Section

1-1 FastAPIとは？

FastAPIは、WebアプリケーションやAPIを構築するためのPythonフレームワークです。この章では「FastAPI」の特徴を説明後、本書で実施するハンズオンの開発環境構築を行います。この章を読み終わった後に「FastAPIのイメージ」を何となく掴んで頂けたら幸いです。

1-1-1 フレームワークとは？

まず、フレームワークとは何でしょうか？フレームワークとは、簡単に言うとソフトウェアやアプリケーション開発を行う事を簡単にする「骨組み」です（**図1.1**）。フレームワークのメリットとして、フレームワークが必要最低限の機能を提供してくれるため、自分ですべての機能を作成する必要がなく、アプリケーションの開発にかかる時間とコストを削減できます。デメリットとして、フレームワークを利用する開発では、フレームワーク特有の使用方法（ルール）を理解する必要があります。

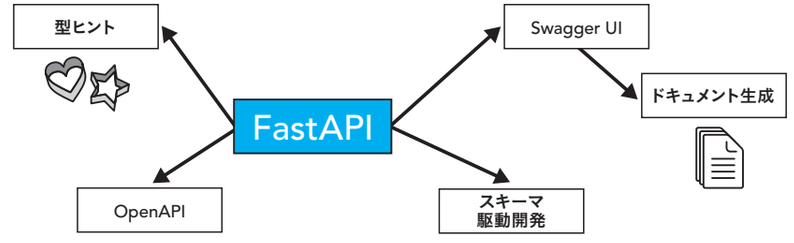
図1.1 フレームワークのイメージ

1-1-2 FastAPIの特徴

先ほども説明しましたが、FastAPIはWebアプリケーションやAPIを構築するためのPythonフレームワークです。大きな特徴は、「型ヒント」を用いた開発ができること、そして自動で「APIのドキュメント」を生成できることです。

以下に「マインドマップ(注1)」でFastAPIの特徴を示します（**図1.2**）。

図1.2 マインドマップ（**FastAPIの特徴**）



「マインドマップ」で記述した、FastAPIの各特徴についてそれぞれ説明します。

型ヒント

Pythonの「型ヒント」は、変数や関数の引数、戻り値がどのような型であるかを明示するための機能です。FastAPIでは、この型ヒントを使用しAPIが受け取るリクエストのデータ型や返すレスポンスのデータ型を定義します。これにより、データの検証やエディタの補完が自動で行われ、開発の効率が大きく向上します。

OpenAPI

「OpenAPI」は、RESTful APIの仕様を記述するための標準フォーマットです。FastAPIは型ヒントを基に、自動的にこの「OpenAPI」の仕様書を生成します。これにより、APIのエンドポイント(注2)、リクエスト、レスポンスの構造が明確に文書化され、APIの使用方法が容易に理解できるようになります。

SwaggerUI

「SwaggerUI」は、OpenAPIの仕様書をもとに、ブラウザ上で動作するインタラクティブ(注3)なAPIドキュメントを提供します。FastAPIでAPI開発をすると、SwaggerUIによるドキュメントが自動で用意され、開発者や利用者は、ドキュメントを見ながらリアルタイムでAPIのテストがで

（**注1**） マインドマップとは、アイデアや情報を視覚的に整理し、関連付けるためのツールです。中心となるテーマから枝分かれするようにキーワードや概念を書き出し、それらの間の関連を線でつなぎます。
（**注2**） 「エンドポイント」とは、インターネット上の特定の場所（アクセスポイント）を示します。簡単に言うとWebアプリケーションやサービスが「機能やデータ」を提供するためのアクセスポイントです。
（**注3**） インタラクティブとは、コンピューターやデバイスが、人の指示や行動に応じて反応することです。

Section

2-1 WebAPIの基礎知識

FastAPIを学習する前に、まずは「WebAPI」の基本を一緒に学んでいきましょう。「WebAPIって何?」というところから始めて、実際に自分でプログラムを作成します。APIがどんなものか、プログラムを通して理解しましょう。

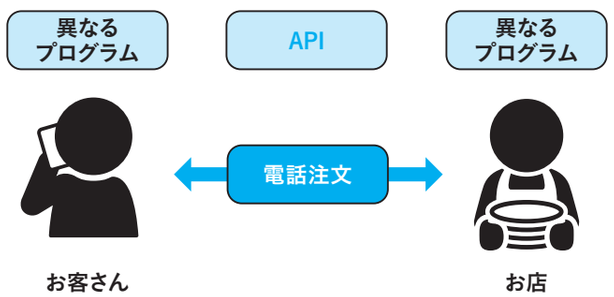
2-1-1 APIとは?

API（Application Programming Interface）とは、異なるソフトウェア間で情報をやり取りするためのルールや約束事、ざっくりイメージすると「方法」です。APIを使うことで、異なるプログラムが互いにコミュニケーションを取り合い、機能を利用し合うことができます。

現実世界で例える

APIを現実世界で例えると、デリバリーの「電話注文」のようなものです。「お客さん」が「お店」に電話で要求を伝えることで、料理を注文でき、サービスとして「食べ物」を受け取れます（**図2.1**）。

図2.1 現実世界での例（API）



2-1-2 WebAPIとは?

「WebAPI」は、インターネットを介して使用される「API」の一種です。Webの技術を利用して他のソフトウェアと通信を行います。HTTPプロトコル(注1)を使い、Webサーバーとクライアント（Webブラウザや他のWebアプリケーション）間で情報のやり取りを行うことができます。

WebAPIを利用することで、他Webサイトのデータを取得したり、SNSへ投稿したり、オンラインサービスの機能を使用することが可能になります。

WebAPIを使用している具体例

WebAPIの使用例を**表2.1**に示します。

表2.1 WebAPIの使用例

使用例	説明	具体例
天気予報アプリ	最新の天気情報を取得してユーザーに表示します	OpenWeatherMap APIを使用して、世界中の都市の天気予報を取得できます
ソーシャルメディアの統合	アプリ内で直接投稿を共有したり、ソーシャルメディアのフィードを表示します	X APIを使用することで、Webサイトを通さなくてもポストを予約・投稿ができたり、新たなソフトウェアやアプリの開発ができます
地図とナビゲーションサービス	アプリ内で地図を表示したり、経路案内を提供します	Google Maps APIを使用して、店舗の場所を表示または経路を案内できます

2-1-3 WebAPIプログラムの作成

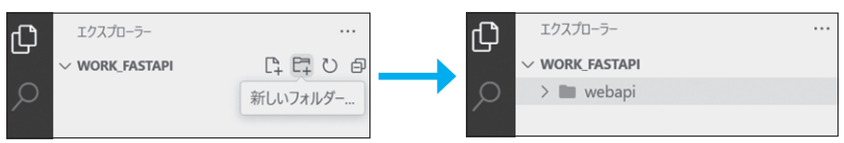
WebAPIを利用したプログラムを作成し、理解を深めましょう。

プロジェクトフォルダとファイルの作成

「**1-4-3 ハンズオン環境の作成**」で作成した「C:¥work_fastapi」ディレクトリに、今回作成するプログラム用のフォルダを作成します。

VSCode画面にて「新しいフォルダを作る」アイコンをクリックし、プロジェクトフォルダ「webapi」を作成します（**図2.2**）。

図2.2 フォルダの作成



（注1） HTTPプロトコルとは、インターネット上でWebページやデータをブラウザとサーバー間でやり取りするためのルールのことです。

Section

2-3 Swagger UIによるドキュメント生成

FastAPIは、「Swagger UI」を活用して自動的にドキュメントを生成します。開発サーバーが動いているとき、指定されたURLにアクセスするだけで、ドキュメントを簡単に閲覧できます。「Swagger UI」で作成されたドキュメントを体験しましょう。

2-3-1 Swagger UIとは？

Swagger UIは、FastAPIの強力な機能の１つで、APIがどのように動作するかを理解しやすくするために、エンドポイント、リクエストの方法、そしてパラメータなどを文書化し、テストできるインターフェースを提供します。

表示する

ターミナルでカレントディレクトリを「fastapi_hello」に合わせ、以下のコマンドを実行し「Uvicorn」を起動します。

```
uvicorn main:app --reload
```

ブラウザのアドレスバーに「http://127.0.0.1:8000/docs」を入力することで「Swagger UI」にアクセスできます（**図2.11**）。

図2.11 Swagger UI①

画面項目について**表2.4**に示します。

表2.4 Swagger UI項目

項目	説明
GET / Get Hello	GETかつエンドポイント「'/'」で呼ばれる「ルーティング」を示します。 Get Helloはget_hello関数を示しています

※FastAPIのルーティングは、Webアプリケーションにおける URLのパスに対応する特定の関数を割り当てる仕組みのことです。

図2.12の画面枠をクリックして、詳細表示にします。「Parameters」と「Responses」の項目が表示されます（**図2.13**、**図2.14**）。

図2.12 Swagger UI②

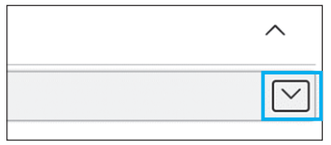


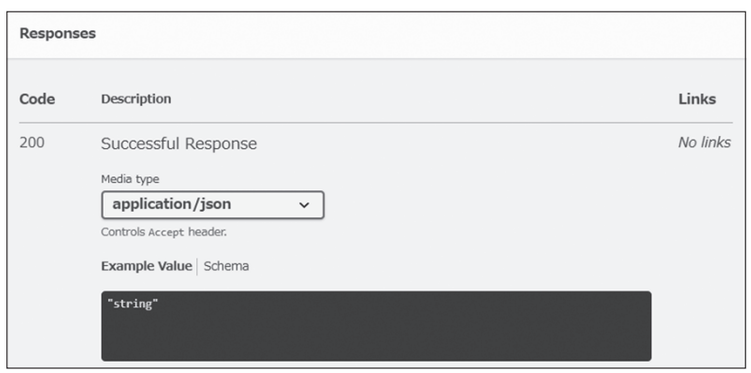
図2.13 Swagger UI③

「Parameters」画面項目について**表2.5**に示します。

表2.5 「Parameters」画面項目

項目	説明
Parameters	このリクエストにはパラメータが必要ないことを示しています（「No parameters」と記載）
Try it out	このボタンを押すと、実際にリクエストを試すことができます

図2.14 Swagger UI④



Section

3-3 型ヒントの使用方法（Annotated）

ここでは、型ヒントに追加情報を付け加える「Annotated」について説明します。Annotatedを使うと、型ヒントに追加情報を付け加えることができます。これは、ただの型を示すだけでなく、その型がどのような条件を満たすべきか、またはどのように使われるべきかという追加の説明を付けたいときに役立ちます。

3-3-1 Annotatedとは？

例えば、ある関数が受け取るパラメータが文字列であることはわかっていても、その文字列が特定のフォーマット（例えば、メールアドレス）を満たしている必要があるとします。Annotatedを使用して、そのような情報を「型ヒント」に付け加えることができます。

例

```
from typing import Annotated

def process_email(email: Annotated[str, "有効なメールアドレスの必要がある"]):
    # emailを処理するコード
```

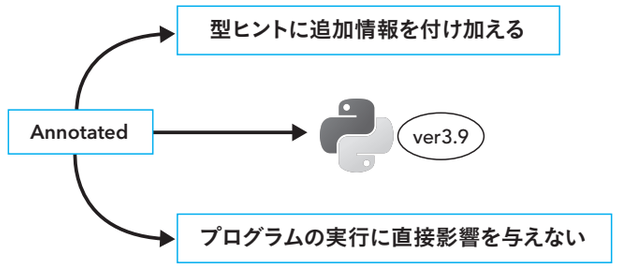
「from typing import Annotated」をすることで「Annotated」を使用できます。これはPythonの「型ヒント」で変数や関数の引数に追加の情報や制約を指定するために使用される、Python 3.9以降で導入された機能です。

この例では、emailパラメータは文字列型ですが、「Annotated」を使用して「有効なメールアドレスの必要がある」という注釈を付けています。この注釈により、他のプログラマーがこのコードを見た時に、ただの文字列ではなく、メールアドレスとしての文字列が期待されていることを明確に伝えるのに役立ちます。

Annotated型は実際のプログラムの実行には影響しませんが、コードを読む人がより多くの情報を得られるようになります。また、様々なツールやライブラリがこの追加情報を使用して、より強力なチェックを行ったり、コードを自動的に処理したりすることが可能になります。

マインドマップを用いて、「Annotated」の内容を整理します（**図3.6**）。

図3.6 マインドマップ（Annotated型）



3-3-2 Annotated型を使用するプログラムの作成

プロジェクトフォルダとファイルの作成

「**1-4-3 ハンズオン環境の作成**」で作成した「C:¥work_fastapi」ディレクトリに、今回作成するプログラム用のプロジェクトフォルダを作成します。

VSCode画面にて「新しいフォルダを作る」アイコンをクリックし、フォルダ「annotated」を作成し、作成したフォルダを選択後「新しいファイルを作る」アイコンをクリックし、ファイル「main.py」を作成します。

コードを書く

作成した「main.py」に**リスト3.3**のコードを記述します。

リスト3.3 main.py

```
001: from typing import Annotated
002:
003: # 引数で渡された整数値が指定された範囲内にあるかを
004: # チェックする関数
005: # 引数：数値型（Annotated）
006: # 戻り値：None
007: def process_value(
008:     value: Annotated[int, "範囲: 0 <= value <= 100"]
009:     )-> None:
010:     # 値が指定された範囲内にあるかチェックする
011:     if 0 <= value <= 100:
012:         # 値が範囲内の場合の処理
013:         print(f"受け取った値は範囲内です: {value}")
014:     else:
015:         # 値が範囲外の場合の処理
016:         raise ValueError(f"範囲外の値です。受け取った値: {value}")
017:
018: # ============================================================
```

Section

4-1 リクエスト処理（パスパラメータ）

「パスパラメータ」は、WebAPIにおけるリクエストの重要な処理で、URLの一部として指定される値のことを指します。これは、WebAPIで特定のリソース（Web上でアクセスまたは操作できるデータやサービス）を指定するために使用されます。FastAPIでは、関数の引数をパスパラメータとして受け取ることができ、エンドポイントの定義において、パスの一部に「{ }」を使用してパラメータ名を指定します。

4-1-1 パスパラメータの基本

パスパラメータは、Webアドレス（URL）の一部を変数として使用し、特定の情報やデータを指定するために使用されます。パスパラメータを使用することで、同じURLパターン内で異なるデータを取得できます。

例

```
@app.get("/users/{user_id}")
async def get_user(user_id: int):
    # 例えば、データベースからuser_idに対応するユーザー情報を取得する処理を行う
```

この例では、「/users/123」のようにアクセスすると、user_idに123が入り、そのIDに対応するユーザー情報を取得する処理が実行されます。{user_id}はパスパラメータで、URLの一部としてユーザーIDを受け取り、そのIDを使用して特定のユーザー情報にアクセスします。「パスパラメータ」と「関数の引数」を同名にすることで、URLから取得した値を直接関数の引数として渡すことができます。

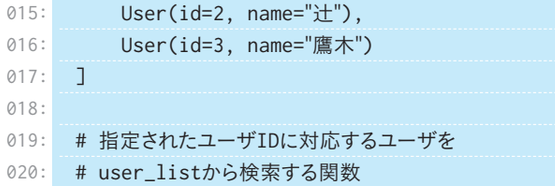
4-1-2 FastAPIプログラム（パスパラメータ）の作成

プロジェクトフォルダとファイルの作成

「1-4-3 ハンズオン環境の作成」で作成した「C:¥work_fastapi」ディレクトリに、今回作成するプログラム用のプロジェクトフォルダを作成します。

VSCode画面にて「新しいフォルダを作る」アイコンをクリックし、フォルダ「fastapi_path_parameter」を作成し、作成したフォルダを選択後「新しいファイルを作る」アイコンをクリックし、ファイル「main.py」と「data.py」を作成します（**図4.1**）。

図4.1 フォルダとファイルの作成



コードを書く

作成した「data.py」に**リスト4.1**のコードを記述します。

リスト4.1 data.py

```
from typing import Optional

# Userクラス
# ユーザのIDと名前を属性として持つ
class User:
    def __init__(self, id: int, name: str):
        # ユーザId
        self.id = id
        # ユーザ名
        self.name = name

# ダミーデータベースとして機能するユーザリスト
user_list = [
    User(id=1, name="内藤"),
    User(id=2, name="辻"),
    User(id=3, name="鷹木")
]

# 指定されたユーザIDに対応するユーザを
# user_listから検索する関数
# 引数：ユーザID（整数）
# 戻り値：Userオブジェクトまたは None（見つからない場合）
def get_user(user_id: int) -> Optional[User]:
    for user in user_list:
        if user.id == user_id:
            # 指定されたIDを持つユーザが見つかった場合
            # そのユーザを返す
            return user
    # ユーザが見つからない場合はNoneを返す
    return None
```

Section

5-3 CRUDアプリケーションの作成

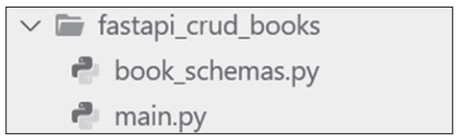
先ほど紹介した「書籍情報」を扱うエンドポイントを参考にして、CRUDアプリケーションを作成しましょう。また、「Pydantic」を利用してリクエストとレスポンスで使用するデータの構造も考慮しながらソースコードを作成していきましょう。初心者にもわかりやすいようにステップバイステップで説明します。

5-3-1 アプリケーションの作成

「1-4-3 ハンズオン環境の作成」で作成した「C:¥work_fastapi」ディレクトリに、今回作成するプログラム用のプロジェクトフォルダを作成します。

VSCode画面にて「新しいフォルダを作る」アイコンをクリックし、フォルダ「fastapi_crud_books」を作成し、作成したフォルダを選択後「新しいファイルを作る」アイコンをクリックし、ファイル「main.py」、「book_schemas.py」を作成します（**図5.6**）。

図5.6 フォルダとファイルの作成



ファイルの説明

ファイル名	説明
book_schemas.py	書籍のデータ構造を定義するスキーマを含みます。BookSchemaは基本情報、BookResponseSchemaはレスポンス用にIDも含めた情報を扱います
main.py	FastAPIを使用したAPIのエンドポイントを定義しています。書籍のCRUD処理（登録、取得、更新、削除）の操作をBookSchemaとBookResponseSchemaを使用して行います

スキーマとは？

「スキーマ」は、データ構造の定義や形式を規定するための枠組みです（**図5.7**）。データがどのように構成されるべきか、どのような型を持つべきかといった情報を提供し、データの整合性や互換性を保証するために使用されます。APIやデータベースなど、データを扱うさまざまな場所でスキーマが活用されています。

図5.7 スキーマ

コードを書く

作成した「book_schemas.py」に**リスト5.1**のコードを記述します。このソースコードは、書籍に関する情報を扱うためにPydanticスキーマをBaseModelを継承したクラスで定義します。

リスト5.1 book_schemas.py

```
001: from pydantic import BaseModel
002:
003: # 書籍の作成と更新に使用するスキーマ
004: class BookSchema(BaseModel):
005:     # タイトル
006:     title: str
007:     # カテゴリ
008:     category: str
009:
010: # レスポンス用のスキーマには
011: # 書籍スキーマを継承してidも含める
012: class BookResponseSchema(BookSchema):
013:     # ID
014:     id: int
```

4～8行目「BookSchema」クラスは書籍の作成や更新に使用されるスキーマです。タイトルとカテゴリーという基本的な情報を保持します。

12～14行目「BookResponseSchema」クラスは4～8行目で作成した「BookSchema」を継承し、追加で書籍のIDも保持することで、「レスポンスデータ」に適した形を提供しています。これにより、APIを介して書籍データを扱う際に、一貫性のあるデータ構造を維持することができます。

作成した「main.py」に**リスト5.2**～**リスト5.7**のコードを記述します。このソースコードは、FastAPIを使用したシンプルな書籍管理のCRUD処理を行うAPIです。「BookSchema」を使用し

Section 6-2

FastAPIでの非同期処理

FastAPIを使用した非同期処理の例として、外部APIからデータを取得するシンプルなエンドポイントを考えてみましょう。ここでは「httpx」を使用して非同期にHTTPリクエストを行い、レスポンスを返すプログラムを作成します。

6-2-1 httpxのインストール

「httpx」はPythonでHTTPリクエストを送るための非同期対応のライブラリです。requestsライブラリに似ていますが、httpxは非同期プログラミングに対応しており、「async / await」構文を使用して非同期にHTTPリクエストを行うことができます。これにより、Webアプリケーションや API クライアントなどで効率的な非同期通信を実現できます。

pipコマンド

仮想環境：fastapi_envに「httpx」をpipコマンドを使用してライブラリをインストールしましょう。

```
pip install httpx
```

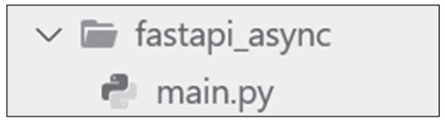
6-2-2 「httpx」を使用するプログラムの作成

プロジェクトフォルダとファイルの作成

「1-4-3 ハンズオン環境の作成」で作成した「C:¥work_fastapi」ディレクトリに、今回作成するプログラム用のプロジェクトフォルダを作成します。

VSCode画面にて「新しいフォルダを作る」アイコンをクリックし、フォルダ「fastapi_async」を作成し、作成したフォルダを選択後「新しいファイルを作る」アイコンをクリックし、ファイル「main.py」を作成します（**図6.7**）。

図6.7 フォルダとファイルの作成



コードを書く

作成した「main.py」に**リスト6.3**のコードを記述します。このコードは、「2-1-3 WebAPIプログラムの作成」で使用した住所検索APIサービスを利用します。FastAPIを使用して、北海道、東京、沖縄の「郵便番号」を基に住所情報を非同期に取得するWebアプリケーションです。

リスト6.3 main.py

```
001: from fastapi import FastAPI
002: import asyncio
003: import httpx
004:
005: app = FastAPI()
006:
007: # 郵便番号APIを利用する関数
008: # 郵便番号APIのURLを指定
009: # (例) 郵便番号「7830060」で検索する場合
010: # https://zipcloud.ibsnet.co.jp/api/search?zipcode=7830060
011: async def fetch_address(zip_code: str):
012:     async with httpx.AsyncClient() as client:
013:         response = await client.get(
014:             f"https://zipcloud.ibsnet.co.jp/api/search?zipcode={zip_code}"
015:             )
016:         return response.json()
017:
018: # エンドポイント
019: @app.get("/addresses/")
020: async def get_addresses():
021:     zip_codes = [
022:                 '0600000',  # 北海道
023:                 '1000001',  # 東京
024:                 '9000000'   # 沖縄
025:                 ]
026:     return await asyncio.gather(*(fetch_address(zip_code) for zip_code in zip_codes))
```

11～16行目は、「httpx.AsyncClient()」を使用して非同期に外部のAPI（zipcloudの郵便番号検索API）にリクエストを送り、特定の郵便番号に関する住所情報を取得します。「async with」を使用してHTTPクライアントを非同期に開始し、awaitでリクエストのレスポンスを待ちます。

13行目「client.get」は、httpxライブラリの「AsyncClient」クラスを使用して、指定されたURL

Section

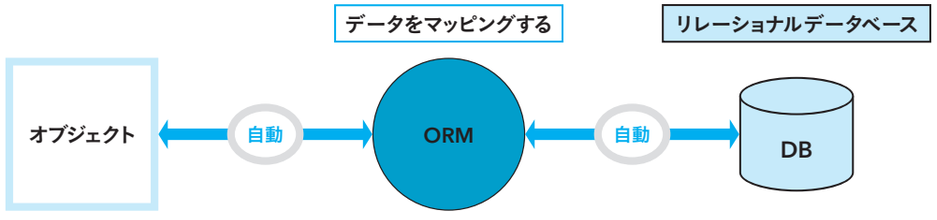
8-1 ORMとは?

現在のプログラム開発では、DBとのアクセス処理には「O/Rマッパー」というフレームワークを使用することが一般的です。Pythonでは、「ORM（Object Relational Mapper）」と呼ばれます。本文では「ORM」について簡単に説明してから、PythonのORMの1つである「SQLAlchemy」を使用してプログラムを作成する方法を説明します。

8-1-1 ORMの概要

「ORM」とは、アプリケーションで扱う「O：オブジェクト」と「R：リレーショナルデータベース」とのデータをマッピングするものです。より詳しく説明すると、「ORM」はあらかじめ設定された「O：オブジェクト」と「R：リレーショナルデータベース」との対応関係情報に基づき、インスタンスのデータを対応するテーブルに書き出したり、データベースから値を読み込んでインスタンスに代入したりする操作を自動的に行います（**図8.1**）。

図8.1 ORMのイメージ



8-1-2 SQLAlchemyとは?

「SQLAlchemy（エスキューエルアルケミー）」とはPythonでよく利用される「ORM」です（**図8.2**）。「SQLAlchemy」を使用するメリットには以下のようなものがあります。

○ SQLの記述が不要

SQLAlchemyを使用すると、Pythonのオブジェクト操作だけでデータベースを扱えます。そのため、複雑なSQLコマンドを直接記述する必要がなく、プログラミングでデータベースを直感的かつ簡単に扱うことができます。これは、データベースやSQLの知識が少ない人でもコードが書きやすくなるという利点です。

○ ORMのクロスプラットフォーム性

ORM（オブジェクトリレーショナルマッピング）を利用することで、異なるデータベースシステム間での移植性が向上します。これにより、同一のコードを異なるデータベースで使用することが可能になり、開発の効率が大幅に向上します。

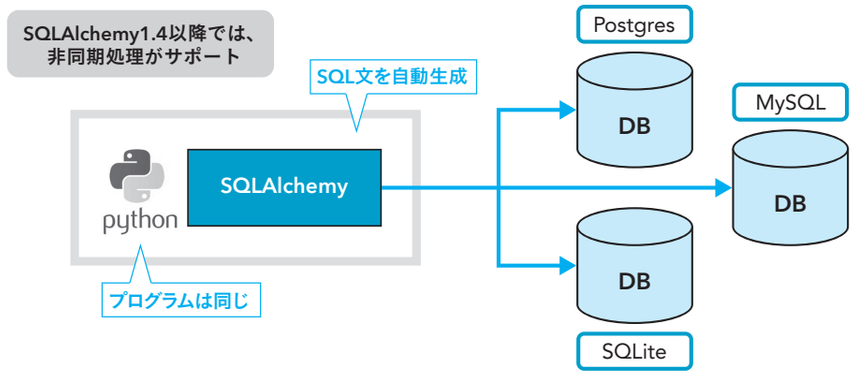
○ セッション管理

SQLAlchemyはセッション管理を行うことができ、データベースへの接続やトランザクションの管理を自動化します。これにより、トランザクションが適切に管理され、データの整合性が保たれるため、安全にデータベース操作が行えます。

○ 非同期処理のサポート

SQLAlchemy 1.4以降では、非同期処理がサポートされており、データベース操作の速度が向上しました。これにより、アプリケーション全体のパフォーマンスが改善され、ユーザー体験が向上します。

図8.2 SQLAlchemyのイメージ



インストール

仮想環境：fastapi_envに、pipコマンドを実行し「SQLAlchemy」をインストールします（**図8.3**）。

```
pip install sqlalchemy
```

図8.3 SQLAlchemyのインストール

```
(fastapi_env) C:\work_fastapi>pip install sqlalchemy
Collecting sqlalchemy
  Downloading SQLAlchemy-2.0.29-cp312-cp312-win_amd64.whl.metadata (9.8 kB)
```

Section

10-1 スキーマ駆動開発

この章からは、前章まで学習した内容を活用し、Webアプリケーションを作成していきます。作成方法は「スキーマ駆動開発」を使用します。

10-1-1 スキーマ

「1-1-2 FastAPIの特徴」でスキーマ駆動開発の概要を説明しましたが、ここでは深掘りして説明していきます。

スキーマとは？

スキーマ駆動開発における「スキーマ」とは、データ構造やAPIの形式などを定義した「設計図」のことです。これにより、データがどのように構築され、どのように通信されるべきかを明確に示します。例えば、Web APIの場合、どのようなデータが「リクエスト」と「レスポンス」で送受信されるかを事前に定義します。

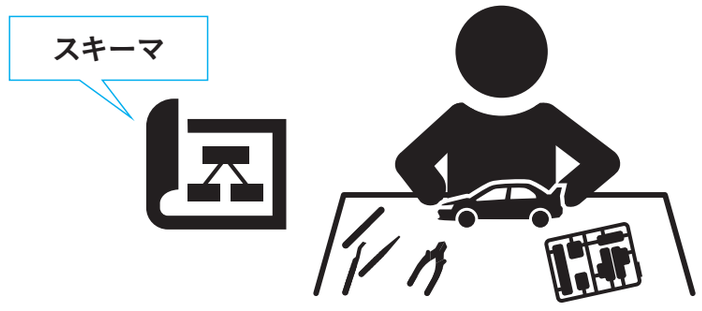
図10.1 スキーマ（再掲）

スキーマを現実世界で例える

スキーマを現実世界で例えると、「モノ作りの手順書」みたいなものです。プラモデルを作るときに使う説明書を思い浮かべてみてください。説明書には、どのパーツをどの順番でどこにつけるのかが書いてあります。

ITの世界でのスキーマも同じで、リクエストのスキーマでは「どんなデータを送るか」、レスポンスのスキーマでは「どんなデータが返ってくるか」が定義されています。これにより、クライアントとサーバーは、お互いにどんな情報をやり取りすべきかがわかります。開発者はこのスキーマを基に、正確に通信を行うシステムを構築していきます。

図10.2 スキーマ（現実世界の例）



スキーマがあると何が嬉しいか？

型ヒントを通じたドキュメント化

型ヒントを使用して関数や変数の期待される型を明示することで、コードのドキュメントとして機能し、使い方が理解しやすくなります。ドキュメントは「FastAPI」経由で自動生成されます。

型チェックによるエラーの早期発見

Pythonの型ヒントを使用すると、開発環境や静的型チェッカーが型の不一致を検出して、コードを実行する前にエラーを見つけ出すことができます。

10-1-2 スキーマ駆動開発

スキーマ駆動開発とは？

スキーマ駆動開発は、事前にAPIの設計図（スキーマ）を作り、その設計に基づいてバックエンド（データを処理する側）とフロントエンド（ユーザーが操作する画面側）の開発を同時に進める手法です。OpenAPIのような仕様書を使うことで、開発の初期段階からAPIの動きを明確にし、フロントエンドの開発者が実際のデータが利用可能になる前に画面設計を進めることができます。

つまりスキーマ駆動開発を使用すると、プロジェクトの初期段階でAPIのスキーマ（設計図）を定義し、そのスキーマに基づいて開発を進めることが可能です。これにより、開発チーム間での認識齟齬を防ぎ、「フロントエンド」と「バックエンド」がスムーズに連携できます。

一方、スキーマ駆動開発を使用しない場合、スキーマを事前に定義せずに開発を進めます。これにより、柔軟に開発を行うことができる反面、チーム間での認識のずれや、「フロントエンド」と「バックエンド」の接続での問題が発生しやすくなります。また、ドキュメントの整合性を保つために、修正作業などの余分な努力をする可能性が高くなります。

Section
11-2 CRUD処理の作成

ここでは、データベース操作のロジック（CRUD：作成、読み取り、更新、削除）を含む処理を作成していきます。

11-2-1 非同期CRUD処理

CRUD処理の作成

作成したフォルダ：cruds→ファイル：memo.pyに**リスト11.8**～**リスト11.13**のコードを記述します。このコードは、非同期でメモのデータベース操作を行うための基本的なCRUD関数を提供します。各関数は非同期で動作し、データベースとのやり取りを効率的に行います。これにより、アプリケーションが効率的にデータベース操作を行うことができます。ファイルが大変長くなるため分割して説明します。

既に説明させて頂いていますが、ソースコードについては「技術評論社のサポートページ」からリストや完成プロジェクトをダウンロード可能です。初回の学習時には是非、サポートページからダウンロードして有効活用してください。なお、今まで提供してきたソースコードは、ビギナーがわかりやすいようにコメントを付与していますが、特に「ドキュメントストリング（docstring）」を意識していませんでした。ドキュメントストリングは、Pythonの関数やクラス、モジュールに追加される特別なコメントで、コードの説明を提供します。これにより、コードの使用方法や動作を他の開発者（や自分自身）が簡単に理解できるようになります。このファイルではドキュメントストリングも記述していきます。

モジュールのインポート

リスト11.8は、SQLAlchemyとFastAPIを使用してデータベースとやり取りするために「必要なモジュールのインポート」です。各モジュールについて**表11.7**に示します。

リスト11.8 memo.py①（インポート）

```
001: from sqlalchemy import select
002: from sqlalchemy.ext.asyncio import AsyncSession
003: import schemas.memo as memo_schema
004: import models.memo as memo_model
005: from datetime import datetime
006:
```


表11.7 モジュール詳細

モジュール／クラス	説明
sqlalchemy	SQLAlchemyはPythonのORM（オブジェクト関係マッピング）ライブラリで、データベース操作を容易にします
select	SQLAlchemyの関数で、データベースからデータを選択（取得）するために使用されます
sqlalchemy.ext.asyncio	非同期操作をサポートするSQLAlchemyのモジュールです
AsyncSession	非同期データベースセッションを扱うクラスです
schemas.memo	データの構造（スキーマ）を定義するためのモジュールです
models.memo	データベースのテーブルに対応するモデルを定義するモジュールです
datetime	日時を扱うためのPython標準ライブラリです

「新規登録」関数

memo.pyに新規登録を行うコードを追加します（**リスト11.9**）。7～29行目「新しいメモを登録する」関数です。各処理を分けて説明していきます。

リスト11.9 memo.py②（新規登録関数）

```
007: # ================================================
008: # 非同期CRUD処理
009: # ================================================
010: # 新規登録
011: async def insert_memo(
012:         db_session: AsyncSession,
013:         memo_data: memo_schema.InsertAndUpdateMemoSchema) -> memo_model.Memo:
014:     """
015:         新しいメモをデータベースに登録する関数
016:         Args:
017:             db_session (AsyncSession): 非同期DBセッション
018:             memo_data (InsertAndUpdateMemoSchema): 作成するメモのデータ
019:         Returns:
020:             Memo: 作成されたメモのモデル
021:     """
022:     print("=== 新規登録：開始 ===")
023:     new_memo = memo_model.Memo(**memo_data.model_dump())
024:     db_session.add(new_memo)
025:     await db_session.commit()
026:     await db_session.refresh(new_memo)
027:     print(">>> データ追加完了")
028:     return new_memo
029:
```

11～13行目は「関数定義と引数と戻り値」です。詳細を**表11.8**に示します。