

# ブログモデルをつくろう

1つ前のレッスンでは、ウェブページを表示させることができるようになりました。続いて、このレッスンでは、ブログ記事をデータベースに登録するための方法を学びます。

このレッスンでは、ブログオブジェクト（ブログ記事）を扱うことになるので、オブジェクト指向の考え方が重要になります。不安な人はこちらの記事([【Python】オブジェクト指向を理解するための超重要ワードまとめ](#))でもう一度復習しておくことをおすすめします。

## データベースってどんなもの？

まずは、データベースについて理解しましょう。名前を聞いただけで、なんとなく「データが保存される場所」であることはイメージがつくのではないのでしょうか。

その認識で全く問題ありません。ウェブサービスを運営していると、様々なデータを扱うことになりますが、それを保管している場所がデータベースです。

例えば、新規にユーザー登録した人がいればその人の名前や年齢などのユーザー情報を保管しなければなりませんし、新しくブログが投稿されたときのためにブログ情報を保管しておくデータベースも必要です。

データベースがあればデータを保管するだけでなく、データを操作したり検索したりすることも可能となります。

では、具体的にデータベースはどのような構成になっているのでしょうか。

データベースは、以下のような表になっており、この表のことを **テーブル**、横の行の1つ1つを **データ（またはレコード）**、縦の列を **フィールド（またはカラム）** といいます。

1つの行につき1つのインスタンスを保管し、各フィールドでそのインスタンスのプロパティを保管しています。

# Users テーブル

データ (行) →	フィールド (列)				
	id	名前	年齢	性別	趣味
	1	ジョブス	32	男	プログラミング
	2	ローラ	26	女	サッカー
	3	ジョン	18	男	ゲーム
	4	ケイ	52	男	読書
	5	キャサリン	29	女	料理

## データベースを作ってみよう

それでは、実際にブログ記事を保管するデータベース： **Blogsテーブル** を作ってみましょう。

新たにテーブルを作るときは、 **モデル** というものを使います。 **モデルとは、データベースとやりとりするクラス** のことです。

クラスとは、オブジェクトの設計図のことでしたね。ブログ管理サイトでは、様々なブログ記事（ブログオブジェクト）が作成されることになっていきますが、ブログ記事に必要な情報（プロパティ）をクラスをによって定義してあげます。ブログ記事に必要な情報とは、タイトル、本文、作成日、更新日などのことです。

本来、データベースを操作するときはSQLという言語を使うのですが、この「モデル」があることでPythonコードでデータベースの構造を記述し、簡単に操作できるようになります。

また、Djangoにおけるモデルとは、データベースに保存したいデータの構造を指定したもののことです。通常、一つのモデルに対して一つのデータベーステーブルが割り当てられ、各アプリケーション内の `models.py` に記述していきます。

今回作成するブログモデルのコードは以下の通りです。

django\_blog/blogs/models.py

```
from django.db import models

class Blog(models.Model):
    title = models.CharField(blank=False, null=False, max_length=150)
    text = models.TextField(blank=True)
    created_datetime = models.DateTimeField(auto_now_add=True)
    updated_datetime = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.title
```

先頭で、 `from django.db import models` をインポートします。これはDjangoにおけるモデルの型の基本形であり、それを継承させた `Blog` モデルクラスを作成していきます。これによって、このクラスを元に作られたインスタンスはデータベースに保存すべきオブジェクトであるという設定をしています。

モデルを作るときは、「Blogs」の様に複数形で書かずに「Blog」と単数形で書きます。これによって「Blogs」という名称のテーブルを自動生成してくれます。また、Pythonでクラスを作る時同様最初の文字は大文字で始めます。

次に、「title」や「text」の部分では、Blogsテーブルのフィールドを作成しています。この記述により、以下のようなテーブルを作ることができます。

Blogs テーブル

	id	title	text	created_datetime	updated_datetime
(例)	1	読書	今日は読書をしました。	2018/03/18	2018/03/21
	2				
	3				
	4				
	・				
	・				
	・				

フィールドについての記述の内容について詳しく説明します。

```
django_blog/blogs/models.py

title = models.CharField(blank=False, null=False, max_length=150)
```

まず、フィールドに `title` という名前（フィールド名）をつけて、右辺で `models.CharField` という型を指定しています。このようにフィールドにはいくつか型の種類があり、型を指定することによって意図しないデータが入力されることを防いでいます。この `models.CharField` は文字列を入力できるようにした型になります。当然、タイトルには文字列が入ることになりますからね。（CharFieldは、文字を意味するCharacterからきていると思われます）

さらに `models.CharField` の中にいくつか条件を指定することができます。例えば、`blank=False, null=False` は、このフィールドが空欄であってはいけないということを示しています。ブログ記事のタイトルが空欄なのはまずいですもんね。

※blankとnullの違いは、別のチュートリアルで詳しく説明しますが、ざっくり、入力フォームでの空欄を許容するかどうかと、データベース上での空欄を許容するかどうかの違いだと思っています。

また、`max_length=150` はタイトル文字数を最大150字に制限しています。

このように、フィールドの型と、各種条件をつけてデータベースのデータの形を設計していきます。

一部未翻訳ですが、[こちらの公式ドキュメント](#)で指定できるフィールド型とオプションを確認することができます。よく利用する型やフィールドについては今後のチュートリアルでも解説していきます。

`created_datetime`, `updated_datetime` のフィールドでは、`models.DateTimeField` という型を使って、日時情報を保存するように指定しています。もちろん、日時情報を自分で手動入力して保存することもできますが、今回は `auto_now_add`, `auto_now` という引数を与えることで、インスタンスが作成された日時、更新された日時を自動的に保存するよう設定しています。

最後の `def __str__(self)` の部分は、このモデルで作成されたインスタンス（一つ一つのブログ記事のこと）自体を指し示すときに利用する文字列を指定しています。これにより、管理ページなどで各インスタンスを表示するときはブログ記事のタイトルで表示されることになります。（この記述だけだとよくわからないと思いますので、後で詳しく説明します。）

## マイグレーション

ここまでで、Blogモデルを作成することができました。ただ、モデルはあくまで「設計図」であるため、現段階だとデータベースの設計図を作ったに過ぎません。この設計図を元に実際にデータベースを作成するには、もう少しだけしなければならないことがあります。

それが「マイグレート」という作業です。マイグレートとは、`models.py`ファイルで定義したデータベースの設計を、実際にデータベースに反映させることを言います。マイグレートするためには、以下2つのコマンドを実行します。

`django_blog/`

```
$ python manage.py makemigrations
$ python manage.py migrate
```

`python manage.py makemigrations` コマンドでは「マイグレーションファイル」というものを作っています。マイグレーションファイルとは、`models.py`ファイルで作成したデータベースの設計情報がまとめられたファイルです。このコマンドによって、`django_blog/blogs/migrations` ディレクトリの中にマイグレーションファイルが新たに追加されます。

`python manage.py migrate` コマンドは、マイグレーションファイルの情報をデータベースに反映させるコマンドです。

これにより、models.pyファイルで作成したBlogモデルをデータベースに反映させることができました。

このように、データベースを作るときは、「**models.pyファイルで設計 → マイグレート処理で反映**」という手順で行います。models.pyファイルに何か変更を加えた時（例えばタイトルフィールドの文字数制限を150文字から200文字に変えた時）は、その都度マイグレート処理を行い変更内容をデータベースに反映させることを忘れないようにしてください。

## Adminページを利用する

続いて、Adminページを利用できるような設定をしましょう。Adminページとは管理者のみが利用できるページのことです。通常は自前で作成しなければならないものなのですが、なんとDjangoはこの管理ページを自動的に生成してくれるという素晴らしい特徴を持っています！これがDjangoが多くの海外デベロッパーに愛されている理由の一つです。

Djangoでは、`models.py` で指定したモデルクラスから自動的に管理ページのインターフェースを提供してくれます。これにより、新しいデータの追加や編集、削除などの機能が管理者ページから簡単に利用できるのです。

Adminページは`<ドメイン>/admin` というURLからアクセスすることができ、最初になんのモデルも作成していない状態でもデフォルトのページを利用することができます。今回は、作成したBlogモデルをAdminで利用できるようにしてみます。また、実際にブログ記事を管理ページ上から作成してみましょう。

まずは、管理ページにアクセスするために管理者アカウントを作成しましょう。Djangoでは、管理者のことを`スーパーユーザー`と呼びます。スーパーユーザーのアカウントはコマンドラインから作成することができます。

※スーパーユーザーアカウントを作成

django\_blog/

```
$ python manage.py createsuperuser
```

コマンドを実行するとユーザー名、メールアドレス、パスワードを聞かれるので、自分の好きなように入力してください。パスワードは、入力してもターミナルには何の文字も表示されませんが、表示されないだけでちゃんと入力されているので心配しないでください。ユーザー名は、`Username (leave blank to use 'User_Name'):` の様に聞かれますが、書いてあるとおり、ブランク、つまりなにも入力せずにEnterを押すと'User\_Name'の部分がそのままユーザー名になります。

これで管理者アカウントが作成されました！実際にAdminページにアクセスしてみましょう。ローカルサーバーを立ち上げて、`http://127.0.0.1:8000/admin` というURLからアクセスすることができます。

最初にログインが要求されると思いますが、いま作成した管理者アカウントでログインできます。ログインするとこのような画面が表示されます。



管理するためのページといっても、まだデフォルトの状態なのでユーザー情報くらいしか管理することができません。次は、この画面でBlogsテーブルの情報を管理できるように設定していきましょう。ちなみにですが、この管理ページが日本語で表示されているのは、settings.pyファイルのLANGUAGE\_CODE を'ja'に設定しているおかげです。

Adminページを編集するためには `admin.py` ファイルを使います。まずはBlogモデルを `admin.py` に登録します。admin.pyファイルを開いて、このように書いてください。

※BlogモデルをAdminに追加

django\_blog/blogs/admin.py

```
from django.contrib import admin
from .models import Blog

admin.site.register(Blog)
```

`from .models import Blog` は、admin.pyファイルと同じ階層にある `models.py` ファイルで定義したBlogモデルをインポートして、このファイルで扱えるようにしています。modelsの前にあるドットは、同じ階層にあることを示しています。

そして `admin.site.register(Blog)` の部分でインポートしたBlogモデルを、Adminページで利用できるようにしています。これにより、AdminページでBlogの情報を見ることができるようになります。

ようになります。エディタを保存して、再度 `http://127.0.0.1:8000/admin` にアクセスしてください。



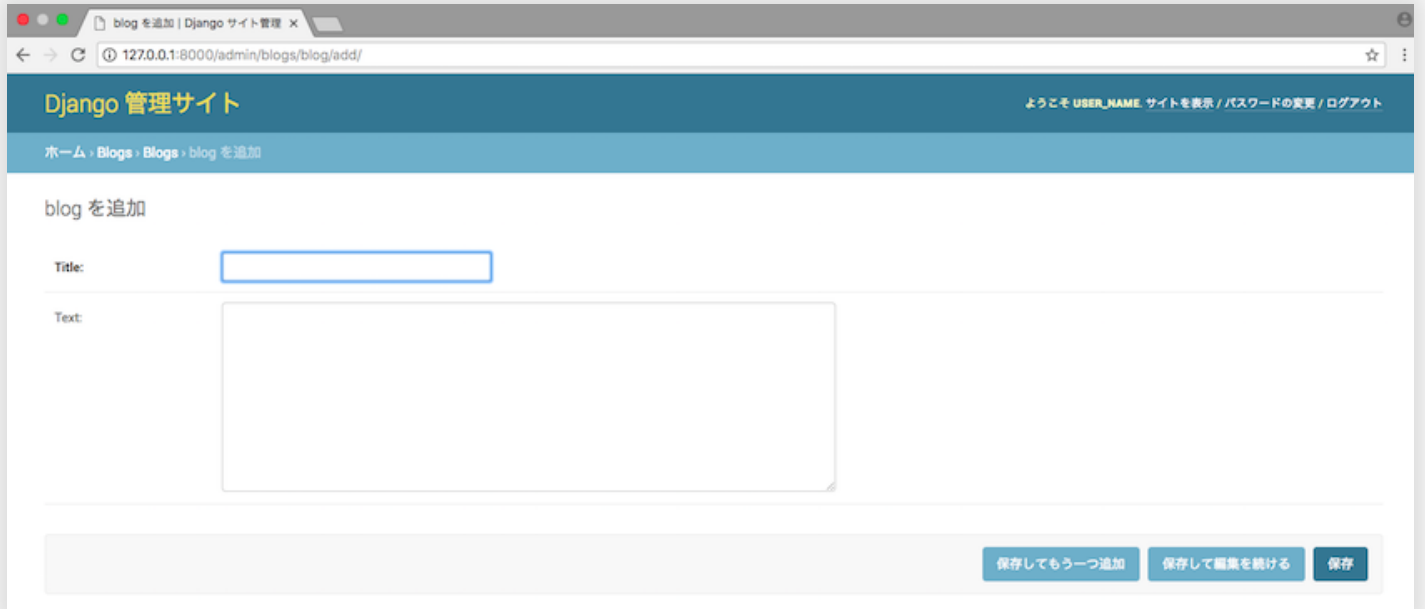
Blogsというパートが表示されるようになっているはずです。これでブログ記事の管理ができるようになりました！このように、Blogモデルを作ると自動的に `Blogs` と複数形にしてテーブルを作ってくれます。

## 管理ページからブログを投稿してみよう

では、実際にブログの記事を作成してみましょう。

Blogsという欄の「追加」ボタンをクリックしてください。すると、こんな画面が表示されましたね。





The screenshot shows the Django admin interface for adding a new blog post. The browser address bar shows the URL `127.0.0.1:8000/admin/blogs/blog/add/`. The page title is 'Django 管理サイト'. The breadcrumb navigation shows 'ホーム > Blogs > Blogs > blog を追加'. The form is titled 'blog を追加' and contains two main fields: 'Title' (a single-line text input) and 'Text' (a larger text area). At the bottom right, there are three buttons: '保存してもう一つ追加', '保存して編集を続ける', and '保存'.

「Title」と「Text」と表示されていますが、`models.py`ファイルで定義したBlogクラスのフィールド名が表示されています。ここには文字列型に入れられる設定にしていたよね。

このページから新規のブログ投稿を行うことができるので実際に一つブログを投稿してみましょう。好きなようにタイトルと本文を入力し、右下の「保存」ボタンで保存しましょう。ちなみに、タイトルを入力しないまま保存ボタンを押すとエラーメッセージが表示されます。このフィールドは`blank=False`を指定してますからね！

これでBlogクラスを元にBlogインスタンスを作ることができました。クラスは「オブジェクトを量産するための設計図」、インスタンスは「クラスを元に作られた個々のオブジェクトのこと」でしたね。

テスト用にもう一つか二つ程度、記事を投稿してみてください。

ここで先ほどちょっと説明した、`def __str__(self)`の説明をします。ブログオブジェクトの一覧ページ (`http://127.0.0.1:8000/admin/blogs/blog/`) にいくと、それぞれのブログのタイトルが表示されていますよね。ここにタイトルが表示されるのは以下のように`self.title`と記述しているからです。

`django_blog/blogs/models.py`

```
...
def __str__(self):
    return self.title
```

例えば、ここを`self.text`と書き換えると、ブログの本文が見出しとして表示されるようになります。

このように`def __str__(self)`の部分では、そのオブジェクトを扱う上で使用するフィールドを指定しているのです。



また、このページには現在タイトルだけが表示されていますが、ここには複数のフィールドを表示させることができます。admin.pyファイルを編集することで可能です。以下のように書き換えてみてください。

django\_blog/blogs/admin.py

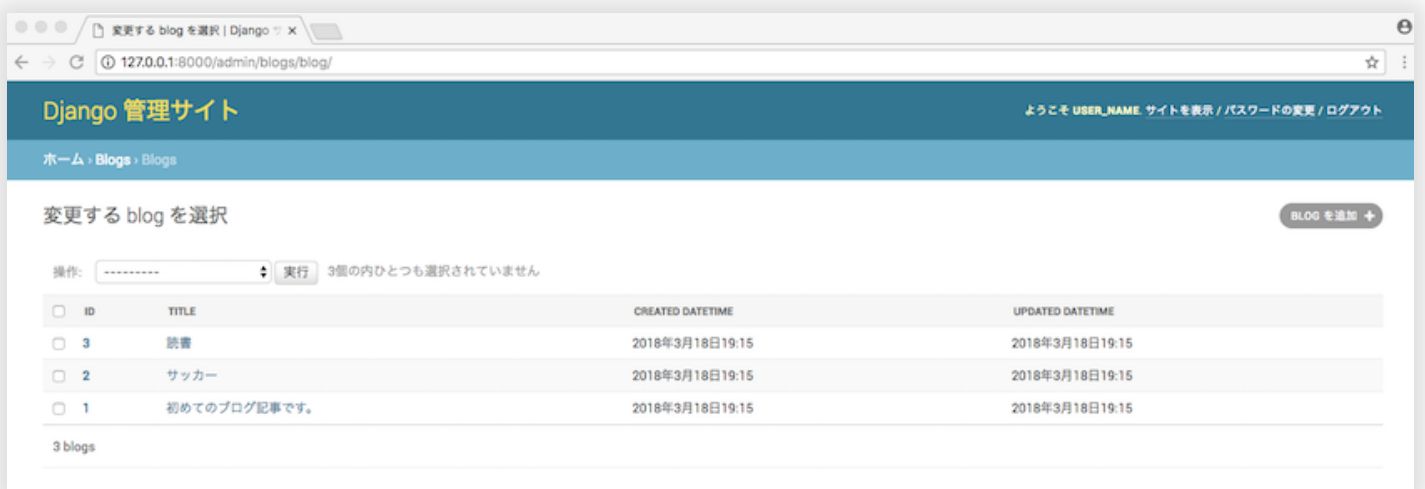
```
from django.contrib import admin
from .models import Blog

class BlogAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'created_datetime', 'updated_datetime')
    list_display_links = ('id', 'title')

admin.site.register(Blog, BlogAdmin)
```

`list_display` で指定したフィールドが管理ページに表示されるようになります。

`list_display_links` で指定したフィールドはリンクがつくようになります。



ここでちょっと見慣れないフィールドが出てきましたね。そうです、`id` のことです。`ID` は、インスタンスがデータベースに保存される時に自動的に割り振られる番号です。**1つのインスタンスに対して1つの番号が割り振られます。**

IDはこの後のレッスンで扱いますので、1つのインスタンスに1つのIDがつくことを覚えておいてください。

これでこのレッスンは終わりです！長くて難しい内容だったと思いますが、うまくできましたか？データを扱うことで少しずつWebサービスっぽくなってきましたね！

まだブログ記事は管理者ページにログインした人しか見ることができません。次はこれらの記事をトップページに表示して、サイトを訪れた人が読めるようにしていきましょう。