

Python Django入門 (4)

Python Django

Bootstrapの導入

Django で CRUD を作る説明の前に、Bootstrapの導入を行います。

我々エンジニアが作るHTMLは、デザインの的にも味気ないものとなってしまいます。

そこで、CSSフレームワークのBootstrapを使うことにします。

ドキュメントは英語ですが、実例が載っているので、やりたいことは見つかると思います。

Bootstrap

Bootstrap で必要な jQuery を含めて、以下からダウンロードします。

- [Bootstrap](http://getbootstrap.com/) - <http://getbootstrap.com/> (v4.0.0)
- [jQuery](http://jquery.com/) - <http://jquery.com/> (v3.4.1)

ダウンロードしたファイルを、mybook/cms/static/cms/ というディレクトリを作って、以下のように配置します。

*.map ファイルは、ブラウザーの開発者ツールで、JavaScriptのデバッグをしたり、HTMLの要素を見てCSSを検証したりする際に、元の行番号を出すためのものです。なくても動作はします。

bootstrap.bundle.min.js は、Popper.js を含む（バンドルしている）ものです。

```
mybook
├── cms
│   ├── static
│   │   ├── cms
│   │   └── css
```

```
|   |— bootstrap.min.css
|   └─ bootstrap.min.css.map
└─ js
    |— bootstrap.bundle.min.js
    |— bootstrap.bundle.min.js.map
    |— jquery-3.4.1.min.js
    └─ jquery-3.4.1.min.map
```

なぜ mybook/cms/static/cms/ なのかというと、

- Django は static ディレクトリを、プロジェクト（mybook）配下の各アプリケーション（cms）の static ディレクトリを探します（mybook/cms/static）
- 各アプリケーションの static ディレクトリは1つにまとめられて解釈されるため、識別のため自分のアプリケーション名（cms）を更に追加しておきます。
（mybook/cms/static/cms）

これらは、後に出てくる template の探し方の仕組みとも似ていますが、アプリケーションを独立して再利用するためのものです。

今はわからなくてもかまいません。そういうものだと思って下さい。

django-bootstrap4

一覧系HTMLは、Bootstrapで必要なclassを手書きして、登録/修正のフォーム系HTMLは django-bootstrap4 という Python モジュールを使います。

- [django-bootstrap4](https://github.com/zostera/django-bootstrap4) - <https://github.com/zostera/django-bootstrap4>

インストールは、pip コマンドで行います。

```
python manage.py runserver
```

 で動いている時は、Ctrl + c で停止してから行ってください。

(env1) 仮想環境にいることを前提にしています。

残念ながら、今まで使っていた [django-bootstrap-form](#) は Bootstrap3 で更新が止まっていて、4には対応していないようなので、こちらに乗り換えました。

```
$ pip install django-bootstrap4
```

現時点では、1.1.1 が入りました。

```
$ pip freeze -l
asgiref==3.2.3
beautifulsoup4==4.8.2
Django==3.0.2
django-bootstrap4==1.1.1
pytz==2019.3
soupsieve==1.9.5
sqlparse==0.3.0
```

確認できたら、また `python manage.py runserver` で起動しておきましょう。

`mybook/settings.py` の `INSTALLED_APPS` に `'bootstrap4'` を追加します。

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
'django.contrib.sessions',  
'django.contrib.messages',  
'django.contrib.staticfiles',  
'cms.apps.CmsConfig',      # cms アプリケーション  
'bootstrap4',              # django-bootstrap4  
)
```

Django の CRUD

ビューをざっと作る

一覧、登録、修正、削除のフアンクションが必要なので、
'cms/views.py' にひな形を作ります。

登録、修正は編集としてひとまとめにしています。(book_id
の指定がなければ登録、あれば修正)

コードスタイルとして、defの前も2行あけないと波線が出て、うるさいです。

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
```

```
def book_list(request):
    """書籍の一覧"""
    return HttpResponseRedirect('書籍の一覧')
```

```
def book_edit(request, book_id=None):
    """書籍の編集"""
    return HttpResponseRedirect('書籍の編集')
```

```
def book_del(request, book_id):
    """書籍の削除"""
    return HttpResponseRedirect('書籍の削除')
```

URL スキームの設計

cms/urls.py というファイルは存在しないので、新しく作ります。

左側の cms の所で右クリックして、New > Python File
で `urls.py` と指定します。

この中で、URLと、ビューのフアンクションの紐付けを行います。

URLとフアンクション名は一致させる必要はありません。
このあたりは自由です。

```
from django.urls import path
from cms import views

app_name = 'cms'
urlpatterns = [
    # 書籍
    path('book/', views.book_list, name='book_list'),    # 一覧
    path('book/add/', views.book_edit, name='book_add'), # 登録
    path('book/mod/<int:book_id>/', views.book_edit, name='book_mod'),
    path('book/del/<int:book_id>/', views.book_del, name='book_del'),
]
```


次に、 `cms/urls.py` をプロジェクト全体の `mybook/urls.py` の中でインクルードします。

```
from django.contrib import admin
from django.urls import path, include    # ←, includeを追加

urlpatterns = [
    path('cms/', include('cms.urls')),    # ←ここを追加
    path('admin/', admin.site.urls),
]
```

最終的にURLは以下ようになります。

ブラウザで確認してみてください。

```
http://127.0.0.1:8000/cms/book/
http://127.0.0.1:8000/cms/book/add/
http://127.0.0.1:8000/cms/book/mod/5/
http://127.0.0.1:8000/cms/book/del/7/
```

一覧のビュー

一覧を表示するビューを、きちんと書いてみます。

cms/views.py の def book_list を以下のように修正します。

```
from django.shortcuts import render
from django.http import HttpResponseRedirect

from cms.models import Book

def book_list(request):
    """書籍の一覧"""
    # return HttpResponseRedirect('書籍の一覧')
    books = Book.objects.all().order_by('id')
    return render(request,
                    'cms/book_list.html',      # 使用するテンプレート
                    {'books': books})          # テンプレートに渡す
```

一覧のテンプレートを作る

mybook プロジェクトの cms アプリケーションで使う book_list.html というテンプレートを作成します。

位置は、以下のようになります。

```
mybook/cms/templates/cms/book_list.html
```

その前に、これの継承元となる base.html というテンプレートを作成しましょう。

左側の cms を右クリックして、New > Directory
で templates を指定
template を右クリックして、New > Directory で cms を
指定
cms を右クリックして、New > HTML File
で base.html を指定、といった感じです。

```
mybook/cms/templates/cms/base.html
```

base.html の中身は、以下のようになります。

```
{% load i18n static %}
<!DOCTYPE html>{% get_current_language as LANGUAGE_CODE %}
<html lang="{{ LANGUAGE_CODE|default:"en-us" }}">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-sca
<link rel="stylesheet" href="{% static 'cms/css/bootstrap.min.'
{% block extra_css %}{% endblock %}
<title>{% block title %}My books{% endblock %}</title>
</head>
<body>
  <div class="container">
    {% block content %}
      {{ content }}
    {% endblock %}
  </div>
<script src="{% static 'cms/js/jquery-3.4.1.min.js' %}"></scri
<script src="{% static 'cms/js/bootstrap.bundle.min.js' %}"></
{% block extra_js %}{% endblock %}
</body>
</html>
```

それでは、この base_html を継承して
mybook/cms/templates/cms/book_list.html を作成します。

こちらは必要な所のみを書いていく形となります。

class に指定しているのは、Bootstrapで必要な class です。

```
{% extends "cms/base.html" %}
```

{% block title %}書籍の一覧{% endblock title %}

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

書籍の一覧

```
<a href="{% url 'cms:book add' %}" class="btn btn-primary |
```

[illegible]|
 scope="col">ID | 書籍名 | 出版社 | ページ数 | 操作 |

</thead>

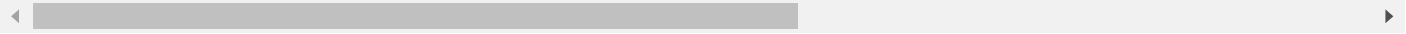
<tbody>

```
{% for book in books %}
```

|
 <th scope="row">{{ book.id }}</th> |

```
<td>{{ book.name }}</td>
```

```
<td>{{ book.publisher }}</td>
<td>{{ book.page }}</td>
<td>
    <a href="{% url 'cms:book_mod' book_id=book.id %}"
    <a href="{% url 'cms:book_del' book_id=book.id %}"
</td>
</tr>
{% endfor %}
</tbody>
</table>
{% endblock content %}
```



それではブラウザで確認してみましょう。

<http://127.0.0.1:8000/cms/book/>

書籍の一覧

追加

ID	書籍名	出版社	ページ数	操作
1	Django入門	GeekLab.Nagano	100	修正 削除
2	書籍 2	GeekLab.Nagano	200	修正 削除

追加、修正のフォーム

cms/forms.py というファイルを作って、以下のように記述します。

ここでは、 cms/models.py の Book モデルを追加、修正するための元となるフォームを作成します。

```
from django.forms import ModelForm
from cms.models import Book
```

```
class BookForm(ModelForm):
    """書籍のフォーム"""
```

```
class Meta:
    model = Book
    fields = ('name', 'publisher', 'page', )
```

追加、修正のビュー

cms/views.py の def book_edit を以下のように修正します。

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

```
from cms.models import Book
from cms.forms import BookForm
```

```
:
```

```
def book_edit(request, book_id=None):
    """書籍の編集"""
    # return HttpResponseRedirect('書籍の編集')
    if book_id:    # book_id が指定されている (修正時)
        book = get_object_or_404(Book, pk=book_id)
    else:          # book_id が指定されていない (追加時)
        book = Book()
```



```
if request.method == 'POST':
    form = BookForm(request.POST, instance=book) # POST さ
    if form.is_valid(): # フォームのバリデーション
        book = form.save(commit=False)
        book.save()
        return redirect('cms:book_list')
else: # GET の時
    form = BookForm(instance=book) # book インスタンスからこ

return render(request, 'cms/book_edit.html', dict(form=form
```

追加、修正のテンプレート

mybook/templates/base.html を継承して

mybook/cms/templates/cms/book_edit.html を作成します。

```
{% extends "cms/base.html" %}
```

```
{% load bootstrap4 %}
```

```
{% block title %}書籍の編集{% endblock title %}
```

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

```
<h4 class="mt-4 mb-5 border-bottom">書籍の編集</h4>
```

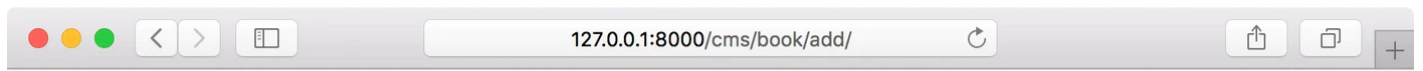
```
{% if book_id %}
```

```
<form action="{% url 'cms:book_mod' book_id=book_id %}" me
{% else %}
<form action="{% url 'cms:book_add' %}" method="post">
{% endif %}
    {% csrf_token %}
    {% bootstrap_form form layout='horizontal' %}
    <div class="form-group row">
        <div class="offset-md-3 col-md-9">
            <button type="submit" class="btn btn-primary">送信</b
        </div>
    </div>
</form>
<a href="{% url 'cms:book_list' %}" class="btn btn-seconda
{% endblock content %}
```

cms/forms.py の BookForm で定義した項目を、
cms/views.py で form という変数でテンプレートに渡し、
{% bootstrap_form form layout='horizontal' %} と
書いている form の部分で、フォームの内容がHTMLで展開
されます。

ここではさらに、 bootstrap_form タグを使うことにより、
Bootstrapの書式に変換しています。

書籍の登録ページは、以下のようになります。



書籍の編集

書籍名

出版社

ページ数

送信

戻る

削除のビュー

cms/views.py の def book_del を以下のように修正します。

```
def book_del(request, book_id):  
    """書籍の削除"""  
    # return HttpResponse('書籍の削除')  
    book = get_object_or_404(Book, pk=book_id)  
    book.delete()  
    return redirect('cms:book_list')
```

それでは、書籍の一覧ページから「削除」ボタンを押して、書籍を削除してみてください。



書籍の一覧

追加

ID	書籍名	出版社	ページ数	操作
1	Django入門	GeekLab.Nagano	100	修正 削除
2	書籍 2	GeekLab.Nagano	200	修正 削除

いきなり消していますが、本当は Bootstrap の モーダルダイアログを出して、確認メッセージを出した方がいいでしょう。

これは後に取り組むこととします。

[Python Django入門 \(5\)](#) に続きます。