Qオブジェクトを使って検索機能の実装

Djangoには「Q object」という複雑な処理を実装できるクエリを持っています。

Django組み込みのfilter関数とセットで使われることが多く、Qオブジェクトに格納にされた文言をfilter関数に渡すことで検索機能として実装できるようになります。

より詳しい内容は「Django公式ドキュメント：Q object」をご参照ください。

まずは関数定義の方から実装していきたいと思います。

関数定義：def

「views.py」にて、必要なモジュールをインポートしindex関数内に定義していきます。

```
# project/blog/views.py

from django.shortcuts import render, get_object_or_404
from .models import Blog

""" 追加 """
from django.contrib import messages
from django.db.models import Q


def index(request):
    blog = Blog.objects.order_by('-id')
    """ 検索機能の処理 """
    keyword = request.GET.get('keyword')

    if keyword:
        blog = blog.filter(
                 Q(title__icontains=keyword)
               )
        messages.success(request, '「{}」の検索結果'.format(keyword))

    return render(request, 'blog/index.html', {'blog': blog })
```

検索フォームからリクエストされた文言（keyword）をQオブジェクトに渡しているのが分かると思います。

この処理内容は、与えられた文言がデータベース内のデータと１つでも一致していれば取得するという処理です。

「title」はモデルのフィールド名、「__icontains」（部分一致）はQオブジェクトのプロパティです。

他にも色々なプロパティが用意されているので試してみましょう。「Django公式ドキュメント：Fieldルックアップ」

では「base.html」にて検索フォームを追記します。

```
<!-- project/blog/templates/blog/base.html -->

<!doctype html>
<html>
    <head>

        <title>Blog</title>

   </head>
   <body>

        <!-- 追記 -->
        <form method="GET" action="{% url 'blog:index' %}">
            <input type="text" name="keyword">
            <button type="submit">検索</button>
        </form>

        <h1>Blog</h1>
        {% block content %}
        {% endblock %}

    </body>
</html>
```

inputタグの「name='keyword'」としたところで、ブラウザに表示させてみましょう。

Blog

検索

一覧

- 2019年12月15日 係り受け解析器のCaboChaを使って文章の関係性をNetworkXで描画する 詳細
- 2019年12月15日 【Python】構文解析器（係り受け解析）CaboChaの導入（エラー回避＆成功例） 詳細
- 2019年12月15日 【Python】MeCab（形態素解析器）を使用して文章をカテゴリー分類する 詳細
- 2019年12月15日 【Python】形態素解析器のMeCabを使って自然言語処理の実装 詳細
- 2019年12月15日 【Django】モデルを利用して管理サイトからWebサイトのカラーを変更する 詳細
- 2019年12月15日 【Python】簡単なコードで人工無能（チャットボット）の実装 詳細
- 2019年12月15日 【Django】Blog（ブログ）サイトの「下書き保存」を実現させる２種類の方法 詳細

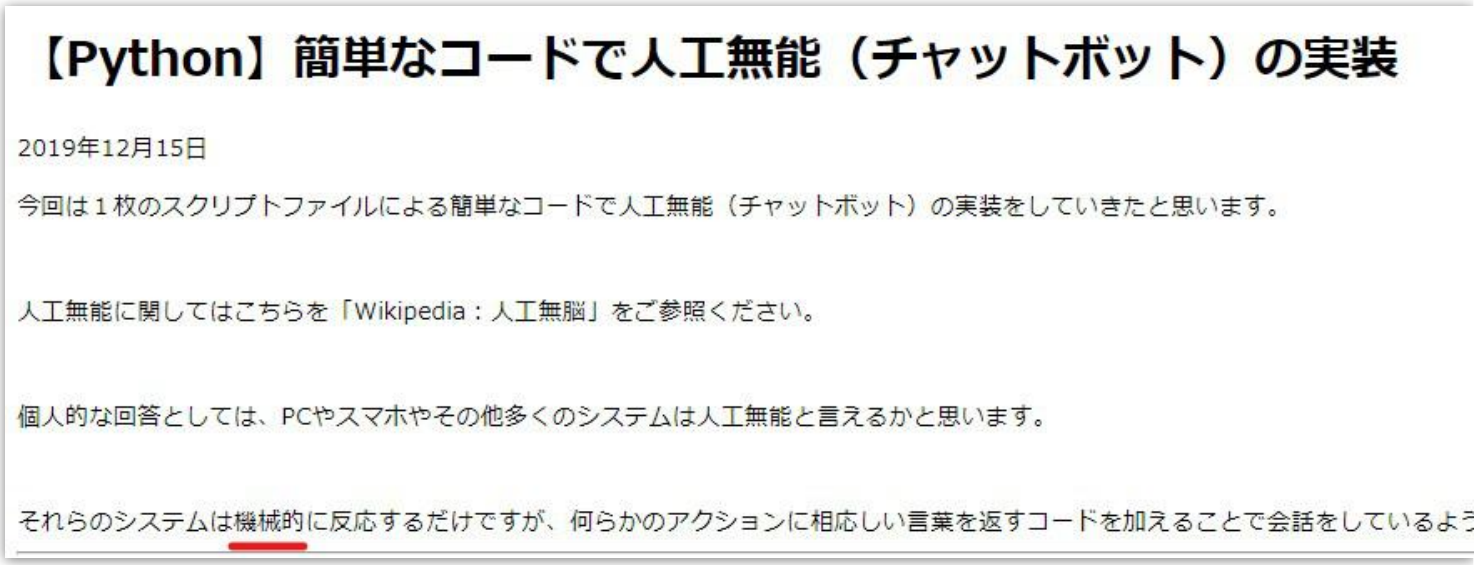
「django」と検索すると、タイトルに記述されている通りの記事を取得できます。

次はタイトルだけでなく記事の内容を含めた処理も追記していくので、「views.py」を編集します。

```
# project/blog/views.py

from django.shortcuts import render, get_object_or_404
from .models import Blog

from django.contrib import messages
from django.db.models import Q


def index(request):
    blog = Blog.objects.order_by('-id')
    """ 検索機能の処理 """
    keyword = request.GET.get('keyword')

    if keyword:
        """ テキスト用のQオブジェクトを追加 """
        blog = blog.filter(
                 Q(title__icontains=keyword) | Q(text__icontains=keyword)
               )
        messages.success(request, '「{}」の検索結果'.format(keyword))

    return render(request, 'blog/index.html', {'blog': blog })

.......
```

「&」もしくわ「|」の条件演算子を定義することでQオブジェクトを追加することができます。

追加できたらリロードを行い、タイトルには無い記事の内容だけに含まれている文言を検索してみます。



上手く表示されれば成功です。

しかしこの処理内容では複数のキーワードを選択することができないので、後ほどカスタマイズしていこうと思います。

次はクラス定義での実装を行ってみましょう。

クラス定義：class

関数定義から、クラス定義に変換させていきます。

まずはアプリディレクトリ内の「urls.py」「views.py」とテンプレートファイルの「index.html」という順番で編集します。

```
# project/blog/urls.py

from django.urls import path
from . import views


app_name = 'blog'

urlpatterns = [
        path('', views.IndexView.as_view(), name='index'),   # 編集
        path('detail/<int:blog_id>/', views.detail, name='detail'),
]
```

次に必要なモジュールをインポートし、クラスリストビューを定義します。

```
# project/blog/views.py

from django.shortcuts import render, get_object_or_404
from .models import Blog

from django.contrib import messages
from django.db.models import Q
""" 追記 """
from django.views import generic


class IndexView(generic.ListView):
    model = Blog
    template_name = 'blog/index.html'

    """ オーバーライド """
    def get_queryset(self):
        queryset = Blog.objects.order_by('-id')
        keyword = self.request.GET.get('keyword')

        if keyword:
            queryset = queryset.filter(
                        Q(title__icontains=keyword) | Q(text__icontains=keyword)
                    )
            messages.success(self.request, '「{}」の検索結果'.format(keyword))

        return queryset


def detail(request, blog_id):
    ........
```

クラスメソッドで定義すると、内部的に「blog_list」というオブジェクトが生成されるのでテンプレートファイルを編集します。

```
<!-- project/blog/templates/blog/index.html -->

{% extends 'blog/base.html' %}

{% block content %}

    <h1>一覧</h1>
    <br>

    <!-- blog_listに変更 -->
    {% for blog in blog_list %}
        <ul>
            <li>
                {{ blog.created_at }}
                {{ blog.title }}
                <a href="{% url 'blog:detail' blog.pk %}">詳細</a>
            </li>
        </ul>
    {% endfor %}

{% endblock %}
```

これだけでクラス定義としての実装が完了しました。

複数キーワード検索の実装

これまで通りの実装では、複数のキーワード（「django python」というワード）で絞って検索することができません。

例えば下図のような半角スペースや全角スペースを挟んで検索を試みてもクエリは空の結果を出力します。

これはSQLの性質上の問題なので、クエリが発行されるようQオブジェクトに何らかの処理を加える必要があります。

Django公式ドキュメント内をくまなく探せば簡単でスマートな実装方法が見つかるかもしれませんが、ここではPython標準パッケージの「functools」モジュールと「operator」モジュールを使ってカスタムを図ります。

そして今回の機能を実現できたのはこちら「検索画面を作る」を参考にしたからです。ありがとうございます。

まずは「functools」と「operator」モジュール、そしてブラックボックス的な位置にいる「Q object」の内部を簡単に見ていきます。

functoolsとoperator

functoolsには「reduce」という関数があり、これと「operator」を組み合わせて処理を行います。

fuctoolsは高階関数と言われており、関数に何かしらの処理を加えるための関数となります。

operatorは関数形式でPython組み込みの標準演算子を実行することができます。

例えば、このようなリスト「[1, 2, 3, 4, 5]」に格納された数字を左から右に足して行くならば、for文を使った処理を行うと思います。

```
# 実行

$ python3 manage.py shell
```

```
>>> x = [1, 2, 3, 4, 5]
>>> y = 0
>>> for i in x:
...     y += i
...     print(y)
1
3
6
10
15
>>> from operator import add
>>>
>>> add(1, 1)
2
>>> add(5, 5)
10
>>> from functools import reduce
>>>
>>> reduce(add, [5, 5])
10
 # 先ほどのxをreduceの第二引数に渡す
>>> reduce(add, x)
15
>>> exit()
```

このようにざっくりですが、関数内に関数を適用させてイテラブルな処理を実行できました。

これらをDjangoのQオブジェクトと組み合わせることで柔軟性に優れたクエリを発行することができます。

気になってしまった方はPython公式ドキュメントをご参照ください。

Q objectの内部構造

ここからは、Qオブジェクトに与えられた文言はどのように処理されているのかを見ていこうと思います。

再びDjangoShellを起動しモジュールをインポートします。

```
# 実行

$ python3 manage.py shell
```

```
>>> from blog.models import Blog
>>> from django.db.models import Q
>>>
>>> keyword = 'mecab　形態素解析 python'
>>> x = Q(title__icontains=keyword)
>>> print(x)
(AND: ('title__icontains', 'mecab\u3000形態素解析 python'))
>>>
>>> query = Blog.objects.filter(x)
>>> print(query)
<QuerySet []>
>>>
>>> from functools import reduce
>>> from operator import and_ # 論理演算子をインポート
>>>
>>> y = reduce(and_, [Q(title__icontains=keyword)])
>>> print(y)
(AND: ('title__icontains', 'mecab\u3000形態素解析 python'))
>>>
>>> y = reduce(and_, [Q(title__icontains=q) for q in keyword])
>>> print(y)
(AND: ('title__icontains', 'm'), ('title__icontains', 'e'), ('title__icontains', 'c'),
('title__icontains', 'a'), ('title__icontains', 'b'), ('title__icontains', '\u3000'),
('title__icontains', '形'), ('title__icontains', '態'), ('title__icontains', '素'),
('title__icontains', '解'), ('title__icontains', '析'), ('title__icontains', ' '),
 ('title__icontains', 'p'), ('title__icontains', 'y'), ('title__icontains', 't'),
 ('title__icontains', 'h'), ('title__icontains', 'o'), ('title__icontains', 'n'))
>>>
>>> query = Blog.objects.filter(y)
>>> print(query)
<QuerySet []>
>>>
>>> # 全角と半角の空文字列を除外する
>>> exclusion = set([' ', '　'])
>>> q_list = ''
>>> for i in keyword:
...     if i in exclusion:
...         pass
...     else:
...         q_list += i
...
>>> print(q_list)
mecab形態素解析python
>>>
>>> y = reduce(and_, [Q(title__icontains=q) for q in q_list])
>>> query = Blog.objects.filter(y)
>>> print(query)
<QuerySet [&lt;Blog: 【Python】形態素解析器のMeCabを使って自然言語処理の実装>,
<Blog: 【Python】MeCab（形態素解析器）を使用して文章をカテゴリー分類する>]>
>>>
>>> exit()
```

上記を例に、Qオブジェクトに与えられた文字列は「'mecab\u3000形態素解析 python'」のようになっており、データと一致されない事が分かります。

関数内関数を扱える「reduce」と論理演算関数の「operator.and_」に第二引数としてリスト内包表記で「keyword」をひとつずつQオブジェクトに与えることで、一文字一文字を論理演算子（and）によって抽出することができます。

そこで、タイトルには含まれていない全角空文字（\u3000）と半角空文字（ ）を除外することにより部分一致することができるようになりました。

先ほどの例を「views.py」に記述して実行してみましょう。

検索エンジンのカスタマイズ

まずは関数定義から実装していきます。

※「uls.py」や「index.html」も関数用に統一しておきましょう。

「views.py」を開き必要なモジュールをインポートして追記しましょう。

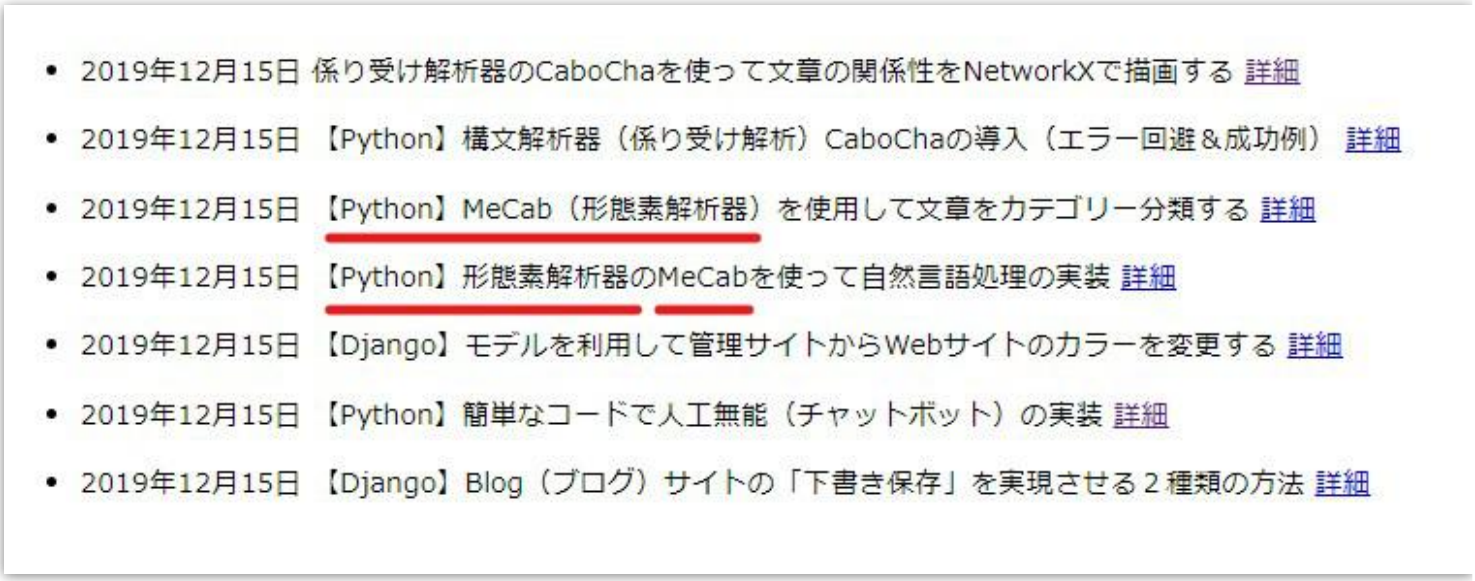
```
# project/blog/views.py

from django.shortcuts import render, get_object_or_404
from .models import Blog

from django.contrib import messages
from django.db.models import Q
""" 追加 """
from functools import reduce
from operator import and_


def index(request):
    blog = Blog.objects.order_by('-id')
    keyword = request.GET.get('keyword')

    if keyword:
        """ 除外リストを作成 """
        exclusion_list = set([' ', '　'])
        q_list = ''

        for i in keyword:
            """ 全角半角の空文字が含まれていたら無視 """
            if i in exclusion_list:
                pass
            else:
                q_list += i

        query = reduce(
                    and_, [Q(title__icontains=q) | Q(text__icontains=q) for q in q_list]
                )
        blog = blog.filter(query)
        messages.success(request, '「{}」の検索結果'.format(keyword))

    return render(request, 'blog/index.html', {'blog': blog })

......
```

クラス定義では以下のようになります。

※「urls.py」や「index.html」もクラス用に統一しておきましょう。

```
# project/blog/views.py

from django.shortcuts import render, get_object_or_404
from .models import Blog

from django.contrib import messages
from django.db.models import Q
from django.views import generic  # クラスリストビュー用
""" 追加 """
from functools import reduce
from operator import and_


class IndexView(generic.ListView):
    model = Blog
    template_name = 'blog/index.html'

    def get_queryset(self):
        queryset = Blog.objects.order_by('-id')
        keyword = self.request.GET.get('keyword')

        if keyword:
            exclusion = set([' ', '　'])
            q_list = ''

            for i in keyword:
                if i in exclusion:
                    pass
                else:
                    q_list += i

            query = reduce(
                        and_, [Q(title__icontains=q) | Q(text__icontains=q) for q in q_list]
                    )
            queryset = queryset.filter(query)
            messages.success(self.request, '「{}」の検索結果'.format(keyword))

        return queryset

......
```

さっそく複数のキーワードを入力して検索をしてみます。



キーワードの順番を入れ替えても機能することが分かります。

今回サイト内検索のパフォーマンスを上げるうえで非常に役立った「functoos」と「operator」でしたが、実際存在すら知らなかった代物でした。

参考にさせて頂いたサイトには本当にありがたく思います。

他にも便利な検索機能を組み込めるDjangoのサードパーティがあるという噂があるので、機械があれば試して行こうと思います。