リクエストボディ

クライアント (ブラウザなど) からAPIにデータを送信する必要があるとき、データを **リクエストボディ (request body)** として送ります。

リクエスト ボディはクライアントによってAPIへ送られます。**レスポンス** ボディはAPIがクライアントに送るデータです。

APIはほとんどの場合 **レスポンス** ボディを送らなければなりません。しかし、クライアントは必ずしも **リクエスト** ボディを送らなければいけないわけではありません。

リクエスト ボディを宣言するために Pydantic モデルを使用します。そして、その全てのパワーとメリットを利用します。

!!! info "情報" データを送るには、 `POST` (もっともよく使われる)、 `PUT` 、 `DELETE` または `PATCH` を使うべきです。

`GET` リクエストでボディを送信することは、仕様では未定義の動作ですが、`FastAPI` でサポートされており、非常に複雑な（極端な）ユースケースにのみ対応しています。

非推奨なので、`Swagger UI`を使った対話型のドキュメントには`GET`のボディ情報は表示されません。さらに、中継するプロキシが対応していない可能性があります。

Pydanticの `BaseModel` をインポート

ます初めに、 `pydantic` から `BaseModel` をインポートする必要があります:

```
{!../../../docs_src/body/tutorial001.py!}
```

データモデルの作成

そして、 `BaseModel` を継承したクラスとしてデータモデルを宣言します。

すべての属性にpython標準の型を使用します:

```
{!../../../docs_src/body/tutorial001.py!}
```

クエリパラメータの宣言と同様に、モデル属性がデフォルト値をもつとき、必須な属性ではなくなります。それ以外は必須になります。オプショナルな属性にしたい場合は `None` を使用してください。

例えば、上記のモデルは以下の様なJSON「 `オブジェクト` 」(もしくはPythonの `dict`) を宣言しています:

```
{
    "name": "Foo",
    "description": "An optional description",
    "price": 45.2,
    "tax": 3.5
}
```

... `description` と `tax` はオプショナル (デフォルト値は `None`) なので、以下のJSON「 `オブジェクト` 」も有効です:

```
{
    "name": "Foo",
    "price": 45.2
}
```

パラメータとして宣言

パスオペレーション に加えるために、パスパラメータやクエリパラメータと同じ様に宣言します:

```
{!../../../docs_src/body/tutorial001.py!}
```

...そして、作成したモデル `Item` で型を宣言します。

結果

そのPythonの型宣言だけで **FastAPI** は以下のことを行います:

- リクエストボディをJSONとして読み取ります。
- 適当な型に変換します（必要な場合）。
- データを検証します。
    - データが無効な場合は、明確なエラーが返され、どこが不正なデータであったかを示します。
- 受け取ったデータをパラメータ `item` に変換します。
    - 関数内で `Item` 型であると宣言したので、すべての属性とその型に対するエディタサポート（補完など）をすべて使用できます。
- モデルのJSONスキーマ定義を生成し、好きな場所で使用することができます。
- これらのスキーマは、生成されたOpenAPIスキーマの一部となり、自動ドキュメントのUIに使用されます。

自動ドキュメント生成

モデルのJSONスキーマはOpenAPIで生成されたスキーマの一部になり、対話的なAPIドキュメントに表示されます:

そして、それらが使われる *パスオペレーション* のそれぞれのAPIドキュメントにも表示されます:

エディターサポート

エディターによる型ヒントと補完が関数内で利用できます (Pydanticモデルではなく `dict` を受け取ると、同じサポートは受けられません):

型によるエラーチェックも可能です:

これは偶然ではなく、このデザインに基づいてフレームワークが作られています。

全てのエディターで機能することを確認するために、実装前の設計時に徹底的にテストしました。

これをサポートするためにPydantic自体にもいくつかの変更がありました。

上記のスクリーンショットはVisual Studio Codeを撮ったものです。

しかし、PyCharmやほとんどのPythonエディタでも同様なエディターサポートを受けられます:

!!! tip "豆知識" PyCharmエディタを使用している場合は、Pydantic PyCharm Pluginが使用可能です。

以下のエディターサポートが強化されます:

* 自動補完
* 型チェック
* リファクタリング
* 検索
* インスペクション

モデルの使用

関数内部で、モデルの全ての属性に直接アクセスできます:

```
{!../../../docs_src/body/tutorial002.py!}
```

リクエストボディ + パスパラメータ

パスパラメータとリクエストボディを同時に宣言できます。

FastAPI はパスパラメータである関数パラメータは**パスから受け取り**、Pydanticモデルによって宣言された関数パラメータは**リクエストボディから受け取る**ということを認識します。

```
{!../../../docs_src/body/tutorial003.py!}
```

リクエストボディ + パスパラメータ + クエリパラメータ

また、**ボディ**と**パス**と**クエリ**のパラメータも同時に宣言できます。

FastAPI はそれぞれを認識し、適切な場所からデータを取得します。

```
{!../../../docs_src/body/tutorial004.py!}
```

関数パラメータは以下の様に認識されます:

- パラメータが**パス**で宣言されている場合は、優先的にパスパラメータとして扱われます。
- パラメータが**単数型** (`int`、`float`、`str`、`bool` など)の場合は**クエリ**パラメータとして解釈されます。
- パラメータが **Pydantic モデル**型で宣言された場合、リクエスト**ボディ**として解釈されます。

!!! note "備考" FastAPIは、`= None` があるおかげで、`q` がオプショナルだとわかります。

`Optional[str]` の`Optional` はFastAPIでは使用されていません（FastAPIは`str`の部分のみ使用します）。しかし、`Optional[str]` はエディタがコードのエラーを見つけるのを助けてくれます。

Pydanticを使わない方法

もしPydanticモデルを使用したくない場合は、**ボディ**パラメータが利用できます。Body - Multiple Parameters: Singular values in body{.internal-link target=_blank}を確認してください。