

## 2-11. 画像アップロード機能を付与する

2019年4月16日 [コメントをする](#)

今回のテーマは「画像アップロード機能を付与する」です。掲示板に画像のアップロード機能をつけることでファイルの扱いについて見ていきましょう。全てのファイルの種類を扱うのは難しいので画像データのアップロードという点に焦点を当てて見ていきたいと思います。

※本ページは[Cookieへのデータの保存と読み出し](#)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。

### MEDIA\_ROOTの設定

Djangoでファイルを扱うにはMEDIA\_ROOT変数でファイルを取り扱うディレクトリを指定する必要があります。このディレクトリはDjangoの実行ユーザーの書き込み権限がある必要があります。今回はmysiteプロジェクト内にmediaディレクトリを作成することにします。少々ややこしいのでディレクトリ構成を図示します。

```
mysite
├─mysite
├─base
├─thread
├─search
├─static
└─media
```

```
(venv)$ cd mysite #プロジェクトディレクトリ
(venv)$ mkdir -p media
```

このmediaディレクトリを認識させるためにmysite/settings.pyに以下を追記します。

#### mysite/settings.py(一部抜粋)

```
+ MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
+ MEDIA_URL = '/media/'
```

MEDIA\_URL変数はWEBアプリでファイルを指し示すURLのルートとなります。今回は'/media/'としましたが、他のフレーズでもOKです。

次にmysite/urls.pyの設定をします。urlpatterns変数を以下の様に変更します。

#### mysite/urls.py(一部抜粋)

```
+ from django.conf.urls.static import static
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('django.contrib.auth.urls')),
    path('', include('base.urls')),
    path('thread/', include('thread.urls')),
    path('api/', include('api.urls')),
    path('search/', include('search.urls')),
    path('sitemap.xml', sitemap, {'sitemaps': sitemaps}),
- ]
+ ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

## モデルの作成

今回はFileFieldを継承したImageFieldを使っていきます。ImageFieldを使う場合はPillowが必要になるためインストールします。

```
(venv)$ pip install pillow
```

コメント投稿時に画像をアップロードできるようにするためthread/models.pyを修正していきます。

#### thread/models.py(Commentクラス関連部分のみ)

```

class CommentManager(models.Manager):
    # Comment操作に関する処理を追加
    def create_comment(self, user_name, message, topic_id, image=None):
        comment = self.model(
            user_name=user_name,
            message=message,
            image=image
        )
        comment.topic = Topic.objects.get(id=topic_id)
        comment.no = self.filter(topic_id=topic_id).count() + 1
        comment.save()

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'
    )
    image = models.ImageField(
        verbose_name='投稿画像',
        validators=[FileExtensionValidator(['jpg', 'png'])],
        upload_to='images/%Y/%m/%d/',
        null=True,
        blank=True,
    )
    pub_flg = models.BooleanField(
        default=True,
    )
    created = models.DateTimeField(
        auto_now_add=True,
    )
    objects = CommentManager()

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

```

併せてビューも修正しておきます。

### thread/views.py(一部抜粋)

```
class TopicViewAndCommentCreateView(FormView):
    template_name = 'thread/detail_topic.html'
    form_class = CommentModelForm

    def form_valid(self, form):
        Comment.objects.create_comment(
            user_name=form.cleaned_data['user_name'],
            message=form.cleaned_data['message'],
            topic_id=self.kwargs['pk'],
+           image=form.cleaned_data['image']
        )
        response = super().form_valid(form)
        return response

    def get_success_url(self):
        return reverse_lazy('thread:topic', kwargs={'pk': self.kwargs['pk']})

    def get_context_data(self):
        ctx = super().get_context_data()
        ctx['topic'] = Topic.objects.get(id=self.kwargs['pk'])
        ctx['comment_list'] = Comment.objects.filter(
            topic_id=self.kwargs['pk']).annotate(vote_count=Count('vote')).order_by(
        return ctx
```

ではテンプレートを修正していきましょう。templates/thread/detail\_topic.htmlは以下のように修正されます。validatorsで拡張子によるバリデーション処理を行うように指定しています。特定の拡張子しか受け付けたくないときには便利です。upload\_toにはmediaディレクトリ内のアップロードファイルを指定します。%Yのような指定子を用いることで日付や時間をディレクトリ名とすることも出来ます。今回は画像なしでもコメント投稿できるようにするためにnull,blankはTrueとしています。

### templates/thread/detail\_topic.html

```

{% extends 'base/base.html' %}
{% block title %}{{topic.title}} - {{ block.super }}{% endblock %}
{% block content %}
{% load threadfilters %}
{% load static %}
<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 href="{% url 'thread:category' url_code=topic.category.url_code %}" class="
            <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 segment">
                    <p><pre>{{topic.message}}</pre></p>
                </div>
            </div>
        </div>
    <!--コメント表示-->
    <div class="ui segment">
        {% if comment_list %}
        {% for comment in comment_list %}
        <div class="ui segment secondary">
            <p>{{comment.no}}. {{comment.user_name}}<br>{{comment.created}}</p>
            {% if comment.pub_flg %}
            <p>{{comment.message | comment_filter | safe}}</p>
+             {% if comment.image %}
+             <a href="{{comment.image.url}}" target="_blank" rel="noopener norefe
+             {% endif %}
            <div class="vote_button ui right aligned vertical segment" style="cursor
            data-comment-id="{{comment.id}}" data-count="{{comment.vote_count}}"
            <i class="heart outline icon"></i>
            <span class="vote_counter">
                {% if comment.vote_count > 0 %}{{comment.vote_count}}{% endif %}
            </span>
            </div>
            {% else %}
            <p style="color: grey">コメントは非表示とされました</p>
            {% endif %}
        </div>
        {% endfor %}
        {% else %}
        <div class="ui warning message"><p>まだコメントはありません</p></div>

```

```

        {% endif %}
    </div>
    <!--//コメント表示-->
    <!--コメント投稿-->
    <h4>コメント投稿</h4>
    <div class="ui segment">
-       <form class="ui form" action="" method="POST">
+       <form class="ui form" action="" method="POST" enctype="multipart/form-data">
            {% csrf_token %}
            {{form.as_p}}
            <button class="ui button orange" type="submit">コメント投稿</button>
        </form>
    </div>
    <!--//コメント投稿-->
</div>
{% include 'base/sidebar.html' %}
</div>
{% endblock %}
{% block js %}
<script src="{% static 'js/vote.js' %}" type='text/javascript'></script>
{% endblock %}

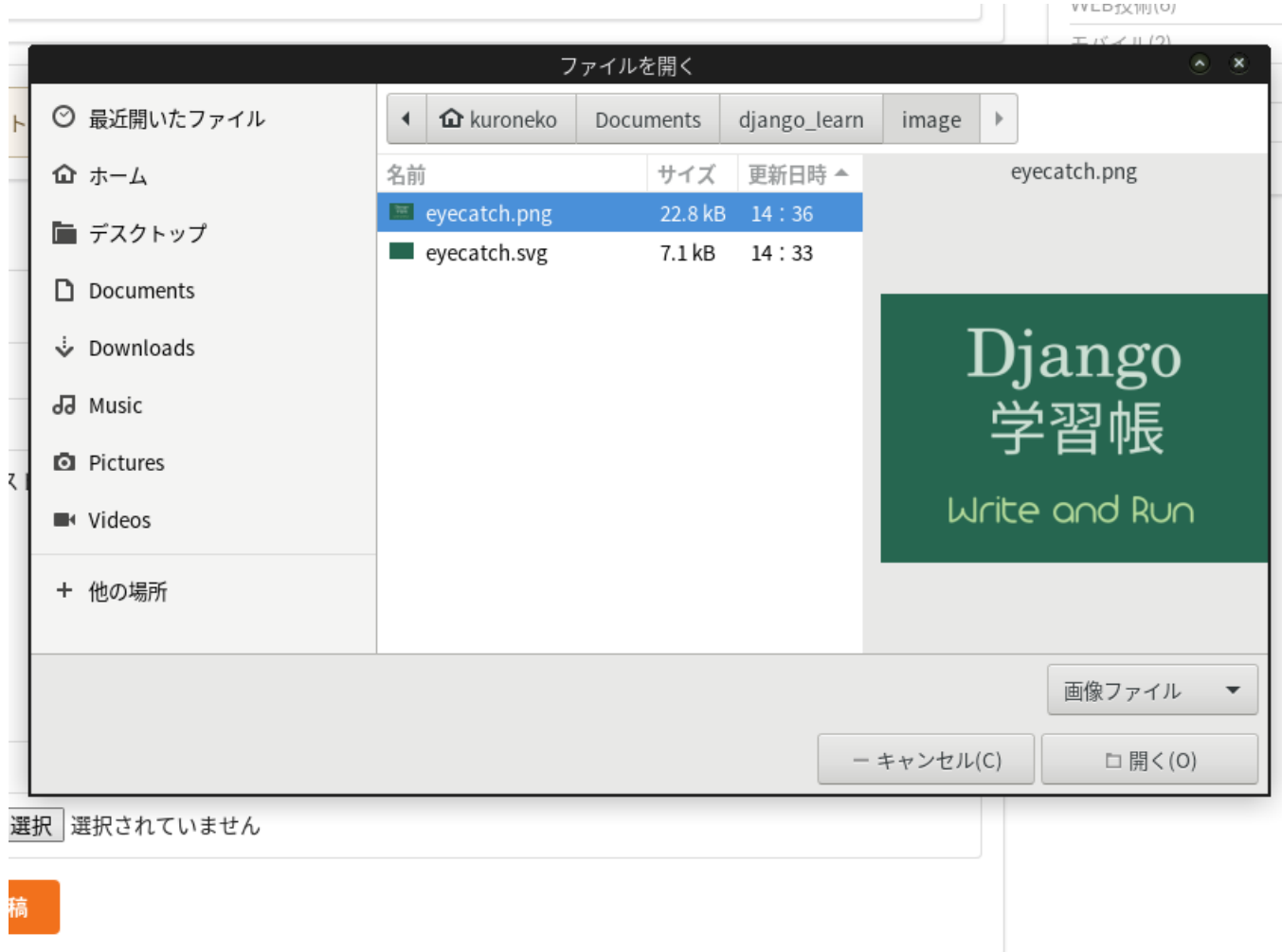
```

まず、投稿したファイルをimgタグで呼び出しています。ファイルのpathはimageのurl属性で取得することができます。MEDIA\_ROOTでしたいした'/media/'を含むpathが返されます。

注意点ですが、formタグのenctypeを"multipart/form-data"としないとファイルが送信出来ませんので忘れず修正して下さい。

実は画像をDjangoで扱う場合には上記だけでは不十分なことが多く、画像表示が重たくならないようにサムネイルやサイズごとの画像を用意する等の工夫がされることが多いのですが、サードパーティの機能の解説になることもあり、別の機会にできればと考えています。

では、コメント投稿時に画像が投稿できるか確かめてみましょう。コメント投稿欄の「投稿画像」部分のボタンを押すと画像選択用のウィンドウが開きます。（ブラウザに酔って挙動は異なります）



そのまま投稿するとコメント欄に画像が表示されます。

## 画像投稿テスト

名無し - 2019年3月25日 14:53

これは画像投稿用のトピックです

1. 名無し

2019年3月25日 14:55

画像投稿テスト



## 最後に

今回は基本的なファイルの扱いについて見てきました。実際はdjango-imagekitライブラリ等を使用することが多いと思いますが、それについては別の機会にしたいと思います。

Sponsored Link



## 2-10. Cookieへのデータの保存と読み出し

2019年4月14日 [コメントをする](#)

今回のテーマは「Cookieへのデータの保存と読み出し」です。前回セッションでのデータの扱いを見てきました。今回はCookieのデータ保存とデータの読み出しを見ていきます。

※本ページは[セッションへのデータの保存と読み出し](#)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。

Cookieにデータを保管することについて

Djangoに限らない話ですがクッキーはユーザーサイドでデータの操作が可能である点、クッキーが盗難にあう可能性があることを考慮し、セキュリティ重要なデータや個人情報などを保存してはいけません。あくまでページを跨ぐ情報の一時的な保管場所として利用するのが望ましいかと考えています。

## Cookieへ値をセットと取得

HttpResponseクラスのset\_cookieメソッドを利用してセットします。ここでは例としてトピック作成した際にカテゴリーIDを保存してみます。あまりいい例が思いつかず申し訳ないです。次回のトピック作成時には前回作成したカテゴリーが予め選択されているようにします。thread/views.pyの先程作成したTocicCreateViewBySessionクラスを修正します。

**thread/views.py**(一部抜粋)

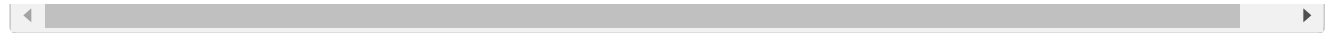
```

class TopicCreateViewBySession(FormView):
    template_name = 'thread/create_topic.html'
    form_class = TopicModelForm

    def post(self, request, *args, **kwargs):
        ctx = {}
        if request.POST.get('next', '') == 'back':
            if 'input_data' in self.request.session:
                input_data = self.request.session['input_data']
                form = TopicModelForm(input_data)
                ctx['form'] = form
            return render(request, self.template_name, ctx)
        elif request.POST.get('next', '') == 'create':
            if 'input_data' in request.session:
                Topic.objects.create_topic(
                    title=request.session['input_data']['title'],
                    user_name=request.session['input_data']['user_name'],
                    category_id=request.session['input_data']['category'],
                    message=request.session['input_data']['message']
                )
            # メール送信処理は省略
+         response = redirect(reverse_lazy('base:top'))
+         response.set_cookie('categ_id', request.session['input_data']['category'])
+         request.session.pop('input_data') # セッションに保管した情報の削除
+         return response
        elif request.POST.get('next', '') == 'confirm':
            form = TopicModelForm(request.POST)
            if form.is_valid():
                ctx = {'form': form}
                # セッションにデータを保存
                input_data = {
                    'title': form.cleaned_data['title'],
                    'user_name': form.cleaned_data['user_name'],
                    'message': form.cleaned_data['message'],
                    'category': form.cleaned_data['category'].id,
                }
                request.session['input_data'] = input_data
                ctx['category'] = form.cleaned_data['category']
                return render(request, 'thread/confirm_topic.html', ctx)
            else:
                return render(request, self.template_name, {'form': form})

+     def get_context_data(self):
+         ctx = super().get_context_data()
+         if 'categ_id' in self.request.COOKIES:
+             form = ctx['form']
+             form['category'].field.initial = self.request.COOKIES['categ_id']
+             ctx['form'] = form
+         return ctx

```



簡単に解説します。set\_cookieメソッドはHttpResponseの属性です。なのでredirect関数で生成されたHttpResponseオブジェクトresponseからset\_cookieメソッドを呼びます。引数にキーとバリューを入れて設定しています。有効期限を引数に入れることで有効期限の設定も出来ます。

セットした値を取得するにはrequest.COOKIESにアクセスすることで取得できます。今回はget\_context\_data関数の中でCookieの値を取得してカテゴリーの初期値を設定する操作を行っています。

## 最後に

ちょっと強引な例となってしまいましたが、Cookieの使用方法を紹介しました。次回はファイルを扱う方法を見ていきます。

Sponsored Link

## 2-9. セッションへのデータの保存と読み出し

2019年4月13日 [2件のコメント](#)

今回のテーマは「セッションへのデータの保存と読み出し」です。掲示板アプリもそれっぽくなってきましたね。あと少し頑張りましょう。

今回はセッションの使い方を学ぶための実装を例示しますが、学習用に用意したため掲示板の機能にはあまり影響がありません。予めご了承ください。

※本ページは[Djangoでメールを送信する](#)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。

### セッション

セッションについては[公式ドキュメント](#)も参照してください。Djangoのセッションは完全にクッキーベースでありセッションIDをURLとして渡す等の手法は用いません。また、セッションとはブラウザが起動中のみ有効でブラウザと閉じると切れるものと考えている方もいるかも知れませんが、Djangoの標準設定ではセッションはブラウザを閉じても生き続ける永続的なもので、意図的に削除されるか、有効期限（標準設定では2週間）となるまで削除されません。もちろん、ブラウザを閉じたらセッションも切れるようにも設定出来ます。

### セッションを使ったトピック作成画面

[確認画面付きのトピック作成画面を作る](#)で作成したトピック作成画面をセッションを用いる方式に修正してみましょう。トピック作成画面はユーザー作成画面→確認画面→（トピック作成処理）→TOP画面と遷移していきます。現行処理は確認画面にhiddenのフォームを入れることでユーザー入力画面の情報をトピック作成処理に渡すことにしていました。今回はセッションを使ってこの処理を実装してみます。

まずthread/views.pyに新しいクラスを作りましょう。以下のようなクラスを生成します。

**thread/views.py**（一部抜粋）

※2020/9/13一部ソースコードを修正しました。

```

class TopicCreateViewBySession(FormView):
    template_name = 'thread/create_topic.html'
    form_class = TopicModelForm

    def post(self, request, *args, **kwargs):
        ctx = {}
        if request.POST.get('next', '') == 'back':
            if 'input_data' in self.request.session:
                input_data = self.request.session['input_data']
                form = TopicModelForm(input_data)
                ctx['form'] = form
            return render(request, self.template_name, ctx)
        elif request.POST.get('next', '') == 'create':
            if 'input_data' in request.session:
                form = self.form_class(request.session['input_data'])
                form.save()
                # Topic.objects.create_topic(
                #     title=request.session['input_data']['title'],
                #     user_name=request.session['input_data']['user_name'],
                #     category_id=request.session['input_data']['category'],
                #     message=request.session['input_data']['message']
                # )
                request.session.pop('input_data') # セッションに保管した情報の削除
                # メール送信処理は省略
                return redirect(reverse_lazy('base:top'))
            elif request.POST.get('next', '') == 'confirm':
                form = TopicModelForm(request.POST)
                if form.is_valid():
                    ctx = {'form': form}
                    # セッションにデータを保存
                    input_data = {
                        'title': form.cleaned_data['title'],
                        'user_name': form.cleaned_data['user_name'],
                        'message': form.cleaned_data['message'],
                        'category': form.cleaned_data['category'].id,
                    }
                    request.session['input_data'] = input_data
                    ctx['category'] = form.cleaned_data['category']
                    return render(request, 'thread/confirm_topic.html', ctx)
                else:
                    return render(request, self.template_name, {'form': form})

```

templates/thread/confirm\_topic.html内でhiddenのインプットタグを削除します。  
**templates/thread/confirm\_topic.html**(差分のみ)

```
{% 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>
                    <p>内容を確認してください</p>
                    <table class="ui celled table table table-hover" >
                        <tr><td>タイトル</td><td>{{form.title.value}}</td></tr>
                        <tr><td>お名前</td><td>{{form.user_name.value}}</td></tr>
                        <tr><td>カテゴリー</td><td>{{category}}</td></tr>
                        <tr><td>本文</td><td><pre>{{form.message.value}}</pre></td></tr>
                    </table>
                    <form class="ui form" action="{% url 'thread:create_topic' %}" method="P
                        {% csrf_token %}
-                     {% for field in form %}
-                         {{field.as_hidden}}
-                     {% endfor %}
                        <button class="ui button grey" type="submit" name="next" value="back
                        <button class="ui button orange" type="submit" name="next" value="cr
                    </form>
                </div>
            </div>
        </div>
        {% include 'base/sidebar.html' %}
    </div>
{% endblock %}
```

**thread/urls.py**も変更します。

**thread/urls.py** (一部抜粋)

```
+ path('create_topic/', views.TopicCreateViewBySession.as_view(), name='create_topic'),
- path('create_topic/', views.TopicCreateView.as_view(), name='create_topic'),
```

これで最初に作成した確認画面付きのトピック作成画面と同等の機能をセッションを使って実装することが出来ました。セッションは基本的にはビュー内部で扱うことが多く、request.sessionに対してキーと値を持たせることで保存します。セッションエンジンにはDB, ファイル, キャッシュがありますが、今回はデフ

オルト設定されているDBを用いました。変更したい場合には公式ドキュメントの[セッションエンジンを設定する](#)を参考にして下さい。

セッションをビュー外部から使用することも出来ます。その場合は[ビューの外でセッションを使う](#)を参考にして下さい。

## 最後に

セッションは会員制のサイトなどでユーザー情報を保持しておく際によく用いられます。次回はCookieの扱いについて見ていきたいと思います。

Sponsored Link

## 2-8. Djangoでメールを送信する



今回のテーマは「Djangoでメールを送信する」です。ウェブアプリケーションではメールの送信は良くあるアクションの1つですね。今回はDjangoの機能を用いてメール送信する処理を見ていこうと思います。

公式ドキュメントの該当箇所としては[メールを送信する](#)に該当します。

※本ページは[サイトマップを作成する](#)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。

## メールバックエンドについて

メールの送信処理はメールバックエンドによって処理されます。メールバックエンドにはいくつか種類があり、目的に応じて使われます。まず、開発時に試験的に(実際にメールを送信せず)送信テストをしたい場合はコンソールバックエンドの使用をオススメします。これはメールをコンソールに出力するのみで、送信しません。またファイルバックエンドもファイルに出力するのみで実際には送信しません。メールバックエンドは独自のものを使用することもできますが多くの場合はSMTPバックエンドを使用する機会が多いと思いますので、今回は開発環境ではコンソールバックエンド、送信用にはSMTPバックエンドを使用していきます。

## メールバックエンドの設定

まずはmysite/settings.pyにて設定を行います。

**mysite/settings.py**(一部抜粋)

```
+ EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

まずは開発用にコンソールバックエンドを指定しました。ではトピックの追加された場合にメールを送信する処理を行いましょう。

**thread/views.py**(一部抜粋)

```

+ from django.core.mail import send_mail, EmailMessage

class TopicCreateView(CreateView):
    template_name = 'thread/create_topic.html'
    form_class = TopicModelForm
    model = Topic
    success_url = reverse_lazy('base:top')

    def form_valid(self, form):
        ctx = {'form': form}
        if self.request.POST.get('next', '') == 'confirm':
            ctx['category'] = form.cleaned_data['category']
            return render(self.request, 'thread/confirm_topic.html', ctx)
        elif self.request.POST.get('next', '') == 'back':
            return render(self.request, 'thread/create_topic.html', ctx)
        elif self.request.POST.get('next', '') == 'create':
+             # メール送信処理
+             send_mail(
+                 subject='トピック作成: ' + form.cleaned_data['title'],
+                 message='トピックが生成されました。',
+                 from_email='hoge@hoge@example.com',
+                 recipient_list = [
+                     'admin@example.com',
+                 ]
+             )
            return super().form_valid(form)
        else:
            # 正常動作ではここは通らない。エラーページへの遷移でも良い
            return redirect(reverse_lazy('base:top'))

```

ここで用いたsend\_mail関数はEmailMessageのラッパーで、単順なメール送信では重宝します。タイトル、本文、送信アドレス、受信アドレスを設定します。ではトピック作成してメールがコンソールに出力されるか見てみましょう。

出力例

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Subject:
=?utf-8?b?440I440U440D44Kv5L2c5oiQ0iDntKDmlbXjgarjg4vjg6Pjg7PjgrPjga7kuJY=?=
=?utf-8?b?55WM?=
From: hoge@hoge@example.com
To: admin@example.com
Date: Wed, 20 Mar 2019 07:58:17 -0000
Message-ID: <155306869746.10086.12768691481838639735@arch.localdomain>
```

トピックが生成されました。

-----

このように出力されます。

テンプレートを使う

多くのウェブサービスではメール用のテンプレートを用意しておいて変数化された部分のみ変更してメール送信する処理も多いと思います。直接のメール機能というわけではないですが、紹介しておきたいと思います。まずはテンプレートファイルを生成します。

**templates/thread/mail/topic\_mail.html**

以下のトピックが登録されました。

-----

タイトル : `{{title}}`  
ユーザー名 : `{{user_name}}`  
本文 :  
`{{message}}`

**thread/views.py**(一部抜粋)

```

+ from django.core.mail import send_mail, EmailMessage
+ from django.template.loader import get_template

class TopicCreateView(CreateView):
    template_name = 'thread/create_topic.html'
    form_class = TopicModelForm
    model = Topic
    success_url = reverse_lazy('base:top')

    def form_valid(self, form):
        ctx = {'form': form}
        if self.request.POST.get('next', '') == 'confirm':
            ctx['category'] = form.cleaned_data['category']
            return render(self.request, 'thread/confirm_topic.html', ctx)
        elif self.request.POST.get('next', '') == 'back':
            return render(self.request, 'thread/create_topic.html', ctx)
        elif self.request.POST.get('next', '') == 'create':
+             # メール送信処理
+             template = get_template('thread/mail/topic_mail.html')
+             mail_ctx={
+                 'title': form.cleaned_data['title'],
+                 'user_name': form.cleaned_data['user_name'],
+                 'message': form.cleaned_data['message'],
+             }
+             send_mail(
+                 subject='トピック作成: ' + form.cleaned_data['title'],
+                 message=template.render(mail_ctx),
+                 from_email='hoge@hoge@example.com',
+                 recipient_list = [
+                     'admin@example.com',
+                 ]
+             )
            return super().form_valid(form)
        else:
            # 正常動作ではここは通らない。エラーページへの遷移でも良い
            return redirect(reverse_lazy('base:top'))

```

このようにテンプレートのレンダリングを用いることでコンテキストをテンプレートに渡してメール本文を作成することができます。

[出力例]

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Subject:
  =?utf-8?b?440I440U440D44Kv5L2c5oiQ0iDntKDmlbXjgarjg6/jg7PjgrPjga7kuJbnlYw=?=
From: hoge@hoge@example.com
To: admin@example.com
Date: Wed, 20 Mar 2019 08:09:26 -0000
Message-ID: <155306936665.10237.4864318700173116195@arch.localdomain>
```

以下のトピックが登録されました。

```
-----
タイトル： 素敵なワンコの世界
ユーザー名： 名無し
本文：
ようこそ。ワンコの世界へ
-----
```

## EmeilMessageオブジェクトを使用してメールを送信する

冒頭で書いた通りsend\_mail関数はEmailMessageのラッパーです。CCやBCCを使う等の複雑な処理はEmailMessageオブジェクトを使用します。

**thread/views.py**(一部抜粋)

```

+ from django.core.mail import send_mail, EmailMessage

class TopicCreateView(CreateView):
    template_name = 'thread/create_topic.html'
    form_class = TopicModelForm
    model = Topic
    success_url = reverse_lazy('base:top')

    def form_valid(self, form):
        ctx = {'form': form}
        if self.request.POST.get('next', '') == 'confirm':
            ctx['category'] = form.cleaned_data['category']
            return render(self.request, 'thread/confirm_topic.html', ctx)
        elif self.request.POST.get('next', '') == 'back':
            return render(self.request, 'thread/create_topic.html', ctx)
        elif self.request.POST.get('next', '') == 'create':
+             # メール送信処理
+             template = get_template('thread/mail/topic_mail.html')
+             mail_ctx={
+                 'title': form.cleaned_data['title'],
+                 'user_name': form.cleaned_data['user_name'],
+                 'message': form.cleaned_data['message'],
+             }
+             EmailMessage(
+                 subject='トピック作成: ' + form.cleaned_data['title'],
+                 body=template.render(mail_ctx),
+                 from_email='hogehoge@example.com',
+                 to=['admin@example.com'],
+                 cc=['admin2@example.com'],
+                 bcc=['admin3@example.com'],
+             ).send()
            return super().form_valid(form)
        else:
            # 正常動作ではここは通らない。エラーページへの遷移でも良い
            return redirect(reverse_lazy('base:top'))

```

トピックを追加してみましょう。

[出力例]

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Subject:
=?utf-8?b?440I440U440D44Kv5L2c5oiQ0iDntKDmmbTjgonjgZfjgY1weXRob27jga7kuJY=?=
=?utf-8?b?55WM?=
From: hoge@hoge@example.com
To: admin@example.com
Cc: admin2@example.com
Date: Wed, 20 Mar 2019 08:29:19 -0000
Message-ID: <155307055904.10506.3851232136327226257@arch.localdomain>
```

以下のトピックが登録されました。

```
-----
タイトル： 素晴らしきpythonの世界
ユーザー名： 名無し
本文：
ようこそ。pythonの世界へ
-----
```

## SMTPバックエンドでメールを送信する

ではSMTPバックエンドによるメール送信を見ていきたいと思います。まず、メールサーバーの設定をmysite/settings.pyに追加します。今回は擬似的に[MailCatcher](#)を用いてメールの受信を体験します。MailCatcherの設定については[Debian 9にMailCatcherを導入する](#)を参考にして下さい。

**mysite/settings.py**(一部抜粋)

```
- EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
+ EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
+
+ EMAIL_USE_TLS = False
+ EMAIL_PORT = 1025
+ EMAIL_HOST = '192.168.0.10'
+ EMAIL_HOST_USER = ''
+ EMAIL_HOST_PASS = ''
```

これでトピックを追加してみます。MailCatcherにメールが送られました。

 **MailCatcher**

Search messages... Clear Quit

From	To	Subject	Received
<hoge@hoge@example.com> size=611	<admin@example.com>, <admin2@example.com>, <admin3@example.com>	トピック作成: Django学習サイトについて	Wednesday, 20 Mar 2019 5:51:49 PM

Received  
From  
To  
Subject

HTML Plain Text Source

Download

以下のトピックが登録されました。

-----

タイトル： Django学習サイトについて  
ユーザー名： 名無し  
本文：  
Django学習帳ってサイトがあるらしい

## 最後に

いかがだったでしょうか。SMTPバックエンドを使用すればGmail経由でもメールを送ることができ、個人のウェブサービスでもメールを手軽に使うことが出来ると思います。是非活用してみてください。

Sponsored Link



## 2-7. サイトマップを作成する

2019年4月11日 [コメントをする](#)

今回のテーマは「サイトマップを自動作成する」です。SEO対策として検索エンジンにウェブサイトの更新を通知するためサイトマップを用意したいケースも多いと思います。今回はDjangoを用いてサイトマップを動的に生成する方法を見ていきます。

今回の内容は公式ドキュメントでは[The sitemap framework](#)の部分に相当します。

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

## 準備

まずsettings.pyを変更する必要があります。

### mysite/settings.py

```
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',  
    'api',  
    'search',  
]  
  
+ SITE_ID = 1
```

次にデータベースのマイグレーションをします。

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

管理画面にアクセスしてドメイン名と表示名を変更します。今回はmydomain.comに修正しました。このサイトオブジェクトのIDが先程settings.pyで設定したSITE\_IDと一致している必要があります。

ホーム > サイト > サイト

変更する サイト を選択

サイ



検索

操作:

実行

1個の内ひとつも選択されていません



ドメイン名



表示名



mydomain.com

mysite

1 サイト

## Sitemap継承クラスを作る

ではサイトマップを作っていきます。DjangoにおいてサイトマップはSitemapクラスを継承したクラスにて作成します。thread/sitemaps.pyを作成します。尚、sitemaps.pyはDjangoで決まっているファイルではありませんので別の名前でもいいですし、他のファイルに書いても構いません。

### thread/sitemaps.py

```
from django.contrib.sitemaps import Sitemap
from django.shortcuts import resolve_url

from . models import Topic, Category

class TopicSitemap(Sitemap):
    priority = 0.5
    changefreq = 'always'

    def items(self):
        return Topic.objects.all()

    def location(self, obj):
        return resolve_url('thread:topic', pk=obj.id)

class CategorySitemap(Sitemap):
    priority = 0.5
    changefreq = 'never'

    def items(self):
        return Category.objects.all()

    def location(self, obj):
        return resolve_url('thread:category', url_code=obj.url_code)
```

Sitemapクラスを継承したクラスを作成しています。このクラスではサイトマップを作成に必要な情報をクラスの属性として与えます。この際、静的な情報はクラス変数、動的な情報は関数を使って指定出来るようになっています。

指定できる属性は以下です。

items	[必須]ページのオブジェクト。イテレータブルなオブジェクトを指定する
location	[オプション]itemsのURL
lastmod	[オプション]ページ更新日時
changefreq	[オプション]ページの更新頻度
priority	[オプション]ページの重要度 0 ～ 1

limit	[オプション]ページネーションの区切り。デフォルトは50000
protocol	[オプション]httpプロトコル。デフォルトはhttp
i18n	[オプション]URL表示に設定言語を適用するか True/False

まず、items()メソッドでトピックのリストを与えています。この関数が返すイテレータブルなitemsの1つ1つに対してサイトマップが生成されます。ではこのitemsの要素であるitemのURLのを指定するためにlocation関数でURLを指定しましょう。トピックのURLは動的に変更するのでlocationは関数で与えます。このようにitemsで指定したリストの1要素を仮引数とする関数を作ることで動的なサイトマップを生成できます。

カテゴリー用のサイトマップも同様です。CategorySitemapクラスを作成して要素を指定しています。カテゴリーの場合はurl\_codeを指定することに注意して下さい。

では、次に静的なページである利用規約やプライバシーポリシーについて見ていきましょう。これらのページは動的要素がないために非常に簡単です。base/sitemaps.pyを生成しましょう。

### base/sitemaps.py

```
from django.contrib.sitemaps import Sitemap
from django.shortcuts import resolve_url

class BaseSitemap(Sitemap):

    def items(self):
        items = [
            'base:top',
            'base:policy',
            'base:terms',
        ]
        return items

    def location(self, obj):
        return resolve_url(obj)

    def changefreq(self, obj):
        if obj == 'base:top':
            return 'always'
        return 'never'

    def priority(self, obj):
        if obj == 'base:top':
            return 0.8
        return 0.1
```

基本的な考え方はthread/sitemaps.pyと同じです。ただitemsで'base:policy'のようにページのショートカット名でリストを生成することで効率的にサイトマップを生成することができます。上記でクラス変数として指定した属性についても関数で指定しています。両者を比べると理解がより進むのではないのでしょうか。

## URLの設定

では、作成したサイトマップを表示してみましょう。ここから先はmysite/urls.pyを変更していきます。

### mysite/urls.py

```
from django.contrib import admin, auth
from django.urls import path, include
from django.conf import settings
from django.contrib.sitemaps.views import sitemap

+ from thread.sitemaps import TopicSitemap, CategorySitemap
+ from base.sitemaps import BaseSitemap

+ sitemaps = {
+     'topic': TopicSitemap,
+     'cateogry': CategorySitemap,
+     'base': BaseSitemap,
+ }

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('django.contrib.auth.urls')),
    path('', include('base.urls')),
    path('thread/', include('thread.urls')),
    path('api/', include('api.urls')),
    path('search/', include('search.urls')),
+    path('sitemap.xml', sitemap, {'sitemaps': sitemaps}),
]

if settings.DEBUG:
    import debug_toolbar
    urlpatterns = [
        path('__debug__/', include(debug_toolbar.urls)),
    ] + urlpatterns
```

このように各アプリケーションでSitemap継承クラスを作成し、urls.pyでまとめるというのはDjangoのルールではないですが、分かりやすいので筆者は気に入っています。

では確認してみましょう。localhost:8080/sitemap.xmlにアクセスします。

This XML file does not appear to have any style information associated with it. The document 1

---

```
▼<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  ▼<url>
    <loc>http://mydomain.com/thread/1/</loc>
    <changefreq>always</changefreq>
    <priority>0.5</priority>
  </url>
  ▼<url>
    <loc>http://mydomain.com/thread/2/</loc>
    <changefreq>always</changefreq>
    <priority>0.5</priority>
  </url>
  ▼<url>
    <loc>http://mydomain.com/thread/3/</loc>
    <changefreq>always</changefreq>
    <priority>0.5</priority>
  </url>
  ▼<url>
    <loc>http://mydomain.com/thread/4/</loc>
    <changefreq>always</changefreq>
    <priority>0.5</priority>
  </url>
  ▼<url>
    <loc>http://mydomain.com/thread/5/</loc>
```

## 最後に

Djangoのサイトマップ生成機能を使うと簡単に動的なサイトマップを作成することができますね。今回は基礎的な部分に着目したので、テンプレートをカスタマイズしたいという場合には公式ドキュメントを呼んでみて下さい。

Sponsored Link

## 2-6. ページネーションを使う

2019年4月10日 [コメントをする](#)

今回のテーマは「ページネーションを使う」です。今回はWEBアプリでつきもののページネーションについて触れていきます。Djangoでは標準で手軽に使えるページネーション機能が組み込まれています。

※本ページは[検索画面を作る](#)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。

クラスベースビューでページネーションを使う

ページネーションは関数ベースでもクラスベースビューでも使えるのですが、クラスベースビューで使いたほうが楽なので、まずこちらから紹介します。手軽なのはListViewを継承する方法です。ちょうど、thread/views.pyのCategoryViewクラスがListViewを継承したクラスですのでこれにページネーションを適用することを考えてみましょう。

まずページネーションを表示するためのテンプレートを用意しましょう。  
templates/base/pagination.htmlを作成します。

### templates/base/pagination.html

```
{% if is_paginated %}
<div class="ui basic segment center aligned">
  <div class="ui pagination menu">
    <!-- 左矢印 -->
    {% if page_obj.has_previous %}

      <a class="item" href="?p={{page_obj.previous_page_number}}"><i class="chevron left"
    {% else %}
      <a class="disabled item"><i class="chevron left icon"></i></a>
    {% endif %}
    <!-- // 左矢印 -->
    <!-- ページ番号 -->
    {% for link_page in page_obj.paginator.page_range %}
      {% if link_page == page_obj.number %}
        <a class="disabled item">{{link_page}}</a>
      {% else %}
        <a class="item" href="?p={{link_page}}">{{link_page}}</a>
      {% endif %}
    {% endfor %}
    <!-- // ページ番号 -->
    <!-- 右矢印 -->
    {% if page_obj.has_next %}
      <a class="item" href="?p={{page_obj.next_page_number}}"><i class="chevron right ic
    {% else %}
      <a class="disabled item"><i class="chevron right icon"></i></a>
    {% endif %}
    <!-- // 右矢印 -->
  </div>
</div>
{% endif %}
```

テンプレートに渡されたpage\_objというパラメータで組み立てていきます。page\_objはその名のとおりPageオブジェクトです。pageオブジェクトはpaginatorから生成されますが、ページングに関する様々な情報を備えています。詳細は公式ドキュメントの[ページネーション](#)を一読することをオススメします。pageオブジェクトはインスタンス変数としてページ番号、paginatorオブジェクト、オブジェクトリスト



を保有しており、この情報を基に、次にページがあるか、ないかなど情報を取得するインターフェースを備えています。今回はpageオブジェクトのインスタンス変数であるpagenateオブジェクトのpage\_rangeを呼び出してループしています。

※実はListViewを使用した場合はpaginatorオブジェクトは別に渡されているので、そちらを使っても構わないのですが、次に扱う関数ベースのビューの場合のため今回はpageオブジェクトから情報を全て取得します。

この段階ではまだ良くわからないと思います。後半の関数ベースでの処理まで見て再度見直すとテンプレートの意味が何となく見えてくるとと思います。

次にこのpagination.htmlをcategory.htmlでインクルードしましょう。

#### templates/thread/category.html(一部抜粋)

```
<div class="ui segment">
  <div class="content">
    <div class="header"><h3>{{category.name}}</h3></div>
+    {% include 'base/pagination.html' %}
    <div class="ui divided items">
      {% if topic_list %}
      {% for topic in topic_list %}
```

単順に見出しの下にインクルード下だけです。

ではthread/views.pyのCategoryViewを修正しましょう。

#### thread/views.py(一部抜粋)

```
class CategoryView(ListView):
    template_name = 'thread/category.html'
    context_object_name = 'topic_list'
+    paginate_by = 1 # 1ページに表示するオブジェクト数 サンプルのため1にしています。
+    page_kwarg = 'p' # GETでページ数を受けるパラメータ名。指定しないと'page'がデフォルト

    def get_queryset(self):
        return Topic.objects.filter(category__url_code = self.kwargs['url_code'])

    def get_context_data(self):
        ctx = super().get_context_data()
        ctx['category'] = get_object_or_404(Category, url_code=self.kwargs['url_code'])
        return ctx
```

修正はこれだけです。実はListViewはBaseListViewを継承したクラスなのですが、このクラスはMultipleObjectMixinという複数のオブジェクトを表示する機能を持ったクラスを継承しており、MultipleObjectMixinがページネーション機能を有しているためクラス変数を指定するだけで使えたので

す。尚、`paginate_by`には1ページに表示するオブジェクトの数、`page_kwargs`はGETで受けるページのパラメータ名でありデフォルトは'page'です。自作のクラスベースビューにListViewのようなページネーション機能を持たせる場合にはMultipleObjectMixinを継承させると機能を付与することができます。

では、表示して見ましょう。



## 関数ベースのビューでページネーションを使う

クラスベースビューでは予め用意されたクラス変数をオーバーライドすれば良かったので楽でした。ただし何が行われているのかが見えづらくて分かりづらい部分もあったと思います。そこで関数ベースのビューで同じ挙動をするビューを作成してみることにします。先程はMultipleObjectMixinが自動で行っていた部分を自分で書いていきます。

`thread/views.py`(一部抜粋)

```

from django.core.paginator import Paginator, PageNotAnInteger, EmptyPage
def show_catgegory(request, url_code):
    if request.method == 'GET':
        page_num = request.GET.get('p', 1)
        pagenator = Paginator(
            Topic.objects.filter(category__url_code=url_code),
            1 # 1ページに表示するオブジェクト数
        )
        try:
            page = pagenator.page(page_num)
        except PageNotAnInteger:
            page = pagenator.page(1)
        except EmptyPage:
            page = pagenator.page(pagenator.num_pages)

        ctx = {
            'category': get_object_or_404(Category, url_code=url_code),
            'page_obj': page,
            'topic_list': page.object_list, # pageでもOK
            'is_paginated': page.has_other_pages,
        }
        return render(request, 'thread/category.html', ctx)

```

```

urlpatterns = [
    path('create_topic/', views.TopicCreateView.as_view(), name='create_topic'),
    path('/', views.TopicViewAndCommentCreateView.as_view(), name='topic'),
-   path('category//', views.CategoryView.as_view(), name='category'),
+   path('category//', views.show_catgegory, name='category'),
]

```

thread/urls.pyも修正します。

見た目は全く変わらないので画像は省略します。恐らくこちらの方が分かりやすいという方が多いと思います。先程見たListViewの継承クラスではpage\_obj, is\_paginatedは自動で渡されていたのです。また、topic\_listも区切りの数字に併せて調整されていましたが、これもpageのobject\_listを渡すことで対応しています。

## 最後に

クラスベースビューはコード量が少なく自動でいろいろやってくれる分、フレームワークで何をやっているのかが分かりづらい面もあります。時にはDjangoのソースを追って関数ベースビューで書き直してみても勉強にもなるんじゃないかな・・・と思っていますが忙しい方には大変ですね。このサイトがお役に立てるよう頑張ります。次は心機一転でサイトマップを作ってみます。

Sponsored Link

## 2-5. 検索画面を作る

2019年4月8日 [1件のコメント](#)

今回のテーマは「検索画面を作る」です。今回はDjangoの便利機能という訳ではないんですが、クエリセットの扱いについて紹介できればと思います。

※本ページは[DjangoのAPIとAjax通信する「いいねボタン」を作成する](#)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。

## 検索アプリケーションの作成

まずは検索を担当するアプリケーションを追加しましょう。Djangoは機能別のアプリケーションを集めてWebアプリケーションを構築します。もちろん、threadアプリケーション内に検索機能を持たせることもできますが、将来別のアプリケーションが追加された場合にはサイト全体の検索を誰が担うのか所在がばやける恐れがあります。検索を専門に行うsearchアプリケーションを作ることになります。

```
(venv)$ ./manage.py startapp search
```

アプリケーションを追加したら、いつもどおりmysite/settings.pyとmysite/urls.pyを変更します。

### mysite/settings.py

```
(venv)$ ./manage.py startapp search
```

### mysite/urls.py

```
(venv)$ ./manage.py startapp search
```

## テンプレートの作成

ではテンプレートを作っていきます。検索結果を表示するテンプレートはこのようになります。

### templates/search/result.html

```

{% extends 'base/base.html' %}
{% block title %}検索結果 - {{ 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">検索結果</a>
    </div>
    <div class="ui segment">
      <div class="content">
        <div class="header"><h3>検索結果</h3></div>
        <form action="{% url 'search:result' %}" method="GET">
          <div class="ui action input" style="width: 100%;">
            <input type="text" placeholder="検索" value="{{query}}" name="q">
            <button class="ui button"><i class="search icon"></i></button>
          </div>
        </form>
        {% if result_list %}
        {% for result in result_list %}
        <div class="ui segment message">
          <h3><a href="{% url 'thread:topic' pk=result.id %}">{{result.title}}</a>
          <p>{{result.message | truncatewords:30}}</p>
        </div>
        {% endfor %}
        {% else %}
        <div class="ui segment warning message">
          <p>検索結果はありません。</p>
        </div>
        {% endif %}
      </div>
    </div>
  </div>
  {% include 'base/sidebar.html' %}
</div>
{% endblock %}

```

特に問題ないと思います。今回はGETでアクセスするのでcsrfについては気にしなくてOKです。

## ビューの作成

次にビューを作ります。ビューにはthreadアプリケーションのモデルをインポートします。

**search/views.py**

```

from django.shortcuts import render
from django.db.models import Q
from django.views.generic import ListView
from functools import reduce
from operator import and_
from thread.models import Topic

class SearchResultView(ListView):
    template_name = 'search/result.html'
    context_object_name = 'result_list'

    def get_queryset(self):
        if self.request.GET.get('q', ''):
            params = self.parse_search_params(self.request.GET['q'])
            query = reduce(
                lambda x,y : x & y,
                list(map(lambda z: Q(title__icontains=z) | Q(message__icontains=z), params
                )
            # 下記でもOK
            # query = reduce(and_, [Q(title__icontains=p) | Q(message__icontains=p) for p
            return Topic.objects.filter(query)
        else:
            return None

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        ctx['query'] = self.request.GET.get('q', '')
        return ctx

    def parse_search_params(self, words: str):
        search_words = words.replace(' ', ' ').split()
        return search_words

```

今回はQオブジェクトが出てきました。Djangoでは複雑なクエリセットを組み立てる時にはQオブジェクトを使用します。Qオブジェクトを使用すると"|"演算子でOR条件、"&"演算子でAND条件を表現できます。Qオブジェクトはビット演算子によって新たなQオブジェクトを生成します。よって `Q(title__icontains=z)|Q(message__icontains=z)` は1つのQオブジェクトとなります。icontainsでは大文字、小文字を区別せず検索します。

それを念頭に置いた上で上記のソースコードを見ると分かりやすいと思います。見やすいようにlambdaとmapで書きましたが、and\_関数とリスト内包表記でも同じことができます。

このビューにアクセスするURLを作成します。

**search/urls.py**

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

app_name = 'search'

urlpatterns = [
    path('', views.SearchResultView.as_view(), name='result'),
]
```

## サイドバーの修正

次にサイドバーの検索バーから検索できるようにtemplates/base/sidebar.htmlを修正しましょう。

### templates/base/sidebar.html

```
{% load threadtags %}
<div class="five wide column">
    <form action="{% url 'search:result' %}" method="GET">
        <div class="ui action input" style="width: 100%;">
            <input type="text" placeholder="検索" name="q">
            <button type="submit" class="ui button"><i class="search icon"></i></button>
        </div>
    </form>
    <div class="ui items">
        <div class="item">
            <a href="{% url 'thread:create_topic' %}" class="ui fluid teal button">トピック
        </div>
    </div>
    <div class="ui segment">
        <div class="content">
            <div class="header"><h4>カテゴリー</h4></div>
            {% category_tag %}
        </div>
    </div>
</div>
```

では確認してみましょう。 サイドバーの検索窓から検索を行って検索結果画面に検索結果が表示されればOKです。



検索窓にキーワード入力

ログイン

ユーザー登録

ネコ

Q

トピックを作成

カテゴリー

WEB技術(7)

モバイル(2)

DjDT

検索結果画面

TOP > 検索結果

検索結果

ネコ

Q

素晴らしいネコの世界

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

最後に

今回のポイントはQオブジェクトの使い方ですね。ビット演算子で連結することで複雑なクエリを表現できますので、活用してみてください。次回はページネーションを扱っていきます。だんだん掲示板っぽくなってきましたね。

Sponsored Link

## 2-4. DjangoのAPIとAjax通信する「いいねボタン」を作成する

2019年4月6日 [2件のコメント](#)

今回のテーマは「DjangoのAPIとAjax通信する「いいねボタン」を作成する」です。今回はDjangoで作られたAPIにjavascriptでAjax通信する処理を見ていきたいと思います。APIの設計等は本題から外れるためお粗末だとは思いますが、気になる点があればコメントいただければと思います。

※本ページは[テンプレートのフィルターを使う](#)まで読まれた方を対象としています。そのためサンプルソースコードが省略されている場合があります。

## いいねボタンを作るための準備

いいねボタンがどんなものかは説明不要だと思いますが、想定を簡単に説明しておきます。ユーザーがいいねボタンを押すと1POINT加点されリアルタイムで数字が変化します。連打防止はjavascriptおよびIPアドレスで判定します。

これを実装するにあたりvoteモデルを用意します。

**thread/models.py**(一部抜粋)

```
class VoteManager(models.Manager):
    def create_vote(self, ip_address, comment_id):
        vote = self.model(
            ip_address=ip_address,
            comment_id = comment_id
        )
        try:
            vote.save()
        except:
            return False
        return True

class Vote(models.Model):
    comment = models.ForeignKey(
        Comment,
        on_delete=models.CASCADE,
        null=True,
    )
    ip_address = models.CharField(
        'IPアドレス',
        max_length=50,
    )

    objects = VoteManager()

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

ここはモデル作成の部分で説明済みですので特に問題ないと思います。モデルの準備が出来たらマイグレーションをしましょう。もう大丈夫ですね。

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

voteの数を数えるようにthread/views.pyを変更しましょう。

**thread/views.py**(一部抜粋)

```

class TopicAndCommentView(FormView):
    template_name = 'thread/detail_topic.html'
    form_class = CommentModelForm

    def form_valid(self, form):
        # comment = form.save(commit=False)
        # comment.topic = Topic.objects.get(id=self.kwargs['pk'])
        # comment.no = Comment.objects.filter(topic=self.kwargs['pk']).count() + 1
        # comment.save()
        Comment.objects.create_comment(
            user_name=form.cleaned_data['user_name'],
            message=form.cleaned_data['message'],
            topic_id=self.kwargs['pk'],
        )
        return super().form_valid(form)

    def get_success_url(self):
        return reverse_lazy('thread:topic', kwargs={'pk': self.kwargs['pk']})

    def get_context_data(self):
        ctx = super().get_context_data()
        ctx['topic'] = Topic.objects.get(id=self.kwargs['pk'])
-       ctx['comment_list'] = Comment.objects.filter(
-           topic_id=self.kwargs['pk']).order_by('no')
+       ctx['comment_list'] = Comment.objects.filter(
+           topic_id=self.kwargs['pk']).annotate(vote_count=Count('vote')).order_by(
        return ctx

```

特に説明は不要かと思います。annotateで各コメントが参照しているVoteモデルの数を数えて追加しています。このvote\_countが投票されたポイントとなります。

## テンプレート側の準備

いいねボタンのUIを準備しましょう。templates/thread/detail\_topic.htmlを修正します。

**templates/thread/detail\_topic.html** (一部抜粋)

```

    {% if comment.pub_flg %}
+ <p>{{comment.message | comment_filter | safe}}</p>
+ <div class="ui right aligned vertical segment">
+     <div class="vote_button" style="cursor: pointer;"
+         data-comment-id="{{comment.id}}" data-count="{{comment.vote_count}}">
+         <i class="heart outline icon"></i>
+         <span class="vote_counter">
+             {% if comment.vote_count > 0 %}{{comment.vote_count}}{% endif %}
+         </span>
+     </div>
+ </div>
+ {% else %}
+ <p style="color: grey">コメントは非表示とされました</p>
    {% endif %}

```

こんなハートマークが表示されると思います。

1. 名無し  
2019年3月14日 15:40

テストテスト  
テスト



## APIの作成

さて、これで事前準備ができました。API用のアプリケーションを用意しましょう。

```
(venv)$ ./manage.py startapp api
```

アプリケーションを追加したのでmysite/settings.py, mysite/urls.pyをいつもどおり設定します。

mysite/settings.py

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

mysite/uls.py

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('accounts/', include('django.contrib.auth.urls')),  
    path('', include('base.urls')),  
    path('thread/', include('thread.urls')),  
+    path('api/', include('api.urls')),  
]
```

ここまでは問題ないですね。ではAPIを作成していきましょう。

api/views.pyを以下のようにします。

**api/views.py**

```

from django.views.generic import View
from django.http import JsonResponse

from thread.models import Vote

class CreateVoteView(View):
    '''
    いいね投票作成処理を行う
    '''
    def post(self, request, *args, **kwargs):
        res = {
            'result': False,
            'message': '処理に失敗しました。'
        }
        # POST値に'comment_id'がなければBAD REQUESTとする
        if not 'comment_id' in request.POST:
            return JsonResponse(res, status=400)

        # コメントIDとIPアドレスの取得
        comment_id = request.POST['comment_id']
        ip_address = get_client_ip(request)

        # 既にIP登録があればコンフリクト
        if Vote.objects.filter(comment_id=comment_id, ip_address=ip_address):
            res['message'] = '投票済みです'
            return JsonResponse(res, status=409)

        # Voteの保存に成功した場合のみ成功
        if Vote.objects.create_vote(ip_address, comment_id):
            res['result'] = True
            res['message'] = 'ポイント追加しました'
            return JsonResponse(res, status=201)
        else:
            return JsonResponse(res, status=500)

    def get_client_ip(request):
        '''
        IPアドレスを取得する
        '''
        x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
        if x_forwarded_for:
            ip = x_forwarded_for.split(',')[0]
        else:
            ip = request.META.get('REMOTE_ADDR')
        return ip

```

ソースを追っていただければ処理自体は非常に簡単だと思います。コメントIDとIPアドレスをセットで格納し、新たな投票時にはIPアドレスによって重複投票を防いでいます。今回はViewを継承したクラスビューを用意しました。関数タイプで書いても問題ないですよ。今回はテンプレートがないですが、ユー

ザーに見せるべき情報を用意するというビューの役割は変わりません。

JSON形式でレスポンスを返すために JsonResponse を用いています。JsonResponse は辞書型の変数を引数として JSON 形式にエンコードしたボディを返します。また、HTTP ステータスコードを status 引数で渡しています。API 設計の拙い点は目をつぶっていただければと思います。また、一般的に、HTTP レスポンスコード 201 の場合はヘッダに作成したオブジェクトを示す URL を入れることが推奨されていますが、今回は投票ということで結果のみを返しています。

このビューにアクセスするための URL を設定します。api/urls.py を生成します。

### api/urls.py

```
from django.urls import path
from . import views
# from django.views.decorators.csrf import csrf_exempt

app_name = 'api'

urlpatterns = [
    path('v1/vote/', views.CreateVoteView.as_view(), name='create_vote'),
]
```

これで API 側の準備はできました。

javascript の実装を見てみましょう。今回は jQuery フレームワークを使用して実装します。使いたくない方は生の javascript でも問題ありません。static/js/vote.js を用意しましょう。

### static/js/vote.js



```

$(function(){
    // setup for ajax
    var csrftoken = getCookie('csrftoken');
    $.ajaxSetup({
        beforeSend: function(xhr, settings) {
            if (!csrfSafeMethod(settings.type) && !this.crossDomain) {
                xhr.setRequestHeader("X-CSRFToken", csrftoken);
            }
        }
    });

    var votedList = []; // 連打防止用のコメントID格納リスト
    // いいねボタン押下時の処理
    onClickVoteButton();

    function getCookie(name) {
        var cookieValue = null;
        if (document.cookie && document.cookie !== '') {
            var cookies = document.cookie.split(';');
            for (var i = 0; i < cookies.length; i++) {
                var cookie = jQuery.trim(cookies[i]);
                // Does this cookie string begin with the name we want?
                if (cookie.substring(0, name.length + 1) === (name + '=')) {
                    cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
                    break;
                }
            }
        }
        return cookieValue;
    }

    function csrfSafeMethod(method) {
        // these HTTP methods do not require CSRF protection
        return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
    }

    function onClickVoteButton() {
        $('.vote_button').on('click', function() {
            var commentId = $(this).data('comment-id');
            var currentCount = $(this).data('count');
            var countViewer = $(this).find('.vote_counter');
            if (votedList.indexOf(commentId) < 0) {
                vote(commentId, currentCount, countViewer);
            }
        });
    }

    // ajax通信して投票結果を反映する
    function vote(commentId, currentCount, countViewer) {

```

```

let url = '/api/v1/vote/';
$.ajax({
  type: 'POST',
  url: url,
  data: {
    comment_id: commentId
  }
}).then(
  data => {
    if (data.result) {
      countViewer.text(currentCount + 1);
      votedList.push(commentId);
    }
  },
  error => {
    if (error.responseJSON.message) {
      alert(error.responseJSON.message);
    }
  }
);
});

```

POSTメソッドでAPIにアクセスする場合は注意が必要です。DjangoのPOSTメソッドにはクロススク립トフォージェリ対策が必要であるためにキャッシュからトークンを取得してヘッダにセットする処理をしています。これさえ注意すれば後は特に問題ありません。Ajaxの通信結果に応じてUIに変更を加える処理をしています。尚、GETでアクセスする場合にはCSRFの認証を気にする必要はありません。

ではこのvote.jsが読み込まれるようにtemplates/thread/detail\_topic.htmlに追加しましょう。

**templates/thread/detail\_topic.html**(一部抜粋)

```

<--!ファイルの末尾に追加-->
{% block js %}
<script src="{% static 'js/vote.js' %}" type='text/javascript'></script>
{% endblock %}

```

これで、vote.jsが読み込まれるようになったはずです。ではブラウザで確認してみましょう。ハートマークを押してポイントがプラス1されればOKです。同一コメントに対しては同じIPからは一回しか投票できないはずです。

1. 名無し

2019年3月14日 15:40

テストテスト  
テスト

♡ 1

## 最後に

単にAjax通信を説明する割には大げさなサンプルになってしまいました。Ajax通信だけでなく、JSONレスポンスを返す場合の処理についても紹介できたのではないかと思います。尚、csrfを解除してPOSTしたいという要望もあると思います。これに関しては別の機会に扱おうと思います。

Sponsored Link

## 2-3. テンプレートのフィルターを使う

2019年4月5日 [2件のコメント](#)

今回のテーマは「テンプレートのフィルターを使う」です。前回テンプレートタグを作成しました。今回はテンプレート上で用いる独自フィルターを使用していきます。

※本ページは[テンプレートタグを使ってサイドバーを作成する](#)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。

### Djangoテンプレートのフィルター

Djangoのテンプレートには他のフレームワーク同様にテンプレート上でフィルター処理を行えます。そのため、予め組み込まれた組み込みフィルターが用意されています。組み込みフィルターの種類については公式ドキュメントの[組み込みタグとフィルタ](#)をご覧ください。例えば、truncatecharsという組み込みフィルターがありますが、これは`{{ context.hoge | truncatechars:10 }}`と書くと10文字に切り詰めて表示するという機能を持つフィルターです。

### カスタムフィルターを作る

ではカスタムフィルターを作成していきましょう。今回はコメントにURLが記載された場合リンクとして表示するフィルターを作成しましょう。thread/templatetags/threadfilters.pyを作成します。

**thread/templatetags/threadfilters.py**

```
from django.template import Library

register = Library()

@register.filter
def comment_filter(text):
    return ''.join(list(map(convert_url, text.split('\n'))))

def convert_url(text_line):
    """
    URLリンク行をaタグ付きの行に変換
    """
    if 'https://' in text_line or 'http://' in text_line:
        return '<a href="' + text_line + '" target="_blank" rel="noopener noreferrer">' +
    else:
        return text_line
```

register.filterメソッドは引数nameを省略すると関数の名前がフィルター名として適用されます。今回はcomment\_filterですね。前回のテンプレートタグと同様にthreadfiltersをロードしてcomment\_filterを適用することにします。

**templates/thread/detail\_topic.html**

```

{% extends 'base/base.html' %}
{% block title %}トピック作成 - {{ block.super }}{% endblock %}
{% block content %}
{% load threadfilters %}
<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 href="{% url 'thread:category' url_code=topic.category.url_code %}" class="
      <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 segment">
          <p><pre>{{topic.message}}</pre></p>
        </div>
      </div>
    </div>
    <!--コメント表示-->
    <div class="ui segment">
      {% if comment_list %}
      {% for comment in comment_list %}
      <div class="ui segment secondary">
        <p>{{comment.no}}. {{comment.user_name}}<br>{{comment.created}}</p>
        {% if comment.pub_flg %}
        <p>{{comment.message | comment_filter}}</p>
        {% else %}
        <p style="color: #aaa">このコメントは非表示となりました。</p>
        {% endif %}
      </div>
      {% endfor %}
      {% else %}
      <div class="ui warning message"><p>まだコメントはありません</p></div>
      {% endif %}
    </div>
    <!--//コメント表示-->
    <!--コメント投稿-->
    <h4>コメント投稿</h4>
    <div class="ui segment">
      <form class="ui form" action="" method="POST">
        {% csrf_token %}
        {{form.as_p}}
        <button class="ui button orange" type="submit">コメント投稿</button>
      </form>

```

```
</div>
<!--//コメント投稿-->
</div>
{% include 'base/sidebar.html' %}
</div>
{% endblock %}
```

このようになります。{% load threadfilters %}でフィルターを読み込んでいます。  
{{comment.message | comment\_filter}}の部分でコンテキストで渡された文字列を変換しています。  
ではブラウザで確認してみましょう。

URLを含んだ投稿をします。（今回のケースではURL行は改行される必要があります。）

## コメント投稿

お名前

名無し

投稿内容

<https://django.kurodigi.com>  
Django学習帳

コメント投稿

変換された結果

5. 名無し  
2019年3月8日10:01

サンプル<br>サンプル

6. 名無し  
2019年3月8日10:12

サンプル<br>サンプル

7. 名無し  
2019年3月13日10:15

<a href="https://django.kurodigi.com" target="\_blank">https://django.kurodigi.com</a>  
<br>Django学習帳

あれ？ちょっとおかしい結果になりましたね。原因を考えていきましょう。

## Django テンプレート エスケープ処理

確かに文字列は変換されていますが、HTMLタグがそのまま表示されてしまいました。これはどういうことでしょうか？実はDjangoテンプレートはデフォルトでエスケープ機能がONになっていて、コンテキストで渡された文字列はエスケープ処理が施されて出力されることになっています。PHPでいうとhtmlspecialchars関数が適用されたような状態です。これにより万が一コンテキストに悪意のあるスクリプトが渡されたとしても実行を防ぐ作用をしています。（ただし過信は禁物です。）

今回のようにフィルターでHTMLを生成する場合にはエスケープ機能をOFFにする必要があります。しかし、この機能を外す際にはプログラマーは十分に注意する必要があります。今回はbleachライブラリを使用してエスケープ処理をした後にHTML変換を書けることで対応してみます。

まずはbleachをインストールしましょう。

```
(venv)$ pip install bleach
```

次にthread/templatetags/threadfilters.pyを下記のように書き換えます。

**thread/templatetags/threadfilters.py**



```
from django.template import Library

import bleach

register = Library()

@register.filter
def comment_filter(text):
    return ''.join(list(map(convert_url, bleach.clean(text).split('\n'))))

def convert_url(text_line):
    '''
    URLリンク行をaタグ付きの行に変換
    '''
    if 'https://' in text_line or 'http://' in text_line:
        return '<a href="' + text_line + '" target="_blank" rel="noopener noreferrer">'
    else:
        return text_line
```

先で説明した通り、テンプレートの機能でエスケープ処理が行われないために、関数内部でエスケープ処理をしています。ではテンプレートを修正しましょう。

**templates/thread/detail\_topic.html**(一部抜粋)

```
- <p>{{comment.message | comment_filter}}</p>
+ <p>{{comment.message | comment_filter | safe}}</p>
```

このようにsafeフィルターを付けることでテンプレートのエスケープ処理が外れます。これはプログラマが'safe'な文字列であることを保証するという意味です。その意味を考えたら気軽には使えませんよね。今回はbleach関数で手軽に事前エスケープで対応しましたが、実際はもっと気を配るケースが多いと思います。

ではブラウザでどのように表示されるかを確認してみましょう。URLリンクが想定通り出力されたでしょうか？

5. 名無し  
2019年3月8日10:01

サンプル  
サンプル

6. 名無し  
2019年3月8日10:12

サンプル  
サンプル

7. 名無し  
2019年3月13日10:15

<https://django.kurodigi.com>  
Django学習帳

## 最後に

今回はフィルターのカスタマイズとDjangoテンプレートのエスケープ機能を見てきました。ネットで調べ物をしているとセキュリティに無頓着でユーザーの入力によってスクリプトが実行できてしまうような実装をしているサンプルも目にします。よく考えずに参考にとすると痛い目を見られると思います。当然、この記事も鵜呑みにせずに使用する際には吟味しないとダメですよ。次回はjavascriptでAjax通信をすることを考えていきます。

Sponsored Link

## 2-2. テンプレートタグを使ってサイドバーを作成する

2019年4月4日 [コメントをする](#)

今回のテーマは「テンプレートタグを使ってサイドバーを作成する」です。第一章から宿題となっていたサイドバーですね。いつまでもdummyのままでは格好悪いのでサイドバーにカテゴリーが表示されるようにしていきましょう。

※本ページは[デバッグツールバーの導入](#)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。

## テンプレートタグって何

これまで説明は省いてきましたが組み込みのテンプレートタグを使用してきました。例えば、{% extends %}や{% url %}などです。テンプレートタグは{% テンプレートタグ名 %}という使い方でテンプレート上で呼び出して使います。このようにテンプレート上で予め決められたルールでレンダリングしてくれるものをテンプレートタグと言います。今回はこのテンプレートタグを自作していきます。

## 何故必要なのか？

今回はサイドバーに表示されるカテゴリー表示について考えてみましょう。カテゴリーは動的に管理ページで追加や変更、削除することができます。そしてサイドバーは様々なページで表示されます。テンプレートタグを使わずに動的にカテゴリーを表示しようと思うとサイドバーが表示されるページには全てにビューからコンテキストを渡さなければいけなくなります。非常に効率が悪いですね？そこでテンプレートタグを使用します。

## カテゴリーを表示するテンプレートタグを作成する

まずテンプレートタグ用のテンプレートを用意します。

**tempates/thread/tags/category\_teg.html**

```
<div class="ui relaxed list small divided link">
  {% for category in category_list %}
  <a class="item">{{category.name}}({{category.count}})</a>
  {% endfor %}
</div>
```

次にテンプレートタグを表示するロジック部分を作成します。thread/templatetagsディレクトリを作成し、その中にthreadtags.pyを作成します。

**thread/templatetags/threadtags.py**

```
from django.template import Library
from django.db.models import Count
from ..models import Category

register = Library()

@register.inclusion_tag('thread/tags/category_tag.html')
def categorytag():
    ctx = {}
    ctx['category_list'] = Category.objects.annotate(
        count=Count('topic')).order_by('sort')
    return ctx
```

これはinclusion\_tag関数のデコレータを使ってcategorytag関数をテンプレートタグとして登録しています。categorytag関数は単純にデータベースからクエリセットの評価によって得られたCategoryオブジェクトの辞書をコンテキストとして返す関数です。

このクエリセットですが、annotateを用いて各カテゴリーに属するトピックの数を数えて'count'という名前をつけて情報を付与しています。

このコンテキストがcategory\_tag.htmlテンプレートに当てられます。では、登録したcategorytagを使ってみましょう。

templates/base/sidebar.htmlを書き換えます。

### templates/base/sidebar.html

```
{% load threadtags %}

<div class="five wide column">
  <div class="ui action input" style="width: 100%;">
    <input type="text" placeholder="検索">
    <button class="ui button"><i class="search icon"></i></button>
  </div>
  <div class="ui items">
    <div class="item">
      <a href="{% url 'thread:create_topic' %}" class="ui fluid teal button">トピック
    </div>
  </div>
  <div class="ui segment">
    <div class="content">
      <div class="header"><h4>カテゴリー</h4></div>
      {% categorytag %}
    </div>
  </div>
</div>
```

このようにcategorytagをロードして必要な場面で使うことで必ずしもビューからテンプレートにコンテキストを渡さなくても動的なページを作れることが分かりました。今回はinclusion\_tagを用いましたが他にもsimple\_tag関数などが用意されており、引数を取る処理も書けます。公式ドキュメントの[独自のテンプレートタグとフィルタ](#)を参考にしてください。

では確認してみましょう。localhost:8080にブラウザでアクセスします。



トピックを作成

カテゴリー

WEB技術(2)

モバイル(0)

プログラミング(2)

OS関連・インフラ(0)

業界ネタ・憩いの場(2)

データベースに登録してあるカテゴリーとトピック数が表示されましたね。便利な機能ですので積極的に使っていきましょう。

## 最後に

今回はDjangoのテンプレートタグをサイドバーで使うという内容で扱いました。必ずしもビューから渡す必要のない動的な処理はテンプレートタグを使用することで実現できる場合もあります。覚えておくと便利ですよ。次回はテンプレートタグと少し似ているフィルターについて扱っていきます。