

1-16. FormとHTMLレンダリングの関係を理解する

2019年3月30日 [コメントをする](#)

今回のテーマは「FormとHTMLレンダリングの関係を理解する」です。Django使い始めてフォームの扱いがややこしいという話はよく聞きます。その1つにフォームが担う役割が多くて混乱するというのがあると思います。今回はテンプレートに渡されたフォームをどのように扱えば自分の望むHTMLが得られるのかを中心に見ていきます。

※本ページは**FormとModelForm**まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

ここまでのおさらい

現時点までで、テンプレートに渡されたフォームは`{{form}}`や`{{form.as_p}}`、`{{form.as_table}}`等で表示できるという説明をしてきました。そうすることで入力させたい項目全ての入力フォームをまとめてレンダリングしてくれる便利機能として扱ってきました。しかし、見た目の細かい調整をさせようと思うと突然融通がきかない不便さを感じます。つまり`{{form.as_p}}`は常に

```
<p><label for="id_title">タイトル:</label> <input type="text" name="title" maxlength="2" r
<p><label for="id_user_name">お名前:</label> <input type="text" name="user_name" maxlength
<p><label for="id_category">カテゴリー:</label> <select name="category" required id="id_ca
  <option value="" selected>選択して下さい</option>

  <option value="1">WEB技術</option>
  <option value="2">モバイル</option>
  <option value="3">プログラミング</option>
  <option value="4">OS関連・インフラ</option>
  <option value="5">業界ネタ・憩いの場</option>

</select></p>
<p><label for="id_message">本文:</label> <textarea name="message" cols="40" rows="10" requ
</textarea></p>
```

のように表示されます。例えば「タイトル」と「お名前」の間に何か挿入したいと思っても出来ません。順序を入れ替えたい。CSSのクラスを挿入したい。もっとフレキシブルにフォームをレンダリングする方法はないのでしょうか？

フォームを使わないで全てHTMLを手書きする

まず極論しますとテンプレートに渡されたフォームを使わなくても全てHTMLで手書きすればOKです。name属性の名前さえ間違えなければ無事POSTすることは出来ます。バリデーション処理を行う関数を別々に書いて、結果をテンプレートに渡してエラー表示させるということもフォームを使わずとも出来ます。とにかくサイトを早く完成させたい。フォームなどに付き合ってもらえない。という場合はこれでもサイトは出来ます。ただし、非常にもったいないです。

入力項目ごとにフォームをレンダリングする

実はフォームはバラバラにレンダリングすることが可能です。views.py等のプログラム上ではform['{フィールド名}']で、テンプレート内ではform.{フィールド名}で各フィールドにアクセスできます。各フィールドには以下の属性があります。

- label : ラベル用のテキスト
- label_tag : ラベルのHTMLタグ
- id_for_label : ラベルタグ用のid
- value : インプットタグ用のvalue要素の値
- help_text : 説明文言
- html_name : インプットタグのname要素の値
- 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

errorsはErrorDict型の集合体なのでfor文を使うなど注意が必要です。fieldに至っては謎ですよ。 [公式ドキュメント](#)を読んだ時も{{ field.field }}と書いてあり唖然としました。実はこれはフォームクラスの各フィールドにアクセスしているのです。つまり前回扱ったTopicFormで説明すると各プロパティであるtitleやmessageにアクセス出来ます。これを利用してviews.pyでもwidgetやmax_lengthにアクセス出来るようになるというわけです。

話を戻して、ここでは入力項目毎にレンダリングできるということを説明します。例えばタイトルの入力フォームだけをレンダリングする場合は以下の様にできます。

```
<p>
    {{form.title.label_tag}}
    {{form.title}}
    {% for error in form.title.errors %}
        {{error}}
    {% endfor %}
</p>
```

ここでテンプレート中にforループが出てきましたね。上記のようにpythonの文法と非常によく似た方法でforループを書くことが出来ます。endforを忘れないように注意して下さい。

フォームごとforループでレンダリングすることも可能です。よって{{form.as_p}}は以下の様に書くことが出来ます。

```
{% for field in form %}
<p>
    {{field.label_tag}}
    {{field}}
    {% for error in field.errors %}
        {{error}}
    {% endfor %}
</p>
{% endfor %}
```

今回の掲示板の例ではforループを使って以下のようにレンダリングすることにします。今回はSemanticのクラスも追加しながらHTML化しています。

templates/thread/create_topic.html

```
{% extends 'base/base.html' %}
{% block title %}トピック作成 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
  <div class="eleven wide column">
    <div class="ui breadcrumb">
      <a href="{% url 'base:top' %}" class="section">TOP</a>
      <i class="right angle icon divider"></i>
      <a class="active section">トピック作成</a>
    </div>
    <div class="ui segment">
      <div class="content">
        <div class="header"><h3>トピック作成</h3></div>
        <form class="ui form" action="{% url 'thread:create_topic' %}" method="POST">
          {% csrf_token %}
          {% for field in form %}
            <div class="field">{{field.label_tag}} {{field}}</div>
            {% for error in field.errors %}
              <p style="color: red;">{{error}}</p>
            {% endfor %}
          {% endfor %}
          <button type="submit" class="ui button">作成</button>
        </form>
      </div>
    </div>
  </div>
  <div class="ui segment">
    {% include 'base/sidebar.html' %}
  </div>
</div>
{% endblock %}
```

ブラウザで確認すると以下のように見える筈です。

[TOP](#) > [トピック作成](#)

トピック作成

タイトル

お名前

カテゴリー

選択して下さい ▼

本文

作成

エラーメッセージはインプットタグの下に赤字で表示されるように設定しています。errorsは複数ですので注意して下さい。

最後に

ここまでフォームの使い方を中心にユーザーが入力した情報をDBに保存する方法を見てきました。次回はクラスベースビューを使って少ないコード量でデータの登録を実装する方法を見ていきましょう。

Sponsored Link

1-15. FormとModelForm

2019年3月30日 [コメントをする](#)

今回のテーマは「FormとModelForm」です。前回までフォームはModelFormを扱ってきました。今回は次の説明のためにもFormとModelFormについて触れておきます。

※本ページは[簡単なトピック投稿画面の作成する](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

Formを使う

これまでDjangoにおけるフォームの役割を説明してきました。このフォーム機能を具現化するクラスがFormクラスです。Djangoの便利さを体感するためModelFormから見てきましたが、ModelFormはFormを継承したクラスです。ではTopicModelFormをFormを継承したクラスで書き直して見ましょう。

`thread/forms.py`

```

from django import forms
from . models import Topic, Category

class TopicModelForm(forms.ModelForm):
    class Meta:
        model=Topic
        fields=[
            'title',
            'user_name',
            'category',
            'message',
        ]

    def __init__(self, *args, **kwargs):
        # kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)
        self.fields['category'].empty_label = '選択して下さい'
        self.fields['user_name'].widget.attrs['value'] = '匿名'
        # self.fields['title'].widget.attrs['class'] = 'huga'

class TopicForm(forms.Form):
    title = forms.CharField(
        label='タイトル',
        max_length=255,
        required=True,
    )
    user_name = forms.CharField(
        label='お名前',
        max_length=30,
        required=True,
        widget=forms.TextInput(attrs={'value': '名無し'}),
    )
    category = forms.ModelChoiceField(
        label='カテゴリー',
        queryset=Category.objects.all(),
        required=True,
        empty_label='選択して下さい',
    )
    message = forms.CharField(
        label='本文',
        widget=forms.Textarea,
        required=True,
    )

```

次に先程作成したTopicFormを使ってトピックを作成する関数を見てみましょう。処理をわかりやすくするためクラスベースビューでなく敢えてtopic_create関数に変更を加えています。TopicFormのインポートを忘れないようにしてくださいね。

thread/views.py(一部抜粋)


```

from . forms import TopicModelForm, TopicForm

def topic_create(request):
    template_name = 'thread/create_topic.html'
    ctx = {}
    if request.method == 'GET':
        form = TopicForm()
        ctx['form'] = form
        return render(request, template_name, ctx)

    if request.method == 'POST':
        topic_form = TopicForm(request.POST)
        if topic_form.is_valid():
            # topic_form.save()
            topic = Topic()
            cleaned_data = topic_form.cleaned_data
            topic.title = cleaned_data['title']
            topic.message = cleaned_data['message']
            topic.user_name = cleaned_data['user_name']
            topic.category = cleaned_data['category']
            topic.save()
            return redirect(reverse_lazy('base:top'))
        else:
            ctx['form'] = topic_form
            return render(request, template_name, ctx)

```

views.pyのcreate_topic関数を使うようにurls.pyも変更しておきましょう

thread/urls.py

```

from django.urls import path

from . import views
app_name = 'thread'

urlpatterns = [
    # path('create_topic/', views.TopicFormView.as_view(), name='create_topic'),
    path('create_topic/', views.topic_create, name='create_topic'),
]

```

前回説明を省きましたが、formのコンストラクタにrequest.POSTを入れることでデータと紐付けられたフォームが出来ます。このフォームはis_valid()関数を呼ぶことでデータの検証を行えます。

Topicオブジェクトの保存の仕方が変更されたことにお気づきでしょうか？Formの場合はis_valid()関数が呼ばれた後にcleaned_dataから検証済みのデータを取り出し、Modelのオブジェクトに値をセットしてオブジェクトのsave()関数を呼び保存します。尚、データの検証

はFormに書かれた各フィールドの内容に応じてチェックされます。

一方、ModelFormはis_valid()関数が呼ばれるとModelに記載されたルールでデータ検証が行われます。そして保存する場合はModelForm自体が備えているsave()を呼び出して保存をします。ここが大きな違いですね。

セレクトタグの未選択の文言やValueの初期値を与える

カテゴリーはセレクトタグを使用していますが、（正確にはデフォルトのwidgetがselectという意味）このセレクトタグの未選択状態では「――」というように点線で表現されています。この文言を変更したいという需要は多いと思います。Formの場合は以下のようにempty_labelを指定するだけです。また、入力値の初期値を与えるなどタグの要素を変更する場合は以下のようにします。

thread/forms.py

```
user_name = forms.CharField(
    label='お名前',
    max_length=50,
    required=True,
+    widget=forms.TextInput(attrs={'value': '名無し'}),
)
category = forms.ModelChoiceField(
    label='カテゴリー',
    queryset=Category.objects.all(),
    required=True,
+    empty_label='選択して下さい',
```

確認してみましょう。以下のように変更されていればOKです。

トピック作成

タイトル:

お名前:

カテゴリー: 選択して下さい ▼

- 選択して下さい
- WEB技術
- モバイル
- プログラミング
- OS関連・インフラ
- 業界ネタ・憩いの場

本文:

作成

ではModelFormの場合はどうすれば良いのでしょうか？ ModelFormの場合は以下のようにします。

thread/forms.py

```
class TopicModelForm(forms.ModelForm):
    class Meta:
        model=Topic
        fields=[
            'title',
            'user_name',
            'category',
            'message',
        ]

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['category'].empty_label = '選択して下さい'
        self.fields['user_name'].widget.attrs['value'] = '名無し'
```

ModelFormの場合、プロパティを直接変更できないため、__init__関数をオーバーライドして対応します。この方法はインプットタグのclass要素を修正したい場合にも使えます。また、ModelFormでwidgetを変更する場合には以下のように変更することが出来ます。widgetの指定をしながらタグの要素を指定するのです。

thread/forms.py

```

class TopicModelForm(forms.ModelForm):
    class Meta:
        model=Topic
        fields=[
            'title',
            'user_name',
            'category',
            'message',
        ]
+     widgets = {
+         'title' : forms.TextInput(attrs={'class': 'hoge'}),
+         'user_name' : forms.TextInput(attrs={'value': '名無し'}),
+     }

    def __init__(self, *args, **kwargs):
        # kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)
        self.fields['category'].empty_label = '選択して下さい'

```

これでレンダリングされた'title'のinputタグには'hoge'クラスが付与されています。クラスだけではなくattrsを用いるとタグの様々な属性について操作をすることが出来ます。

label_suffixを変更する

label_suffixとはラベルの末尾につく記号でデフォルトではコロンがついています。たとえば「タイトル:」や「お名前:」のように全てコロンが付いていますよね。場合によってはこれを変更したい、或いは削除したいということもあると思います。そういう場合にlabel_suffixを設定します。フォーム生成時にコンストラクタに渡す方法もあるのですが、__init__関数をオーバーライドしてlabel_suffixを設定する方法が簡単で良いと思います。

thread/forms.py(一部抜粋)

```

class TopicModelForm(forms.ModelForm):
    class Meta:
        model=Topic
        fields=[
            'title',
            'user_name',
            'category',
            'message',
        ]

    def __init__(self, *args, **kwargs):
+         super().__init__(*args, **kwargs)
        self.fields['category'].empty_label = '選択して下さい'
        self.fields['user_name'].widget.attrs['value'] = '名無し'

class TopicForm(forms.Form):
    label_suffix = ''
    title = forms.CharField(
        label='タイトル',
        max_length=255,
        required=True,
    )
    user_name = forms.CharField(
        label='お名前',
        max_length=30,
        required=True,
        widget=forms.TextInput(attrs={'value': '名無し'}),
    )
    category = forms.ModelChoiceField(
        label='カテゴリー',
        queryset=Category.objects.all(),
        required=True,
        empty_label='選択して下さい',
    )
    message = forms.CharField(
        label='本文',
        widget=forms.Textarea,
        required=True,
    )

    def __init__(self, *args, **kwargs):
+         kwargs.setdefault('label_suffix', '')
        super().__init__(*args, **kwargs)

```

これでラベルのコロンが消えました。もちろん、矢印等にカスタムすることも可能です。公式ドキュメントでは[フォームAPI](#)で扱っています。

最後に

ModelFormとFormどちらも似たようなことが出来るので使い方に迷うかも知れません。モデルドリブンなWebアプリの場合にはModelFormの方が望ましいのではないかと考えています。さて、次回はフォームによるレンダリングについて少し詳しく見ていこうと思います。

Sponsored Link

1-14. 簡単なトピック投稿画面の作成する

2019年3月30日 [2件のコメント](#)

今回のテーマは「簡単なトピック投稿画面の作成する」です。おまたせ致しました。ここまで時間が掛かりましたね。Django使っても全然Webアプリが出来上がらないじゃないかとお叱りを受けそうですが、何事も基礎が大事です。第一章でDjangoの使い方を大まかに捉えると応用は比較的簡単だと思います。

では掲示板の新規トピックを登録する画面を作っていきます。まずはフォームがあるだけのシンプルな登録画面を作ります。登録が完了するとTOP画面に作成したトピックが表示されるようにしましょう。

※本ページは[DetailViewを使った詳細表示画面の作成](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

トピック登録画面のテンプレート作成

新しい画面を作成するのでまずはテンプレートを作りましょう。作成するファイルは `templates/thread/create_topic.html` です。

`thread/create_topic.html`

```
{% extends 'base/base.html' %}
{% block title %}トピック作成 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
  <div class="eleven wide column">
    <div class="ui breadcrumb">
      <a href="{% url 'base:top' %}" class="section">TOP</a>
      <i class="right angle icon divider"></i>
      <a class="active section">トピック作成</a>
    </div>
    <div class="ui segment">
      <div class="content">
        <div class="header"><h3>トピック作成</h3></div>
        <form class="" action="{% url 'thread:create_topic' %}" method="POST">
          {% csrf_token %}
          {{form.as_p}}
          <button type="submit" class="ui button">作成</button>
        </form>
      </div>
    </div>
  </div>
  {% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

キーポイントが二つあります。1つは{% csrf_token %}ですね。これはクロスサイトリクエストフォージェリという脆弱性に対するセキュリティ対策でトークンによって認証されないリクエストは受け付けない仕組みです。POSTメソッドでは必須です。（意図的csrf_tokenなしでアクセスすることもできますが、セキュリティ上の脆弱性を熟考の上判断して下さい。）

もう一つは{{form.as_p}}ですね。これはビュー側から‘form’というコンテキストを受け取るために記述しているのですが、もう少し先で解説します。

フォームを使う

さて、ここでフォームというものが登場します。フォームというとHTMLの入力フォームを思い浮かべると思いますが、Djangoにおけるフォーム機能はもう少し幅の広い機能を提供します。ウェブアプリがフォームを持つ場合、以下のような処理が必要にあります。

- データをレンダリングするための準備
- HTMLとしてフォームをレンダリングすること
- フォームから送信されたデータを処理すること

Djangoではフォームというパーツがこれらの処理を守備範囲として受け持ちます。詳細は[公式ドキュメント](#)も参照下さい。

フォームにはついては別の機会にもう少し詳しく扱う予定ですが、ここではとにかく使ってみましょう。今回はModelFormというフォームを使います。まず、thread/forms.pyを作成します。

thread/forms.py

```
from django.forms import ModelForm
from . models import Topic

class TopicCreateForm(ModelForm):
    class Meta:
        model=Topic
        fields=[
            'title',
            'user_name',
            'category',
            'message',
        ]
```

ModelFormを継承したTopicCreateFormを用意します。ModelFormを継承したクラスはMetaクラスにmodelとfieldsを指定することでフォームを作成出来ます。今回のようにモデルに対応するフォームを作成する場合は便利な方法です。（今回は扱いませんが、ModelFormを使わずフォームを生成する方法も当然あります。）

ビューを作成する

次にthread/views.pyに関数ベースのビューを追加していきます。

thread/views.py

```

from django.shortcuts import render, redirect
from django.urls import reverse_lazy

from . forms import TopicCreateForm
from . models import Topic

def topic_create(request):
    template_name = 'thread/create_topic.html'
    ctx = {}
    if request.method == 'GET':
        ctx['form'] = TopicCreateForm()
        return render(request, template_name, ctx)

    if request.method == 'POST':
        topic_form = TopicCreateForm(request.POST)
        if topic_form.is_valid():
            topic_form.save()
            return redirect(reverse_lazy('base:top'))
        else:
            ctx['form'] = topic_form
            return render(request, template_name, ctx)

```

GETメソッドでアクセスされた場合とPOSTメソッドでアクセスされた場合に処理を分けています。とくに明示しない場合はGET

でのアクセスとして扱われます。TopicCreateFormのインスタンスをコンテキストで渡しています。これがthread/create_topic.htmlで描写されるというわけです。テンプレートの部分で説明を保留していましたが、先程フォームにはHTMLフォームとしてレンダリングする機能もあると説明しました。一番簡単なレンダリングは`{{form}}`とすることです。これは単順にHTMLのinputタグとlabelタグを並べるだけです。これをもう少し整形するための手法が以下の方法です

`{{form.as_p}}` : Pタグで囲んで段落毎に整形する。

`{{form.as_ul}}` : LIタグで囲んでリスト整形表示する。ただしULタグは別途記載する必要あり

`{{form.as_table}}` : テーブル表示するための整形。ただしTABLEタグは別途記載する必要あり

今回はform.as_pを使って整形したということです。もうお気づきかも知れませんが、as_p, as_ul, as_tableはそれぞれ整形されたHTMLタグを返す関数です。

POSTメソッドで受けた場合にはis_valid()関数を呼んでデータの精査を行います。今回はModelFormを使用しているためTopicモデルが有している情報に適しているか精査されます。文字の長さやNullの許容等が正しくない場合は精査に失敗します。正しいデータが来た場合はフォームのsave()関数を呼んで保存しています。この方法はModelFormの場合のみ使える方法です。ややこしいので別の機会に触れます。データの精査に失敗した場合はtopic_formにエラーメッセージが入っていますので、これをコンテキストとして渡して再度create_topic.htmlを表示します。

では、追加した関数とURLを結びつけるためにthread/urls.pyに追加していきます。

thread/urls.py

```
from django.urls import path

from . import views
app_name = 'thread'

urlpatterns = [
    path('create_topic/', views.topic_create, name='create_topic'),
]
```

これで準備は整いました。localhost:8080/thread/create_topic/にアクセスしてみましょう。

[TOP](#) > [トピック作成](#)

トピック作成

タイトル:

お名前:

カテゴリー:

本文:

最後に

無事、トピックは登録出来ましたでしょうか？次回はDjangoにおけるフォームの役割についてもう少し見ていきたいと思います。

Sponsored Link

1-13. DetailViewを使った詳細表示画面の作成

2019年3月29日 [4件のコメント](#)

今回のテーマは「DetailViewを使った詳細表示画面の作成」です。前回に引き続きクラスベースビューの使い方を覚えていきましょう。DetailViewは名前の通り詳細情報を表示するビューです。今回は前回リスト表示したトピックの詳細を表示するページを作っていきます。

※本ページは[DetailViewを使った詳細表示画面の作成](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

トピック詳細ページを作る

まずはtemplates/thread/detail_topic.htmlを作成します。

templates/thread/detail_topic.html

```
{% extends 'base/base.html' %}
{% block title %}トピック作成 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
  <div class="eleven wide column">
    <div class="ui breadcrumb">
      <a href="{% url 'base:top' %}" class="section">TOP</a>
      <i class="right angle icon divider"></i>
      <a class="section">{{topic.category.name}}</a>
      <i class="right angle icon divider"></i>
      <a class="active section">{{topic.title}}</a>
    </div>
    <div class="ui segment">
      <div class="content">
        <div class="header"><h3>{{topic.title}}</h3></div>
        <p>{{topic.user_name}} - {{topic.created}}</p>
        <div class="ui secondary segment">
          <p><pre>{{topic.message}}</pre></p>
        </div>
      </div>
    </div>
  </div>
  <div>
    {% include 'base/sidebar.html' %}
  </div>
</div>
{% endblock %}
```

次にthread/views.pyに追記しましょう。

thread/views.py

```
from django.shortcuts import render, redirect
from django.views.generic import CreateView, FormView, DetailView

from . models import Topic

class TopicDetailView(DetailView):
    template_name = 'thread/detail_topic.html'
    model = Topic
    context_object_name = 'topic'
```

非常に簡単ですね。DetailViewはtemplate_nameとmodelに値を渡してあげると、URLで渡されたpk(primary key)に対応したオブジェクトを呼び出してテンプレートに渡してくれます。その際、例えばTopicオブジェクトならばtopicという名前で渡されます。もしテンプレートに渡す名前を指定したい場合はcontext_object_nameで指定します。今回の場合はcontext_object_nameがなくても問題なく機能しますが、できるだけcontext_object_nameは書いたほうが良いと思います。

URLを作成する

ページにアクセスするURLを決めましょう。{domain_root}/thread/{トピックID}というURLでアクセス出来るようにしましょう。

thread/urls.pyを以下のようにします。

thread/urls.py

```
from django.urls import path

from . import views
app_name = 'thread'

urlpatterns = [
    path('<int:pk>/', views.TopicDetailView.as_view(), name='topic'),
]
```

ここで<int:pk>がポイントです。URLからpkを渡すことでDetailViewがpkを元にオブジェクトを取得してくれます。

ブラウザで確認してみましょう。localhost:8080/thread/1にアクセスしてます。番号はトピックのIDであればOKです。

IT学習ちゃんねる

このサイトはなに？

トピック作成

[TOP](#) > [プログラミング](#) > [Python入門サイトを教えてください](#)

Python入門サイトを教えてください

名無し - 2019年3月6日15:17

サンプルサンプルサンプルサンプル
サンプルサンプルサンプル
サンプルサンプル
サンプル

さて、ここで読者諸氏はTemplateViewクラスや関数で書いた場合にどうなるか興味があると思いますので、記載例を見てみましょう。

まずTemplateViewで書いた場合です。

thread/views.py(一部抜粋)

```
from django.shortcuts import render, redirect, get_object_or_404
from django.views.generic import CreateView, FormView, DetailView, TemplateView

class TopicTemplateView(TemplateView):
    template_name = 'thread/detail_topic.html'

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        ctx['topic'] = get_object_or_404(Topic, id=self.kwargs.get('pk', ''))
        return ctx
```

オブジェクトの取得とテンプレートへの受け渡しをget_context_dataをオーバーライドして行う必要があります。この際に、不適切なpkが渡された場合にはNot found 404を返すようにget_object_or_404関数を使用します。これはよく使う手法です。importも忘れないようにして下さいね。また、DetailViewでは自動的に処理されていたpkがself.kwargsから取得している点も注目して下さい。pkから渡されたパラメータはpkに関わらず、このように取り出すことが出来ます。

次に関数ベースで書いた場合ですが、以下例のようになります。

thread/views.py (一部抜粋)

```
from django.shortcuts import render, redirect, get_object_or_404

def detail_topic(request):
    ctx = {}
    template_name = 'thread/detail_topic.html'
    if request.method == 'GET':
        ctx['topic'] = get_object_or_404(Topic, request.kwargs.get('pk', ''))
        return render(request, template_name, ctx)
```

特に問題はないと思いますが、pkの取得の仕方が、変わっている点に注意して下さい。
最後に、この本題ではないですが、トップページの各トピックへのリンクを修正しておきましょう。

templates/base/top.html(一部抜粋)

```
<div class="item">
    <div class="content">
        <div class="header">
            <a href="{% url 'thread:topic' pk=topic.id %}"><h4>{{topic.title}}</h4></a>
        </div>
        <div class="meta">
            <span class="name">{{topic.user_name}}</span>
            <span class="date">{{topic.created}}</span>
        </div>
    </div>
</div>
```

引数をpk=topic.idで渡しているところがポイントですね。これでトップページから各トピックにアクセスすることが出来るようになりました。

最後に

ここまで初期投入したデータや管理画面から入力したデータを表示することを見てきました。
次回からはユーザーがデータを登録できる画面を作っていきますよ。

Sponsored Link

1-12. ListViewを使ったリスト表示画面の作成

2019年3月28日 [コメントをする](#)

今回のテーマは「ListViewを使った一覧表示画面の作成」です。クラスベースビューとしては現在までにTemplateViewを見ていきましたが、おそらくその次に初心者の方が使いやすなのがListViewとDetailViewだと思います。ListViewはリスト表示、DetailViewは詳細表示を作成するのに向いているクラスベースビューです。名前の通りですね。

※本ページは[ListViewを使ったリスト表示画面の作成](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

準備

今回はトピックの一覧表示をするので管理画面からトピックに関するダミーデータを挿入しましょう。管理画面のTOPIC画面から「TOPICを追加」を押します。

ようこそ **ADMIN**. [サイトを表示](#) / [パスワードの変更](#) / [ログアウト](#)

TOPIC を追加 +

項目を入力していきます。入力したら保存を押せばトピックが出来上がります。一覧表示の練習なので4つぐらい登録しておきましょう。

ホーム › Thread › Topics › topic を追加

topic を追加

お名前:

タイトル:

本文:

カテゴリー:

----- ▼   

保存してもう一つ追加

保存して編集を続ける

保存

トピック一覧表示の作成

今回はbaseアプリケーションに作成したトップページに新着トピックをリスト表示するページを作成します。theadアプリケーションのモデルをbaseアプリケーションで使うのが気になる方はthread/views.pyでビューを作成してbase/urls.pyもしくはmysite/urls.pyでビューを呼び出してもいいと思います。筆者はトップページなどのbaseアプリケーションはモデルを持たず、他のアプリケーションからインポートして使うことが多いですが、好みの問題かと思います。

まずはテンプレートを以下のように修正します。

templates/base/top.html

```

{% extends 'base/base.html' %}
{% block title %}ITについて切磋琢磨する掲示板 - {{ block.super }}{% endblock %}
{% block content %}
<div class="ui grid stackable">
  <div class="eleven wide column">
    <div class="ui breadcrumb">
      <a class="active section">TOP</a>
    </div>
    <div class="ui segment">
      <div class="content">
        <div class="header"><h3>新着トピック</h3></div>
        <div class="ui divided items">
          {% for topic in topic_list %}
            <div class="item">
              <div class="content">
                <div class="header">
                  <a href=""><h4>{{topic.title}}</h4></a>
                </div>
                <div class="meta">
                  <span class="name">{{topic.user_name}}</span>
                  <span class="date">{{topic.created}}</span>
                </div>
              </div>
            </div>
          {% endfor %}
        </div>
      </div>
    </div>
  </div>
  {% include 'base/sidebar.html' %}
</div>
{% endblock %}

```

次にbase/views.pyを修正します。

base/views.py

```

from django.shortcuts import render
from django.views.generic import TemplateView, ListView

from thread.models import Topic

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

class TopView(TemplateView):
    template_name = 'base/top.html'

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        ctx['title'] = 'IT学習ちゃんねる(仮)'
        return ctx

class TopicListView(ListView):
    template_name = 'base/top.html'
    model = Topic
    context_object_name = 'topic_list'

```

base/urls.pyも変更しましょう。

base/urls.py(一部抜粋)

```

urlpatterns = [
-     path('', views.TopView.as_view(), name='top'),
+     path('', views.TopicListView.as_view(), name='top'),
    path('terms/', TemplateView.as_view(template_name='base/terms.html'), name='terms'),
]

```

これで一覧表示されました。簡単ですね。ちなみにcontext_object_name='topic_list'としましたが、これがなくても問題なく表示されます。理由はListViewを使用したときのコンテキストで渡すデフォルトの名前が"モデル名_list"だからです。今回はデフォルトでtopic_listが渡されています。ただ、この暗黙の命名はDjangoに馴染みのないエンジニアに混乱をもたらしますのでテンプレート側に何を渡しているのかcontext_object_nameで明確に宣言したほうが良いと思います。

さて、この新着スレッドですが、作成日で降順に並べるように変更しましょう。非常に簡単に出来ます。base/views.pyを少し変更するだけです。

base/views.py(一部抜粋)

```
class TopicListView(ListView):
    template_name = 'base/top.html'
-    model = Topic
+    queryset = Topic.objects.order_by('-created')
    context_object_name = 'topic_list'
```

modelの変わりにquerysetを使用します。ListViewはmodelもしくはquerysetが必須です。get_queryset関数をオーバーライドしても良いです。Topic.objectにはBaseManageを継承したクラスであり、一般的にDjangoのデータベースCRUD処理はこのobjectを通して行うことができます。今回はorder_by()関数を用いて作成日の降順に並べました。'created'にマイナス(-)がつくことで降順を表現しています。慣れない内はちょっとギョッと・・・しません？

ブラウザで確認するとこのような画面が見えると思います。

TOP

新着スレッド

[Python入門サイトを教えて下さい](#)

名無し 2019年3月6日15:17

[儲かるWebアプリが作りたい](#)

マイケル 2019年3月6日15:16

[DjangoとLaravel](#)

匿名 2019年3月6日15:15

[Djangoモデル作成について](#)

名無し 2019年3月6日15:15

杉

ナ

di
—
di
—
di
—
di
—
di

最後に

モデルに働きかけViewがユーザーに見せるべきデータを揃え、テンプレートに渡すというDjangoのMVT全ての役者が揃いましたね。次回はDetailViewを使用して表示処理を行っていきます。

Sponsored Link

1-11. 管理画面と管理者の作成

2019年3月27日 [コメントをする](#)

今回のテーマは「管理画面と管理者の作成」です。ちょっとここで話が変わりますが、管理画面が使える方が今後の作業が進むので、このタイミングで挿入しました。Djangoを使うなら管理画面を使わないと損ですよ。

※本ページは[初期データの投入](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

Djangoの管理画面

ここで一度Djangoの管理画面に目を向けてみましょう。Djangoが注目を集めた理由の1つに管理画面の自動作成機能がありました。ここでいう管理画面とはモデルのCRUD*1)を行うことが出来るWEB画面という意味です。多くのフレームワークでは管理画面作成ライブラリこそあれ、フレームワークの機能としてGUI管理画面が自動生成されるというのは画期的でした。もちろん、この管理画面を使用せずに自作することも出来ますし、プロジェクトによっては自動生成された管理画面では諸々の理由から使うことが出来ないかも知れません。しかし、せっかく自動生成される管理画面があるのですから、使わない手はない。と筆者は考えております。

*1) CRUD: create, read, update, delete操作の頭文字


管理者の作成

管理画面に入るためには管理者を生成しないといけません。管理者の生成は以下のコマンドで行います。

```
(venv)$ ./manage.py createsuperuser
ユーザー名 : [入力]
メールアドレス : [入力]
パスワード : [入力]
パスワード確認 : [入力]
```

これで管理者が生成されました。では管理画面に入ってみましょう。
localhost:8080/admin/にアクセスします。

こんな画面が見えればOKです。先程作成した管理者アカウントでログインしてみましょう。



Django 管理サイト

ユーザー名:

パスワード:

ログイン

しかし、threadアプリケーションで生成したモデルがありません。管理画面から見えるように設定する必要があります。

thread/admin.pyに追記していきます。

thread/admin.py

```
from django.contrib import admin

+ from . import models
+ admin.site.register(models.Topic)
+ admin.site.register(models.Comment)
+ admin.site.register(models.Category)
```

これで管理画面に反映されるようになりました。

Django 管理サイト

ようこそ **ADMIN** サイトを表示 / パスワードの変更 / ログアウト

サイト管理

THREAD	
Categories	+ 追加 ✎ 変更
Comments	+ 追加 ✎ 変更
Topics	+ 追加 ✎ 変更

認証と認可	
グループ	+ 追加 ✎ 変更
ユーザー	+ 追加 ✎ 変更

最近行った操作

自分の操作

利用不可

最後に

現段階では特にカスタマイズしないで管理画面を使用していきます。次回からいよいよプログラミングっぽくなっていきます。ようやくですね。

Sponsored Link

1-10. 初期データの投入

2019年3月26日 [コメントをする](#)

こんにちは。今回のテーマは「初期データの投入」です。データベースに予めデータを入れておきたいケースは珍しくないと思います。今回はカテゴリ情報の初期データを投入してみます。

※本ページは[データベースのマイグレーション](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

Djangoにおける初期データの投入

データを予め投入しておきたいということは様々な場面であると思います。Djangoではマイグレーションで投入する方法とフィクスチャーを使う場合の2通りがあります。詳細は[公式ドキュメント-モデルに対する初期データを投入する](#)を参照して下さい。

今回はフィクスチャーを使った初期データの投入をしていきます。

fixturesファイルの準備

フィクスチャーを使ったデータ投入にはfixturesファイルを作り、それを読み込む作業によって行います。fixturesファイルはJSON,YAML,XMLの形式で書くことが出来ます。今回はJSONで書いてみます。

thread/fixtures/thread/category_data.json

```
[
  {
    "model" : "thread.category",
    "pk" : 1,
    "fields" : {
      "name" : "WEB技術",
      "url_code" : "web_app",
      "sort" : 1
    }
  },
  {
    "model" : "thread.category",
    "pk" : 2,
    "fields" : {
      "name" : "モバイル",
      "url_code" : "mobile",
      "sort" : 2
    }
  },
  {
    "model" : "thread.category",
    "pk" : 3,
    "fields" : {
      "name" : "プログラミング",
      "url_code" : "programing",
      "sort" : 3
    }
  },
  {
    "model" : "thread.category",
    "pk" : 4,
    "fields" : {
      "name" : "OS関連・インフラ",
      "url_code" : "os",
      "sort" : 4
    }
  },
  {
    "model" : "thread.category",
    "pk" : 5,
    "fields" : {
      "name" : "業界ネタ・憩いの場",
      "url_code" : "chatter",
      "sort" : 5
    }
  }
]
```

YAML等での同様の構造であれば問題ありません。

データの投入

それでは早速投入していきましょう。Djangoがfixturesファイルを探すのはデフォルトでは各アプリケーションのfixturesフォルダです。以下のようなコマンドでロードします。

```
(venv)$ ./manage.py loaddata thread/category_data.json
```

これでデータが投入されました。確認してみましょう。

```
[forum_data] select * from thread_category;
```

[出力結果]

```
+-----+-----+-----+
| id | name                | url_code | sort |
+-----+-----+-----+
| 1 | WEB技術             | web_app  | 1    |
| 2 | モバイル            | mobile   | 2    |
| 3 | プログラミング      | programing | 3    |
| 4 | OS関連・インフラ    | os       | 4    |
| 5 | 業界ネタ・憩いの場  | chatter  | 5    |
+-----+-----+-----+
```

最後に

投入したデータに変更を加えて再度loaddataを行うとデータが初期化されてしまうので注意が必要です。次回は管理画面について紹介していきます。

Sponsored Link

1-9. データベースのマイグレーション

2019年3月25日 [1件のコメント](#)

今回のテーマは「データベースのマイグレーション」です。Djangoにはデータベースマイグレーション機能があります。これを使うことで作成したモデルを元に自動的にデータベースへテーブルを作成してくれます。Ruby on Railsに馴染みのある方にはお馴染みの機能だと思います。余談ですがCakePHPにはマイグレーションとは逆方向のデータベースからモデルを生成するbake model機能があり、とても面白いと思います。

※本ページは[モデルの作成](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

初回のマイグレーション

まず、Django標準のモデルのマイグレーションを行いましょう。

```
(venv)$ ./manage.py migrate
```

これで初回のマイグレーションが終わりました。現段階では自作のモデルはマイグレーションされません。自作モデルをマイグレーションするにはmakemigrationsを実行する必要があります。

```
(venv)$ ./manage.py makemigrations
```

```
Migrations for 'thread':
thread/migrations/0001_initial.py
- Create model Category
- Create model Comment
- Create model Topic
- Add field topic to comment
- Add field user to comment
```

上記のような出力が得られマイグレーションの準備が出来ました。確認してみましょう。

```
(venv)$ ./manage.py showmigrations
```

```
thread
[ ] 0001_initial
```

このような出力が得られます。[]内がからということは未だマイグレーションされていないということです。マイグレーション前にどのようなSQLが実行されるのか確認する癖はつけておいた方が良いでしょう。

```
(venv)$ ./manage.py sqlmigrate thread 0001
```

これでthreadアプリケーションの0001番のマイグレーション時のSQLを閲覧できます。意図せぬSQLが発行されないか確認しておきましょう。ではマイグレーションしてみましょう。


```
(venv)$ ./manage.py migrate
```

以下の様に出力されればOKです。

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, thread
Running migrations:
  Applying thread.0001_initial... OK
```

MariaDB/MySQLの場合、以下のようなWARNINGが出るかも知れません。

```
WARNINGS:
?: (mysql.W002) MySQL Strict Mode is not set for database connection 'default'
   HINT: MySQL's Strict Mode fixes many data integrity problems in MySQL, such as dat
```

この場合mysite/settings.pyのDATABASEを修正します。参考:[Django-MySQL](#)

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'forum_data',
        'USER': 'forum_user',
        'PASSWORD': 'devpassword009',
        'HOST': 'localhost',
+       'OPTIONS': {
+         'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
+       },
    }
}
```

これでthreadアプリケーションのモデルをマイグレーション出来ました。今後、モデルを変更することがあれば同様の手順でmakemigrationsを行いマイグレーションを行います。makemigrationsを実行する度にthread/migrationsディレクトリにマイグレーションの記録を残していきます。履歴はshowmigrationsで確認できますし、sqlmigrateでSQLも確認できます。

最後に

他のウェブフレームワークを使用したことがある方は特に躓くところはなかったのではないのでしょうか？次回は初期データの投入をしていきます。

Sponsored Link

1-8. モデルの作成

2019年3月22日 [コメントをする](#)

今回のテーマは「モデルの作成」です。ここまで非常に初歩ですが、ViewとTemplateについて見てきました。残りはModelですね。Djangoではモデルに様々な情報をもたせることでデータベースとの連携やバリデーション処理をスマートに行える仕組みを持っています。今回は掲示板ということで簡単なモデルを作って学習することにしましょう。

※本ページは[TemplateViewでテンプレートを表示する](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

threadアプリケーションの追加

まずはthreadアプリケーションを作ります。これは掲示板のスレッドに関することを処理するアプリケーションです。

```
(venv)$ ./manage startapp thread
```

baseアプリケーションと同様にthread/urls.pyの追加、mysite/settings.pyにアプリケーションの追加、mysite/urls.pyにURLの追加を行います。この辺りはbaseアプリケーションと同様です。

mysite/settings.py(一部抜粋)

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

mysite/urls.py(一部抜粋)

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

thread/urls.py

```
from django.urls import path  
  
from . import views  
app_name = 'thread'  
  
urlpatterns = [  
]
```

これでアプリケーションの追加処理はOKです。

モデルの追加

threadアプリケーションのモデルは以下の様なものを想定します。まずは掲示板の最低限の機能を有するように以下のようなモデルを考えます。

Topic: ID、タイトル、本文、ユーザー名、作成日、更新日

Comment: ID、本文、ユーザー、作成日、公開フラグ

Category: ID、タイトル、ソート番号、URLコード

まずはソース全体がどうなるかを見てみましょう。thread/models.pyは以下のようになります。

thread/models.py

```
from django.db import models

class TopicManager(models.Manager):
    # Topic操作に関する処理を追加
    pass

class CommentManager(models.Manager):
    # Comment操作に関する処理を追加
    pass

class CategoryManager(models.Manager):
    # Category操作に関する処理を追加
    pass

class Category(models.Model):
    name = models.CharField(
        'カテゴリー名',
        max_length=50,
    )
    url_code = models.CharField(
        'URLコード',
        max_length=50,
        null=True,
        blank=False,
        unique=True,
    )
    sort=models.IntegerField(
        verbose_name='ソート',
        default=0,
    )
    objects = CategoryManager

    def __str__(self):
        return self.name

class Topic(models.Model):
    user_name = models.CharField(
        'お名前',
        max_length=30,
        null=True,
        blank=False,
    )
    title = models.CharField(
        'タイトル',
        max_length=255,
        null = False,
        blank = False,
    )
    message = models.TextField(
```

```

        verbose_name='本文',
        null=True,
        blank=False,
    )
    category = models.ForeignKey(
        Category,
        verbose_name='カテゴリー',
        on_delete=models.PROTECT,
        null=True,
        blank=False,
    )
    created = models.DateTimeField(
        auto_now_add=True,
    )
    modified = models.DateTimeField(
        auto_now=True,
    )
    objects = TopicManager()

    def __str__(self):
        return self.title

class Comment(models.Model):
    id = models.BigAutoField(
        primary_key=True,
    )
    no = models.IntegerField(
        default=0,
    )
    user_name = models.CharField(
        'お名前',
        max_length=30,
        null=True,
        blank=False,
    )
    topic = models.ForeignKey(
        Topic,
        on_delete=models.PROTECT,
    )
    message = models.TextField(
        verbose_name='投稿内容'
    )
    pub_flg = models.BooleanField(
        default=True,
    )
    created = models.DateTimeField(
        auto_now_add=True,
    )
    objects = CommentManager()

```

```
def __str__(self):  
    return '{}-{}'.format(self.topic.id, self.no)
```

簡単な解説

モデルフィールド

モデルの各クラスのプロパティはモデルフィールドで規定していきます。[公式のモデルフィールドリファレンス](#)にモデルフィールドの種類がまとまっています。Djangoではモデルが持つ関連情報はできるだけ箇所にまとめることを理想としていて、データベース格納時のデータ型、リレーションに関する情報、デフォルト値、nullの許容等の情報を付与していきます。多くのWeb開発ではAPIの設計、データベース設計を終えた後に実装に入ると思いますので、設計を素直にモデルに書き起こしていく作業となります。

nullとblank

nullとblankが出てきました。nullはデータベース上でNULLを許容するかどうか？であり、blankはユーザーが入力項目として必須項目にするかどうかということです。具体的にはblankにFalseを当てるとフォームでインプットタグにrequiredが付きます。

IDの自動生成

さて、IDに関する記述がないのに気づきましたか？IDに関しては明記しない場合はDjangoによって自動生成されます。その場合はデータ型はintとなりPrimary Keyが設定されます。

外部キー

今回、1対多の関係を構築するためにトピックはカテゴリーのidをコメントはトピックのidを保有しています。このような場合DjangoではForeignKeyを用います。その場合、外部キーとして保有してるモデルが削除された場合の削除方法をon_deleteで指定します。種類としては以下のようなものがあります。

- models.CASCADE：外部キーのモデルと一緒に参照しているモデルも削除される
- models.PROTECT：保護される(参照されている場合には削除できない)
- models.SET_NULL：外部キーのモデルが削除された場合IDはNULLとなる
- models.SET_DEFAULT：デフォルトで設定されている値がセットされる。

オススメはPROTECTでしょうか。CASCADEは少々リスクが高いですが、設計次第では問題ないと思います。

マネージャーについて

また、今回はTopicManagerとCommentManager,CategoryManagerを用意し、各モデルのobjectsと紐づけました。今後、各モデルに関する操作に関わる処理はこのマネージャーに追加していき、ビューからはその処理を呼び出し、モデルに作用するようにします。慣れない内はViewで多くの処理をしてしまい肥大化しがちですが、モデルが担うビジネスロジックの処理はモデルで行うべきかと（自戒をこめて）考えています。

最後に

解説が駆け足となってしまう、説明が不十分な点もあるかと思いますが、まずはDjangoのモデルがどういう性質のものを体感してもらうことが大事かと思います。次回は作成したモデルを使っていきますよ。

Sponsored Link

1-7. TemplateViewでテンプレートを表示する

2019年3月20日 [8件のコメント](#)

今回のテーマは「TemplateViewでテンプレートを表示する」です。テンプレートを表示するためにはViewに書いた関数でrenderを使えば良いことを紹介してきました。ここでもう1つの表示方法を紹介しておきます。

※本ページは[1-6. テンプレートの継承とinclude](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

関数のビューとクラスベースビュー

Djangoのビューの書き方は関数で書く場合とクラスベースビューというクラスを用いる方法があります。歴史的には関数によるビューが先で、オブジェクティブな拡張性をもとめてクラスベースのビューが登場したようです。(参考：[クラスベースビュー入門](#))

クラスベースビューには汎用的に用いられるものがDjango側で予め用意しており、それらのクラスをオーバーライドして拡張して使うことが頻繁にあります。関数型のビューだけでなくクラスベースビューにも早いうちから慣れておくことが良いと思います。

TemplateViewを使ってみる

汎用的なクラスベースビューにはTemplateView, DetailView, ListViewなどがあり、より少ないコードで表示できるよう工夫してあります。今回はTemplateViewを使ってみます。TemplateViewはViewクラスをオーバーライドしたテンプレートを表示するためのビュークラスです。現在のbase/views.pyを書き換えてみましょう。

base/views.py

```
from django.views.generic import TemplateView

class TopView(TemplateView):
    template_name = 'base/top.html'

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        ctx['title'] = 'IT学習ちゃんねる(仮)'
        return ctx
```

base/urls.pyも書き換えます。

base/urls.py

```
urlpatterns = [
    -   path('', views.top, name='top')
    +   path('', views.TopView.as_view(), name='top'),
]
```

これだけでは何のメリットがあるのか分からないかも知れませんが、クラスベースビューは頻繁に使うので覚えておいて損はないです。また、テンプレートにコンテキストを渡す際に、`get_context_data`関数をオーバーライドする方法もよく用います。（この他にも方法はあるのですが、今はこの方法だけ覚えておけばよいかと思います。）

TemplateViewのもう少し便利な使い方を見ていきましょう。もしコンテキストをviews.pyから渡す必要がない場合はurls.pyのみで完結することも出来ます。単順にHTMLファイルを出力したいだけの場合はviews.pyからコンテキストを渡す必要はありません。今回は利用規約ページがそのようなページに相当する想定で実装してみます。

まずはテンプレートを用意します。

templates/base/terms.html

```

{% extends 'base/base.html' %}
{% block title %}利用規約 - {{ block.super }}{% endblock %}
{% block content %}

<div class="ui grid stackable">
  <div class="eleven wide column">
    <div class="ui breadcrumb">
      <a href="{% url 'base:top' %}" class="section">TOP</a>
      <i class="right angle icon divider"></i>
      <a class="active section">利用規約</a>
    </div>
    <div class="ui segment">
      <div class="content">
        <div class="header"><h3>利用規約</h3></div>
        <p>あんなことやこんなことを守って下さい。などなど</p>
        <p>.....</p>
        <p>.....</p>
        <p>.....</p>
      </div>
    </div>
  </div>
  {% include 'base/sidebar.html' %}
</div>
{% endblock %}

```

base/urls.pyは以下のようにします。

base/urls.py

```

from django.urls import path
+ from django.views.generic import TemplateView
from . import views
app_name = 'base'

urlpatterns = [
    path('', views.top, name='top'),
+   path('terms/', TemplateView.as_view(template_name='base/terms.html'), name='terms'),
]

```

ブラウザで確認すると以下ようになります。

[TOP](#) > [このサイトはなに？](#)

このサイトはなに？

このサイトはDjangoのサンプルアプリです。などなど

ここでbase/terms.htmlの中でリンク先の指定時に{% url 'base:top' %}を用いました。このように書くとbase/urls.pyでapp_nameで指定した'base'アプリケーション名の'top'のURLを自動的に挿入してくれます。便利なのでよく使います。また、views.pyの中でもリダイレクト時などに良く出てくる表現です。さて、本題ですが、これでテンプレートを表示するだけであればurls.pyだけで事足りることが分かりましたね。