Django応用講座 1
(Class Based View)

Class Based View（一覧）

https://docs.djangoproject.com/ja/3.1/ref/class-based-views/

これまで、勉強してきたDjangoのViewは関数を用いたViewだが、ここからはクラスを用いたViewを利用していく。
クラスを用いることで、別のViewの機能を継承して利用できたりする。

View	全てのViewの元になるView
TemplateView	テンプレートを表示する。ホーム画面などに用いる。
CreateVIew	データベースにデータを挿入するView。データ作成画面で用いる
UpdateView	データを更新するView
DeleteView	データを削除するView
ListView	特定のテーブルのデーター覧を表示するView
DetailView	テーブルのレコードの詳細を表示するView
FormView	Formを表示してデータを送信するView。
RedirectView	リダイレクトを行うView

Class Based View（View）

最も基本的なView(https://docs.djangoproject.com/ja/3.1/ref/class-based-views/base/#view)
(django.views.generic.base.View)

```
# vies.py
class MyView(View):
    pass

# urls.py
path('url/', MyView.as_view(), name='')
```

GET, POSTなどリクエストに応じて、実行する処理を変えたい場合には、get(), post()をメソッドとして定義する。

```
class MyView(View):
    def get(self, request, **kwargs):
        pass # GETの場合の処理
    def post(self, request, **kwargs):
        pass # POSTの場合の処理
```

Class Based View（TemplateView）

ホーム画面など、シンプルなテンプレート画面を作成する際に用いる
(https://docs.djangoproject.com/ja/3.1/ref/class-based-views/base/#view)
(django.views.generic.base.TemplateView)

 # urls.py
from django.views.generic.base import TemplateView

path(‘’, TemplateView.as_view(template_name=‘index.html’)) **# index.htmlを表示するViewとして定義**

 # views.py(TemplateViewの継承)

class MyTemplateView(TemplateView):

    template_name = ‘’ **# 表示するテンプレートの名前**
    def get_context_data(self, **kwargs): **# テンプレートに渡す値を指定する**

作成したら以下の用にurls.pyにパスを通す
path(‘url/’, templateView.as_view(), name=‘○○’)

Class Based View（DetailView）

挿入したデータの詳細を表示したい場合に用いる。
(django.views.generic.detail.DetailView)

```
class MyDetailView(DetailVew):
    model = MyModel # 画面に表示したいモデルを定義
    template_name = '' # 表示するテンプレート
```

 # urls.py(pkで更新したいオブジェクトを取得する)

```
path('detail_view/<int:pk>', MyDetailView.as_view(), name=' detail_view'),
```

 # テンプレート側

object: **取得したオブジェクトが格納されている。**

Class Based View（ListView）

挿入したデータの一覧を表示したい場合に用いる。
(django.views.generic.list.ListView)

```
class MyListView(ListVew):
    model = MyModel # 一覧表示するモデルを定義
    template_name = '' # 表示するテンプレート


    def get_queryset(self): # データを取得する際のSQLを定義する。

# urls.py
path('list_view/', MyListView.as_view(), name='list_view'),


# テンプレート側
object_list: 取得したオブジェクトの一覧が格納されている。
```

Class Based View（CreateView）

作成したテーブルにデータを挿入したい場合に用いる。
(django.views.generic.edit.CreateView)

```
class MyCreateView(CreateVew):
    template_name = '' # 表示するテンプレート
    model = MyModel # データを挿入するモデルを定義
    fields = ['','''] # 入力するデータ
    or
    form_class = '' # 利用するFormを定義

    success_url = '' # Create成功時の遷移先を定義する
    or
    (Model)get_absolute_url = '' # こちらは、対象のモデルに定義する

    def get_success_url() # 成功時の遷移先を定義する
    def form_valid(self, form) # formの送信時の処理をカスタマイズする
    def get_initial(self, **kwargs): # 初期値を設定する
```

Class Based View（CreateView）

django.urls.reverse_lazy **# 関数名から、その関数を呼び出すパスを返す(success_url, get_absolute_urlに用いる)**

 # テンプレート側

form: データを挿入するさいに用いるFormが入っている。

 # データを挿入する処理

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="保存">
</form>
```

Class Based View（UpdateView）

挿入したデータを更新したい場合に用いる。
(django.views.generic.edit.UpdateView)

```
class MyUpdateView(UpdateVew):
    # 作成するフィールド、メソッド
    template_name = '' # 表示するテンプレート

    model = MyModel # データを挿入するモデルを定義

    fields = ['','''] # 入力するデータ　or　form_class = '' # 利用するFormを定義

    success_url = '' # Update成功時の遷移先を定義する
    or
    (Model)get_absolute_url = '' # こちらは、対象のモデルに定義する

    def get_success_url(self) # 成功時の遷移先を定義する
    def form_valid(self, form) # formの送信時の処理をカスタマイズする。

# urls.py(pkで更新したいオブジェクトを取得する)
path('update_view/<int:pk>', MyUpdateView.as_view(), name=' update_view'),
```

Class Based View（DeleteView）

挿入したデータを削除したい場合に用いる。
(django.views.generic.edit.DeleteView)

```
class MyDeleteView(DeleteVew):
    model = MyModel # データを挿入するモデルを定義
    template_name = '' # 表示するテンプレート
    success_url = '' # 成功時の遷移先を定義する

    def get_success_url() # 成功時の遷移先を定義する


# urls.py(pkで削除したいオブジェクトを取得する)
path('delete_view/<int:pk>', MyDeleteView.as_view(), name=' delete_view'),

# テンプレート側
<form method="post">{% csrf_token %}
    <input type="submit" value="Confirm">
</form>
```

Class Based View（FormView）

一般にFormを用いる場合使う。
(django.views.generic.edit.FormView)

```
class MyDetailView(FormVew):
    template_name = ‘’ # 表示するテンプレート
    form_class = ‘’ # 利用するフォームを定義
    success_url = ‘’ # フォーム処理成功時の処理

    def get_initial(self):  # Formの初期値を定義する

    def form_valid(self, form): # POST実行時の処理を定義する
```

Class Based View（RedirectView）

別のView, 別のページにリダイレクトをしたい場合に用いられる

 # urls.pyに直接定義する

```
from django.views.generic.base import RedirectView

urlpatterns = [
    path('/search/<term>/',
        RedirectView.as_view(url='https://google.co.jp/?q=%(term)s')),
]
```

 # RedirectViewを継承してクラスを作成する

```
class SearchRedirectView(RedirectView):
  url = ‘https://google.co.jp’ # 静的に定義

class SearchRedirectView(RedirectView):
  def get_redirect_url(self, *args, **kwargs):
      return redirect(○○) # 動的に定義
```

Class Based View（SuccessMessageMixin）

データ更新時、削除時などにメッセージを表示するには、SuccessMessageMixinを用いると良い
(django.contrib.messages.views.SuccessMessageMixin)

```
class BookUpdateView(SuccessMessageMixin, UpdateView):

    model = Books

    success_message  # 成功した場合のメッセージを定義（静的）

    def get_success_message(self, cleaned_data): # 成功した場合のメッセージを定義（動的）
```