CHAPTER 3

基本的なデータ操作（CRUD処理）

この章では、Djangoの関数ベースビュー（Function-Based Views：FBV）を使って、データの作成（Create）、読み取り（Read）、更新（Update）、削除（Delete）を行う方法を学びます。ステップバイステップで一緒に進めていきましょう！

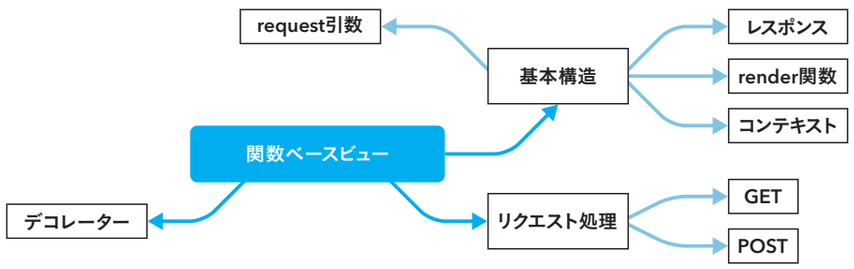
3-1 FBVとは？（特徴と使いどころ）

Djangoにおける関数ベースビュー（Function-Based Views：FBV）を簡単に言うと「処理をシンプルな Python 関数として記述し、リクエストを受け取ってレスポンスを返す」仕組みのことです。

3-1-1 ｜ FBVのマインドマップ

この章で説明するDjangoの関数ベースビュー（FBV）をマインドマップ(注1)で示します。

図3.1 マインドマップ（関数ベース）



図のように、FBVは以下のような要素で構成されます。

- 基本構造（request引数、レスポンス、render関数、コンテキスト）
- リクエスト処理（GET, POST）
- デコレーター（後述）

（注1） マインドマップは、アイデアや情報を視覚的に整理・表現するための図です。

3-1-2 ｜ 基本構造（request引数、レスポンス、render関数、コンテキスト）

Djangoの関数ベースビュー（FBV）の基本構造や、よく使用するオブジェクトや関数について以下に示します。

request引数

ビュー関数は、通常のPython関数として定義します。すべてのビュー関数は、最初の引数として「requestオブジェクト」を受け取ります。このrequestオブジェクトには、ユーザーから送られてきたデータや、HTTPメソッドの情報などが含まれます。

レスポンス

HttpResponseオブジェクトを使用して、直接HTTPレスポンスを返すことができます。HttpResponseは、ビュー関数がクライアント（ユーザーのブラウザ）に返すデータ（HTMLやテキストなど）を含むレスポンスオブジェクトです。

以下にソースコードの例を記述しますが、後ほどアプリケーションを作成しながら動くソースコードで説明しますので、ここではあくまでも構文例として捉えてください。

```
001: from django.http import HttpResponse
002:
003: def sample_view01(request):
004:     return HttpResponse("Hello, World!")
```

このコードは、「Hello, World!」というテキストをそのまま表示するビュー関数です。HttpResponseについては、すでに2章で作成したhelloappで使用しました。

render関数

Djangoの「render関数」は、テンプレートを使用してHTMLを生成し、レスポンスとして返します。これにより動的なWebページを簡単に作成できます。

コンテキストデータ

テンプレートに渡すデータは、「辞書形式」で管理します。「render関数」を使用して、テンプレートに渡すデータを「コンテキストデータ」と呼びます。

CHAPTER 4

ユーザーへの通知（Messagesフレームワーク）

この章では、Djangoの関数ベースビュー（FBV）を使い、ユーザーにメッセージを表示する「Messagesフレームワーク」と、画像ファイルの登録方法を学びます。ステップバイステップで進めていきましょう！

4-1 Messagesフレームワークとは？（画面上にメッセージを表示）

先ほど作成した「書籍アプリ」では、登録、更新、削除後にPRGパターンにより「書籍一覧画面」を表示します。しかし、表示された「書籍一覧画面」を見ただけでは、何の処理が実行されたかわかりません。そんな時は、ユーザーに短いメッセージを表示して知らせましょう。

4-1-1 ｜ DjangoのMessagesフレームワーク

Djangoの「Messages フレームワーク」は、ユーザーに対して短い通知メッセージを表示するためのツールです。たとえば、フォームの送信に成功したときや、データの削除が完了したときに、ユーザーへ結果をフィードバックする目的で使われます。このような通知メッセージは「フラッシュメッセージ」とも呼ばれ、1度だけ画面に表示された後、自動的に消えます。

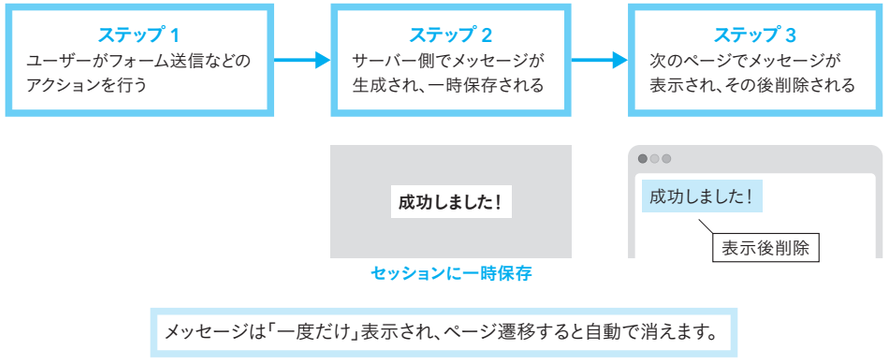
フラッシュメッセージの特性

フラッシュメッセージは「一度だけ表示される」ことが特徴です。その仕組みを以下にまとめます。

① セッションまたはクッキーに保存(注6)
　フラッシュメッセージは、Djangoの「Messages フレームワーク」を通じて、一時的に保存されます（サーバー側のセッションやブラウザ側のクッキーを使用）
② 次のリクエストで表示＆削除
　次のリクエスト（ページ移動など）が行われると、メッセージが画面に表示され、その直後に削除されます。これにより、同じメッセージは再表示されず、「一度きり」の通知が実現されます

(注6)　セッションとは、サーバー側で一時的にユーザーの情報を記録する仕組みで、クッキーは、ブラウザに保存される小さなデータで、一時的な情報の管理に使われます。

図4.1　フラッシュメッセージの仕組み



この仕組みにより、ユーザーがフォーム送信やその他の操作を行った後に、成功やエラーのフィードバックを受け取り、操作の成功・失敗を確認できます。確認後は不要なメッセージが表示され続けることがないため、ユーザー体験（User Experience：UX）が向上します。

Djangoの「Messages フレームワーク」では、メッセージの重要度や緊急度を示すために、いくつかの「メッセージレベル」が用意されています。それぞれのレベルには、「強さ（緊急度）」の違いがあり、表示内容に応じた使い分けが推奨されます。**表4.1**に、主なメッセージレベルとその強さ、用途をまとめます。

表4.1　メッセージレベル

メッセージレベル	強さ	説明
debug	非常に低い	開発者向けのデバッグ情報を表示します。通常、ユーザーには表示されません。開発時の問題調査や動作確認に役立ちます
info	低い	一般的な通知やお知らせを表示します。特に緊急性のない、日常的なフィードバックに使います
success	中程度	成功した操作に対するフィードバックを提供します。ユーザーが操作を正しく行ったことを確認するためのメッセージです
warning	高い	ユーザーに対して注意を喚起する必要がある状況を提供します。重大な問題ではないが、改善が必要な場合に使用します
error	非常に高い	ユーザーが直ちに対処すべき重大な問題を提供します。操作が失敗したり、重要なエラーが発生した場合に使用します

Messagesフレームワークの設定方法

Djangoの「Messagesフレームワーク」を使用するには、プロジェクトの「settings.py」に

CHAPTER 6

Django管理画面の使い方

Djangoには「管理画面（Django Admin）」という、とても便利な機能があります。この管理画面を使うと、Webブラウザからアプリのデータを追加・参照・編集・削除（CRUD）などの操作が簡単にできます。早速「管理画面（Django Admin）」の使い方をステップバイステップで一緒に学びましょう！

6-1 Django管理画面とは？（標準機能の活用）

6-1-1 | Django管理画面とは？

Django管理画面は、Djangoのデータベースに直接アクセスできる「管理者専用ページ」です。開発者や管理者が、下記のような用途で利用します。

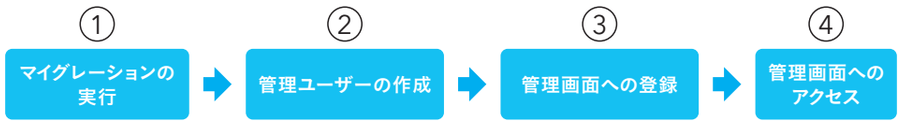
表6.1 Django管理画面の機能

機能	用途
データの追加・更新・削除	モデルに登録されたデータを追加・更新・削除することができます
データの一覧表示	データベースに保存されているデータを一覧表示し、詳細を確認できます
権限による操作制限	管理者ごとに権限（追加、編集、削除、表示など）を設定することで、モデルや操作にアクセス可能なユーザーを制限できます

6-1-2 | 管理画面を使うための準備

Django管理画面を使用するには、以下の4ステップを行う必要があります。

図6.1 4ステップ



① マイグレーションの実行

まずはマイグレーションを行いましょう。作業ワークスペース「work_django/myproject」フォルダを開き、以下のコマンドを実行します。

項目	対象
作業フォルダ	work_django/myproject
コマンド	python manage.py migrate

Djangoでは、ユーザー情報や認証に関するデータ（ユーザー名やパスワードなど）を保存するための「auth（認証）」というデフォルト機能が用意されています。「python manage.py migrate」コマンドを実行することで、Djangoのデフォルトの設定に基づいてデータベースにテーブルが作成され、認証機能が正常に動作できるようになります。これらのテーブルを作成しないと、次に行うスーパーユーザーの作成が失敗しエラーが発生してしまいます。

図6.2 マイグレーションの実行

```
(django_env) C:\work_django\myproject>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, bookapp, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

マイグレーションにより、DBファイル「db.sqlite3」へ以下テーブルが作成、更新されます。

CHAPTER 7

メニュー画面の作成とデータの表示

この章では、Djangoのクラスベースビュー（CBV）を使って、TemplateView・ListView・DetailViewの3つの基本的な画面の作り方を学習します。CBVを使うと、ページの表示や一覧・詳細画面がシンプルに作れます。では、ステップバイステップで一緒に学びましょう!

7-1 CBVとは？（FBVとの違いを知る）

Djangoのクラスベースビュー（Class-Based Views：CBV）とは、「ビューの処理ロジックをクラスとして定義し、共通処理を継承・拡張して構築する方法」です。CBVを使うことで、複雑な処理や機能をオブジェクト指向の考え方で整理でき、HTTPリクエストに応じたレスポンスを効率的に返すことができます。簡単に言うと「Djangoが用意されているクラスを継承する」ことで、様々な処理を簡単に実現できます。

7-1-1 ｜ CBVのマインドマップ

この章で説明するDjangoのクラスベースビュー（Class-Based Views：CBV）の主要クラスをマインドマップで示します。

図7.1 マインドマップ（クラスベース）

各クラスについて**表7.1**に示します。

表7.1 クラスベースの主要クラス

クラス	説明
TemplateView	指定したテンプレートを表示するためのビュークラスです 単純な画面表示に適しています
ListView	一覧表示用のビュークラスです データベースの複数のオブジェクトを一覧で表示するのに便利です
DetailView	特定のオブジェクトの詳細を表示するためのビュークラスです データの「読み取り」機能を提供します
CreateView	新規作成用のビュークラスです オブジェクトを新たに作成するためのフォームを提供します
UpdateView	既存のオブジェクトを更新するためのフォームを提供するクラスです データの「更新」に使用します
DeleteView	既存のオブジェクトを削除するビュークラスです 削除確認画面も自動的に生成されます

7-1-2 ｜ 基本構造

Djangoのクラスベースビュー（Class-Based Views: CBV）の基本構造について**表7.2**に示します。

表7.2 クラスベースの重要点

項目	概要
ビューのクラス	Djangoの「クラスベースビュー（CBV）」は、django.views モジュールのクラスを継承して作成します。クラスを使うことで、共通の処理をまとめられ、再利用しやすくなり、保守がしやすくなります
定義	クラスベースビューは、Djangoの組み込みクラス（例：TemplateView や ListView）を継承して定義します。関数ベースビュー（FBV）よりも、コードの整理がしやすく、拡張しやすいのが特徴です

それでは、早速クラスベースビューを利用したアプリケーションを作成し、理解を深めましょう。

7-2 静的ページの作成（TemplateViewの使用）

クラスベースビューの概要が理解できたので、これから実際に「クラスベースビュー（CBV）」を使って簡単なアプリケーションを作成しましょう。今回作成するのは、定番ではありますが学習に適した「ToDoアプリ」です。

CHAPTER 9

データ分析の実装

Pythonには、データ分析のための便利な道具（ライブラリ）がたくさんあります。たとえば、表のデータをきれいに扱う「Pandas」、データを見やすくグラフにする「Matplotlib」や「Seaborn」、計算をスムーズにする「NumPy」などです。この章では、こうしたライブラリを使って「データの読み込み」「分析」「グラフ化」の流れを実際に体験しましょう。

9-1 データ分析の準備

9-1-1 ｜ 必要なライブラリのインストール

以下のコマンドをターミナルで実行します。

項目	対象
仮想環境	django_env
コマンド	pip install pandas matplotlib django-pandas

図9.1 インストール

```
問題　出力　デバッグ コンソール　ターミナル　ポート

(django_env) C:\work_django\myproject>pip install pandas matplotlib django-pandas
Collecting pandas
  Using cached pandas-2.2.3-cp312-cp312-win_amd64.whl.metadata (19 kB)
Collecting matplotlib
  Using cached matplotlib-3.10.1-cp312-cp312-win_amd64.whl.metadata (11 kB)
Collecting django-pandas
```

今回使用するライブラリについて**表9.1**に示します。

Djangoでは、さまざまなライブラリを使って、データを整理したり、グラフにして見やすくすることができます。初学者には少し難しく感じるかもしれませんが、まずは完璧に理解しようとせず、「こんなこともできるんだな」と気軽に体験してみてください。

表9.1 使用するライブラリの特徴

ライブラリ	主な用途	できること
Pandas（パンダス）	表データの整理と分析	・表を読み込む（CSVなど）
		・並び替え、集計、絞り込みができる
Matplotlib（マットプロットリブ）	グラフの作成	・折れ線グラフ、棒グラフ、円グラフなどが作れる
		・データを見える化できる
django-pandas（ジャンゴ・パンダス）	Djangoのデータをpandasで扱えるようにする	・Webアプリのデータをpandasで簡単に使える
		・データ分析とDjangoの連携ができる

この3つのライブラリを使えば、「データを集める → 整理する → 見える化する」までを実行できます。

9-2 データ分析機能の追加（Pandas・Matplotlib・django-pandas）

インストールした「pandas（パンダス）」、「Matplotlib（マットプロットリブ）」、「django-pandas（ジャンゴ・パンダス）」を利用してDjangoアプリケーションのToDoデータをもとに、円グラフと棒グラフを使ってタスクの分析結果を可視化してみましょう！

9-2-1 ｜「データ分析」作成手順

図9.2に示す順番で「データ分析」を作成していきます。

図9.2 作成手順

1 models.pyへ「データ分析」を追加

フォルダ「todoapp」配下のファイル「models.py」の末尾に「データ分析メソッド」を追加します（**リスト9.1**）。

項目	対象
作業フォルダ	work_django/myproject/todoapp
修正ファイル	models.py

CHAPTER 10

認証と認可（ログインと権限管理）

この章では、Djangoで用意されている「認証」と「認可」について学びます。認証は「ユーザーが本人かどうかを確かめるしくみ」、認可は「ユーザーがどこまで操作できるかを決めるしくみ」です。これらを理解することで、アプリの安全なユーザー管理やアクセス制限の方法が身に付きます。

10-1 認証（ログイン機能の実装）

Djangoには、ログインやログアウトなどの認証機能があらかじめ用意されており、簡単に実装できます。まずは「認証」とは何かを説明し、その仕組みを理解したうえで、認証機能を実際のアプリケーションに組み込んでいきましょう。

10-1-1 ｜ 認証（Authentication）とは？

認証（Authentication）とは、ユーザーが自分の身元を証明する方法で、簡単に言うと「ログイン」のことです。

Djangoで用意されている認証に関連するクラスを**表10.1**にまとめます。

表10.1 Djangoで用意されている認証関連クラス

クラス名	説明	主な用途
LoginView	ユーザーがログインするためのビューです。ユーザー名とパスワードの入力フォームを提供し、認証が成功すれば指定のページにリダイレクトします	ログイン画面の表示とログイン処理を行う
LoginRequiredMixin	この「Mixin」をビューに追加することで、そのビューにアクセスするにはログインが必要となります。未認証のユーザーはログイン画面にリダイレクトされます	特定のページをログインユーザー（認証されたユーザー）のみアクセス可能にする

以下の順番でプロジェクトに対して「認証機能」を作成していきます。

図10.1 作成手順

① URLパターンの設定（プロジェクト）

「認証」は、どのアプリでも共通して必要となる全体的な処理であるため、プロジェクト全体の「urls.py」に設定します。

項目	対象
作業フォルダ	work_django/myproject/myproject
修正ファイル	urls.py

フォルダ「myproject」→ ファイル「urls.py」に「ログイン」のURLパターンを追加します（**リスト10.1**）。

リスト10.1 urls.py（プロジェクト）

```
001: ・・・既存コード：インポート部分・・・
002: # ▽▽▽▽▽ 10.1 ▽▽▽▽▽
003: from django.contrib.auth.views import LoginView
004: # △△△△△ 10.1 △△△△△
005:
006: urlpatterns = [
007: ・・・既存コード・・・
008:     # ▽▽▽▽▽ 10.1 ▽▽▽▽▽
009:     # ログイン画面のURLパターン
010:     # ルートURL('')にアクセスすると、LoginViewが呼び出される
011:     # 'login.html'というテンプレートが表示される
012:     path('', LoginView.as_view(template_name='login.html'), name='login'),
013:     # △△△△△ 10.1 △△△△△
014: ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

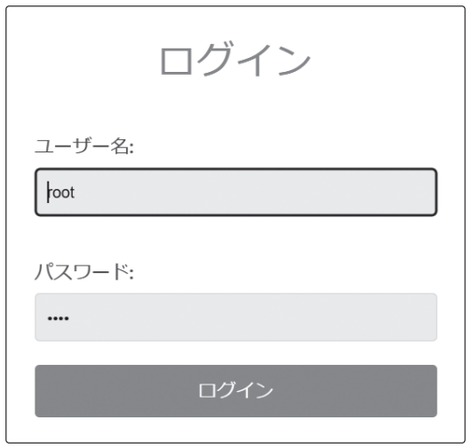
3行目の「django.contrib.auth.views」には、Djangoが提供する「ログイン」や「ログアウト」、パスワードリセットなどの「認証」に関するビューモジュールが含まれています。

12行目の「''」は、URLが空の場合、つまり「/（ルートURL）」にアクセスされたときに表示されます。「LoginView.as_view(template_name='login.html')」は、Djangoがあらかじめ用意している「ログイン用のビュー（LoginView）」を使用して、ログインページを表示する処理です。

「template_name='login.html'」では、ログイン画面の表示に使うテンプレートとして、これ

図10.7 ログアウト①

図10.8 ログアウト②



10-2 認可（権限管理の実装）

次に、「認可」の実装に進みます。Djangoには「認可」機能もあらかじめ用意されており、ユーザーに対してアクセス権限を簡単に設定できます。まずは「認可」とは何かを説明し、その基本的な考え方を理解したうえで、実際にアプリケーションに適用してみましょう。

10-2-1 ｜ 認可（Authorization）とは？

認可（Authorization）とはユーザーがアクセスできる機能やリソースを制限・管理する方法。簡単に言うと「権限を確認する」ことです。

Djangoで用意されている認可に関連するクラスを以下に示します。

項目	説明
LoginRequiredMixin	ログイン済みのユーザーのみアクセスを許可するクラスです。未ログインの場合、自動的にログイン画面へリダイレクトされます。ログインユーザー専用のページに「アクセス制限」をかけたいときに使用します。「10-1-1　認証（Authentication）とは?」で解説済みです
PermissionRequiredMixin	指定した権限（パーミッション）を持つユーザーだけにアクセスを許可するためのクラスです。必要な権限がない場合は、自動的にログイン画面やエラーページにリダイレクトされます。たとえば「編集権限を持つユーザーだけが閲覧・操作できるページ」に制限をかけたい場合などに使用します

「LoginRequiredMixin」と「PermissionRequiredMixin」を組み合わせることで「ログインしているかつ、特定の権限を持つユーザーのみがページにアクセスできる」ような高度なアクセス制御も実装できます。

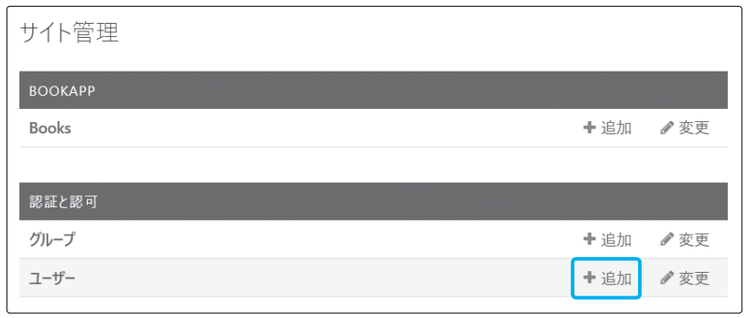
以下の順番で「認可機能」を作成していきましょう。

図10.9 作成手順

① ユーザー作成（管理画面）

サーバーを起動し（コマンド：python manage.py runserver）、ブラウザで「http://127.0.0.1:8000/admin/」にアクセスします。Django管理サイトに（ユーザー名：root、パスワード：pass）ログイン後、表示される画面にて「認証と認可」→「ユーザー」の「＋追加」リンクをクリックします。

図10.10 ユーザー作成①



「ユーザーを追加」画面にて以下の値を入力後、「保存」ボタンをクリックします。

CHAPTER

11 データベース操作（ORMの活用）

この章では、Djangoの「ORM（オーアールエム）」という仕組みについて学びます。ORMを使うと、難しいSQLを書かずに、Pythonの文法だけでデータベースの情報を操作できるようになります。データの取り出し方や、テーブル同士の関係（リレーション）の作り方、データ構造の変更、そして1対1・1対多・多対多といった関係に合わせたデータ取得方法まで、ステップバイステップで一緒に学びましょう。

11-1 データベースとDjangoの連携

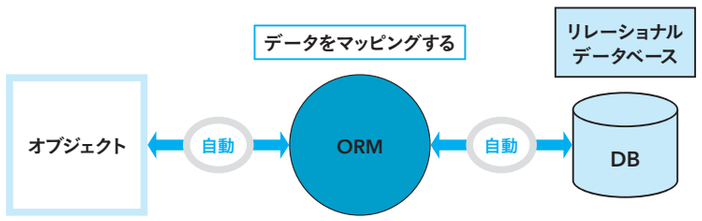
現在のプログラム開発では、データベースとのやり取りには「O/Rマッパー（Object-RelationalMapper：ORM）」というフレームワークを使用するのが一般的です。Pythonではこれを「ORM（オブジェクト・リレーショナル・マッパー）」と呼び、Djangoでも標準機能として提供されています。

今回作成しているプロジェクト内でもすでにDjangoのORMを利用していますが、本章ではORMの仕組みや使い方について、さらに詳しく学んでいきます。

11-1-1 ｜ ORMとは？

「ORM」とは、アプリケーション内で扱う「オブジェクト」と、データを保存する「リレーショナルデータベース」との対応関係を自動で処理してくれる仕組みです。たとえば、Pythonのクラスで定義したオブジェクトのデータを、対応するデータベースのテーブルに保存したり、逆にテーブルのデータを読み込んでオブジェクトとして扱ったりすることができます。このように、あらかじめ設定された対応ルールにもとづいて、データの保存や読み込みを自動で行ってくれるため、SQL文を直接書かなくてもデータベース操作ができるようになります（**図11.1**）。

図11.1 ORMのイメージ



11-1-2 ｜ ORMを使用する主な理由

Djangoでは、SQL文を直接記述せずに「ORM」を使用することが推奨されます。
ORMを使用するメリットを以下に示します。

- データベース操作が簡単
  Djangoの「ORM」を使うと、SQLを直接書かずにデータ操作が可能になります。データの取得・追加・更新・削除をPythonのコードで直感的に記述できるため、初心者でも扱いやすくなります。
- 異なるデータベースでも動く
  DjangoのORMを使えば、SQLite・MySQL・PostgreSQLなど異なるデータベースでも同じコードで動作します。データベースを変更する際も、設定を変更するだけで移行が可能になり、開発の柔軟性が向上します。
- コードが読みやすく、保守しやすい
  SQLを直接書く場合、複雑なクエリは可読性が低くなりますが、DjangoのORMを使うことで、データ操作をPythonのオブジェクトとして表現できます。これにより、コードの可読性が向上し、修正や追加がしやすくなります。
- セキュリティが強化される
  DjangoのORMは、ユーザー入力を適切に処理し、「SQLインジェクション(注1)」などのセキュリティリスクを低減します。手動でSQLを書くよりも、安全なデータ操作が保証されます。
- マイグレーション機能でデータベース管理が簡単
  Djangoの「マイグレーション機能」により、データベースのテーブル構造をPythonのコードで管理できます。モデル（データ構造）を変更しても、コマンド1つでデータベースの変更を適用できるため、開発効率が向上します。

11-2 データの取得（全件・1件・条件検索）

「ORM」の概要をイメージできたので、簡易なアプリケーションを作成していきたいと思います。作成するアプリはORMを利用した「教育アプリ」です。

11-2-1 ｜ アプリケーションの作成

まずはアプリケーションを作成しましょう。作業ワークスペース「work_django/myproject」

（注1）「SQLインジェクション」は、SQL文を悪用して不正なデータを入力し、データベースを攻撃して情報を盗んだり改ざんしたりする攻撃手法です。

APPENDIX 1

HTMLの効率的な管理（テンプレートの継承）

本書の最後に「Appendix（付録）」として3つの機能について説明します。まずは1点目「テンプレートの継承」についてです。

1-1 「テンプレートの継承」の概要

テンプレートの継承とは、「テンプレートの共通部分を再利用できる仕組み」です。Djangoでは、ヘッダーやフッターなどの全ページで使う共通部分を「親テンプレート」にまとめ、それを「子テンプレート」で使いながら必要な部分を追加・変更できます。

1-1-1 ｜ 何故テンプレートの継承を使用するのか？

Webサイトには、全ページで同じ「ヘッダー」や「フッター」など、繰り返し使う部分がたくさんあります。継承を使うことで、これらの共通部分を一箇所にまとめ、他のページで再利用できるため、効率的で管理が簡単になります。継承は、プログラミングの基本原則である「DRY（Don't Repeat Yourself）の原則：繰り返しをなくす」に従っています。

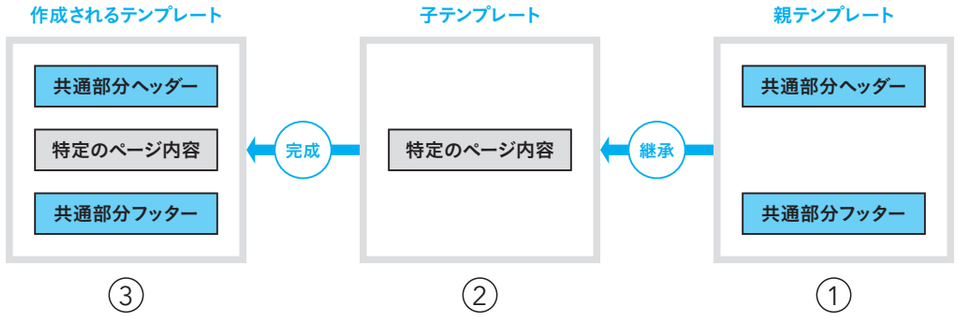
- DRYの考え方
  同じコードを何度も書かず、共通部分を1箇所にまとめる
- メリット
  共通部分を直すだけで、すべてのページに反映されるため、修正が簡単です。その結果、重複が減り、新しいページもスムーズに作成できます

1-1-2 ｜ テンプレートの継承の使い方

テンプレートの継承の基本的な使い方は**図A.1**になります。また、継承の重要事項を**表A.1**にまとめます。

① 親テンプレート（共通部分を定義）を作成する
② 子テンプレート（特定のページ内容を記述）を作成する
③ 共通部分を継承したテンプレートが自動で作成される

図A.1　テンプレートの継承



表A.1　継承の重要事項

タグ / 機能	説明
{% extends %}	個別テンプレート（子テンプレート）の最初の行に記述します。記述時は、継承元のテンプレート（親テンプレート）を指定します（例：{% extends "base.html" %} ※base.htmlが親テンプレートを指します）
{% block %}	子テンプレートで上書き可能な部分を定義します。親テンプレートで名前をつけて定義し、子テンプレートで同じ名前を使用することで上書きを可能にします（例：{% block content %}{% endblock %}　※contentは自分で定義した名前です）
{{ block.super }}	親テンプレートのblock内容を継承しつつ、内容を追加したい場合に使用します。既存の親内容に新しい内容を加える際に使用します

1-2 テンプレートの継承の実装

ここでは、実際に簡単なアプリケーションを作成しながら、テンプレート継承を確認します。

アプリケーションの作成

作業ワークスペース「work_django/myproject」フォルダを開き、以下のコマンドを実行します。

項目	対象
作業フォルダ	work_django/myproject
コマンド	python manage.py startapp appendixapp

アプリケーションの登録

作成したアプリケーション「appendixapp」をプロジェクト「myproject」で使用出来るように

APPENDIX

3 データ表示の改善（ページネーション）

「Appendix（付録）」の最後は「ページネーション」についてです。ページネーションとは、たくさんのデータを1ページに全て表示せず、複数のページに分けて表示する仕組みのことです。

3-1 「ページネーション」の概要

ページネーションの利点

データが多いと、すべてを1ページに表示すると画面がごちゃごちゃして見づらくなり、読み込みにも時間がかかることがあります。そこで「ページネーション」を使うと、データを複数ページに分割して表示できるので、以下のメリットがあります。

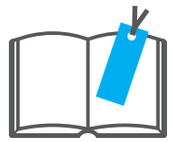
- 画面がスッキリする
  一度に表示するデータを減らし、見やすく整理できる。
- 表示が速くなる
  必要な分だけデータを取得するので、ロード時間を短縮できる。

図A.15　ページネーションのイメージ

「ページネーション」を現実世界で例える

本や雑誌を思い浮かべてみましょう。もしすべての内容が1ページに詰め込まれていたら、読みにくく、必要な情報を探すのも大変です。そこで、内容を複数のページに分けることで、読みやすく整理しています。「ページネーション」もこの考え方と同じで、データを適切な量ごとに区切ることで、画面をスッキリさせ、スムーズに情報へアクセスできるようにする仕組みです。

図A.16　ページネーションのイメージ



3-2 「ページネーション」の作成

「Paginator」とは？

Djangoは、データを自動的にページに分けるための便利なクラス「Paginator」を用意しています。「Paginator」を使用することで、たくさんのデータを指定した件数ごとに区切り、ページネーションを簡単に実現できます。

「Paginator」の引数について

Djangoの「Paginator」には、**表A.3**に示す引数があります。これらは、データを分割してページネーションを作成する際に指定します。

表A.3　Paginatorの引数一覧

引数	必須/任意	説明
object_list	必須	ページに分割したいデータのリストやクエリセットを指定します
per_page	必須	1ページに表示するデータの件数を指定します
orphans	任意	最後のページに表示されるアイテムがこの数未満の場合、それらを前ページに合併します。例：orphans=3の場合、最後のページに3件未満のアイテムがあれば、それらは前のページに表示されます
allow_empty_first_page	任意	最初のページが空（アイテムなし）であることを許可するかどうかを指定します。デフォルトはTrueで、空のページを許可します

以下の順番で「ページネーション」を使用したアプリケーションを作成していきます。