_PATH/yyyyMMdd-hhmmss\*\*\*.logというテキストファイルに出力される | EMAIL\_FILE\_PATH |
| インメモリーバックエンド | locmem |  |  |
| ダミーバックエンド |  | 何もしない |  |
|  |  |  |  |

SMTP バックエンドの時の設定値

|  |  |  |  |
| --- | --- | --- | --- |
| 引数名 | settings.pyでの名称 |  | 初期値 |
| host | EMAIL\_HOST |  |  |
| port | EMAIL\_PORT |  |  |
| username | EMAIL\_HOST\_USER |  |  |
| password | EMAIL\_HOST\_PASSWORD |  |  |
| use\_tls | EMAIL\_USE\_TLS |  |  |
| use\_ssl | EMAIL\_USE\_SSL |  |  |
| timeout | EMAIL\_TIMEOUT |  |  |
| ssl\_keyfile | EMAIL\_SSL\_KEYFILE |  |  |
| ssl\_certfile | EMAIL\_SSL\_CERTFILE |  |  |

公式：<https://docs.djangoproject.com/ja/4.2/topics/email/#obtaining-an-instance-of-an-email-backend>

公式（メール全体）：<https://docs.djangoproject.com/ja/4.2/topics/email/>

##### EmailMessage

公式：<https://docs.djangoproject.com/ja/4.2/topics/email/#django.core.mail.EmailMessage>

#### django.core.paginator

##### Paginatorクラス

paginator = Paginator(queryset, self.paginate\_by)

p = self.request.GET.get(‘page’)

page\_obj = paginator.get\_page(p)

page = paginator.page(page\_num)

paginator.num\_pages

（公式）<https://docs.djangoproject.com/ja/4.0/topics/pagination/>

##### PageNotAnIntegerクラス

##### EmptyPageクラス

#### django.core.validators

##### RegexValidator

from django import forms

from django.core.validators import RegexValidator

class BookForm(forms.Form)

def clean\_titile(self):

data = self.cleaned\_data[]

numeric\_validator=RegexValidator(

regex=r '^[0-9]+$',

message='半角数字のみ入力して下さい',

)

numeric\_validator(data)

retrun data

補足

htmlの入力時に使える文字列を規制する方法はJavaScriptを使って実現するしかないらしい

例

<input type="text" onkeyup="validateInput(event)">

<script>

function validateInput(event) {

var input = event.target;

var value = input.value;

var pattern = /[\x00-\x7F]\*/;

if (!pattern.test(value)){

input.value = value.replace(/[^\x00-\x7F]\*/g, '')

};

}

</script>

'

公式： <https://docs.djangoproject.com/ja/4.2/ref/validators/#regexvalidator>

#### django.conf

##### settings

setting.pyの値を取得できる

from django.conf import settings

context = {

‘book’: book,

‘stripe\_pub\_key’: settings.STRIPE\_PUBLISHABLE\_KEY,

}

###### シリアライズ関連

BUILTIN\_SERIALIZERS

デフォルトでの設定

BUILTIN\_SERIALIZERS = {

"xml": "django.core.serializers.xml\_serializer",

"python": "django.core.serializers.python",

"json": "django.core.serializers.json",

"yaml": "django.core.serializers.pyyaml",

}

SERIALIZATION\_MODULES

公式：<https://docs.djangoproject.com/en/4.2/ref/settings/#serialization-modules>

#### django.contib

##### admin

##### messages

概要

テンプレート（html）の共通変数messagesの値を管理する

メッセージの種類にはinfo, success, warning, error, debugが存在する。

[ views.py ]

if not form.is\_valid():

messages.info(request, ‘データに誤りがあります。’) # info を用いた例

テンプレート（.htmlファイル）における典型例

{% if messages %}

<ul class="messages">

{% for message in messages %}

<li{% if message.tags %} class="{{ message.tags }}"{% endif %}>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

注意点

・テンプレート（.html）で一度表示されると自動的にインスタンスから削除される仕組み

・逆に言うと，表示されなかったメッセージのインスタンスは次に持ち越してしまう為注意。

used

例えば全メッセージを使用済として処理する場合

def del\_all\_messages(request):

if len(messages.get\_messages(request)) > 0:

for msg in messages.get\_messages(request):

msg.used = True

もしくはmessages.get\_messages(request).used で全要素の値が返られる（らしい）

#### django.contrib.admin

##### ModelAdmin

概要

[ myapp.admin.py ]

from django.contrib import admin

class BookModelAdmin(admin.ModelAdmin):

ordering = (‘name’,) # 昇順、降順。※注意！あくまでtuppleなのでコンマが必要。(‘name’**,** )

fields = (‘name’) # 編集できるフィールド

admin.site.register(Book, BookModelAdmin) # ModelAdminの設定を適用する

fields

追加・編集で有効なフィールドを指定する。入力ページではここに指定したフィールドが表示される。

# わかりにくいフィールド

fields = (‘user\_permissions’,)

readonly\_fields

編集用ページで編集できないようにするフィールドを指定。

list\_display

**リストページで**表示されるフィールドを指定する。

メソッドでも指定できるらしい

@admin.display(description=‘Name’)

def upper\_case\_name(obj):

return ("%s %s" % (obj.first\_name, obj.last\_name)).upper()

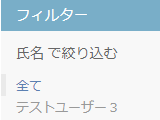
class PersonAdmin(admin.ModelAdmin):

list\_display = (upper\_case\_name,)

list\_editable

**リストページで**編集できるフィールド。

これを指定しなくても、個別編集画面では編集できる点に注意

list\_filter

絞り込み条件の対象となるフィールドを指定する。

右図のようなものが表示されるようになる。

list\_select\_related

絞

次ページに続く

～django.contib.admin　ModelAdminつづき

add\_view

change\_view

delete\_view

これが基本（らしい）

class GyoumuAppUserAdminForm(forms.ModelForm):

class Meta:

widgets = {

'password': forms.PasswordInput(),

}

def change\_view(self, request, object\_id, form\_url='', extra\_context=None):

return super().change\_view(request, object\_id, form\_url, extra\_context=extra\_context,)

情報元：<https://docs.djangoproject.com/ja/4.0/ref/contrib/admin/#django.contrib.admin.ModelAdmin.history_view>

search\_fields 検索ボックスが出現

search\_fields = (‘publisher\_\_name’)

list\_filter チェックボックスによる検索コンテナが出現

list\_select\_related

項目の情報

user\_permissions 各モデルのいわゆるCRUDの権限を１つずつ指定する。

補足：この手の制限はグループ機能でやるべき

groups グループ

資料

（参考；英語）<https://realpython.com/manage-users-in-django-admin/>

（公式）<https://docs.djangoproject.com/ja/4.0/ref/contrib/admin/>

##### Django管理サイトでハッシュ化パスワードを扱う

[ myapp/admin.py ]

from django.contrib import admin

from django import forms

from django.contrib.auth.hashers import make\_password, check\_password

from .models import MyCustomUser

class MyCustomUserAdminForm(forms.ModelForm):

class Meta:

widgets = {

'password': forms.PasswordInput(),

}

def clean\_password(self):

password\_cleaned = self.cleaned\_data['password']

if password\_cleaned:

password\_cleaned = make\_password(password\_cleaned) # make\_passwordメソッドを利用

return password\_cleaned

class MyCustomUserModelAdmin(admin.ModelAdmin):

fields = ('id\_number', 'username','password',) # fieldsはタプルな点に注意

form = MyCustomUserAdminForm

admin.site.register(MyCustomUser, MyCustomUserModelAdmin)

#### django.contrib.auth

##### login()

##### logout()

##### authenticate()

ユーザー名とパスワードを使った認証を行う。

Userオブジェクトのis\_authenticated

（公式）<https://docs.djangoproject.com/ja/4.0/ref/contrib/auth/#django.contrib.auth.models.User.is_authenticated>

##### get\_user\_model()

Userモデルのインスタンスを取得する。直接参照するのではなく，このメソッドを使って取得する方法が推奨されている。

from myapp import some\_module

some\_module.UserModel = get\_user\_model()

以下のようにする事でmakemigrationsなどがうまく行く？

from django.contrib.auth import get\_user\_model

class Book (models.Model):

author = models.ForeignKey(get\_user\_model(), on\_delete=models.CASCADE)

##### urls

urls.pyの記述内容

urlpatterns = [

path('login/', views.LoginView.as\_view(), name='login'),

path('logout/', views.LogoutView.as\_view(), name='logout'),

path('password\_change/', views.PasswordChangeView.as\_view(), name='password\_change'),

path('password\_change/done/', views.PasswordChangeDoneView.as\_view(), name='password\_change\_done'),

path('password\_reset/', views.PasswordResetView.as\_view(), name='password\_reset'),

path('password\_reset/done/', views.PasswordResetDoneView.as\_view(), name='password\_reset\_done'),

path('reset/<uidb64>/<token>/', views.PasswordResetConfirmView.as\_view(), name='password\_reset\_confirm'),

path('reset/done/', views.PasswordResetCompleteView.as\_view(), name='password\_reset\_complete'),

]

#### django.contrib.auth.backends

##### ModelBackend

（公式）<https://docs.djangoproject.com/ja/4.0/topics/auth/customizing/#specifying-authentication-backends>

#### django.contrib.auth.base\_user

##### BaseUserManager

|  |  |  |
| --- | --- | --- |
|  | 対象 | 備考 |
| get\_by\_natural\_key | manage.py createuser |  |

#### django.contrib.auth.forms

##### UsernameField

Django管理ページ（admin）に入るためのユーザーモデルのusername 用のフィールドを返す？

##### UserCreationForm

ユーザー作成に適したフォーム

#### django.contrib.auth.hashers

##### make\_password

#### django.contrib.auth.models

##### AnonymousUser

未ログインのユーザーを扱う（らしい）

##### has\_perm

例えばPostというモデルに対して以下のパーミッションが自動で作成される

app.add\_post　app.change\_post　app.delete\_post　app.view\_post

それに対して "<app label>.<permission>"という書式、つまり"myapp.add\_post" などと記述する。

例

perm = book.has\_perm(‘myapp.change\_book’)

以下の様な仕様がある：

・アクティブでないユーザーの場合、このメソッドは常に False を返す

・アクティブなスーパーユーザーの場合、このメソッドは常に True

・それ以外の場合は、グループや個別ユーザーに付与された権限が適用される。

##### PermissionsMixin

認証にまつわるクラス？例えばis\_superuser はここで定義されている。

#### django.contrib.auth.mixins

##### LoginRequireMixin

Djangoのユーザー管理機能でのログインを義務付ける。

以下のような形でViewの第一引数にしている。

[ views.py ]

from django.contrib.auth.mixins import LoginRequiredMixin

class IndexView(LoginRequiredMixin, View):

##### LoginViewクラス

from django.contrib.auth.views import LoginView

class AccountLoginView(LoginView)

template\_name = ‘accounts/login.html’

login = AccountLoginView.as\_view()

##### LogoutViewクラス

変数

LOGIN\_URL

LOGIN\_REDIRECT\_URL ログイン後のリダイレクト先

LOGOUT\_REDIRECT\_URL ログアウト後のリダイレクト先

#### django.contrib.auth.password\_validation

##### 概要

標準では以下の４つが設定されている：

UserAttributeSimilarityValidator，MinimumLengthValidator，CommonPasswordValidator，NumericPasswordValidator

以下のようにして微調整も可能

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

'OPTIONS': {

'min\_length': 10,

}

},

]

##### UserAttributeSimilarityValidator

|  |  |  |
| --- | --- | --- |
| オプション | 概要 | 備考 |
| user＿attributes |  | 'username', 'first\_name', 'last\_name', 'email'を組み合わせる |
| max\_similarity |  |  |

##### MinimumLengthValidator

|  |  |  |
| --- | --- | --- |
| オプション | 概要 | 備考 |
| password\_list\_path |  |  |
|  |  |  |

##### CommonPasswordValidator

|  |  |  |
| --- | --- | --- |
| オプション | 概要 | 備考 |
| password\_list\_path |  |  |
|  |  |  |

##### NumericPasswordValidator

オプションは特にない

#### django.contrib.auth.signals

##### user\_logged\_in

#### django.contrib.auth.views

##### PasswordChangeView

（公式）<https://docs.djangoproject.com/ja/4.0/topics/auth/default/#django.contrib.auth.views.PasswordChangeView>

#### django.db

##### transaction

トランザクション付きでレコードの更新を行う。

non\_atomic\_requestsデコレータ

settings.pyで 指定されたATOMIC\_REQUESTS を指定のメソッドのみ無効化する。

atomic\_requestsデコレータ

明示的にATOMIC\_REQUESTS を指定のメソッドに有効化する。

with transaction.atomicデコレータ

１つのメソッドを１つのトランザクションで処理する

from django.db import transaction

@transaction.atomic

def sample\_view(request):

do\_stuff()

with transaction.atomicメソッド

特定の区間を１つのトランザクションで処理する。

tryは任意だが、**atomic の内部で例外をキャッチしない**事！

def viewfunc(request):

do\_stuff()

try:

with transaction.atomic(): # この区間を１つのトランザクションで処理

do\_more\_stuff()

except IntegrityError:

handle\_exception()

公式：https://docs.djangoproject.com/ja/4.0/topics/db/transactions/

#### django.db.models

##### Field継承クラス

公式：<https://docs.djangoproject.com/ja/3.2/ref/models/fields/>

参考：[https://qiita.com/okoppe8/items/13ad7b7d52fd6a3fd3fc#](https://qiita.com/okoppe8/items/13ad7b7d52fd6a3fd3fc#対応表)対応表

注意：django.formsにも似たようなフィールド群があるので注意。用途が全く異なる。

概要

django.db.models.fields.Fieldを継承する様々な型。

～ライブラリレファレンス：Field継承　ForeignKey　続き

###### ForeignKey

related\_name

逆向き参照したい場合の変数名を指定する。この値を指定しない時はmymodel\_setでアクセスできるが，この値を ‘+’ にする事で，逆向き参照を禁止する事もできる。

（補足）逆向き参照：非参照側から、自身を参照しているレコードを取得する。

（参考）<https://docs.djangoproject.com/ja/3.2/ref/models/fields/#django.db.models.ForeignKey.related_name>

[models.py]

class Dapartment(models.Model):

name = models.CharField(max\_length=255)

class User(models.Model):

username = models.CharField(max\_length=100, unique=True)

department = models.ForeignKey(Department, on\_delete=models.CASCADE, related\_name= "user\_set")

この場合views.pyなどで以下のようにすると、Department→Userの逆引きが出来る

dep = Department.objects.get(id=1)

for user in dep.user\_set.all():

print(user.username)

補足

この{class\_name}\_set というのはrelated\_nameを省略しても設定される既定値。

但し、公式は**related\_nameの指定を推奨**している。

資料

（公式）<https://docs.djangoproject.com/ja/4.0/ref/models/fields/#foreignkey>

～ライブラリレファレンス：Field継承　続き

###### OneToOneField

一対一フィールドを提供する。

idが分かれば関数的に値も決まるようなフィールド。REFERENCEの被対象となるようなフィールド？

ForeignKeyとの違い。以下のように、classのインスタンスでレコードを特定できる。

class Restaurant(models.Model):

owner = models.OneToOneField(Person, on\_delete=models.CASCADE,)

person1 = Person.objects.get(id=1)

restaurant = Restaurant(owner = person1)

restaurant.save()

print(restaurant == user.restaurant) # True

注意：null許容、unique制約なので、完全に対となるレコードが存在する場合に使う。

参考：<https://docs.djangoproject.com/en/4.0/ref/models/fields/#onetoonefield>

ForeignKeyはあくまでuser = User.objects.get(id=1)とidでアクセス

共通引数

to 必須引数。対象テーブルClassを直接，またはClass名の文字列で指定。

to\_field 参照元となる，参照先テーブルの列名。

on\_delete

OneToOneField, ForeignKeyで使われる引数

**参照先が削除**される際の、関連する**自テーブルのレコードの挙動**を指定する。

例：この例でAuthorレコードを削除すると、関連するStoryレコードも削除される

from django.db import models

class Author(models.Model):

name = models.CharField(max\_length=50)

class Story(models.Model):

author = models.ForeignKey(Author, on\_delete=models.CASCADE)

title = models.CharField(max\_length=50)

|  |  |
| --- | --- |
| 値 | 概要 |
| CASCADE | 自身を参照しているレコードを**すべて削除**する。 |
| PROTECT | 自身を参照しているレコードが１件でもある場合、自身の削除は中止される。（django.db.IntegrityError.ProtectedErrorがraise）  少しややこしいが、参照側のレコードは自由に削除できる。 |
| RESTRICT | PROTECTに似ているが、違いは：  raiseするのがRestrictedError  ※v2.1には無い |
| SET\_NULL | None(=null) が代入される。但しnull = True の時のみ。 |
| SET\_DEFAULT | デフォルト値が代入。但しdefault オプションの設定が必要 |
| SET | メソッドを指定し，そのメソッドが返す値をレコードに代入する |
| DO\_NOTHING | 何もしない |

（公式）<https://docs.djangoproject.com/en/4.0/ref/models/fields/#django.db.models.ForeignKey.on_delete>

補足　デフォルトで参照先のid列の値を参照するので，このフィールドを省略しても大体の場合は問題ない。

～ライブラリレファレンス：Field継承　基本フィールド　続き

###### ManyToMany

このようなモデルがあるとする。

[ models.py]

class Author(models.Model):

name = models.CharField(max\_length=100)

class Book(models.Model):

title = models.CharField(max\_length=100)

authors = models.ManyToManyField(Author) # ManyToMany

# なくても動作する。指定したい場合のみ

class BookAuthor(models.Model):

book = models.ForeignKey(Book, on\_delete=models.CASCADE)

author = models.ForeignKey(Author, on\_delete=models.CASCADE)

order = models.IntegerField()

ManyToManyField.\_\_init\_\_の引数（抜粋）

through 中継に用いるクラスを指定する（str型で指定）

authors = models.ManyToManyField('Author', through='BookAuthor')

逆参照

# 補足　related\_name=が指定されていない場合は 'mymodel\_set' になる

books = Author.objects.filter(book\_set\_\_title='The Great Gatsby')

# 単純に取得する場合

author.objetcts.get(pk=1)

books = author.book\_set.all()

# 補足　related\_name=booksの場合

books = Author.objects.filter(books\_\_title='The Great Gatsby')

[ html（template）]

{% for book in author.book\_set.all %}

{{ book.title }}

{% endfor %}

views.py

# querysetの結果をそのまま使うとエラーになる。all() を使う。

context['object\_list'] = Book.my\_many\_to\_many\_fields.all(),

次ページに続く

作成（INSERT）

モデルクラスにthroughの指定が無い場合

# 先にレコードの追加が必要な点に注意

author= Author.objects.create(name= 'J.K. Rowling')

book.authors.add(author)

throughの指定がある場合は中間テーブルを使用しないとエラーとなる

author= Author.objects.create(name= 'J.K. Rowling')

BookAuthor.objects.create(book=book, author=author, order=1)

公式：https://docs.djangoproject.com/ja/4.0/topics/db/examples/many\_to\_many/

更新（UPDATE）

モデルクラスにthroughの指定が無い場合

set メソッドを用いる

book.authors.set([author1, author2])

throughの指定がある場合は中間テーブルを使用しないとエラーとなる

book\_author = BookAuthor.objects.get(book=book, author=author, order=1)

# 何か修正を行う

book\_author.save()

###### 共通プロパティ

まとめ

|  |  |  |  |
| --- | --- | --- | --- |
|  | データ型 |  | 初期値 |
| editable |  | ModelFormやadminページを利用する際に編集可能なフィールドとして扱うかどうか | True |
| required | bool |  | True |
| verbose\_name |  |  |  |

公式(editable)：<https://docs.djangoproject.com/en/4.2/ref/models/fields/#django.db.models.Field.editable>

verbose\_name

views.pyではこれで取得できるらしい？

verbose\_field\_names = map(lambda x: x.verbose\_name, filtered\_fields)

##### get\_absolute\_url

特定のモデルの詳細画面を表示する

##### limit\_choices\_to

models.ForeignKeyにのみ存在するフィールド。

外部キーを選択する際の制限を指定する。

##### Modelクラス

概要

モデルを操作する「モデルマネージャ」が「クエリセットAPI」を通じてデータベースとのクエリのやり取りを行う。

基本

ファイルパス：/myapp/models.py

from django.db import models

class Book(models.Model):

class Meta:

db\_table = ‘book’ # テーブル名を定義。基底クラスからの継承？

title = models.CharField(verbose\_name=‘タイトル’, max\_length=255)

price = models.IntegerField(verbose\_name=‘価格’, null=True)

def \_\_str\_\_(self):

return self.title

編集が終わったら

python manage.py makemigrations app\_name # models.Model を参考にテーブルが作成される

python manage.py migrate app\_name # 更新

##### update\_counter

??

##### UniqueConstraint

複合ユニークを用いる

from django.db import models

class Reservation(models.Model):

room = models.CharField(verbose\_name="部屋名", max\_length=20)

date = models.DateField(verbose\_name="予約日")

class Meta:

constraints = [

models.UniqueConstraint(fields=[‘room’, ‘date’], name=‘unique\_booking’),

]

注意点

複合ユニークの対象になる列（この場合roomとdate）はnot null制約が必要。

UniqueConstraintはDjango ver 2.2以上。それ未満のバージョンはunique\_togetherを使用する。（2022年4月現在非推奨）

class Reservation(models.Model):

room = models.CharField(verbose\_name="部屋名", max\_length=20)

date = models.DateField(verbose\_name="予約日")

class Meta:

unique\_together = (‘room’, ‘date’,) # tuppleで指定するので、最後のドットを忘れないように

unique\_together = [[ ‘room’, ‘date’ ]] # こんな情報もあるが？

補足　SQLではこんな感じになる（らしい）

CREATE TABLE example(

column1 SERIAL PRIMARY KEY,

column2 INTEGER NOT NULL,

column3 INTEGER NOT NULL,

UNIQUE ( column2, column3 )

);

##### Userクラス

概要

Djangoの色々な機能の恩恵を受けられるユーザークラス。

複数のクラスを継承している

|  |  |  |
| --- | --- | --- |
| クラス名 | 継承元 | 備考 |
| User | django.contrib.auth.models.AbstractUser |  |
| AbstractUser | django.contrib.auth.base\_user.AbstractBaseUser |  |
|  | django.contrib.auth.models.PermissionMixin |  |

・django.contrib.auth.models.Userクラス。createsuperuser で作成したユーザー。

ユーザーの種類

|  |  |  |  |
| --- | --- | --- | --- |
| 種類 | 概要 | 条件 | 備考／運用のメモ |
| スーパーユーザー | すべての権限が与えられる。 | is\_superuser = True | 開発者など、数名のみに与えるべき権限。 |
| スタッフユーザー | 管理画面にログインできる。 | is\_staff  = True | Webアプリを管理するスタッフにこの権限を与える。 |
| アクティブユーザー | 有効なDjangoアプリのユーザーかどうか。 | is\_active  =True | 例えば退職したスタッフや、退会した顧客はアクティブユーザーで無くする。 |

補足：is\_authenticatedは「ログインしているかどうか」を示すらしい。

関連モジュール

django.contrib.auth.models スーパーユーザー関連？

ユーザーの作成

管理者、一般ユーザーそれぞれ作成方法が異なる。

superuser（管理者）

python manage.py createsuperuser # 対話で行う

python manage.py createsuperuser --username=joe --email=joe@example.com # 引数で行う

user（一般ユーザー）

・Django管理サイト（/admin）にログインして作成

・create\_userメソッドを用いる

user = User.objects.create\_user(‘john’, ‘lennon@thebeatles.com’, ‘johnpassword’)

次ページに続く

必須のフィールド

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Abstract  User | Abstract  BaseUser | Perm  Mixin | 対象 | 備考 |
| password | 継承 | 〇 |  | アプリユーザーとして必須 |  |
| last\_login | 継承 | 〇 |  | アプリユーザーとして必須 |  |
| is\_active | 〇 | 〇 |  | アプリユーザーとして必須 | models.BooleanFieldとしてoverrideする必要がある |
| is\_authenticated | 継承 | 〇 |  | アプリユーザーとして必須 | 常にTrueを返す必要がある |
| is\_staff | 〇 |  |  | createsuperuser |  |
| is\_superuser |  |  | 〇 | Django管理サイトへのログイン |  |
| has\_perm |  |  | 〇 | Django管理サイトへのログイン |  |
| has\_module\_perms |  |  | 〇 | Django管理サイトへのログイン |  |
| USERNAME\_FIELD | 〇 |  |  |  | レコード識別フィールド名 |
| REQUIRED\_FIELDS | 〇 | 〇 |  | manage.py createuser |  |

その他主要フィールド

|  |  |  |  |
| --- | --- | --- | --- |
| フィールド名 | データ型 | 提供クラス | 備考 |
| username | CharField | AbstractUser |  |
| email | EmailField | AbstractUser | EmailFieldはCharFieldを継承 |
| first\_name | CharField | AbstractUser |  |
| last\_name | CharField | AbstractUser |  |
| is\_staff | boolean | AbstractUser |  |
| date\_joined | DateTimeField | AbstractUser |  |

補足

Web情報で「is\_admin」というフィールドを定義している例が見られるが、Djangoとしては全く意味のないフィールド。（フィールド名は自由に決めても問題はないが）

is\_superuserと混同して理解している可能性がある。

has\_permは自分で実装、または継承（override）する事もできる

def has\_perm(self, perm, obj=None):

return True

次ページに続く

USERNAME\_FIELD

概要

・Userレコードを識別する為のフィールド。

・Django管理サイトへのログインに使われる。

・対話式のmanage.py createsuperuserの第一パラメータ

・Django標準のユーザークラスではusernameが使われる。

・以下の例では、フィールドを一意に指定するために、identifier フィールドが使われている。

class MyUser(AbstractBaseUser):

identifier = models.CharField(max\_length=40, **unique=True**) # unique制約が必要

...

USERNAME\_FIELD = ‘identifier’

注意点

・対象のフィールドがunique=Trueである必要がある。

・REQUIRED\_FIELDSに対象のフィールドを含めてはならない。

REQUIRED\_FIELDS

定義クラス：AbstractBaseUser (def.= [] )

概要

manage.py createsuperuserの引数として必要となるフィールド名を指定する。

注意

USERNAME\_FIELDのフィールドを対象にしてはいけない。

EMAIL\_FIELD

get\_email\_field\_name()で返すフィールドを文字列で指定。

補足

django.contrib.auth.models.AnonymousUserは未ログインのユーザーを扱う（らしい）

資料

カスタムclass関連の公式資料<https://docs.djangoproject.com/ja/4.0/topics/auth/customizing/>

##### Q

filterメソッドと共に使われる。特にor条件で使われる事で有名。

from db.models import Q

objects = NewModel.objects.filter(Q(title\_startswith=‘1’))

MyUser.objects.filter( Q(name\_\_icontains="taro")|Q(age\_\_lte="20") ) # or検索

listで使う際の考え方

from db.models import Q

q\_list = Q()

q\_list.add(Q(title \_\_contains='hoge'), Q.OR) # 最初のconnector(OR/AND)は無視される（らしい）

q\_list.add(Q(title \_\_contains='fuga'), Q.AND) #

q\_list.add(Q(title \_\_contains='foo'), Q.OR) #

結果：WHERE title LIKE "%hoge%" AND title LIKE "%fuga%" OR title LIKE "%foo%"

動的に条件を付ける

from django.db.models import Q

condition=Q(code=instance.code)

found\_item =Company.objects.filter(condition).first()

# こんなこともできるらしい

queries = [Q(contents\_\_contains=word) for word in words]

#### django.db.models.expressions

##### F

モデルのフィールド値を取り出す。

.aggregate(total=Sum(F('purchase\_info\_\_price\_per\_unit') \* Value(3)))

##### Value

値を直接指定

<https://docs.djangoproject.com/ja/4.2/ref/models/expressions/#value-expressions>

##### Case

関連：When

##### raw

raw(raw\_query, params=(), translations=None)

##### RawSQL

正直、良く分からない

subquery = RawSQL('SELECT 金型番号 FROM mold\_list where 区分 = 1', [])

queryset = RfidList.objects.filter(金型番号\_\_in = subquery)

print('[ raw を用いたSQLの直接実行 ]')

for rfid in queryset:

print(rfid.rfid)

公式：https://docs.djangoproject.com/en/4.2/topics/db/sql/

#### django.db.models.aggregates

##### Sum

#### django.db.models.query

##### QuerySet

#### django.db.models.field

models.Field継承型一覧

<https://docs.djangoproject.com/ja/4.2/ref/models/fields/#field-types>

##### TextField

注意点：forms.TextFieldは存在しない

#### django.db.models.functions

##### Concat

文字列の結合。データベースでの「||」と同等

from django.db.models.functions import Concat

（公式）<https://docs.djangoproject.com/ja/4.0/ref/models/database-functions/>

#### django.db.models.aggregates

##### ArrayAgg

注意点2.2以降で導入

books\_with\_joined\_authors = Book.objects.annotate(

joined\_authors=Concat(

ArrayAgg(‘authors\_\_name’, ordering=’authors\_\_name’, distinct=True, delimiter=’-’),

output\_field=CharField(),

)

)

#### django.forms

コンストラクタ

initial

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

author\_id = self.initial.get(‘author’)

##### queryset

ModelChoiceFieldなどで、選択候補を指定する。

例

form.fields['author'].queryset = Author.objects.filter(is\_target=True)

##### empty\_values

空白とみなされる値のtuple。

djangoではこれに該当する値の場合データベース側ではNULL(None)として扱う。

使い方例

if not value in self.empty\_values:

return None

また、末端では以下のように定義されている。

[ django.core.validators.py ]

EMPTY\_VALUES = (None, '', [], (), {})

##### Field継承クラス（forms特化）

概要

django.db.models.fields内の同名のクラスに対応するフォームクラス群

|  |  |  |
| --- | --- | --- |
| class名 | 標準widget | 備考など |
| ChoiceField | Select |  |
| MultipleChoiceField |  |  |
| ModelMultipleChoiceField |  |  |
| ComboField |  |  |
| ModelCoiceField | django.forms.models | 外部キー先のモデルを選択する。 |

（参考）<https://docs.djangoproject.com/ja/4.0/ref/forms/fields/#built-in-field-classes>

具体的な使い方

以下のようにするとMeta: クラスを使わずとも、似たような事ができる。

class BookForm(forms.ModelForm):

label = {

‘publish\_date’ : ‘発刊日’,

}

publish\_date = forms.DateTimeField( # Modelで定義したフィールドと同名にする

label=‘発刊日’, # 上で定義しているが、再度入力が必要（リセットされる）

widget=forms.DateInput(attrs={"type": "date"}), # widgets ではなく、widgetなので注意！Metaではwidgets

input\_formats=[‘%Y-%m-%d’]

)

補足

django.forms.fields.IntegerFieldとdjango.forms.IntegerFieldは全く同じっぽい 2021-09-24

⇒（エイリアス；別名　みたいのでそうゆう事ができるの？）

基本フィールド

##### ChoiceField()

単一選択。

widget(def.)=Select htmlのinput：<select>

form.fields[‘category’] = forms.ChoiceField(

choices = ((cat.pk, cat.title) for cat in Book.objects.all()), # ((key, value),)のtuple型でないといけない

)

公式：<https://docs.djangoproject.com/ja/4.0/ref/forms/fields/#choicefield>

##### ChoiceField系共通情報

empty\_label 未選択（null）の選択肢として表示される文字列を指定。def.=‘---------’

通常Formの場合

class BookForm(forms.ModelForm):

book\_type = forms.ChoiceField(queryset=..., empty\_label=None) # これでNoneが選択できなくなる

ModelFormの場合

class BookForm(forms.ModelForm):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields['book\_type'].empty\_label = None

##### MultipleChoiceField()

###### 概要

複数選択可能なドロップダウンボックス。

[ forms.py ]

widget(def.)=SelectMultiple htmlのinput：<select multiple>

my\_selection = forms.MultipleChoiceField(

label = "複数選択ボックス",

choices = [(1, 'val1'),(2, 'val2'),(3, 'val3')]

)

関連するwidget

forms.CheckboxSelectMultiple inputがcheckboxに。注意：htmlのrequiredが効かなくなる。

###### POSTメソッドで選択肢を取得する方法

[ views.py ]

selected\_items = form.cleaned\_data['my\_selection']

selected\_items = request.POST.**getlist**('my\_selection', None)

# この方法だと、複数の選択肢の内、最後のものしか受け取れない

selected\_list = request.GET.get ('my\_selection', None)

補足

formを使わずに、html上で複数選択を実装する方法（非推奨）

[ views.py]

class SampleView(ListView):

get():

query\_month = int( request.GET.get(‘month’, None) ) # int型に変換しておくのがポイント

context={

‘month\_list’ : context[‘‘] = range(1,13) # int型のままにしておくのがポイント

}

[ template/sample.html ]

<select name="month" id="mounth">

{% for mth in month\_list %}

<option value={{mth}}>

{% ifequal mth selected\_month %} selected {% endifequal %}>{{mth}}</option>

{% endfor %}

</select>

###### choicesを動的に生成する

[ forms.py ]

class BookForm(forms.Form):

my\_selection = forms.MultipleChoiceField(label='項目')

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields['my\_selection'].choices = [ (book.pk, book) for book in Book.objects.all() ]

補足：choicesはタプルのタプル、またはタプルのリストで指定する

[ views.py ] （views.pyで動的に生成したいとき）

form = BookForm(request.POST)

form.fields['my\_selection'].choices=[ (book.pk, book) for book in Book.objects.filter(author = author ) ]

注意点：views.pyで動的に作成する場合はpostメソッドの時も**['my\_selection'].choicesをgetの時とまったく同じ値に設定**しないと永久にform.is\_validにならない

[ html（テンプレート）の例 ]

<tbody>

{% for pk, obj in form.my\_selection.field.widget.choices %}

<tr>

<td>

<input type="checkbox" name="items" value="{{ pk }}"

{% if pk in form.my\_selection.field.initial %}checked{% endif %}> <!-- 値の復元を行う場合 -->

{% endif %}

</td>

</tr>

{% endfor %}

</tbody>

補足：ModelChoiceFieldではchoicesの代わりにquerysetというフィールドに値を入れる

selection = forms.ModelChoiceField (

queryset = Book.object.all()]

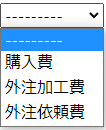
)

（参考）ModelChoiceField

###### 資料

参考　<https://itc.tokyo/django/change-dynamically-values-of-choices/>

##### ModelChoiceField

モデルを直接選択

widget(def.)=Select htmlのinput：<select>

class ProductForm(forms.ModelForm):

brand\_cd = forms.ModelChoiceField(queryset=Product.objects.all()) # このように設定

補足

文字列の入力はできない。

（公式）<https://docs.djangoproject.com/ja/4.0/ref/forms/fields/#modelchoicefield>

##### ModelMultipleChoiceField

class BookCreateForm(forms.Form):

book\_list = forms.ModelMultipleChoiceField(

queryset=Tag.objects,

widget=forms.CheckboxSelectMultiple

)

class Meta:

model = Book

fields = '\_\_all\_\_'

##### ComboField

test = forms.ComboField(fields=[forms.CharField(max\_length=20), forms.ChoiceField(choices = (1,2,3,4,5))])

GUIは特に変更なかった（CharFeild；Textboxのまま）

バリデーションの時に役に立つ、というだけの話？

##### CheckSelectMultiple

例

FAVORITE\_COLORS\_CHOICES = [

('blue', 'Blue'),

('green', 'Green'),

('black', 'Black'),

]

class SimpleForm(forms.Form):

favorite\_colors = forms.MultipleChoiceField(

required=False,

widget=forms.CheckboxSelectMultiple,

choices=FAVORITE\_COLORS\_CHOICES,

)

##### TextChoices

Django3.0からは

##### IntegerChoices

Django3.0からは

##### Form

フォームの基本的な機能を提供する。

Metaクラス

fields 必須オプション

model クラスを直接指定。（クラス名の文字列では無い）

labels コントロールの左に表示する文字列（ラベル）をdict型で指定する

labels = {

‘delivery\_date’ : ‘配達予定日’ # フィールド名delivery\_dateに対するラベル文字列

}

# labels指定がない場合はmodel.pyのverbose\_nameになる

help\_texts ヒント文字列

field\_order 並び順？

widget**s** 入力の際に使うコントロールを選択する。（例：email，日付など）

補足：widgetではなくwidget**s**なので注意。（後述するField継承ではwidget）

⇒ 後述するField継承クラスを用いて、Field毎に設定する方が合理的

コンストラクタ

initial フォームの初期値を指定する。

[ views.py ]

def get(self, request, \*args, \*\*kwargs):

form = BookForm(initial =

{

‘pricet’ : 2000,

‘memo’ : record.memo,

})

return render(request, ‘book/edit.html’, { ‘form’ : form })

以下の方法でも良い

form.fields[‘pricet’] = forms.IntegerField(

initial=2000,

)

form.fields[‘pricet’].initial = 2000

～Form続き

save

ModelFormを使う時のみ利用可能

commit

book\_obj = form.save(commit=False)

<https://docs.djangoproject.com/ja/4.0/topics/forms/modelforms/#the-save-method>

##### Form.fields

queryset

ForeignKeyなどのドロップダウンで表示する項目の制限などを行う。

self.fields[‘bookshelf’].queryset = Bookshelf.objects.filter(user\_id=id)

initial

初期値

from django.utils import timezone

self.fields[‘date’].initial = timezone.now().date()

（公式）<https://docs.djangoproject.com/ja/4.0/ref/forms/fields/>

##### ModelForm

データベースの１レコード（モデルの１インスタンス）を編集するのに向くフォーム。

モデルの登録や変更を伴わない場合（検索フォームなど）でモデルのフィールドを利用したい場合，では少しテクニックが必要になる。

基本形

class MyModelForm(forms.ModelForm):

# 特定フィールドのみオーバーライドすることも可能

detail = forms.CharField(widget=forms.TextInput())

class Meta:

model = Book

fields=['title', 'detail']

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

def clean(self):

super().clean()

特徴

・self.instanceでModelのインスタンスにアクセスできる。

#### django.forms.widgets

##### widget

ブラウザで使用されるコントロール（TextBoxなど）を指定・変更する。

（通常コントロール（Widget）はmodelのFieldクラスによって，自動的に選択される）

指定方法１：Formクラスのフィールドで指定する

class RegisterForm(forms.ModelForm):

password = forms.CharField(

widget=forms.PasswordInput(render\_value=True),

)

指定方法２： Formクラスの Metaクラスで指定する

class RegisterForm(forms.ModelForm):

class Meta:

widgets = {

‘password’: forms.PasswordInput(attrs={‘placeholder’: ‘ユーザー名’}),

}

指定用法３：Formクラス内のdict，fieldsで指定する

class ProfileForm(forms.ModelForm):

def \_\_init\_\_(self, \*args, \*\*kwargs):

self.fields[‘username’].widget=forms.PasswordInput(render\_value=True)

補足：forms.DateTimeFieldもforms.widgets.DateTimeFieldも全く同じ

指定用法４：views.py

modelの条件に応じた対応が必要な場合

def get\_context\_data(self, \*\*kwargs: Any) -> Dict[str, Any]:

context = super().get\_context\_data(\*\*kwargs)

# ModelFormを使う場合

form = self.get\_form()

# 自作のFormクラスを使う場合

form = MyForm()

form.fields['title'].widget = forms.PasswordInput(attrs={‘placeholder’: ‘ユーザー名’})

context['form']=form

return context

# 条件に応じた対応が必要ではない場合はget\_formでも対応できるが、それであればforms.pyで対応した方が良い

def get\_form(self, form\_class=None):

form = super().get\_form(form\_class)

メモ

widget

実際に表示されるhtmlフォームのコントロールの設定を行う

self.fields[‘name’].widget.attrs = {‘autocomplete’ : ‘off’}

form.fields['release\_date'].widget=forms.DateInput(attrs={"type": "date"}) # ウィジェットだけ変更する場合

form.fields['item\_name'].widget.attrs={ 'autocomplete' : 'off', } # attrsだけ変更する場合

form.fields['release\_date'] = forms.DateTimeField( # フィールドごと変更する場合

label=‘発売日’,

widget=forms.DateInput(attrs={"type": "date"}),

input\_formats=[‘%Y-%m-%d’],

required=False,

)

##### attr

厳密にはfo

###### html属性のまとめ

|  |  |  |
| --- | --- | --- |
| 属性名 | データ型など | 概要 |
| autocomplete | 'on'/'off' | 過去の入力値を参考にした自動補完機能 |
| disabled |  | フォームを Submit しても値の送信が行われない |
| class |  | cssのclass |
| id |  |  |
| list |  | 候補表示機能（datalist）で使う具体的なlist名を指定 |
| pattern | 文字列 | 入力に使える文字列を正規表現で指定する |
| placeholder |  | 薄いグレーで表示されるヒント文字列 |
| readonly |  | 値の変更は行えないが、コントロールは有効 |
|  |  |  |

具体例

class

attrs[‘class’] = 'form-control'

pattern

attrs['pattern'] = '[a-zA-Z0-9]'

###### 指定方法

方法１：Formの\_\_init\_\_ で指定

[ forms.py ]

class DailyReportSearchForm(forms.Form):

title = forms.CharField()

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields['title'].widget.attrs = { 'autocomplete' : ‘off’}

self.fields[‘title’].widget.attrs[‘autocomplete’] =‘off’ # dict型なので、このような書き方でもよい

公式：<https://docs.djangoproject.com/ja/4.1/ref/forms/widgets/#styling-widget-instances>

方法２：FormのMeta で指定

[ forms.py ]

class BookForm(forms.Form):

class Meta:

attrs = {

'title': 'off',

}

方法３：views.pyで指定

class BookView(TemplateView):

template\_name=‘myapp/book/detail.html’ # この例におけるテンプレート名は適当

def get\_context\_data(self, \*\*kwargs: Any) -> Dict[str, Any]:

form = BookForm()

form.fields[‘title’].widget.attrs = {‘autocomplete’ : ‘off’}

方法４：Formクラスのwidgetで指定

[ forms.py ]

class BookForm(forms.Form):

title = forms.CharField(

widget=forms.TextInput(attrs={‘autocomplete’ : ‘off’})

備考

・ウィジェットのクラスまで指定する必要があり、やや煩雑

方法５：inputタグに直接

template/index.html

<input type="text" name="hoge" autocomplete="off">

但し、djangoでform内のタグを直接弄るのは現実的ではない。

###### input typeごとの対応

type= "time"

class TimeInputWidget(forms.TimeInput):

input\_type='time' # ここが重要

class MyForm(forms.Form):

time = forms.TextField( # Fieldは バリデーションに影響が出るが、基本的には何でも良い。

widget=TimeInputWidget(),

)

補足

htmlのtype=’time’について

<https://www.tagindex.com/html/form/input_time.html>

##### Widget継承クラス

|  |  |  |  |
| --- | --- | --- | --- |
| クラス名 | 概要 | イメージ | 備考 |
| forms.RadioSelect | ラジオボタン |  | <input type=”radio”> |
| forms.DateTimeField | 年月日選択 |  |  |
| forms.Textarea | 大きな文字列編集 |  | <textarea> |

Textarea

縦幅を変える方法（すべてうまく行かず2022-02-09）

[ forms.py ]

widgets = {

‘detail’ : forms.Textarea(attrs={‘rows’: 10}),

‘date’ : forms.DateInput(attrs={"type": "date"}),

}

detail = forms.CharField(

widget=forms.Textarea(attrs={‘rows’: ‘80’}),

)

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields[‘detail’].widget.update = {‘rows’ : ‘20’}

##### Input継承クラス

###### TextInput

###### NumberInput

###### EmailInput

###### PasswordInput

###### HiddenInput

[ forms.py ]

name =forms.CharField(widget=forms.HiddenInput,)

self.fields['name'].widget = forms.HiddenInput()

関連するDTLのメソッド

is\_hidden hiddenのフィールドかどうか

as\_hidden フィールドをtype=hiddenとする

公式：<https://docs.djangoproject.com/ja/4.1/ref/forms/widgets/>

2023-02-27追記

通常HiddenInputを使用する場合はテンプレート(html)にて使用した方が無難

（動作はWebブラウザによるらしい）

例：

<form>

{{ form.my\_hidden\_field }}

</form>

関連：テンプレート(html)のas\_hidden

#### django.http

##### HttpRequest クラス

|  |  |
| --- | --- |
|  | 説明 |
| path | リクエストされたURLのフルパス（ドメインを含まない）を格納した文字列 |
| method | String containing the HTTP method used by the request (‘GET’ or ‘POST’) |
| GET | http のGET メソッドで与えられたフォームの値を代表するQueryDict |
| POST | http のPOST メソッドで与えられたフォームの値を代表するQueryDict |
| raw\_post\_data | Unprocessed text of the complete POST data, only used in advanced cases, where POST data cannot be parsed into a QueryDict |
| COOKIES | A dictionary of cookies sent by the browser during the request |
| META | A dictionary of included HTTP headers; available headers will vary based on client and server |
| user | A User (or AnonymousUser) object from the ​authentication middleware |
| session | A special read-write dictionary provided by the ​sessions middleware |

GET

ip\_list = request.META.get(‘HTTP\_X\_FORWARDED\_FOR’) # グローバルアドレス

ip\_list = request.META.get(‘ REMOTE\_ADDR’) # グローバルアドレス

参考：https://qiita.com/3244/items/0b47d3ad91968fe15eb9

POST

request.POST.get(‘username’)

##### session

get(key, default=None)

fav\_color = request.session.get(‘fav\_color’, ‘red’)

pop(key, default=\_\_not\_given)

fav\_color = request.session.pop(‘fav\_color’, ‘blue’)

keys() キーのリストを返す？

items() 項目のリストを返す？

clear() セッションの初期化（ログイン状態も初期化される）

clear\_expired() セッションの内、期限切れ項目のみ破棄する。

flush() セッションとクッキーを破棄する。

set\_test\_cookie()

ブラウザがクッキーをサポートしているか判定するために、テスト用のクッキーをセットする。

test\_cookie\_worked() テストクッキーを正しく保存したかどうかを返す（True または False）。

delete\_test\_cookie() テストクッキーを削除する。

set\_expiry(value) セッションの有効期限を設定。

整数 指定した秒数活動が無ければセッションを破棄。

request.session.set\_expiry(300) # 300秒=5分間アクセスが無ければセッションを破棄

request.session.set\_expiry(0) # ブラウザを閉じた時にセッションを破棄

request.session.set\_expiry(None) #グローバルなセッション有効期限ポリシーにしたがう。

datetime または timedelta オブジェクトならば、指定された日時に破棄。

# datetime と timedelta の値がシリアライズできるのは、PickleSerializer を使っている場合のみ。

get\_session\_cookie\_age() クッキーの有効期限を返す

get\_expiry\_age() このセッションの有効期限までの残りの秒数を返す。

modification: セッションを最後に修正した時刻を datetime オブジェクトとして与える。デフォルトは現在の時刻。

expiry: セッションの有効期限の情報を datetime オブジェクト、int (秒数で)、または None として与えます。デフォルトの値は、もしあれば、セッションに保存されている set\_expiry() で得られる値、なければ None です。

（公式）<https://docs.djangoproject.com/ja/4.0/topics/http/sessions/>

setting.pyの関連項目

SESSION\_ENGINE = django.contrib.sessions.backends.db # データベースで保持(def.)

SESSION\_ENGINE = django.contrib.sessions.backends.cache # 永続化されないが高速。

SESSION\_ENGINE = django.contrib.sessions.backends.cached\_db # キャッシュ＋データベース。

SESSION\_ENGINE = django.contrib.sessions.backends.file # ファイルで保持

SESSION\_ENGINE = django.contrib.sessions.backends.signed\_cookies # クッキーで保持（SECRET\_KEY を使う）

SESSION\_FILE\_PATH # backends.file の時の保存場所。def. = /tmp

##### Http404 クラス

##### urlencode メソッド

（参考）https://chuna.tech/detail/84/

#### django.http.response

##### HttpResponseクラス

def hand\_crafted\_redirect\_view(request):

response = HttpResponse(status=302)

response[‘location’] = ‘/redirect/success/’

return response

###### set\_cookie

構文： set\_cookie(key, value=‘‘, max\_age=None, expires=None, path=‘/’, domain=None, secure=None)

max\_age クッキーが削除される時間を単位[秒]で指定

###### set\_signed\_cookie

set\_signed\_cookie(‘key’, max\_age=max\_age\_sec)

～django.http.response　HttpResponseクラス　つづき

###### contentプロパティ

内部変数\_containerのバイト配列を返す。

以下のように定義されている。

@property

def content(self):

return b''.join(self.\_container)

viewが生成するhtml文字列を確認する方法

response = render(request, 'my\_template.html', context)

print(response.content.decode())

##### FileResponseクラス

例：

response = FileResponse(open('example.pdf', 'rb'), content\_type="application/pdf")

以下のHttpResponseに相当する

response = HttpResponse(open('example.pdf', 'rb'), content\_type="application/pdf")

response['Content-Disposition'] = 'attachment; filename="example.pdf"'

as\_attachment ダウンロードさせるかどうか def.=False

content\_type MIMEの設定？

エクセルの場合（メモ）

class XlsxDownloadView(View):

def get(self, request, \*args, \*\*kwargs):

response = FileResponse(open('docs/xlsx/mybook.xlsm', 'rb'))

response['content\_type'] = 'application/excel'

response['Content-Disposition'] = f'filename="mybook.xlsx"'

return response

BytesIOなどをもちいて動的に作成する例

import io

from reportlab.pdfgen import canvas

buffer = io.BytesIO()

pdf=canvas.Canvas(buffer, bottomup=False, pagesize=landscape(A4))

# ここでpdf を動的に作成する

buffer.seek(0)

response = FileResponse(buffer, content\_type="application/pdf", filename='example.pdf')

公式：<https://docs.djangoproject.com/ja/4.2/ref/request-response/#fileresponse-objects>

##### HttpResponseNotFoundクラス

##### JsonResponseクラス

補足：querysetからJSONResponseを作る方法

book \_list = Book.objects.values('title')

return JsonResponse({'book\_title\_list': list(book\_list)})

#### django.shortcuts

##### render

テンプレートのレンダリングをする

構文：render(request, template\_name, context=None, content\_type=None, status=None, using=None)

戻り値：django.http.HttpResponse

content\_type 省略可

status 省略可

using 省略可

（補足）urls.py上の引数はcontextで渡す

context = { ‘book\_id’ : 1 }

return render(request, ‘shop/top.html’, context)

（補足）django3.0まではrender\_to\_responseという名前だったらしい

##### redirect

リダイレクトを行う

from django.shortcuts import redirect # 必要なimport

from django.urls import reverse # 必要なimport

return redirect(‘/book/top’) # myapp/book/top へリダイレクト

return redirect(‘/book/’, kwargs={'id': 1}) # kwargs（URL引数）を使ってurls.pyに引数を渡す

return redirect(reverse('myapp:index')) # urls.pyからの逆引き。（URL変数が無い場合）

url = reverse('myapp:index',kwargs={'id': 1 }) # reverseで引数を与える方法。urls.py上の定義変数はすべて渡す。

url += ('?title=%s' % (title)) # urls.pyに記載の無いクエリ文字列がある場合

return redirect(url)

構文：redirect(to, \*args, permanent=False, \*\*kwargs)

HttpResponseRedirectもしくはHttpResponsePermanentRedirectクラスを返す

to

redirect('myapp:index') # ビュー名

redirect('/my/view/') # パス

redirect('https://example.com/') # URL

kwargs

クエリ文字（URL引数）を指定する。

return redirect(reverse("myapp:book\_list", kwargs={"id" : book\_id})) # book\_listはURL引数idを取る必要がある

（参考）https://chuna.tech/detail/84/

##### renderとredirect

違いのまとめ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| メソッド名 | context変数 | 転送先の指定方法 | kwargs |  |
| render | 使える | .htmlファイル名 | 引き継がれる |  |
| redirect | 使えない | url文字列 | 要再指定 | 変数が必要な場合はsessionを使う |

詳細の比較

|  |  |
| --- | --- |
| メソッド名 | 詳細／補足 |
| shurtcuts.redirect | 指定したURLへのリダイレクトを行う。  URL文字列を受け取ってHttpResponseRedirectもしくはHttpResponsePermanentRedirectクラスを返す。 |
| shurtcuts.render | テンプレート（.htmlファイル）にレンダリングを行う。  templatesディレクトリからの相対パスを指定する。  **request変数が引き継がれる**点に注意。postでは原則redirectを使う。 |

関連

urls.reverse

urls.pyの名前からURL文字列の逆引きを行い、**url文字列**を返す。reverse(‘[app名:] url文字’)

URLResolverクラス内メンバに値を格納するだけ。内部の\_\_repr\_\_メソッドでURL文字列にして返す？

##### get\_object\_or\_404

from django.shortcuts import get\_object\_or\_404

book = get\_object\_or\_404(Book, pk=idx) # プライマリーキー=idxのレコード（要素）を取得する

#### django.test

##### TestCase

assertEqual 処理の結果、値が等しいかどうかを判定する？

[ myapp/tests.py ]

class TestUrls(TestCase):

def test\_toppage (self):

found = resolve(“/myapp/top/”)

self.assertEqual(resolve(url).func, signup)

類似のメソッド

|  |  |
| --- | --- |
|  |  |
| assertAlmostEqual |  |
|  |  |
| assertContains |  |
| assertFalse |  |
| assertGreater |  |
| assertGreaterEquall |  |
| assertLess |  |
| assertLessEquall |  |
|  |  |
| assertTemplateUsed |  |
| assertRaises | 例外が発生されるかどうかを確認 |
|  |  |

公式：<https://docs.python.org/ja/3/library/unittest.html#unittest.TestCase.assertNoLogs>

例

assertRaises 例外が発生されるかどうかを確認

def test\_exception2(self):

self.assertRaises(Exception, func)

fixtures

jsonファイルを用いてテストデータを作成する。

本番用データベースからデータをdumpすれば、本番用のデータも使える。

ディレクトリ構成

myproject

├ myapp

│ ├ fixtures

│ │├ dump\_book.json manage.py dumpdata で作成

│ │├ dump\_shop.json 同上

│ ├ tests

│ │├ test\_book.py

補足：settings.pyのFIXTURE\_DIRSで指定しない限り、各アプリのfixturesディレクトリが対象になる。

先にダンプファイルを作成しておく

terminal

python manage.py dumpdata myapp.Book > myapp\fixtures\dump\_book.json

[myapp/tests/test\_book.py ]

from django.test import TestCase

from gyoumu.models import Book

class BookTest(TestCase):

fixtures = [‘dump\_book.json’, ‘dump\_shop.json’] # 複数の場合はコンマ区切りで追加

def test\_add(self):

pass

テスト用のデータベースを残したい場合

DATABASES = {

…

‘default’: {

# この行を追加

‘TEST’: {

‘NAME’: ‘test\_database’,

},

}

}

指定しない場合、データベース名はtest\_projectname（ディレクトリ名？）となる

##### Client

##### RequestFactory

views.pyのテストの為のクラス。（詳細不明2022-05-24）

from django.urls import reverse

from django.test import TestCase, RequestFactory

from django.contrib.auth import get\_user\_model

class ViewTest(TestCase):

fixtures = [‘dump\_users.json’] # 本番データベースからdumpしたもの

def setUp(self):

self.factory = RequestFactory() # 本クラス

self.user = get\_user\_model().objects.get(pk=1) # ユーザーの指定。ここで作成しても良い。

def test\_top\_page(self):

request = self.factory.get(reverse(‘gyoumu:gyoumu\_top’)) # htmlリクエスト

request.user = self.user # 実行ユーザーの指定

response = TopView.as\_view()(request) # viewクラス、またはメソッドの指定

self.assertEqual(response.status\_code, 200)

<https://docs.djangoproject.com/ja/4.0/topics/testing/advanced/>

#### django.template

##### simple\_tagデコレータ

DTLで利用可能なタグ関数を作成する

[ myapp/templatetags/my\_filters.py ]

@register.simple\_tag

def multiply(value1, value2):

return value1 \* value2

{% load my\_filters %}

<!-- num1=3, num2=4のとき、HTML上には12が表示される -->

<span>掛け算:{% multiply num1 num2 %}</span>

##### filterデコレータ

##### Contextクラス

##### Templateクラス

具体例：

def directhtml\_view(request):

context = {'message': 'Hello, World!'}

template = Template("<h1>{{ message }}</h1>")

rendered\_html = template.render(context)

return HttpResponse(rendered\_html)

#### django.urls

##### path()

構文：path(route, view, kwargs=None, name),

route

対象となるurlのエイリアス。**最初がスラッシュではない**事に注意。include以外は重複不可

<> を使うとurlから値が取得できる

path(‘articles/:year>/’, views.year\_archive), # ://samplesite/articles/1999 などに対応する

|  |  |  |
| --- | --- | --- |
| コンバータ | 意味 | 等価な正規表現 |
| int | val>=0 の整数 | [0-9]+ |
| str | 空で無い文字列（/ を除く） | [^/]+ |
| path | 空で無い文字列（/ を除く） | .+ |
| slug | 半角英数字，ハイフン，アンダーバーで構成された文字列 | [a-zA-Z0-9-\_]+ |
| uuid | UUID型（8-4-4-4-12桁の小文字の16進数） |  |

公式：<https://docs.djangoproject.com/ja/4.0/topics/http/urls/#path-converters>

kwargs

対象のビューに渡すdict型の引数

view

実行するview(Controller) とメソッド名を指定する。

name

識別のIDとなる文字列。{% url 'name' %} や逆引きで使用。省略可能だが**強く推奨**。

##### re\_path()

正規表現でURL名前解決を行う。

[ config/urls.py ]

from django.urls import include, re\_path

urlpatterns = [

re\_path(r’^index/$’, views.index, name=‘index’),

re\_path(r’^bio/(?P<username>\w+)/$’, views.bio, name=‘bio’),

re\_path(r’^blog/’, include(‘blog.urls’)),

...

]

（公式）<https://docs.djangoproject.com/ja/4.0/ref/urls/#re-path>

##### include

別のurls.pyをURLディスパッチャとして追加する。

例えばmyproject/config/urls.pyを分離して，myapp用のurls.pyを使う場合は

[ config/urls.py]

from django.urls import include, path

urlpatterns = [

path(‘shop/’, include(‘shop.urls’) # *myproject*myapp/urls.py の利用

]

namespace URLの逆引きなどで使う名前空間の文字列を指定

urlpatterns = [

path(‘users/’, include(‘users.urls’, namespace = ‘users’)),

]

##### resolve

url文字列からview関数を作成する。

対応するビュー関数にURLパスを解決するために使用する。

あまり使う機会はないが、Testなどで使う事もある。

構文：resolve(path, urlconf=None)

[

view = resolve('/myapp/shop/') # url文字列を指定

view = resolve(reverse('myapp:shop\_top')) # reverseと併用しても良い

公式：<https://docs.djangoproject.com/ja/4.0/ref/urlresolvers/#resolve>

##### reverse

URLの逆引きを行う。url.pyに記述してあるURLをnameから抽出する。

構文：reverse(viewname, urlconf=None, args=None, kwargs=None, current\_app=None)

例

[ users/urls.py ]

urlpatterns = [

path("<str:username>/", view=user\_detail\_view, name="detail"),

]

[ users/views.py ]

def get\_detail\_url(self):

return reverse("users:detail", kwargs={"username": self.request.user.username})

URL引数が必要な場合

return redirect(reverse(‘myapp:shop ‘, kwargs={ ‘book\_id’: 1 }))

return redirect(reverse(‘ myapp:shop ‘), book\_id = 1)

return redirect(reverse(‘ myapp:shop ‘), book\_id = self.kwargs['book\_id'])

return redirect(‘myapp:shop’, book\_id = 1)

公式：<https://docs.djangoproject.com/ja/4.0/ref/urlresolvers/#reverse>

##### reverse\_lazy

遅延実行式のreverse。

特にクラス式のビューでクラス内変数としてURL文字列を扱う場合、遅延実行させる必要がある。

例

path(‘alias/’, views.top, name=‘top\_page’), # views.py のメソッドtop を呼び出す。

path(‘‘, include(‘myapp.urls’)), # /myapp/urls.py への参照を追加。ドット≒スラッシュ。

path(‘myapp/’ , include(‘myapp.urls’)) # この場合は/myapp/ で始まるurlの場合にのみ適用

厳密には/project/config/setting.py に記載されているROOT\_URLCONF の値を参照。

例）ROOT\_URLCONF =‘config.urls’　→ BASE\_DIR/config/urls.py

注意点

urls.py で使われているpathメソッドの**第一引数はバックスラッシュで始まってはいけない**。

反面，html ファイル内の<a>で hoge/fuga (最初の / 無し)にすると現在のパスから相対的なパスとなり，

/ 有りだとBASE\_DIR/hoge/fuga/ となる。

また，なぜかhtmlファイル内のurlでは最後に / をつけない方がいいらしい。

htmファイル内で {% url ‘url.pyで定義したurl名’ %}　でアクセスできる。

#### django.utils

##### timezone

timezone.localdate ???

timezone.localtime

##### django.shortcuts

##### staticfiles\_urlpatterns

静的ファイルの利用が可能な適切なURLを返す

[ urls.py ]

from django.contrib.staticfiles.urls import staticfiles\_urlpatterns

# URLパターンを記述

urlpatterns += staticfiles\_urlpatterns()

#### django.utils.encodinjg

##### iri\_to\_uri

IRI URI) を拡張したものである

#### django.views.generic

##### 概要

汎用ビューを提供する

##### base

generic.base.View Djangoで使える様々な汎用Viewの基底クラス。

generic.TemplateView テンプレートの表示に特化した基本汎用ビュー。

generic.RedirectView リダイレクトに特化した基本汎用ビュー。

generic.list.ListView モデルオブジェクトの一覧を表示する

generic.detail.DetailView モデルオブジェクトの詳細を表示する

generic.edit.FromView フォームを利用する

generic.edit.CreateView モデルオブジェクトを登録する

generic.edit.UpdateView モデルオブジェクトを更新する

generic.edit.DeleteView モデルオブジェクトを削除する

（参考）<https://docs.djangoproject.com/ja/3.2/ref/class-based-views/>

##### CreateView系の共通事項

編集するフィールドの指定方法（上から順に推奨；主観ではあるが）

・views.pyでform\_classフィールドを利用する

・views.pyでfieldsフィールドを利用する

・views.pyでmodelフィールドを利用する

共にSingleObjectTemplateResponseMixinを継承している。

CreateView、UpdateViewはModelFormMixinを継承したクラスを継承している（例：BaseCreateViewはModelFormMixinを継承）

##### formclass

[ views.py]

from .forms import BookForm

class BookCreateView(CreateView):

form\_class = BookForm #

[ forms.py]

class BookForm(forms.ModelForm):

class Meta:

model = Book

fields = ('title','price',)

##### fields

基本

[ views.py]

from .models import Book

class BookCreateView(CreateView):

fields = [‘title, price’] # fields = [‘\_\_all\_\_’ ] とする事自体はできるが、非推奨

外部キー対応

[ views.py]

class BookCreateView(CreateView):

fields = [‘author’] # 外部キーauthor（著者）を選択する場合

・2022-05-10 author.author\_name など、外部テーブルの１フィールドを対象にする事はできないらしい

⇒ そのように凝った事をする場合はFormView＋Form、またはViewを継承してフルカスタマイズ。

##### model

CreateView, ListViewなどで使用。

対象となるモデルをクラスで指定（クラスのインスタンスやクラス名の文字列ではない）

[ views.py ]

from .models import Book

class BookCreateView(CreateView):

model = Book

補足：ListViewではmodelでモデルクラスの指定か、get\_querysetメソッドのオーバーライドのどちらかが必要になる。

##### 汎用ビューの継承

RedirectViewを継承する例

from django.views.generic import RedirectView

class IndexView(RedirectView):

url = ‘accounts/login/’

index = IndexView.as\_view()

##### success\_url

class BookCreateView(CreateView):

success\_url = reverse('myapp:book\_top')

補足：get\_success\_urlというメソッドもあり、効果はまったく同じ。（らしい）

##### template\_name

class BookView(TemplateView):

model = Book

template\_name = 'myapp/book.html'

TemplateView限定のフィールドという事は無い。ListViewなど、様々なViewで利用される。

また、以下のように普通のフィールド変数として利用しても良い。

return render(request, self.template\_name, context

##### その他フィールド

context\_object\_name ListView、EditViewなどで使用するモデルクラス。

##### form\_valid

ここでバリデーションをする事もできる事はできる。

参考：フレームワーク情報 - forms.py - バリデーション - views.pyでのバリデーション

##### get\_form

htmlフォームの項目の編集を行う為に、formオブジェクトを取得する

class BookCreateView(CreateView): # Viewは何でもよい

def get\_form(self, form\_class=None):

form = super().get\_form(form\_class)

form.fields[‘shop’].queryset = Shop.objects.all() # ドロップダウンの時の候補？

**return form** # **return する**のを忘れないように

class Meta: # 参考：ウィジェットはこのように変更もできる

widgets = {

‘limit\_date’: forms.DateInput(attrs={"type": "date"}) # ウィジェットだけ変更する場合（Metaクラス）

}

# Metaクラスではできないが？2022-05-10

補足：django.forms.Formクラスのformclassメンバに値が格納されていない場合はsuper().get\_formメソッドが利用できない。利用できるのはCreateViewなどのみ（？らしい2022-06-08）

関連：get\_form\_kwargs

##### get\_initial

ModelForm系のViewでのみ有効。特にCreateViewで初期値を与える時に使用する。

def get\_initial(self):

initial = super().get\_initial()

initial['user'] = test.user #testモデルのuserを初期値としてセット

return initial

あくまで**初期値**である事に注意。htmlフォームのhidden的には扱われない為、バリデーションの時の既定値などを管理したい場合はform\_validメソッドの方が適切。

##### get\_form\_kwargs

Formクラスへ引数を渡せるらしいが？

##### get\_queryset

注意点

・getメソッドをオーバーライドする場合**get\_querysetは無効化**されてしまう！

・postメソッドでは**明示的に呼び出す必要**がある。（後述）

・template側で受け取る変数名はobject\_listとなる。（変更可能；context\_object\_name）

class BookListView(View):

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context[‘object\_list’] = Member.objects.all()[:3]

return context

以下のように、自身（self）のquerysetに格納する方法もある

def get\_queryset(self):

keyword = self.request.GET.get("keyword")

if keyword:

self.queryset = Book.objects.filter(name\_\_contains=keyword).all()

else:

self.queryset = Book.objects.all()

return super().get\_queryset()

メモ：親クラスのget\_querysetを呼び出す例があったが、意味ある？？

def get\_queryset(self, \*\*kwargs):

queryset = super().get\_queryset(\*\*kwargs)

補足：

##### get\_context\_data

templateに渡すdict型の変数、contextをカスタマイズする際に使用する。

また、ListViewでget\_querysetを用いる際にも使う。

[ views.py ]

class BookListView(View):

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs) # contextを使う場合

request = self.request # requestの取得

return context

注意点

以下のようにgetやpostメソッドと同時に使う事もできるが、メリットが少ない。

[ views.py ]

class BookListView(View):

template\_name=‘gyoumu/dairy-report/daily\_list.html’

def get\_context\_data(self, \*\*kwargs: Any) -> Dict[str, Any]:

context = super().get\_context\_data(\*\*kwargs) # 補足：get\_context\_data(kwargs)ではダメ

#context = { # 補足：contextを初期化してはいけない！！

# ‘form’ : BookForm (),

# }

context[‘form’] = BookForm() # 補足：必ずdict式のアクセス方法にする事！

def post(self, request, \*args, \*\*kwargs):

self.object\_list = self.get\_queryset() # postの場合get\_querysetを**明示的に呼び出す**

context = self.get\_context\_data(\*\*kwargs) # また、get\_context\_dataより先に呼ぶ必要がある

return render(request, self.template\_name, context )

補足

<https://docs.djangoproject.com/ja/4.0/ref/class-based-views/generic-display/>

##### DetailView

class BookDetailView(DetailView):

model = Book

<https://docs.djangoproject.com/en/4.2/ref/class-based-views/generic-display/#detailview>

##### CreateView

概要

シンプルなレコードの新規作成処理に利用できる定義済View。

但し、以下の場合には向かない。

・Formクラスのcleanメソッドによるカスタムバリデーションが必要な場合

⇒素直にform\_class=MyFormとしてFormクラス側で処理した方が早い

・postメソッドのオーバーライドが必要な場合

form\_validメソッドやget\_success\_urlメソッドを書くよりpostメソッドで処理した方が工数が短く済む場合がある

以上のような特徴がある為、CreateViewもUpdateViewも**意外と出番が少ない**。

個人的には、まず手軽に作れるCreateViewに飛びついてしまうが、最終的にはカスタマイズが必要な事が判明し、「これなら最初からTemplateView+ModelFormでやっておけばよかった…」という経験が多い。

⇒「ほとんど出番がない」「ちょっとでも迷ったら使わない」方が良い。

基本

[ urls.py ]

urlpatterns = [

path(‘shop/create/<int:pk>‘, views.BookCreateView.as\_view()), name=‘book\_create’),

]

注意点

・views.py側でgetメソッドのオーバーライドをしない限りテーブルレコードのidを示す変数名はpkである必要がある。

[ views.py ]

from django.views.generic import CreateView

from .models import Book

from .forms import BookForm

class BookCreateView(CreateView):

model = Book

template\_name = 'myapp/book/book\_create.html'

fields = ['title', 'price'] # fields= '\_\_all\_\_' ともできるが非推奨

form\_class = BookForm # 任意；使用するフォームを指定。

def get\_success\_url(self): # 必須。form\_valid=Trueとなった際の移動先

return reverse("myapp:shop\_top ")

[テンプレート（.htmlファイル）]

<form calss="form" method="poset">

{% csrf\_token %}

{{ form.non\_field\_errors }}

{{ form }}

</form>

関連するフィールド

form\_invalidメソッド form.is\_valid()がFalseの時の処理を記述する

form\_validメソッド form.is\_valid()がTrueの時の処理を記述する。

注意：cleanメソッドによるバリデーションのカスタマイズはform\_classに指定したFormクラスで行う以外方法は無い。

また、下記のようにform\_invalidメソッドでmessagesをaddしないとエラーメッセージが表示されない？

def form\_invalid(self, form):

messages.add\_message(self.request, messages.WARNING, form.errors)

return super().form\_invalid(form)

get\_formメソッド formに簡単なカスタマイズを行う。

def get\_form(self, form\_class=None):

form = super().get\_form(form\_class)

form.fields['publish\_date'].widget=forms.DateInput(attrs={'type': 'date'})

return form

get\_initialメソッド formに初期値を与える

def get\_initial(self):

initial = super().get\_initial()

book = get\_object\_or\_404(Book, pk=self.kwargs['isbn']) # 主キーでBookレコードをとりだす

initial['title'] = book.titile

（公式）<https://docs.djangoproject.com/ja/4.0/ref/class-based-views/generic-editing/#django.views.generic.edit.CreateView>

create（SQLのINSERT）処理をカスタマイズしたい場合

例

・fieldsには表示したくないが、required=Trueとなっているフィールド（html的にはhiddenな項目）が存在する場合。

方法１：

form\_validメソッドをオーバーライド

def form\_valid(self, form):

book = form.save(commit=False)

# ここで処理を行う

book.save()

return redirect(‘myapp:shop\_top’)

方法２

form\_classフィールドでformクラスを指定し、formクラス側のcleanメソッドで処理を行う。

（補足）forms.TextInputを用いて外部テーブルのprimary keyを編集する場合、CreateViewの基底クラス？でバリデーション処理が自動的に実行される。

class AuthorEditForm(forms.ModelForm):

class Meta:

model = Author

fields = [‘name’,’book’’] # bookが外部キーフィールドだとする。

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields[‘book’].widget=forms.TextInput() # ドロップダウンでなく、テキストボックスで編集したい

⇒ コードの記述としてはこれだけで、バリデーションまでやってくれる

##### UpdateView

[ urls.py ]

path('book\_update/<int:pk>/update',views.BookUpdateView.as\_view()), name='book\_update'),

注意点：idの変数名は**「pk」である必要がある**

class BookUpdateView(UpdateView):

template\_name = 'myapp/book\_update.html'

model = Book

fields = ['title', 'price'] # fields= '\_\_all\_\_' ともできるが非推奨

def get\_success\_url(self):

return reverse('myapp:shop\_top')

補足

・form\_classを指定する場合はget\_querysetメソッドをオーバーライドする必要がある（らしい）

実装しないとdjango.core.exceptions.ImproperlyConfigured:がraiseする。

実装例

def get\_queryset(self):

pk = self.kwargs.get('pk')

return Supplier.objects.filter(pk=pk)

・他のviewで「pk」以外のid変数名を使っている場合の例。（通常kwargsは前のViewからそのまま引き継がれる為、触る必要は少ない。ただ、UpdateViewではidを示すurl引数が「pk」に固定されているのが厄介）

def get\_success\_url(self):

return reverse('myapp:book\_detail', kwargs={'ISBN' : self.kwargs['pk']})

テンプレート（.htmlファイル）

CreateViewとほぼ同じ。箇条書きにすると：

・モデルの変数名は「object」。又はすべて小文字のクラス名で、上の例だと「book」でも可。

（補足）ListViewの「object\_list」とは異なり名前の変更は不可。

・フォームの変数名は「form」

フィールド

template\_name\_suffix

（公式）<https://docs.djangoproject.com/ja/4.0/ref/class-based-views/generic-editing/#django.views.generic.edit.UpdateView>

##### DeleteView

削除画面を作成。（個人的にはお勧めしない）

理由：削除の確認の為だけに画面移行が発生する為。JavaScriptによるポップアップウィンドウの方が良い。

path(‘delete/<int:pk>‘, views.BookDeleteView.as\_view(), name=‘book\_delete’),

[ views.py ]

class BookDeleteView(DeleteView):

template\_name = ‘book/delete.html’

model = Book

success\_url = reverse\_lazy(‘book:book\_list’)

[ template/book/delete.html ]

{% block content%}

<h2>本 削除</h2>

<form method="post">

{% csrf\_token %}

<p>{{ object.name }} を削除してよろしいですか</p>

<input type="submit" value="delete">

</form>

{% endblock %}

##### FormView

例として、CSVファイルを選択するビューの場合

ディレクトリ構成（抜粋）

myproject

├ myapp

│ ├ urls.py

│ ├ views.py

│ ├ forms.py

├ templates

│ ├ myapp

│ │ ├ csv.htm l

[ myapp/urls.py ]

from django.urls import path

from . import views

app\_name="myapp"

urlpatterns = [

path(‘csv/’, views. CsvUploadView.as\_view(), name=‘account\_csv’),

path(‘csv\_read/’, views. CsvReadView.as\_view(), name=‘account\_csv\_read’),

]

[ forms.py ]

class CsvUploadForm(forms.Form):

csvfile = forms.FileField()

def clean\_csvfile(self): # 個別にcleanを記述する場合（任意）

if not form.is\_valid(): # is\_validメソッドを呼び出さないとエラーになる

pass

csv\_file = self.cleaned\_data[‘csvfile’]

if not csv\_file.name.endswith(‘.csv’):

raise forms.ValidationError(‘Error: extension must be .csv’)

return csv\_file

次ページに続く

[ views.py ]

from django.views.generic import FormView

from django.urls import reverse\_lazy

from .forms import CsvUploadForm

class CsvUploadView(FormView):

form\_class = CsvUploadForm

template\_name = ‘myapp/csv.html’ # このviewをレンダリングするテンプレート

success\_url= reverse\_lazy(‘myapp:csv\_read’) # form\_validの時の移動先URL

def get\_form\_kwargs (self):

kwargs = super().get\_form\_kwargs()

return context

def get\_context\_data(self, \*\*kwargs): # contextの処理が必要な場合

context = super().get\_context\_data(\*\*kwargs)

return context

def form\_valid(self, form): # バリデーション成功時　失敗時はform\_invalid

csvfile = io.TextIOWrapper(form.cleaned\_data[‘csvfile’], encoding=‘shift-jis’)

reader = csv.reader(csvfile)

context={ # 変数contextを利用したい場合（通常不要）

‘reader’ : reader, # ⇒ 通常、ここでDBの更新などを行う

‘form’ : form,

}

return super().form\_valid(form) # 通常はこちらを用いる

#return render(self.request, myapp/csv\_read.html’, context) # 変数contextなどを利用したい場合

[ templates/csv.html ]

<form class="form" method="post" enctype="multipart/form-data">

{% csrf\_token %}

{% form %}

<input class="btn btn-primary" type="submit" value="選択">

</form>

memo

redirect ではcontext引数が引き継がれない（クエリ文字列はOK）

##### ListView

ListViewは基本的にはModel（≒データベースレコード）のリストを表示する為のView。

get\_querysetという特殊なgetメソッドをオーバーライドしたり、他のviewとは少し異なる記述もできる。

基本

[ views.py ]

class BookListView(ListView):

model = Book # 対象のモデル

template\_name = ‘shop/book\_list.html’ # レンダリングする.htmlファイル

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

return context

def get\_queryset(self, \*\*kwargs): # 親クラスでは (self) と定義されている

return Book.objects.all()

[ myapp/booklist.html ]

{% for obj in object\_list %}

{{ obj.title }}

...

{% endfor %}

関連するフィールド

context\_object\_name templateで扱う際の変数名。def.= ‘object\_list’

paginate\_by コンテンツの表示上限数（後述）

ordering 並び替えに使うフィールド。複数指定可能。get\_query\_setでないと意味がない？

フォームのカスタマイズがしたい場合

方法１

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context[‘form’] = BookForm() # このように、必ずdict式のアクセス方法にする事！

return context

次ページに続く

～ListView　つづき　filter+paginate\_by

paginate\_byを用いたページネーションの基本

（公式）<https://docs.djangoproject.com/ja/4.0/topics/pagination/>

[ views.py ]

class MyListView(generic.ListView):

template\_name = "myapp/list.html"

model = Book

paginate\_by = 20

page\_kwarg = ‘page\_obj’ # テンプレート(.html)で使うページ変数。デフォルト＝’page\_obj’

def get(self, request, \*args, \*\*kwargs): # 通常はget\_context\_dataでやるのかもしれない

queryset = Project.objects.all()

paginator = Paginator(queryset, self.paginate\_by)

page\_number = request.GET.get(‘page’)

try:

page\_obj = paginator.page(page\_number)

except PageNotAnInteger:

page\_obj = paginator.page(1)

except EmptyPage:

page\_obj = paginator.page(paginator.num\_pages)

context ={

self.page\_kwarg : page\_obj,

‘object\_list’ : ‘project\_list’: page\_obj.object\_list, # 2022-05-25現在使い方が良く分からない

}

return render(request, self.template\_name, context)

[ myapp/list.html ]

{% for book in page\_obj %}

{{ book.title }}

{{ book.author.author\_name }}

{% endfor %}

{% if page\_obj.has\_previous %} <!-- 前ページがある場合 -->

<a href="?page={{ page\_obj.previous\_page\_number }}">前へ</a> <!-- 前へ -->

{% endif %}

{% if page\_obj.has\_next %} <!-- 次ページがある場合 -->

<a href="?page={{ page\_obj.next\_page\_number }}">次へ</a> <!-- 次へ -->

{% endif %}

テンプレート（html）側で使用可能な変数

|  |  |  |
| --- | --- | --- |
| 変数名 | 型 | 説明 |
| page\_obj.previous\_page\_number | int | 前のページ番号 |
| page\_obj.next\_page\_number | int | 次のページ番号 |
| page\_obj.has\_previous | bool | 前のページがあるか |
| page\_obj.has\_next | bool | 次のページがあるか |
| page\_obj.paginator.page\_range | List[int] | 表示可能なページ番号一覧 |

次ページに続く

～ListView　filter+paginate\_by　つづき

補足

検索ワードを含むページの場合固定文字列以外で対応する事ができないのが課題

<!-- この入力内容でviews.py側で絞り込みを行う。実際にはformを使う（） -->

<input type="text" name="author" id="id\_author"></input>

<!-- この時点ではinputタグの内容を取得する事ができない。-->

<a href="?author=‘test’&page={{ page\_obj.next\_page\_number }}">次へ</a>

⇒Java Scriptで対応できないか？

filterによるレコードの絞り込み＋ページネーション。これが色々と厄介。

方法１　input＋form submit

<form class="form">

<button type="submit" class="col-10 btn btn-secondary">絞り込み</button>

</form>

<!-- 必ずしもformタグ内になくても良いらしい -->

<labe>ページ</labe>

<input type="number" name="page">

（参考）<https://thinkami.hatenablog.com/entry/2016/03/17/003140>

（参考）<https://tamapoco.com/archives/7155>

##### TemplateView

class IndexView(TemplateView):

template\_name = ‘index.html’

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data() # get\_context\_data(kwargs) としてはいけない？

return context

これがTemplateViewの正しい使い方らしい。長い間勘違いしていた。2022-03-23

##### RedirectView

基本

class ShopRedirectView(RedirectView):

pattern\_name = ‘myapp:shop\_top’ # リダイレクト先

def get\_redirect\_url(self, \*args, \*\*kwargs): # このメソッドは必要に応じて任意

record = get\_object\_or\_404(MonthlyCharge, pk=kwargs[‘id’])

return super().get\_redirect\_url(\*args, \*\*kwargs)

（公式）<https://docs.djangoproject.com/ja/4.0/ref/class-based-views/base/#redirectview>

kwargsを引き継ぎたくない場合はkwargsなしでget\_redirect\_url を呼び出す

return super().get\_redirect\_url(\*args)

（補足）query\_string=Falseで行けそうな気がするが、うまく行かなかった2022-06-01

url

リダイレクト先のurlを直接指定する

url = reverse\_lazy(‘gyoumu:data-manage\_charge\_list’)

#### PostgreSQL固有の集計関数

ArrayAgg

#### 例外

##### 資料

（公式）<https://docs.djangoproject.com/ja/4.0/_modules/django/core/exceptions/>

##### django.forms

ValidationError バリデーション（clean）中に問題が発生した。

### その他

#### csrf\_token

CSRFの検証を行う。**POST**リクエストにCSRF対策のトークン文字列を加える。

基本的な使い方：

<form action="auth" method="post">

{% csrf\_token %} <!-- このようにformタグの直後に入れる -->

#### aldjemy

Django ORMは実はデータベース操作に適していない。（複雑なクエリは重くなる）

できる限り、人気もあり優れたORMであるSQLAlchemyを使った方が良い。

aldjemyはDjangoでSQLAlchemyが使えるようになるpipパッケージ。

インストール

python -m pip install aldjemy # greenlet, SQLAlchemy などが必要

settings.py

INSTALLED\_APPS = [

…

‘aldjemy’, # 追加

]

後は「モデル名．Sa．query（）」でアクセス

python（views.pyなど）

from .models import User

user\_list = User.objects.all() # 従来のDjangoの書き方

# ↓

user\_list = User.sa.query().all() # Aldjemyを利用した方法

参考：<https://scrapbox.io/PythonOsaka/DjangoでSQLAlchemyを使ってみよう>

postgresSQLのjsonFieldを扱うパッケージがある？？

import django

from django.contrib.postgres.fields import ArrayField, DateRangeField, JSONField

from django.db import models

#### Django+FastAPI

https://qiita.com/kigawas/items/80e48ccce98a35f65fff

→結局2つのhtmlサーバーの起動が必要らしい。

#### html tableで直接編集する方法

例えばテーブルの最終行を追加行とする場合。

[ .html（テンプレート）]

<form clss="form" method="POST">

{% csrf\_token %}

<table>

<tr><!-- 通常のテーブル行表示 --></tr>

</table>

</form>

補足：

table、tbody、trなどにformを入れるのはやめた方が良いらしい。

情報元：<https://teratail.com/questions/54864>

#### 用語

バックエンド

ミドルウェアと何が違う？

### トラブルシューティング

#### 本番環境でstaticが読み込めない

前提条件

・python3.8、Django2.1.15

・アプリサーバーはwinswでwaitressをサービス化

その際の実行コマンドは以下と同等：

waitress-serve --port=8080 config.wsgi:application

発生現象

・.jsファイルが読み込まれない（404）

但し、cssなどはすべてCDN（web）からの読み込みなので、実質statisファイル＝.jsファイル。

つまり、staticファイルはすべて読めない状態

補足情報

・.jsファイルなどはキャッシュで一定期間持ち続けるので、デバッグの際は逐一Webブラウザのキャッシュを削除しながら行う必要がある。

メモ

・settings.pyのDEBUG=Trueにしたら動いた。

解決方法

from django.urls import re\_path

urlpatterns = [

… # 中略

re\_path(r'^static/(?P<path>.\*)$', serve,{'document\_root': settings.STATIC\_ROOT}), # この行を追加

]

Django公式の方法だが、これではうまく動かなかった。バージョンが古すぎる？

urlpatterns = [

… # 中略

] + static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

#### Models aren’t loaded yet.

原因１：SECRET\_KEYが無い

for app\_config in self.app\_configs.values():

app\_config.import\_models()

#### django.db.migrations.exceptions.InconsistentMigrationHistory:

Migration admin.0001\_initial is applied before

複数のアプリのモデルをマイグレーションしている時に発生する事がある。

settings.pyからアプリをコメントアウトし、１つずつマイグレーションすると修復できる。

（情報元）<https://hodalog.com/repair-migration-error-by-social-auth-app-django/>

#### Formのコンストラクタで’initial’が無いと言われる

got an unexpected keyword argument ‘initial

‘

class DailyReportForm(forms.Form):

id\_number =forms.CharField(label=‘社員番号’,)

def \_\_init\_\_(self):

super().\_\_init\_\_()

# ↓このように記述する

def \_\_init\_\_(self, \*args, \*\*kwargs): # ‘initial’ を受け取れるようにkwargs

super().\_\_init\_\_(\*args, \*\*kwargs) # 基底クラスにkwargsを渡す

#### logディレクトリが無いと言われる

loggingパッケージを使用する際のハンドラ設定でclass=logging.FileHandlerにしたさい、

unable to configure handler ‘file’

エラーメッセージ（抜粋）

File "C:\python\seigi\_app\env\lib\site-packages\django\\_\_init\_\_.py", line 19, in setup

configure\_logging(settings.LOGGING\_CONFIG, settings.LOGGING)

File "C:\python\seigi\_app\env\lib\site-packages\django\utils\log.py", line 76, in configure\_logging

logging\_config\_func(logging\_settings)

File "C:\Users\SGGSUSER\AppData\Local\Programs\Python\Python38\lib\logging\config.py", line 808, in dictConfig

dictConfigClass(config).configure()

File "C:\Users\SGGSUSER\AppData\Local\Programs\Python\Python38\lib\logging\config.py", line 570, in configure

raise ValueError(‘Unable to configure handler ‘

ValueError: Unable to configure handler ‘file’

2022-05-27：途中までenv内のpythonを参照しているのに、途中からユーザー（SGGSUSER）のpythonを参照している様子。

Webの情報

・「log」ディレクトリが存在しない ⇒ある

・「log」ディレクトリはmanage.pyと同レベルに配置する ⇒確認済

・ディレクトリの名前は「log」でないといけない ⇒確認済

・ファイルやディレクトリの所有者の問題。⇒ Linuxで動かす場合の話？

（参考）<https://localcoder.org/django-valueerror-unable-to-configure-handler-errno-2>

メモ

・config.settings.pyでLOGGING\_CONFIG ={} と指定したら動くようになった。(2022-05-27)

⇒ 但しログも記録されなくなった。恐らく全ての設定項目がデフォルトになっている状態。

・Lib\site-packages\django\utils\log.pyのconfigure\_loggingメソッド内でlogging.config.dictConfig

LOGGING\_CONFIGの設定方法

import logging.config

logging.config.dictConfig(DICT\_STR) # ここでdict形式の文字列を設定

もしくはLOGGING\_CONFIG を None に設定して自分で設定する

（公式）<https://docs.djangoproject.com/ja/4.0/topics/logging/#configuring-logging-1>

Lib\site-packages\django\\_\_init\_\_.pyを編集すると良い？

（情報元）<https://www.programcreek.com/python/example/115639/django.conf.settings.LOGGING_CONFIG>

#### SQL Serverエラー：文字列またはバイナリデータが切り捨てられます

パスワードはハッシュ化される為、varcharの文字数（max\_length）が多めに必要。

なお、AbstractBaseUserクラスのset\_ passwordメソッドが使われる

同メソッドでdjango.contrib.auth.hashers.make\_passwordメソッドが使用されている。

def set\_password(self, raw\_password):

self.password = make\_password(raw\_password)

self.\_password = raw\_password

#### django.core.exceptions.FieldError: Cannot resolve keyword

データベースに存在しないフィールド名を指定している。

・primary keyのフィールドを途中で変更した際に発生する事がある。

例

class Author

author\_id = models.CharField(verbose\_name='著者ID', primary\_key=True)

# 変更前のソースコード

author\_list = Book.objects.filter(author \_\_id = 1)

# ↓ このように変更しないと、上記エラーが発生する

author\_list = Book.objects.filter(author \_\_author\_id = 1)

・migration忘れの可能性もある

#### manage.py dumpdataで作成されるjsonの文字コードがutf-8にならない

まず前提として、Djangoのdumpdataコマンドの標準文字コードはutf-8となっている。

その為、標準シェルの文字コード設定を疑った方が良い

Powershellの場合

# 確認方法

$OutputEncoding.EncodingName

# dumpdataコマンドの結果を格納する変数

$data = python manage.py dumpdata --exclude myapp.MyAppUser --exclude auth.permission --exclude contenttypes

# bom無しutf-8のオブジェクト

$utf8\_bomless = New-Object System.Text.UTF8Encoding $false

# dump\_main.jsonという名前で書き込む

[System.IO.File]::WriteAllLines("dump\_main.json", $data, $utf8\_bomless)

#### 本番環境でjsファイルが404

・DEBUG=Falseである必要がある →効果なし

・django.contrib.staticfiles が設定ファイルの INSTALLED\_APPSに含まれている →問題無し

・myapp/static ディレクトリを作成する →効果なし

（公式）<https://docs.djangoproject.com/ja/4.0/howto/static-files/#serving-static-files-during-development>

## FastAPI

### 概要

#### 情報

Flaskによく似ている。Starletteのラッパー。

（Starlette：軽量のASGIフレームワーク/ツールキット）

Flaskとの違い

・HTTPメソッドごとに個別のデコレータが存在する

・自動でドキュメンテーションを作成する（Swagger UI）　　例 localhost/docs

補足：OpenAPIを使っているらしい

・GraphQLの構築も容易？

#### 関連パッケージ

pip

|  |  |
| --- | --- |
| パッケージ名 | 内容 |
| fastapi-restful | ？ |
| fasetapi-utils | ？ |
|  |  |
| uvicorn | Webサーバー |
| jinja2 | Htmlテンプレート　　補足：Djangoの内部でも使われている |
| aiofiles | 静的ファイル(static)を利用する |
|  |  |
| python-multipart | fastapi.Formを使う時必要 |
|  |  |
| pydantic | 実行時の型ヒントやデータバリデーション。 |
| starlette | ASGI |
| SQLAlchemy | データベースへの接続 |
| psycopg2 | PostgreSQLを使う場合 |
| pymysql | mysql(MariaDB) を使う場合 |
| pyodbc | SQLServerを使う場合 |

### チュートリアル

#### インストール

python3 -m pip install fastapi # install fastapi uvicorn と続けて打っても良い

python3 -m pip install uvicorn # Webサーバー　sqlalchemy？

補足：fastapiを入れるとstarlette, pydantic!, typing-extensions, anyioが一緒にインストールされるらしい

参考：https://rightcode.co.jp/blog/information-technology/fastapi-tutorial-todo-apps-environment

公式情報：<https://fastapi.tiangolo.com/ja/tutorial/>

##### Dockerを使う場合

ディレクトリ構成

/

├ app/

│ ├ main.py

└ Dockerfile

[ Dockerfile ]

FROM tiangolo/uvicorn-gunicorn-fastapi:python3.7

COPY ./app /app

terminal

docker build –t myimage

docker run –d --name mycontainer –p 80:80 myimage

参考：<https://fastapi.tiangolo.com/ja/deployment/docker/>

こんなパターンもあるらしい

ディレクトリ構成

├ app/

│ ├ app/

│ │ ├ main.py

│└ Dockerfile

├ docker-compose.yml

#### 最小限アプリ

ディレクトリ構成

├ env/ python -m venv envで仮想環境を作成

├ main.py アプリのエントリポイント

[ main.py ]

from fastapi import FastAPI

my\_app = FastAPI() # インスタンス生成

@my\_app.get("/") # ルーティング情報とhttpメソッドを指定するデコレータ

async def root(): # メソッド名はわりと何でも良い

return {"message": "Hello World"} # jsonを返す

Terminal

uvicorn main:my\_app --reload

補足：わかりやすくする為に変数名をmy\_appとしたが、Web上ではappとしている例が多い。

補足：uvicorn modlue\_name:instance\_name となっているらしい。

補足：FastAPIモジュールはStarletteを継承している。

#### データベース

公式情報（英語）：https://fastapi.tiangolo.com/ja/tutorial/sql-databases/

インストール

python3 -m pip install sqlalchemy

python3 -m pip install psycopg2 # postgreSQLの場合

python3 -m pip install pyodbc # SQLServerの場合

ディレクトリ構成

├ env/

├ models/【追加】 modelsをパッケージにするか、１つのファイルにするかはアプリの規模に依る

├ database.py 【追加】

├ main.py

database.py

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

Base = declarative\_base()

engine = create\_engine(

"postgresql+psycopg2://user:password@127.0.0.1/db\_name", # postgresql

)

engine = create\_engine(

"mysql+pymysql://user:password@127.0.0.1/db\_name?charset=utf8mb4" # mysql/MariaDB

)

Session = sessionmaker(bind=engine)

session = Session()

models.py

from .database import Base

from sqlalchemy import Column, String, DateTime, ForeignKey

from sqlalchemy.sql.functions import current\_timestamp

from sqlalchemy.dialects.mysql import INTEGER, BOOLEAN

class User (Base):

\_\_tablename\_\_ = "user" # テーブル名を指定

user\_id = Column(Integer, primary\_key=True)

name = Column(String(256))

age = Column(Integer)

def full\_name(self): # フルネームを返すメソッド

return "{self.first\_name} {self.last\_name}"

#### テンプレート

補足：FastAPIでWebAPIを作成したい場合は読み飛ばして良い。Webアプリの作成が対象。

インストール

python3 -m pip install jinja2 # htmlテンプレートの為のパッケージをインストール

ディレクトリ構成

├ env/

├ templates

│ ├ index.html

│ ├ layout.html

├ main.py

[ main.py ]

from fastapi.templating import Jinja2Templates # 【追記】 starlette.templating でも可

templates = Jinja2Templates(directory="templates") # 【追記】

jinja\_env = templates.env # 【追記】

def index(request: Request):

return templates.TemplateResponse(

‘index.html’,

{‘request’: request})

def login(request: Request):

return templates.TemplateResponse(

‘login.html’,

{‘request’: request, ‘username’: ‘admin’}) # テンプレートに引数を渡す場合

#### Webフォーム

##### ビュー

edit\_user.html

<form method="post">

<input name=”age”> <!-- name要素でUserクラスのフィールドを指定 -->

{% for gp in groups %}

<option value="{{gp[‘id’]}}">{{gp[‘name’]}}</option> <!-- 他テーブルへの外部キーの場合 -->

{% endfor %}

</select>

</form>

補足：htmlは頑張って書くしか無いらしい → その方が却って編集しやすいかも？

補足：csrf対策？

htmlのフォームからの値の受け取り方は以下の２通り（らしい）

controllers.py(またはrouter.py；名称やベストプラクティス的な選択）

from fastapi import Form # pip でpython-multipartのインストールが必要（デフォルト）

@router.post("/user/add/")

async def user\_add(name: str = Form(...)): # 引数にinputタグのnameと同名の引数を用意する

print(name)

@router.post("/user/add/")

async def account\_add(request: Request):

form = await request.form()

user\_name = form[‘name’] # inputタグのnameと同名のキーでdict型にアクセス

print(name)

##### バリデーション

controllers.py(またはrouter.py；名称やベストプラクティス的な選択）

import re # 正規表現を扱う

pattern = re.compile(r’\w{4,20}’) # 任意の4~20の英数字を示す正規表現

@router.post("/user/add/")

async def account\_add(request: Request):

form = await request.form()

user\_name = form[‘name’] # inputタグのnameと同名のキーでdict型にアクセス

error = [] # エラーを格納するリスト型

if pattern.match(username) is None:

error.append(‘ユーザ名は4~20文字の半角英数字にしてください。’)

if error:

return templates.TemplateResponse(‘register.html’,

{‘request’: request, ‘username’: name, ‘error’: error})

##### レコードの追加

レコードの追加は簡単。上の「データベース」で作成したsessionを用いる。

from database import session

async def account\_add(request: Request):

# 中略：上のバリデーションが終了したところまで

user = User(username, age)

session.add(user)

session.commit()

session.close()

return templates.TemplateResponse(‘complete.html’,

{‘request’: request,

‘username’: username})

#### ログイン

username = credentials.username

password = hashlib.md5(credentials.password.encode()).hexdigest() # 暗号化

user = db.session.query(User).filter(User.username == username).first()

task = db.session.query(Task).filter(Task.user\_id == user.id).all() if user is not None else []

db.session.close()

tmp\_user = db.session.query(User).filter(User.username == username).first()

参考」https://rightcode.co.jp/blog/information-technology/fastapi-tutorial-todo-apps-authentication-user-registration

#### プロジェクト

##### ディレクトリ構成

├ env

├ sql\_app

│ ├ \_\_init\_\_.py

│ ├ crud.py

│ ├ database.py

│ ├ main.py

│ ├ models.py

│ └ schemas.py

##### models.py

##### リバースプロキシ

公式情報（uvicorn；英語）https://www.uvicorn.org/deployment/

### 詳細

#### 変数の利用

app.getデコレータ（など）で {} を使うことで変数が使用できる。　例：{arg\_name}

補足：flaskでは<>を用いる

@app.get("/") # ルーティング情報とhttpメソッドを指定するデコレータ

async def root():

return {"message": "Hello World"} # jsonを返す

@app.get("/items/0") # オーバーロード対応；固定引数

async def read\_item(): # 上から順に評価されるため、固定パラメータがある場合は先に記述する

return {"item\_id": "current item" }

@app.get("/items/{item\_id}") # オーバーロードに対応する場合；引数なし

async def read\_item(item\_id: int, name:str): # :int はpythonの「型ヒント」。型の指定では無いので注意

name = "hoge" # パスパラメータで指定していない引数はクエリ文字で指定する。

return {"item\_id": item\_id} # 引数に対応した動的なjsonを返す

webブラウザ

192.168.1.1:8000/items/1

> item\_id: "1"

192.168.1.1:8000/items/foo

> エラーメッセージがjson的に表示される（文字列型の引数「foo」を指定した為）

リクエストボディの文字列からdict ⇔ Classインスタンス　するにはPydanticが用いられる。

#### 限定的な引数

Enumを用いることで、パラメータに制約をかけられる。

from enum import Enum

from fastapi import FastAPI

class ModelName(str, Enum): # strとEnumを継承？？

alexnet = "alexnet"

resnet = "resnet"

lenet = "lenet"

app = FastAPI()

@app.get("/models/{model\_name}")

async def get\_model(model\_name: ModelName):

if model\_name == ModelName.alexnet:

return {"model\_name": model\_name, "message": "Deep Learning FTW!"}

elif model\_name.value == "lenet":

return {"model\_name": model\_name, "message": "LeCNN all the images"}

else

return {"model\_name": model\_name, "message": "Have some residuals"}

#### ベストプラクティス（検討中）

##### ディレクトリ構成

your\_project

├── \_\_init\_\_.py

├── main.py

├── core

│ ├── models

│ │ ├── database.py

│ │ └── \_\_init\_\_.py

│ ├── schemas

│ │ ├── \_\_init\_\_.py

│ │ └── schema.py

│ └── settings.py

├── tests

│ ├── \_\_init\_\_.py

│ └── v1

│ ├── \_\_init\_\_.py

│ └── test\_v1.py

└── v1

├── api.py

├── endpoints

│ ├── endpoint.py

│ └── \_\_init\_\_.py

└── \_\_init\_\_.py

情報元：

<https://stackoverflow.com/questions/64943693/what-are-the-best-practices-for-structuring-a-fastapi-project>

コードジェネレーターが存在するらしい

https://github.com/tiangolo/full-stack-fastapi-postgresql

##### urlディスパッチャについて

urls.pyとcontrollers.pyを使う方法と、routers.pyでやる方法がある。

（いずれも厳密にはモジュール名は何でも良いが）

urls.py

app.add\_api\_route(‘/’, index)

app.add\_api\_route(‘/admin’, admin)

app.add\_api\_route(‘/register’, register, methods=[‘GET’, ‘POST’])

controllers.py

def register(request: Request):

if request.method == ‘GET’:

pass

if request.method == ‘POST’:

pass

#### パスパラメータの型

path

/files/{file\_path:path}

#### APIドキュメント

/docs a

/redoc a

/openapi.json すべてのAPIの説明を行う

### ライブラリレファレンス

#### jinja

※別ドキュメントpythonの

#### app

##### デコレータ

@app.post()

@app.put()

@app.delete()

@app.options()

@app.head()

@app.patch()

@app.trace()

##### add\_api\_route

構文 app.add\_api\_route(‘url\_string’, function\_name)

引数

function\_name url文字列に対応する関数名を記述

methods GET, POSTなどのhtmlメソッドの指定

include\_router

#### responses

##### FileResponse

構文：FileResponse(path, filename)

@app.get("/")

def index():

return FileResponse(path="path-to/file.txt", media\_type="text/plain")

##### JSONResponse

@app.get("/")

def index():

return JSONResponse(content={"mes": "hello"})

##### HTMLResponse

通常はjinjaのTemplateを返す

@app.get("/")

def index(request: Request):

return templates.TemplateResponse("page.html", {"request": request})

##### RedirectResponse

@app.get("/")

def index():

return RedirectResponse(url="http://example.com")

#### security

##### HTTPBasic

##### HTTPBasicCredentials

#### templating

##### Jinja2Templates

#### Query

クエリパラメータを管理する？

@app.get(‘/validation/{path}’)

async def validation(

string: str = Query(None, min\_length=2, max\_length=5, regex=r’[a-c]+.’),

integer: int = Query(..., gt=1, le=3), # 第一引数を…にすると必須パラメータになる

alias\_query: str = Query(‘default’, alias=‘alias-query’),

path: int = Path(10)):

細かい引数の入力規則を設ける事ができる（らしい）

情報元：<https://qiita.com/bee2/items/75d9c0d7ba20e7a4a0e9?utm_campaign=popular_items&utm_medium=feed&utm_source=popular_items>

#### Path

パスパラメータを管理する？

#### Body

リクエストボディを管理する？

#### BackgroundTasks

バックグラウンド処理（非同期処理）を扱う

from fastapi import BackgroundTasks, FastAPI

app = FastAPI()

def write\_notification(email: str, message=""):

with open("log.txt", mode="w") as email\_file:

content = f"notification for {email}: {message}"

email\_file.write(content)

@app.post("/send-notification/{email}")

async def send\_notification(email: str, background\_tasks: BackgroundTasks):

background\_tasks.add\_task(write\_notification, email, message="some notification")

return {"message": "Notification sent in the background"}

公式：<https://fastapi.tiangolo.com/tutorial/background-tasks/>

#### uvicorn

[ python ]

if \_\_name\_\_ == ‘\_\_main\_\_’:

uvicorn.run(app=app, port=8000)

補足情報

自身プロセスを監視して落ちたら再起動するといったような仕組みが無い（らしい）。

⇒ gunicornというパッケージがあるらしい。

WSGI アプリケーション用のPython HTTP サーバー

localhost以外からアクセスする時にはApacheやnginxでリバースプロキシが必要

nginxをリバースプロキシにして、localhostで動いているuvicornに転送する。

ルートパスの設定が必要？　uvicorn api:api --root-path /XXX

[https://qiita.com/uezo/items/847e1911ac486f5a89c4#](https://qiita.com/uezo/items/847e1911ac486f5a89c4#リバースプロキシの設定)リバースプロキシの設定

[ terminal ]

uvicorn main:app --reload --host 0.0.0.0 # ホストの指定方法

## Flask

### 概要

#### 情報

pythonを利用したWebアプリ作成のフレームワークで，小規模なものに向く。

その反面、ユーザーが独自に拡張したり、自身で追加したりする。

・urls.pyのようにルーティングを一括で管理する事はできない

・

#### 関連パッケージ

pip

|  |  |
| --- | --- |
| パッケージ名 | 内容 |
| make\_response |  |
|  |  |

### チュートリアル

#### インストール

mkdir flask # 事前準備：ディレクトリ作成

cd flask/ #

python3 -m venv env # 事前準備：仮想環境作成

source ./env/activate #

python3 -m pip install -U pip # 事前準備：pip のアップデート

python3 -m pip install flask # インストール

#### 最小限アプリ

##### コード

app.py

from flask import Flask

app = Flask(\_\_name\_\_) # Flaskのインスタンス

@app.route("/hello") # ルーティング情報を追加するデコレータ

def hello(): # 上記ルーティングに対応したメソッド。（メソッド名は何でも良い？）

return "Hello, World!" #html文字列としてHello, World!を返す

if \_\_name\_\_ == "\_\_main\_\_":

app.run('0.0.0.0') # listenアドレスを指定して起動

##### 実行

terminal:

python3 ./app.py # 実行

もしくは、.pyファイルの名前がapp.pyの時に限り，以下のようにもできる

terminal:

flask run --host=0.0.0.0 # 実行

→ ブラウザで ttp://192.168.1.0:5000/hello などとするとHello World!と表示される。

注意！https:// でアクセスすると接続できない。

参考：<https://flask.palletsprojects.com/en/latest/quickstart/#a-minimal-application>

#### 基本

##### 変数の利用

app.routeデコレータで<>を使うことで変数が使用できる。　例：<arg\_name>

但し、通常はインジェクション攻撃から保護する為，html文字列にする場合にエスケープ処理を施す。

from markupsafe import escape # エスケープ処理を提供するパッケージをimport

@app.route("/<name>") # ルーティングでnameという引数を取る場合

def hello(name): # pythonのメソッドにも同名の引数を加える

return escape("hello %s" % (name)) # エスケープ処理を施しつつ、html文字列に返還

@app.route("items/") # オーバーロードに対応する場合；引数なし

@app.route("items/<item\_id>") # 引数のある場合

def item(item\_id=None): # pythonのメソッドでもデフォルト値を指定する

return escape("user %s" % (item\_id)) # 引数の有無にも対応したhtml文字列を返す

##### デバッグモード

terminalで指定する場合

export FLASK\_ENV=development

flask run

pythonコードで指定する場合

if \_\_name\_\_ == "\_\_main\_\_":

app.run(‘0.0.0.0’, debug=True) # listenアドレスを指定して起動

#### プロジェクト

##### 概要

ディレクトリ構成：

~/myproject # mkdir でディレクトリを作成

├─ env # venv コマンドで作成される

│├ bin/ # 実際に使用するコマンド群が格納されている

│├ pyvenv.cfg # 設定ファイル？

├─ app #

│├ templates/ # HTMLファイルを格納する

││ ├ index.html # トップページ

│├ static/ #

│├ app.py #

├─ run.py # flaskコマンドで実行されるpythonスクリプト

│などなど

##### テンプレート（レンダリング）

from flask import render\_template, Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

msg = ‘Hello World!’

return render\_template(‘index.html’, message = msg)

templates/index.html

<!DOCUTYPE html>

{% autoescape true %}

<html>

<head lang="ja">

<title>top page</title>

</head>

<body>

{{ message }}

</body>

</html>

{% endautoescape %}

#### データベース

##### インストール

python3 -m pip install flask\_sqlalchemy # インストール

python3 -m pip install psycopg2-binary # postgreSQLの場合

python3 -m pip install # mariaDB(≒mySQL) の場合

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

app.config[‘SQLALCHEMY\_DATABSE\_URI’]

### 具体例

#### ファイルのダウンロード

send\_file()

send\_from\_directory()

make\_response()

#### レスポンス

ステータスコードを返す（共通）

@app.route('/404')

def get\_404():

return 'not found', 404

Responseオブジェクトを返す（これが基本らしい）

from flask import Flask, make\_response

@app.route('/object')

def get\_object():

make\_response('hello') # 引数には下で説明するもののいずれかを入れる（違いは何？）

文字列を返す

@app.route('/str')

def get\_str():

return 'hello'

jsonを返す

from flask import Flask, jsonify

@app.route('/json')

def get\_json():

return jsonify({'message': 'hello'})

return { 'message': 'hello' } # これでも良い？

ファイルを返す

from flask import Flask, send\_file

@app.route('/csv')

def get\_csv():

return send\_file(

file\_path,

download\_name='csv\_file.csv',

mimetype="text/csv",

as\_attachment=True,

)

### レファレンス

#### Flask

##### 概要

##### run

threaded

公式：<https://msiz07-flask-docs-ja.readthedocs.io/ja/latest/api.html?highlight=threaded#flask.Flask.run>

#### app

##### app.logger

##### app.run()

debug=True,

host=‘0.0.0.0’,

port=5000, 既定値は5000

threaded=True

##### @app.route()

引数の指定

@app.route("user/<name>") # URLから引数を取る時

def show\_user(name):

return escape("hello %s" % (name))

@app.route("post/<id>") # URL引数に型を指定する時　string, int, float, path, uuid が指定可能

def main(name):

return escape("post %i" % (id))

methods

from flask import Flask, request

@app.route("/change", methods=["POST"]) # HTTPメソッドのPOSTに対応させる

@app.route(‘/login’, methods=[‘GET,’POST’]) # 2つ以上のHTTPメソッドに対応させる

def login():

if request.method = ‘POST’:

return render\_template(‘login.html’)

else:

pass

templateのレンダリングの一例

@app.route("/")

def index():

return render\_template(‘index.html’, \*\*args) #

##### @app.errorhandler()

@app.errorhandler(500)

def internal\_server\_error.html(error):

return render\_template(‘internal\_server\_error.html’), 500

#### flask\_restful

##### Resource

##### Api

#### flask\_caching

##### 概要

Viewをはじめとした関数のキャッシュ化

公式：<https://flask-caching.readthedocs.io/en/latest/>

・内部ではredisを用いる為、redisサーバーが必要。

from flask import Flask

from flask\_caching import Cache

config = {

"CACHE\_TYPE": "RedisCache",

"CACHE\_REDIS\_HOST": "localhost",

"CACHE\_REDIS\_PORT": 6379,

}

app = Flask(\_\_name\_\_)

app.config.from\_mapping(config)

cache = Cache(app)

##### Cache

#### Flask-SQLAlchemy

##### インストール

pip install flask\_sqlalchemy

pip install psycopg2-binary

##### 使い方

from flask\_sqlalchemy import SQLAlchemy

app.config[‘SQLALCHEMY\_DATABASE\_URI’] = ‘postgresql://localhost/postgres’

app.config[‘SQLALCHEMY\_TRACK\_MODIFICATIONS’] = False

db = SQLAlchemy(app)

参考：https://swallow-incubate.com/archives/blog/20190909

#### Flask-Migrate

#### escape

インジェクション攻撃から保護するためにエスケープを提供する

#### jsonify

JSONのレスポンスを提供する

#### request

request.method HTTPメソッドの種類を保持する

request.files アップロードなど、ファイルを管理する

request.cookies クッキーを管理する。

def index():

username = request.cookies.get(‘username’)

#### url\_for

urlの逆引きを提供する？

url\_for(‘static’, path=‘/styles.css’)

from flask import url\_for

@app.route(‘/login’)

def login():

return ‘login’

with app.test\_request\_context():

print(url\_for(‘login’))

print(url\_for(‘login’, next=‘/’)) # 存在しない引数を指定した場合は，クエリ文字列として処理されるらしい？

#### Jinja

Html文書内で扱えるスクリプトの事。{% %} で囲う。

<!doctype html>

<title>Hello from Flask</title>

{% if name %}

<h1>Hello {{ name }}!</h1>

{% else %}

<h1>Hello, World!</h1>

{% endif %}

補足：Djangoにおける [DTL](#_DTL：テンプレートタグ)に相当するものと思われる。

##### Hamlish Jinja

HAML風の記法を提供する。

#### Terminalコマンド (flask)

構文

サブコマンド

run

terminal

flask run --host=0.0.0.0 # app.pyが実行される

## Responder

### 概要

MVC（Webアプリ）作成に強い

FastAPIと同じくStarlette （スターレット）ベース、Jinja2 を用いたテンプレート

GraphQL APIが簡単に書ける（らしい）

条件

Pythonのバージョンが3.6以上

### チュートリアル

#### インストール

pip install responder

#### 最小限アプリ

[ run.py ]

import responder

api = responder.API()

@api.route('/')

def hello\_world(req, resp):

resp.text = 'hello, world!'

if \_\_name\_\_ == '\_\_main\_\_':

api.run(port=8000)

[ terminal ]

python3 run.py

#### テンプレート

ディレクトリ構成

├ venv/

├ templates

│ ├ index.html

│ ├ layout.html

├ run.py

[ layout.html ]

<!DOCTYPE html>

<html lang="ja">

<head>

<meta charset="UTF-8">

<title>Responder Test</title>

</head>

<div class="container">

{% block content %} <!-- Jinja2を使っているのでDjangoのようにテンプレート読み込みが可能 -->

{% endblock %}

</div>

</body>

</html>

[ index.html]

{% block content %}

Hello, {{word}} <!-- Jinja2を使っているので変数も利用可能 -->

{% endblock %}

[run.py]

import responder

api = responder.API()

@api.route('/{word}')

def hello\_word(req, resp, word):

resp.content = api.template("index.html", word=word)

if name == 'main':

api.run()

クラス方式

@api.route('/{word}')

class Hello:

def on\_get(self, req, resp, word):

resp.content = api.template("index.html", word=word)

#### urlディスパッチャ

urlディスパッチャとcontrollersを用意し、MVCプロジェクト風にする

├ venv/

├ templates

│ ├ index.html

│ ├ layout.html

├ controllers.py 【追加】

├ urls.py　【追加】

├ run.py

[ controllers.py ]

import responder

api = responder.API()

class IndexController:

def on\_get(self, req, resp):

resp.html = api.template('index.html')

[ urls.py ]

from controllers import api

from controllers import IndexController

api.add\_route('/', IndexController)

[ run.py ]（修正）

from urls import api

if \_\_name\_\_ == '\_\_main\_\_':

api.run(port=8000)

実行

[ terminal ]

python3 run.py

補足：urls.pyを用いず、メソッドに直接デコレータを記述する事もできる

@api.route('/api/csv')

async def csv\_data(req, resp):

resp.text = f'Hello'

#### SQLAlchemy

##### インストール

インストール

python3 -m pip install sqlalchemy

python3 -m pip install psycopg2 # postgreSQLの場合

python3 -m pip install pyodbc # SQLServerの場合

##### 基本

ディレクトリ構成

├ venv/

├ templates

│ ├ index.html

│ ├ layout.html

├ models/【追加】 modelsをパッケージにするか、１つのファイルにするかはアプリの規模に依る

├ database.py 【追加】

├ controllers.py

├ urls.py

├ run.py

[ models.py ]

from .database import Base

from sqlalchemy import Column, String

class User (Base):

\_\_tablename\_\_ = "user" # テーブル名を指定

user\_id = Column(Integer, primary\_key=True)

name = Column(String(256))

age = Column(Integer)

def full\_name(self): # フルネームを返すメソッド

return "{self.first\_name} {self.last\_name}"

[ run.py]

@api.route('/db')

def hello\_users(req, resp):

users = session.query(Log) \

.filter(Log.line\_name=='CHボデー', cast(Log.date\_time, Date)=='2022-08-04') #\

resp.text = '%s' % logs.count()

if \_\_name\_\_ == '\_\_main\_\_':

api.run(port=8000)

次ページに続く

～SQLAlchemy　つづき

[ database.py ]

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

Base = declarative\_base()

engine = create\_engine(

"postgresql+psycopg2://user:password@127.0.0.1/db\_name", # postgresql

)

engine = create\_engine(

"mysql+pymysql://user:password@127.0.0.1/db\_name?charset=utf8mb4" # mysql/MariaDB

)

Session = sessionmaker(bind=engine)

session = Session()

##### テンプレートへのレンダリング

@api.route('/users')

[ controllers.py ]

import responder

api = responder.API()

class IndexController:

def on\_get(self, req, resp):

resp.html = api.template('index.html')

【追加】

class UsersController:

def on\_get(self, req, resp):

usrs = session.query(User).filter(Users.age > 20)

resp.content = api.template("users\_list.html", usrs\_list=users)

[ urls.py ]

from controllers import api

from controllers import IndexController

api.add\_route('/', IndexController)

api.add\_route('/users', UsersController)

#### Webフォーム

#### REST

ディレクトリ構成

myproj \

├ static \

│ ├ js\rest\_test.js

├ templates \

│ ├ index.html

│ ├ layout.html 割愛

├ controllers.py

├ run.py

[ controllers.py ]

import json

@api.route('/api/test/')

async def api\_test(req, resp):

resp.media = {'data': 'hoge'}

class IndexController:

def on\_get(self, req, resp):

resp.content = api.template('index.html')

[ static\js\rest\_test.js ]

$(document).ready(function () {

$.ajax({

type: 'GET',

url: '/api/csv/',

data:{

},

dataType: 'json'

}).done(response => {

console.log('get\_csv done!');

$("#result").html(`${response.data}`);

}).fail(function(jqXHR, textStatus, errorThrown) {

console.log('fail');

});

}

})

[ templates\index.html ]

{% extends "layout.html" %}

{% block content %}

<h1>Hello</h1>

<div id="result"></div>

{% endblock %}

参考：<https://qiita.com/nassy20/items/adc59c4b7abd202dda26>

メモ

こんな方式もある？2022-12-07

# resp.media = resp.headers = {"Content-Type": "application/json; charset=utf-8"}

# resp.content = json.dumps("{'data': 'hoge'}", ensure\_ascii=False)

#### トラブルシューティング

現象：module 'typesystem' has no attribute 'SchemaDefinitions'

原因：typesystemのバージョン問題

対処：以下のバージョンのものをインストールし直す

python3 -m pip install typesystem==0.2.5

#### 資料

公式チュートリアル： <https://responder.kennethreitz.org/en/latest/#user-guides>

### 詳細

#### 画像表示

Flaskでの例：<https://qiita.com/Gyutan/items/d3dbdc675ce7cb5cf077>

#### Opencvと併用して画像を表示する

<html>

<head>

<title>カメラ画像取得</title>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script>

$(document).ready(function() {

// ボタンクリック時の処理

$("#btn-capture").click(function() {

$.ajax({

type: "GET",

url: "/image",

dataType: "text", // テキストデータとして取得

success: function(data) {

$("#camera-image").attr("src", "data:image/jpeg;base64," + data); // Base64エンコードして表示

}

});

});

});

</script>

</head>

<body>

<h1>カメラ画像取得</h1>

<button id="btn-capture">画像を取得</button>

<br>

<img id="camera-image" width="640" height="480">

</body>

</html>

セッション変数の利用

### レファレンス

#### API

##### run

api.run(address='0.0.0.0', port=5042)

##### template

#### session

app = Flask(

\_\_name\_\_,

)

app.secret\_key = 'abcdefghijklmn'

app.permanent\_session\_lifetime = datetime.timedelta(minutes=3)

response.set\_cookie

### 資料

公式ドキュメント：<https://responder.kennethreitz.org/en/latest/api.html>

## Angular

Googleを中心としたOSSコミュニティが運営するTypeScriptベースのオープンソースのWebアプリのフレームワーク。

ライセンスMIT

## Gatsby.js

ReactJSで構築され，GraphQLを活用した静的サイトジェネレーター。

まずはNode.jsとNode.jsのモジュールマネージャーをインストール

ラズパイで使うには

curl -sL https://deb.nodesource.com/setup\_14.x | sudo -E bash -

$ sudo apt install nodejs

（情報元）<https://www.ultra-noob.com/blog/2020/164/>

使い方

gatsby-cli

サイトの作成

gatsby new [SITE\_DIRECTORY\_NAME] [URL\_OF\_STARTER\_GITHUB\_REPO]

### 資料

公式サイト<https://flask.palletsprojects.com/en/latest/>

有志による日本語サイト（非公式）：<https://msiz07-flask-docs-ja.readthedocs.io/ja/latest/index.html>

## react

### チュートリアル

アプリを新規作成

npx create-react-app myapp

cd myapp

npm start

### 詳細

#### JSX

別名：JavaScript XML

XML のような構文を使用して DOM ツリーを作成できるようにする JavaScript 拡張機能。

Reactで用いられる

const element = <h1>Hello, world!</h1>;

// React.createElement('h1', null, 'Hello, world!') と同等

# サーバー（html）

## nginx

### 概要

・リバースプロキシに向いている

・大量の動的コンテンツの処理に不向き

→こちらはApacheの得意分野

・

（Windows用）<http://nginx.org/en/download.html>　※stable版を選ぶ

設定ファイル

/etc/nginx/nginx.conf

NGINX Unit オープンソース

NGINX Plus NGINX Unitをベースに機能を追加した。（有料）

ロードバランサ

リバースプロキシ

コンテンツ キャッシュ

APIゲートウェイ

Web Application Firewall

### チュートリアル

#### インストール

sudo apt -y install nginx # インストール

systemctl status nginx # 運転状態の確認

→ 動いている場合は、初期設定でlocalhostでも、リモートからでもアクセスできる。

Webブラウザからhttp://localhost（ポートは80番）へアクセスすると「Welcome to nginx!」が表示されるはず。

server\_name コマンド

サーバーのdns名を指定

#### リバースプロキシ

初期設定のファイルをテンプレートとして使う場合、以下のようにしてコピーしてやる。

[ terminal ]

cp /etc/nginx/sites-available/default /etc/nginx/sites-available/mysite

補足：個人的に新規ファイルを作成して、手書きの方が早い気がする

最もシンプルな例の１つ。バーチャルホストexample.com:8080を作成して、exmaple.com:8080をサーバーマシンのlocalhost（127.0.0.1:8000）に転送する設定の場合。

[ /etc/nginx/sites-available/mysite ]

server{

listen 8080;

listen [::]:8080;

server\_name example.com; # 値を \_ にする事で全てのURIに対応するらしい

location / {

proxy\_pass http://127.0.0.1:8000/;

proxy\_set\_header Host $http\_host;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

シンボリックリンクを作成し、バーチャルホストを有効化する。

Terminal

sudo ln -s /etc/nginx/sites-available/mysite /etc/nginx/sites-enabled/mysite

systemctl restart nginx.service # サービスの再起動

補足：80番ポートを使いたい場合はsites-enabledからdefaultを消せば良い

sudo rm /etc/nginx/sites-enabled/default

参考：<https://office54.net/iot/linux/nginx-setting-file#section4-2>

三流資料：<https://qiita.com/uezo/items/847e1911ac486f5a89c4>

公式資料（英語）：<https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/>

#### ログの確認

/var/log/nginx/access.log

/var/log/nginx/error.log

/etc/nginx/servers/ に

（または /usr/local/etc/nginx/servers/ ）

server {

listen 80; # ポート番号

server\_name fastapitest; #

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-Host $host;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Real-IP $remote\_addr;

location / {

proxy\_pass http://127.0.0.1:8000/;

}

}

server {

listen 443;

server\_name ci.example.com ;

（略）

location / {

（略）

### 方法1

# proxy\_pass http://127.0.0.1:8080;

# proxy\_set\_header Host $http\_host;

# proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

# proxy\_redirect http:// https://;

### 方法2

# proxy\_pass http://127.0.0.1:8080;

# proxy\_set\_header Host $http\_host;

# proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

# proxy\_set\_header X-Forwarded-Proto https;

# proxy\_set\_header X-Forwarded-Port 443;

}

}

参考１：

参考２：<https://www.greptips.com/posts/1238/>

### コマンド

オプション

-g

### 詳細

### 設定ファイル

#### 概要

/etc/nginx/nginx.conf 全般の設定。

中小規模サイトの場合は、以下のファイル分の設定項目をここに集約しても良い。

/etc/nginx/uwsgi\_params uWSGIの設定

/etc/nginx/scgi\_params SCGI設定

/etc/nginx/fastcgi\_params FastCGI設定

/etc/nginx/mime.types MIMEタイプと拡張子を定義

/etc/nginx/conf.d/ 仮想サーバー毎の設定ファイルを配置。nginx.confに追記するのと同等。

/etc/nginx/sites-available/ 複数の仮想ホスト運用する場合に独自の設定ファイルを置く。

/etc/nginx/sites-enabled/ sites-availableのファイルのシンボリックををここに配置すると有効になる。

/etc/nginx/modules-enabled/ ？

/etc/nginx/modules-available/ ？

参考：https://office54.net/iot/linux/nginx-setting-file

#### nginx.conf

##### 概要

デフォルトnginx.conf の説明を兼ねて、設定ファイルの概要を記す。

# この{} 範囲外の部分を「mainコンテキスト」と呼ぶ

user www-data; # 実行ユーザーの名前

worker\_processes auto; # ワーカーのプロセス数

pid /run/nginx.pid; #

include /etc/nginx/modules-enabled/\*.conf; #

# このevents{} などを「ディレクティブ」と呼ぶ

events { # イベント駆動に関する設定

worker\_connections 768; # １つのワーカーが処理できるコネクション数

# multi\_accept on; #

}

http { #

sendfile on; #

tcp\_nopush on; #

tcp\_nodelay on; #

keepalive\_timeout 65; #

types\_hash\_max\_size 2048; #

include /etc/nginx/mime.types;

default\_type application/octet-stream;

ssl\_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE

ssl\_prefer\_server\_ciphers on;

access\_log /var/log/nginx/access.log; # アクセスログの記述先

error\_log /var/log/nginx/error.log; # エラーログの記述先

gzip on;

include /etc/nginx/conf.d/\*.conf;

include /etc/nginx/sites-enabled/\*;

}

##### ディレクティブ

http httpに関する設定（2022-07-12よく分かっていない）

server 文字通り、サーバー設定を行う

##### 設定項目

listen

server{

# デフォルトサーバー：想定しないページの表示を防いだり、IPアドレスを使った不正アクセスから防御

listen 80 default\_server;

return 444;

}

server{

listen 8080;

}

server\_name

本サーバーがリッスンするDNS名前を定める？

値を \_ にすることで、すべてのURLを受け付ける

server\_name example.com; # ttp://example.com でアクセスできるようになる

root

ドキュメントルートの指定

location /static/ { # root以降のURLパス文字を書く

root /var/www/example; # この「ディレクトリ」のルートを示す

}

# root + locationで示したurlとなる。つまりこの「ディレクトリ」のpathは /var/www/example/static/ となる

alias

location /static/ { #

alias /var/www/example/static; # この「ディレクトリ」のルートを示す

}

rootがroot+locationを読み込んできたのに対して、こちらは指定したディレクトリをそのまま読み込んでくる

index

index.htmlファイルを示す？相対パスは指定できない？

index index.html;

try\_files

location

構文

URIのエイリアスを追加

location /name/ {

proxy\_pass http://127.0.0.1/; # 末尾のスラッシュの有無で挙動が変わるらしい

}

→ example.com/name/ となる？

プレフィックス 説明

= 完全一致

^~ 前方一致。一致したら正規表現を適用しない。

~ 正規表現(大文字・小文字を区別する)

~\* 正規表現(大文字・小文字を区別しない)

なし 前方一致

proxy\_pass プロキシの転送先のURL（URI）を指定する

proxy\_set\_header アクセス元のIPやホスト情報を、転送先に通知する

proxy\_redirect

/static URL文字「static」に対しての実際のディレクトリパスを指定する

location /static {

alias /var/www/mysite/static;

}

# この場合、URL文字：www.sample.com/staticが 実際のディレクトリ/var/www/mysite/static に対応する

upstream

proxy\_pass, fastcgi\_pass, uwsgi\_pass, scgi\_pass, memcached\_pass, grpc\_pass ドライバ で参照されるサーバーのグループを定義する

ロードバランサーになる？

upstream hoge {

server backend1.example.com weight=5;

server backend2.example.com:8080;

}

events

events {

worker\_connections 1024;

}

#### sites-available

##### default\_serverの例

[ /etc/nginx/sites-available/default ]

server { # バーチャルホスト名

listen 80 default\_server;

listen [::]:80 default\_server;

# listen 443 ssl default\_server; # SSLを使う時の設定

# listen [::]:443 ssl default\_server; # 同上

〜中略〜

root /var/www/html; # html ファイルの置き場

index index.html index.htm index.nginx-debian.html;

server\_name \_; # \_ は全てのリクエストに対応

location / {

try\_files $uri $uri/ =404;

}

#### httpサーバーのリバースプロキシの例

[ /etc/nginx/sites-available/my\_site ]

server{

listen 80;

listen [::]:80;

server\_name \_;

location /static {

alias /var/www/mysite/static;

}

location / {

proxy\_pass http://127.0.0.1:8000/;

proxy\_set\_header Host $http\_host;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

### Windows版

### 資料

公式資料：https://nginx.org/en/docs/

uvicorn（英語）： <https://www.uvicorn.org/deployment/#running-behind-nginx>

参考資料（カゴヤ）：[https://www.kagoya.jp/howto/it-glossary/web/nginx/](https://www.kagoya.jp/howto/it-glossary/web/nginx/#i)#i

サーバー人気：<https://w3techs.com/technologies/history_overview/web_server/ms/y>

## Apache

### 設定ファイル

#### 概要

/etc/apache2/apache2.conf メインの設定ファイル。

### apache2.conf

メインの設定ファイル。Apacheの起動時に読み込まれる。

環境によってファイル名が違うらしい

httpd.conf：CentOs などの場合

apache2.conf ：Debianなどの場合

ディレクトリごとの許可設定<Directory>

/etc/apache2/port.conf port変更の設定はここ? ``(Listen 8080など)

設定ディレクトリ

/etc/apache2/site-available

個別のサイト設定を 000-default.confをもとに新規作成

ドキュメントルートの変更 e.g. /var/www/html

/etc/apache2/conf-available

必要に応じてsecurity.confなどを修正

security.conf

/etc/apache2/mods-available

モジュール設定

available は選択可能。enabled は現在有効になっているもの。

.htaccess

apacheで使われるhtmlサーバーの分散設定ファイル。root権限が無くても設定可能。

htmlサーバーへのアクセス毎に読み込まれる。

https://qiita.com/sunnyG/items/7e5bd6e8dc9b04c9978e

## waitress

### 概要

flaskで使う例

from flask import Flask

from waitress import serve # Waitressをインポート

app = Flask(\_\_name\_\_)

@app.route('/', methods=['GET'])

def hello\_world():

return 'Hello, World!'

if \_\_name\_\_ == '\_\_main\_\_':

serve(app, host='0.0.0.0', port=8080) # FlaskアプリをWaitressで稼働させる

資料

公式：<https://docs.pylonsproject.org/projects/waitress/en/stable/>

### レファレンス

#### serve

unix\_socket

UNIX環境でのソケットデバイスのパス文字。

公式：<https://docs.pylonsproject.org/projects/waitress/en/stable/arguments.html#arguments>

## 用語（htmlサーバー）

ドキュメントルート

クライアントからのリクエストを受け取った際のレスポンスを返す時に基本となるディレクトリの事。

一般的には静的ファイルのみが配置される事が多い

# 非同期処理

## Celery

# 開発ツール

## VSCode

### 概要

テキストディタだが， C#, C++, C, Python, Java, JavaScript, html, CSS, json, xml

と数えきれないファイル形式に対応している。

また，コーディングで使う場合でもデバッグ機能が強力。

### チュートリアル

インストール

Windows

ポータブル版：右写真のボタン→ other download → .zipを選択

※特にポータブル版で不自由を感じた事が無い

もちろんインストーラー版でも良い。

アンインストール：

以下のフォルダをフォルダごと削除

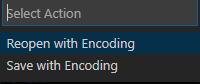
C:\Users\[ユーザー名]\.vscode

C:\Users\[ユーザー名]\AppData\Roaming\Code

Linux

#### 文字コード

文字コードを変更したい場合は画面右下の文字コードの部分  をクリック。

→ 右図のようなウィンドウが出てくる

エンコード付きで開く (Reopen with Encoding)：

今開いているテキストを，指定の文字コードで開く。

今文字化けしている時に選択。

エンコード付きで保存 (Save with Encoding)：

今開いているテキストを，指定の文字コードで保存。

文字コードを変更したい時に選択。

#### 仮想環境

#### ターミナルの変更

コマンドパレット（Ctrl+Shift+P）でshell と入力

#### エクステンション（拡張機能）

 又はCtrl+Shift+X

Remote Development

#### デフォルトパス

code --file-uri フォルダーパス

Usage: code [options][paths...]

Options

--folder-uri <uri> Opens a window with given folder uri(s)

--file-uri <uri> Opens a window with given file uri(s)

#### アンインストール

設定：C:\ユーザー\(ユーザー名)\AppData\Roaming\Code

拡張機能：C:\ユーザー\(ユーザー名)\.vscode

### VSCodeの機能

#### コマンドパレット

##### 概要

Win：Ctrl+Shift+P

##### 文書比較

概要

Linuxのdiffコマンドのように、ファイル差異を表示する

手順

・比較先のファイルを先に開いて閉じる（その方が楽なので）

・比較元のファイルを開く

・コマンドパレット：File Compare Active File With　でファイルを選択

#### settings.json

##### 概要

File → Preferences → Settings または→ Settingsでも良い

補足：検索ボックスを使えば大体の機能の操作には困らない　

スコープがUserとWorkspaceがあり、同じ設定項目を設定した場合はWorkspaceが優先される。

VSCodeの「ワークスペース」を作成しなくても、Workspaceのスコープでの設定は可能（フォルダを開く　の場合など）なお、一つでも設定値を変えると、自動的に .vscode/settings.jsonが作成される。

「ワークスペース」の場合、my\_workspace.code-workspaceというファイルが存在し，

後述するlaunch.jsonなどもこのファイルで一元管理するのが良い。以下がそのテンプレート

{

"folders": [

{

"path": "..."

}

],

"settings": { },

"tasks": { },

"launch": { },

"extensions": { }

}

補足

・画面右上の  を押すとjsonの中身が見られる

・設定画面の  でWorkspaceを選択すると今開いているディレクトリ（ワークスペース）専用の設定ファイルとなる。ディレクトリに.vscode\settings.json というファイルが作成される。

##### 設定項目の大枠

作業ディレクトリ

User：Features → Terminal → Integrated: Cwd

配色テーマ

Workbench → Appearance → Color Theme

文字コードの自動判別

"files.autoGuessEncoding": true

##### 設定項目詳細

settings.pyもしくはworkspace.code-workspaceの”settings”の設定情報

###### python関連

|  |  |  |
| --- | --- | --- |
| 項目 | 値 | 概要／備考 |
| terminal.integrated.shell.linux | 実行ファイルのパス | Linux環境のデフォルトシェル |
| terminal.integrated.shell.windows | 実行ファイルのパス | Windows環境のデフォルトシェル |
| python.pythonPath | /usr/bin/python3など | 昔python.defaultinterpreterPathとして使われていた？ |
| python.defaultinterpreterPath | /usr/bin/python3など | .pyファイルで使う既定のインタープリタ。 |
| python.terminal  .activateEnvInCurrentTerminal | true/false | 立ち上げた仮想環境を現在のターミナル有効にする？ |

python.terminal.activateEnvironment 値：true/false

自動で仮想環境を有効にする？

仮想環境を自動で有効にするには以下の設定をすればよい（らしい）

"settings": {

"python.terminal.activateEnvInCurrentTerminal": true,

"python.terminal.activateEnvironment": true,

}

###### notebook関連

|  |  |  |
| --- | --- | --- |
| 項目 | 値 | 概要／備考 |
| notebook.lineNumbers | "on" | codeセルでの行数の表示 |
| notebook.output.textLineLimit |  | 出力セルの上限行数 |

##### 具体例

###### 区切り文字

Editor: Word Separators

デフォルト：`~!@#$%^&\*()-=+[{]}\|;:'",.<>/?

推奨：

言語ごとに設定する事もできる

[ setting.json ]

"[php]":

{

"editor.wordSeparators":"`~!@#%^&\*()-=+[{]}\\|;:'\",.<>/?",

},

###### 改行文字

files.eol

##### 資料（settings.json）

おすすめ設定：https://www.cview.co.jp/cvcblog/2021.04.20.dYMmFSxAihNBwoEkCys\_X

#### ワークスペース

##### 概要

ディレクトリを開く

複数のファイル，ディレクトリを１まとめにしたもの。.code-workspace というjson形式のファイルで管理する。

※注意！この時点では単に“ディレクトリを開いた”状態。

保存：File → OpenFolder から Save Workspace As が一番楽。

閉じる：File → Close Workspace 開く：File → Open Workspace

補足

・ファイルブラウザでディレクトリを選択して右クリック → アプリケーションで開く→VSCodeでも良い。

ワークスペースとして保存

「ディレクトリを開く」で開いた状態で，メニュー：ファイル → 名前を付けてワークスペースを保存

補足

・各ワークスペースの設定はSettingを開いて Workspaceタブを選ぶ。あとはUser設定と同じ。

##### 設定

###### setting.json（VSCodeの設定）

通常通り  で設定を開く → Userではなく，Workspace を選択

なお，この設定はworkspace.code-workspaceの”settings” のところに記録される

参考設定

"python.envFile": "${workspaceFolder}/env",

"python.defaultInterpreterPath": "${workspaceFolder}/env/bin/python3"

###### launch.json（デバッグ設定）

実行とデバッグ  → launch.jsonファイルを作成します

→ 現在のワークスペースを選択

workspace.code-workspaceの”launch” のところに記録される

→ ワークスペース　を選択

launch.jsonファイルが作成される。

name: デバッグ実行の名前

type: 使用するDebugger

request: デバッグ実行のモード(launchかattach)

program: デバッグ対象のプログラムのパス

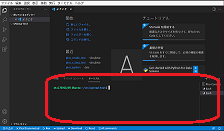
console: デバッグの結果を出力するコンソール

cwd: 現在の作業ディレクトリ

args: デバッグ実行時に渡される引数

（公式情報）<https://code.visualstudio.com/docs/editor/debugging#_launchjson-attributes>

##### 仮想環境(python)

まず，ターミナル（右図参照）を開いていなければ

メニュー：ターミナル → 新しいターミナル

でターミナルを開く。また，この時ターミナルの右端の表示で，現在ワークスペースのディレクトリにいることを確認する。

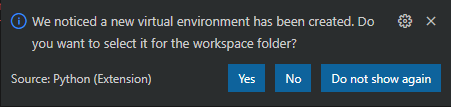
事前準備

sudo apt install python3-venv # venvが無ければインストール

touch test.py # 同ディレクトリやワークスペース内にpythonのファイルが必要

仮想環境を作成

python3 -m venv env # 仮想環境を作成

補足：同ディレクトリに.pyファイルがある場合，「新しい仮想環境を検知しました」　みたいなメッセージが表示される。（右図参照）

「Yes」を選択するとワークスペースを開くたびに仮想環境が自動的にActivateされる。

【手動で設定】Ctrl+Shift+P（コマンドパレット）→ python select

補足：以下のようなコマンドを実行するように設定すればよいらしい（どうやって？2022-02-08）

& d:/tmp/seigi\_app/env/Scripts/Activate.ps1

使い慣れてない人に；補足情報

.¥myvenv¥Scripts¥activate # 仮想環境を有効化(Win) ※powershellの実行ポリシーの変更が必要

source ~/myvenv/bin/activate # 仮想環境を有効化(Linux, Mac)

設定ファイルでのインタープリタ指定で，仮想環境のpythonを選択することで，.pyファイルのデバッグ時に自動的に仮想環境をActivateする事もできる。（参考）[settings](#_settings)

##### workspace.code-workspace

ワークスペースの設定ファイル

folders 特にワークスペース・ルートに存在しないディレクトリを追加する

“folder” : [

{

“path” : “.”

}

{

“path” : “../src”

}

]

#### 拡張機能

##### Draw.io Intergration

以下のものが作成可能

・フローチャート

・レイアウト（ワイヤーフレーム）

・ネットワーク図

・オフィスレイアウト図

・テーブル（表）

・ベン図

.drawioもしくは.dioでファイルを作成するだけ。

.drawio.pngにするとそのまま画像ファイルとして扱われる。

##### PlantUML

https://marketplace.visualstudio.com/items?itemName=jebbs.plantuml

##### copilot

月10＄必要

### 詳細

#### codeコマンド

構文：code [オプション] [パス]

オプション

--folder-uri [パス] 指定したフォルダを開く

code --folder-uri D:\src

これをショートカットにしておく手もある

Windowsの場合

--file-uri [パス]

#### tasks.json

##### 概要

プログラムコードの記述以外のさまざまなタスク（例えばプロジェクトのテストやビルド、デバッグなど）を自動化するためのもの。≒昔のセットアップ用.batファイルみたいな感じ。

##### チュートリアル

コマンドパレット（Ctrl+Shift+P）→ tasks: configure task

→ Create task.json file from template（.vscode\task.jsonが無い場合）→ Others（右図参照）

補足：2022-02-09現在、右図の違いはよく分かっていない

実行

コマンドパレットでtasks: run

タスクのスキャンとは、エラーの種類、位置情報などを出力する際の機能。

必要なければ「タスクの出力をスキャンせずに続行」を選べば良い。

##### 詳細

version: tasks.jsonのバージョン。（バージョンにより書き方が異なると推測される）  
label: タスクに名前を付ける。タスクを実行するときの一覧に表示される。

type: commandを実行するアプリケーションを示す。

値：

"shell" OSのデフォルトShell。Linuxならbash/sh、Windowsならpowershell（？）

command : 実際に実行するコマンド文字列そのものを入力。

args : コマンド引数

commandで使うパラメータ。カンマ区切りで複数指定可。

options : オプション

タスク実行時のオプション指定になります。ここでは、"cwd": "${workspaceRoot}/debug" を指定することで 作業ディレクトリを ${workspaceRoot}/debug としています。

problemMatcher : 問題出力イメージのマッチング（正規表現）

結果出力の文字列のジャンプ先取得の正規表現になります。これは、gcc および g++ が出力するエラーおよび警告メッセージを拾うように指定してあります。

runOptions:

実行条件を変更する？

"runOptions": {

"runOn": "folderOpen"

},

##### その他

実行環境（OS）毎にコマンドを変えられるらしい

"windows": {

"command": "echo",

"args": [

"hello windows"

]

},

"osx": {

"command": "echo",

"args": [

"hello macos"

]

}

#### launch.json

"name": "(gdb) 起動",

"type": "cppdbg",

"request": "launch",

"program": "${workspaceFolder}/hello",

"args": [],

"stopAtEntry": false,

"cwd": "${fileDirname}",

"environment": [],

"externalConsole": false,

"MIMode": "gdb",

"setupCommands": [

　 {

　　"description": "gdb の再フォーマットを有効にする",

　　"text": "-enable-pretty-printing",

　　"ignoreFailures": true

<https://code.visualstudio.com/docs/cpp/launch-json-reference>

#### extensions.json

ワークスペースにおいて推奨される拡張及び推奨しない拡張を指定。

#### 環境変数

${file} 現在開いているファイルの名前

${fileDirname} ディレクトリ名

${fileBasename} base名＝ディレクトリ名を省いたファイル名

${fileBasenameNotExtension} base名かつ拡張子無し

${workspaceRoot} VSCodeで開いたディレクトリのパス

${fileExtname} 現在開いているファイルの拡張子

${CWD} 起動時の現在ディレクトリ

（参考）<https://code.visualstudio.com/docs/editor/variables-reference>

### プログラム言語別

#### 共通

##### ターミナルの自動起動

"settings": {

"terminal.integrated.defaultProfile.windows": "PowerShell",

"terminal.integrated.profiles.windows": {

"PowerShell": {

"source": "PowerShell",

"args": [

"-NoExit",

"-Command",

".\\env\\Scripts\\activate"

],

}

},

}

2022-02-08うまく行かなかった

#### Python

##### python pipの使い方

(Win)PSコマンドプロンプト　(Linux)Terminal

python -m pip install --upgrade pip

##### インタープリタを追加

Ctrl+Shift+P（コマンドパレット）→ python Select Interpreter

 → 

##### インタープリタの変更

左下のこの部分をクリック

##### インタープリタ関連の設定項目

setting.pyかワークスペースの”settings”で設定する

{

（中略）

"settings": {

"python.pythonPath": "myvenv/bin/python", # 古い情報

"python.defaultInterpreterPath ": "myvenv/bin/python", # 比較的新しい情報

}

}

（詳細）[settings](#_settings)

##### pylint

pythonのコードチェッカーの1つ。後述するPEP8コーディング規則に則ってチェックをしてくれる。

PSコマンドプロンプト

python -m pip install pylint

#### C/C++

##### 概要

tasks.jsonの作成

コマンドパレットで「tasks:c」と入力し，Task: Configure Taskを選択

実行（F5）を行っても，.vscode/launch.isonと.vscode/tasks.jsonが自動的に作成される。

launch.jsonの作成

実行とデバッグ 　またはCtrl+Shift + D

### プログラム補助

#### 遠隔編集

##### リモート機（ssh）

###### 概要

ネットワークに接続している別のマシンに保存されているファイルをそのまま編集し，そのマシンの環境でデバッグもできる。

注意点　ARMはv7l, v8lしか対応していない。<https://code.visualstudio.com/docs/remote/linux>

解決策：

Windows

SCP通信ができるアプリでディレクトリを同期する

WinSCPの場合  Syncronizeを選択 → 対象ディレクトリを選択。

あとはローカル側の対象ディレクトリ内のファイルをVSCodeで編集。

Linux(Ubuntu)

ファイルブラウザでsftpができるので，それで対応。

＋ほかの場所 → sftp://192.168.1.1 などと入力 → 対象ディレクトリのファイルをVSCodeで編集。

###### インストール

拡張機能を追加

拡張機能を表示： 又はCtrl+Shift+X → Remote Development  を選択してインストール。

CONTAINERS  のアイコンが追加される。（これでリモートデバイスの選択などを行う）

OpenSSHの設定ファイル

[ ~/.ssh/config] (Linux) 補足：~./ssh/ssh\_configに同じ設定を書いてもダメらしい

[%USERPROFILE%.ssh\config] (Windows) 補足：同じくssh\_configファイルではダメ

Host test

HostName srv\_url\_or\_ip # リモートの場所。URLやIPアドレスなど

User srv\_user\_name # リモートOSでのユーザー名

IdentityFile ~/.ssh/test/gehogeho\_a.key # 必要なら。（パスワード認証も可能）

補足%USERPROFILE% はC:/Users/user\_nameのフォルダ

詳細：[OpenSSH - ssh\_config](#_ssh_config)

接続

をクリックしてSSH Targetsを開く

→ configで設定した名前（上の例だとtest）の左にある新しいウィンドウで開く  をクリック

（サーバーOSの種類を問われる事がある）

右下に と表示（初回は3～5分くらいかかる）

画面の左下が  →  となる。

フォルダを開く

エクスプローラー →  → Open Folder でリモート上のフォルダを選択できる。

次ページに続く

～VSCode リモート　続き

###### 設定

拡張

今まで使ってきたローカルの設定「~/.vscodeディレクトリ」とサーバー側の「~/.vscode-remote」は全く別な為，VSCodeの各種拡張機能は入れ直す必要がある。

> リモート接続時に  又はCtrl+Shift+X > 拡張

ターミナル

Windows環境の場合，ターミナルをbashからpowershellに変える必要がある（2021-09-16未確認

 → Select Default Profile（既定のプロファイルの設定）

pythonインタープリタ

表示→コマンドパレット　又はCtrl+Shirt+P　でコマンドパレットを表示。

注意：

・python.setInterpreter’ not found と表示された場合はpython拡張が上手く起動していない。

「VSCodeはこの大きなワークスペースでファイルの変更を監視できません」の対策

[ /etc/sysctl.conf ]（リモート側）

fs.inotify.max\_user\_watches=524288

Terminal:

sudo sysctl -p

（情報元）<https://qiita.com/miriwo/items/1871d889fb810ad9368a>

##### GitHub

###### 概要

###### 使い方

gitがコンピューターにインストールされている必要がある。[OS別]

[ Linux ]

git --version # これでインストールされているかどうか確認できる。

必要な情報を先に設定しておく [共通]

git config --global --list # これで確認できる。何も表示されない→設定が必要

git config --global user.name ‘ユーザー名’

git config --global user.email ‘メールアドレス’

クローンまたは初期化(init)

cd ~/git/next-netlify-starter # init する場合はまずinitするディレクトリへ移動

init # initする。

git clone https://github.com/user1/next-netlify-starter # クローンする場合

補足

上記はVSCode上のターミナルで実行しても良いし、ターミナル（またはcmd/powershell）で実行してからVSCodeでFile → Open Folderしても良い。

###### 補足

・Web上にはVSCodeの拡張GitHubやGit Historyをインストールする、という情報があるが，もともとソースコントロール  により，Gitのイベントは監視されているらしい。

・Stage All Changes git add -A -- と同等

###### 参考資料

コマンドと-vscode-の-git-関連操作：<https://qiita.com/satokaz/items/4660ce57ca8eb456a096#git->

#### スニペットの登録

コマンドパレット → Configure User Snippets → 言語を選択：Pythonの場合はpython.jsonが作成される

作成される場所

|  |  |  |
| --- | --- | --- |
| Debian系 |  |  |
| Windows | ~\AppData\Roaming\Code\User\snippets\python.json |  |

最もシンプルな例

"sample1": {

"prefix": "sample1",

"body": "print('hoge')"

}

この状態で.pyファイル上でsample1とタイプすると、print('hoge')が表示される

|  |  |  |  |
| --- | --- | --- | --- |
|  | 概要 | 備考 |  |
| $n | 可変値を扱う | $1, $2 とする。実際の入力の時はTabで移動  $1:hoge で初期値を指定 |  |
| $ARG | 特定の変数 | 参考：<https://code.visualstudio.com/docs/editor/userdefinedsnippets#_variables> |  |
| body | 実際に変換されるスニペット | [ "","", … ]の形式で複数行の記述が可能 |  |
| description | 説明の文字列 |  |  |
|  |  |  |  |

### 色々なVSCode

#### github.dev

github.comをgithub.devに変えるだけでgithub上のソースコードを編集できる

補足

1sというのがあったような？

#### vscode.dev

オンライン上で動くエディタ。編集対象はローカルのファイル。

当然とも言えるが、ターミナルは動作しない。

### 特定用途

#### WSL＋Docker

VSCodeの拡張

Remote - WSL

Remote-Containers

Remote Development

#### ganttdown

.ganttのファイルを作成

画面右上のをクリック

をクリック（右図参照）するか、3列のタブ区切り行を追加する。

例：

<https://github.com/anydown/vscode-ganttdown>

→使ってみた感想。ガントチャートの下書きには使えるかもしれない。2022-04-21

### その他

## Git

### 概要

Git（ギット）は各ディレクトで働くアプリで，ディレクトリ内に含まれるファイルの変更履歴などを取り扱う。

リポジトリ：各ディレクトリ（プロジェクト）に割り当てられる，開発管理対象の１単位の事。

ローカルリポジトリ：ローカルクライアントにリポジトリの事で，通常ここで更新などを行う。

リモートリポジトリ：サーバー上にあるリポジトリ。ロカールリポジトリが同期される。

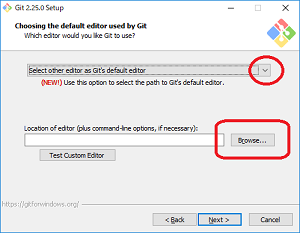
### チュートリアル

#### 最初の準備

##### 最初の準備(Win)

git bash linuxライクなCUI

git cmd 通常のcmdだが，bashへのPATHが通っていて，ハイブリッドなcmd



・インストール時にエディタを聞かれるので，ここだけはしっかりと選択しておく。

（左図参照）

・恐らくBashだけで良いので，UseBashOnly を選択しておく。

（GUIは使うにせよ，サードパーティのもので良いと思う）

補足

MSYSのような物を使うのも手。

VirtualBoxなどを使ってUbuntuなどを使う方が却って楽かもしれない。

##### 最初の準備(Linux)

Linuxの場合は基本的にgitに対応しているものが多い

Debian系：

sudo apt -y install git

あとは特定のディレクトリに移動して，git initかgit clone などを行うだけ。めちゃくちゃシンプル。

#### オンライン操作(GitHub)

※オンラインレポジトリを使用する場合のみで良い。

##### アカウント作成

まずはGitのオンラインサービスであるGitHubでアカウントを作成する。

##### レポジトリの削除

レポジトリのSetting の下の方にDangerZoneというグループがある。

その下の方にDelete this repositoryがある。（右図参照）

##### README.md

![](file\_name) ファイルを選択

#### config

再びローカル機の設定に戻る。

git config --global user.name *username* # ユーザー名の設定（先にwebで作っておいたもの）

git config --global user.email *m**ail\_address* # メールアドレス

※これを設定しないとcommitが使えない。オンラインレポジトリを使用しない場合は適当な値を入れておく。

補足

local 各リポジトリのみの設定 global 現在ユーザーの設定 system システム全体の設定

#### git基本（addとcommit）

ここではGitBashでの例。GUIなどでも基本的にやる事は同じ。

ローカルとリポジトリの新規作成とリモートとの同期の例：

setx HOME *”C:\data\hogehoge”* # 必要があればhomeディレクトリの変更

# エディタの変更（インストール中に設定し忘れたり，別のエディタを使いたい場合のみ）

git config --global core.editor "*‘C:\Program Files\Sublime Text 3\subl.exe’* --new-window --wait"

# ローカル環境の設定

mkdir “*directory\_name*” # 必要あればディレクトリを作成する

cd “*directory\_name*” # 対象ディレクトリに移動しておく。

git init # 対象ディレクトリでのGitの開始。（既存ファイルは削除されない）

# 更新を開始（先にリモートレポジトリを作っておく）

git add -A # フォルダ内にあるすべてのファイルを追加（または更新）

git commit -m “短いコメント” # 変更をcommit

git push -u https://github.com/*user*/*reposit* master # リモートへpush -u：--set-upstreamの事

# あとは更新毎にadd -A 以下を繰り返す。

# 以前の状態に戻りたいがブランチ（後述）を作り忘れた場合 ※resetやrevertを使う方法もある

git log # 過去のcommit名のリスト表示

git reset 消したいコミットID # 過去のcommitに戻る

git checkout feature/test # 最新のcommitに戻す

# 標準のリモートレポジトリを示す変数originを設定しておくと後々楽

git remote -v # 現在のorigin の設定値　空白ならば設定が必要

git remote add origin https://github.com/*user*/*reposit* #

pull（ダウンロード）する場合：

git clone https://github.com/*user*/*reposit*  # 他人のプロジェクトへの初参加時など，最初はクローンする

git clone https://github.com/*user*/*reposit dir\_path* # ディレクトリ名指定バージョン

# 注意：cloneコマンドでリポジット名のディレクトリが自動作成されるので，homeで実行するか，下側のディレクトリ指定バージョンを使う。

# リモートリポジトリで何かしらの変更があった

git pull origin master # 変更の適用。※一度cloneするとorigin という変数が使える

<https://qiita.com/gold-kou/items/7f6a3b46e2781b0dd4a0>

#### branchのチュートリアル

git branch -a # branchの一覧を見る

git checkout -b *develop* # 新branchをdevelop作成し，更に切り替える。

# -b オプション＝ git branch branch\_name + checkout branch\_name

git push -u origin *develop* # remote も新ブランチに切り替えつつpushする場合

git checkout -D *wrong\_name*  # ブランチの削除。名前を打ち間違えた時など。

# 何かしらの変更やテストをここで行う

# ブランチdevelop に問題が起こり，前の状態に戻りたいとき

git checkout master # masterに切り替える（前の状態に戻す）

# ブランチdevelopの変更をmasterに適用するとき(merge)

git checkout master # 一度masterに切り替える

git merge *develop* # ブランチの変更を適用する。master ← develop　となる。

git branch -d *develop* # 必要なくなったブランチを削除する。（当然，.残しておいても良い）

# リモートのブランチ

git push -d origin *branch\_name* # リモートのbranchを削除　--delete でも良い

git fetch -p # 削除されたリモートbranchをローカルに適用する　--pruneでも良い

補足）

・よく用いられるbranch名として，develop，issue などがある。

#### ssh公開鍵

# GitBash

# 公開鍵認証

ssh-keygen -t rsa -C "メールアドレス" # ~/.ssh/id\_rsa が作成される

# ssh-keygen -t rsa -b 4096 -C "メールアドレス" -Cと言う情報もある

# Enter file in which… ()内のパスがSSH keyの保存先となる。問題なければそのままEnter

xclip -selection c [target\_path] # 公開鍵の内容をクリップボードにコピー xclip は要インストール

# GitHub で公開鍵を登録

# 1.GitHubにログインし，右上のプロフィールをクリック -> Settingsにて設定画面に遷移

# 2.左のメニューからSSH keysをクリック

# 3. Add SSH Keyをクリック

# 4. Titleに識別できる文字列を入力

# 5. Bodyに先ほどクリップボードに入れた値をペースト

# 6. Add Keyをクリック

# 秘密鍵のパーミションを変更

chmod 600 (秘密鍵のファイル名)

# /.ssh/config ファイルの作成

# 以下内容

Host github.com

HostName github.com

IdentityFile ~/.ssh/id\_rsa # ここで指定するのは秘密鍵の方 .pub の付いていない方のファイル

User git

# ssh を用いて接続できればOK

<https://qiita.com/jun6231jp/items/5596ef0c912c7f57ef0a> Webアプリの情報

#### GitBash操作

ショートカットキー

|  |  |
| --- | --- |
| Ctrl+A 行の先頭へカーソルを移動 | 行の最後へカーソルを移動 |
| Alt+←(Alt+→) 単語単位でカーソル移動 | Atrl+I 画面をクリアする |
| ↑ 一動作前のコマンドを表示 | ↓ 一動作後のコマンドを表示 |
| Tab 入力中のコマンド補完 | Ctrl+R 履歴検索 |

### gitコマンド

#### 概要

まとめ

|  |  |
| --- | --- |
| コマンド | 概要 |
|  |  |
| submodule | 外部のリポジトリを現在のプロジェクトに追加する。 |
| rebase |  |

#### clean

git管理（追跡）の対象外のファイルを削除する。

#### commit

変更を適用する

オプション

--amend

git commit --amend -m "refs #25 htmlを修正"

#### config

##### オプション

--list

--system システム全体

--global 現在のユーザが対象 ※globalとは言葉がややこしい。

--local 現在のリポジトリのみが対象

##### サブコマンド

credential

git config --global credential.helper cache # 次回以降、認証を記憶する

git config --global --unset credential.helper # もとに戻す

cache 15分経つとメモリーから除去。

store テキストファイルでディスクに保存。暗号化なしのテキストファイルで保存。

~/.git-credentials に保存される。

--file ~/.my-credentials # 保存先の指定

osxkeychain Mac においてOS のキーチェーンへの保存。

wincred Windowsにおけるosxkeychain” と同じような仕組み。

（公式）<https://git-scm.com/book/ja/v2/Git-のさまざまなツール-認証情報の保存>

時間設定

git config --global credential.helper 'cache –timeout=3600' # 一時間

#### log

gitを使った変更の履歴（log）を確認できる

オプション

--oneline

--graph

#### ls-files

git ls-files --other --ignored --exclude-standard

#### merge

-no-ff

#### rebase

オプション

--continue

-i 修正したいコミットの範囲を指定 （最新のコミットが1）

git rebase -i HEAD~3 # 3つまえのコミットを修正

#### reset

コミットそのものを取り消す。※ローカルでのみ使用する事！

--soft commitのみ取り消し

--mixed commitとaddの取り消し [def.]

--hard 全部を取り消し

#### revert

コミットxxxを打ち消したコミットという形でcommitを取り消す

#### rm

例

git rm path/to/file

git rm -r --cached .

#### stash

変更差分をコミットせずに一時的に退避させる。

大体，ブランチを移動する為のサブ機能として使うらしい。

例

# コミットした状態 ・・・①

# 何かしらの変更をする。 ・・・②

# 他ブランチに移動（checkout）したい。がcommitか変更の破棄が必要

git stash # ②の変更を退避して，一時的に①の状態に戻る。

# 他ブランチに移動（checkout）する事ができる。

# 元のブランチに戻ってきた。

git stash apply # ②のを適用。

サブコマンド

save コメント付きでstashする。

git stash save コメント

apply [num] 退避させた変更を適用する。numを省略すると最新のもの（@0）が適用される。

pop [num] ≒apply＋対象stashをリストから削除

clear stashの全リスト削除

list

下のようにstashしたもののリストが表示される。

stash@{0}: \*\*\*\* # 通常のstashでは実行内容が表示されるが解読不可能。

stash@{1}: On master : コメント # saveでstashした場合は、コメントが表示される

オプション

-u untracked file も含めて変更差分を退避させる。

#### submodule

外部のリポジトリを現在のプロジェクトに追加する。

git submodule # サブモジュールの参照

git submodule add https://github.com/raspberrypi/pico-sdk pico-sdk # 外部リポジトリを追加

### 詳細

#### .gitconfig

#### .gitignore

/dir1/childdir/ # 特定のディレクトリ以下のファイル全てを非対象にする場合

/dir1/some\_module.py # 特定のファイルだけ非対象にしたい場合

\*.jpg # すべてのjpgを無視

\*\_\_pycache\_\_/ # 全てのpythonキャッシュを無視（かなり実用的）

### GitHub

gitのオンラインレポジトリの内，最も有名なもの。

#### VS Code風エディタ

GitHub上でソースコードを閲覧，編集しやすくする機能

.dev

ttps://github.com/user\_name/repo\_name/…

↓ このように変更。

ttps://github.**dev**/user\_name/repo\_name/…

編集後はVSCodeのように Source ControlからAdd（Checkout）→Commitで変更を適用できる。

1s

ttps://github.com/user\_name/repo\_name/…

↓ このように変更。

ttps://github**1s**.com/user\_name/repo\_name/…

2021-09-28 正直，この２つの違いは分からない。

旧情報

設定ファイル

Linux(Ubuntu 18.04)

~/.vscode

Windows10

%APPDATA%/Code/User/settings.json

Mac(macOS 10.14.6 Mojave)

~/Library/Application\ Support/Code/User/settings.json

#### Github Codespaces

有料サービスで，動作環境のインスタンスを構築して編集と実行を行えるようにしたもの。

#### Gist

オンライン上でgitコマンドの練習ができるシミュレータ。

https://gist.github.com/

#### Gitへのssh接続

#### アクセストークン

自分のアイコン→Settings→Developer settings→Personal access tokens

#### ソース検索機能

GUI（Webブラウザ）を使った方法

基本： <https://github.com/search>

詳細な検索：<https://github.com/search/advanced>

### 資料

#### Pro Git

Gitのオンライン解説書。

<http://progit-ja.github.io/>

#### LearningBranching

特にブランチ機能に関して、視覚的に学べるツール

<https://pcottle.github.io/learnGitBranching>

#### tryGit

[http://try.github.io](http://try.github.io/)

⇒ こちらにリダイレクトされる。（なくなった？）

<https://docs.github.com/en/get-started/quickstart/set-up-git>

#### コンテナ専用OS

Atomic Host

CoreOS

RancherOS

Snappy Ubuntu Core

### メモ

他にもSVN(Subversion)やTFS(Team Foundation Server)というツールがある。

please clean your repository working tree

#### GUI

Source Tree

git GUI

GitHub Desktop リモートレポジトリにのみアクセスする？

Tortoise Git Windowsのエクスプローラー上で動く拡張機能。←微妙だった20.02.10

## CMake

別ドキュメント「C/C++」に記載

## Bazel

ベイゼル(/ˈbeɪzəl/) と読むらしい（Wikipediaより）

Googleが社内で使っていたビルドツールで、AndroidやiOS向きソフトのコンパイラ。

cmakeの問題点を改善したものと謳われている。

Android、Go、C/C++、Java、Objective-C、Python、シェルスクリプト、C#　など幅広く対応。

用途

pythonのtensorflowのコンパイル

補足：2022-03-30現在、cmakeはc言語専用のコンパイルシステムだと思っていたが、色々な言語に対応しているらしい。

補足：bazelはあまりおすすめしないらしい。理由は少しマニアックなツールな為、など。

情報元：https://mixi-developers.mixi.co.jp/byebye-bazel-welcome-make-b966bfd37fce

## Gradle

Java向けに開発されたビルドツール

## Jupyter Notebook

### 概要

### コードや実行結果をそのままの形で保存できる。

実験的にコードを実行する場合や、教育などで使われる。

venvに入れる場合

source venv/bin/activate # 仮想環境のアクティベート

pip3 install jupyter # pipを用いてインストール

### インストール

sudo snap install jupyter

sudo apt install jupyter # python3が予めインストールされている必要がある

補足：venvやDockerで事足りる事が多い為、直接インストールする事は少ないと思われる。

#### VSCodeで使う

あらかじめVSCodeで"Python"拡張機能をインストール。

venv仮想環境にpipを使ってjupyterをインストール

"Select Interpreter"でjupyterが入った仮想環境を選択。

#### Dockerで使う

dockerhubに各種イメージがある

docker pull jupyter/scipy-notebook

docker pull jupyter/tensorflow-notebook

docker pull jupyter/datascience-notebook

### 詳細

#### .ipynb\_checkpoints

Jupyter labにおいて自動的に生成されるディレクトリで、Notebookの状態を記録する役割。

Gitなどでは管理対象から外しておいた方が良い（.gitigonreなどを用いる）

### 具体例

#### pythonでカレントディレクトリのパス文字列を得る

Jupyter Notebookでは\_\_file\_\_が使えない

os.getcwd() を用いる

### トラブルシューティング

print(sys.path)

sys.path.append()

## Jenkins Circle

使用実績：弥生会計

## Screwdriver.cd

CI/CDツール

使用実績：Yahoo

## Ansible

RedHat社が開発するオープンソースのネットワーク機器の構成管理ツール

設定ファイルはYAML形式。

構成管理ツールを導入することにより、それ自体がドキュメント及び手順書的な意味合いを持つことになる。

参考：Ansible以外の構成管理ツール

|  |  |
| --- | --- |
|  |  |
| Kickstart | 自動インストール（RHEL系） |
| Preseed | 自動インストール（Debian GNU/Linux系） |
| Puppet | Rubyで記述されたサーバ、クライアント型アプリケーション |
| FAI | 自動インストール（Debian GNU/Linux系） |
| Chef | Rubyで記述されたサーバ、クライアント型アプリケーション |

# APIドキュメント管理

## OpenAPI

### 概要

REST APIで作成したアプリケーションのリファレンス資料自動作成ツール。

Swagger 2.0を拡張して実装。

# VNC

## TigerVNC

比較

RealVNC VNC本家

UltraVNC Windowsに特化

# マイクロシステム

## MSYS

### 概要

Windows用の最小Unixシステムの事。

通常，C:\msys64　にインストールされる。

pacman --needed -Sy zsh pacman pacman-mirrors msys2-runtime # MSYS2の基盤ソフトウェアを更新

デフォルトのターミナル：mintty

シェル：bash，

パッケージマネージャ：pacman

### パッケージ管理

pacman -S mingw-w64-x86\_64-cmake # こんな感じで.

pacman

-h ヘルプ。

-S -h -Sオプションのヘルプ。

-Su パッケージのアップデート(事前にupdate-coreが必要)。

-Ss [pattern] パッケージを検索。

-Sl | grep [pattern]でもだいたい分かる。

-S [package] インストール。

-Rs [package]アンインストール。

-Sg パッケージグループの一覧を表示。

-Qqe インストール済みのパッケージ一覧を表示。

-Si [package] 詳しい情報を表示。

-Sc 使っていないキャッシュを削除。

資料

<http://wakita.github.io/gldojo/sb0/msys2/>

# 用語

## ASGI；Asynchronous Server Gateway Interface

非同期サーバーゲートウェイインターフェース

非同期で行われるWSGIの事

## WSGI ；Web Server Gateway Interface

ウィスキー，ウィズギー

html経由でpythonを動かすためのAPI仕様で、その仕様を実装したサーバーをWSGIサーバーと呼ぶ

## スキーマ駆動開発

フロントエンドとバックエンド間のインターフェイスを取り決め、それぞれの開発を同時にスタートする開発方式。

## asyncio

# 資料

## Open Standia

オープンソース団体？

<https://openstandia.jp/>

## WBS；Work Breakdown Structure

作業分解構成図

## MEANスタック

LAMPスタックを意識して作られたと思われる、Webアプリケーションの新たなフレームワークセット。

（但し、2022年現在、すでに一部では古いと言われている）

M：「MongoDB」 – ドキュメント指向のデータベース

E：「Express」 – サーバーサイドのJavaScriptフレームワーク

A：「Angular」 – フロントエンドのJavaScriptフレームワーク

N：「Node.js」 – サーバーサイドのJavaScript実行環境