パスパラメータ

Pythonのformat文字列と同様のシンタックスで「パスパラメータ」や「パス変数」を宣言できます:

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

パスパラメータ `item_id` の値は、引数 `item_id` として関数に渡されます。

しがたって、この例を実行して http://127.0.0.1:8000/items/foo にアクセスすると、次のレスポンスが表示されます。

```
{"item_id":"foo"}
```

パスパラメータと型

標準のPythonの型アノテーションを使用して、関数内のパスパラメータの型を宣言できます:

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

ここでは、 `item_id` は `int` として宣言されています。

!!! check "確認" これにより、関数内でのエディターサポート (エラーチェックや補完など) が提供されます。

データ変換

この例を実行し、ブラウザで http://127.0.0.1:8000/items/3 を開くと、次のレスポンスが表示されます:

```
{"item_id":3}
```

!!! check "確認" 関数が受け取った（および返した）値は、文字列の `"3"` ではなく、Pythonの `int` としての `3` であることに注意してください。

したがって、型宣言を使用すると、**FastAPI**は自動リクエスト <abbr title="HTTPリクエストで受け取った文字列をPythonデータへ変換する">"解析"</abbr> を行います。

データバリデーション

しかしブラウザで http://127.0.0.1:8000/items/foo を開くと、次のHTTPエラーが表示されます:

```
{
    "detail": [
        {
            "loc": [
                "path",
                "item_id"
            ],
            "msg": "value is not a valid integer",
            "type": "type_error.integer"
```

```
            }
        ]
    }
```

これは、パスパラメータ `item_id` が `int` ではない値 `"foo"` だからです。

http://127.0.0.1:8000/items/4.2 で見られるように、intのかわりに `float` が与えられた場合にも同様なエラーが表示されます。

!!! check "確認" したがって、Pythonの型宣言を使用することで、**FastAPI**はデータのバリデーションを行います。

表示されたエラーには問題のある箇所が明確に指摘されていることに注意してください。

これは、`API`に関連するコードの開発およびデバッグに非常に役立ちます。

ドキュメント

そしてブラウザで http://127.0.0.1:8000/docs を開くと、以下の様な自動的に生成された対話的なドキュメントが表示されます。

!!! check "確認" 繰り返しになりますが、Python型宣言を使用するだけで、**FastAPI**は対話的なAPIドキュメントを自動的に生成します（Swagger UIを統合）。

パスパラメータが整数として宣言されていることに注意してください。

標準であることのメリット、ドキュメンテーションの代替物

また、生成されたスキーマが OpenAPI 標準に従っているので、互換性のあるツールが多数あります。

このため、**FastAPI**自体が代替のAPIドキュメントを提供します（ReDocを使用）。これは、http://127.0.0.1:8000/redoc にアクセスすると確認できます。

同様に、互換性のあるツールが多数あります（多くの言語用のコード生成ツールを含む）。

Pydantic

すべてのデータバリデーションは Pydantic によって内部で実行されるため、Pydanticの全てのメリットが得られます。そして、安心して利用することができます。

`str` 、 `float` 、 `bool` および他の多くの複雑なデータ型を型宣言に使用できます。

これらのいくつかについては、チュートリアルの次の章で説明します。

順序の問題

path operations を作成する際、固定パスをもつ状況があり得ます。

`/users/me` から、現在のユーザに関するデータを取得するとします。

さらに、ユーザIDによって特定のユーザに関する情報を取得するパス `/users/{user_id}` ももつことができます。

path operations は順に評価されるので、 `/users/me` が `/users/{user_id}` よりも先に宣言されているか確認する必要があります:

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

それ以外の場合、 `/users/{users_id}` は `/users/me` としてもマッチします。値が「"me"」であるパラメータ `user_id` を受け取ると「考え」ます。

定義済みの値

*パスパラメータ*を受け取る *path operation* をもち、有効な*パスパラメータ*の値を事前に定義したい場合は、標準のPython `Enum` を利用できます。

`Enum` クラスの作成

`Enum` をインポートし、 `str` と `Enum` を継承したサブクラスを作成します。

`str` を継承することで、APIドキュメントは値が `文字列` でなければいけないことを知り、正確にレンダリングできるようになります。

そして、固定値のクラス属性を作ります。すると、その値が使用可能な値となります:

```
{!../../../docs_src/path_params/tutorial005.py!}
```

!!! info "情報" Enumerations (もしくは、enums)はPython 3.4以降で利用できます。

!!! tip "豆知識" "AlexNet"、"ResNet"そして"LeNet"は機械学習モデルの名前です。

*パスパラメータ*の宣言

次に、作成したenumクラスである `ModelName` を使用した型アノテーションをもつ*パスパラメータ*を作成します:

```
{!../../../docs_src/path_params/tutorial005.py!}
```

ドキュメントの確認

*パスパラメータ*の利用可能な値が事前に定義されているので、対話的なドキュメントで適切に表示できます:

Python*列挙型*の利用

*パスパラメータ*の値は*列挙型メンバ*となります。

*列挙型メンバ*の比較

これは、作成した列挙型 `ModelName` の*列挙型メンバ*と比較できます:

```
{!../../../docs_src/path_params/tutorial005.py!}
```

*列挙値*の取得

`model_name.value` 、もしくは一般に、 `your_enum_member.value` を使用して実際の値 (この場合は `str`) を取得できます。

```
{!../../../docs_src/path_params/tutorial005.py!}
```

!!! tip "豆知識" `ModelName.lenet.value` でも `"lenet"` 値にアクセスできます。

*列挙型メンバ*の返却

path operation から*列挙型メンバ*を返すことができます。JSONボディ（ `dict` など）でネストすることもできます。

それらはクライアントに返される前に適切な値 (この場合は文字列) に変換されます。

```
{!../../../docs_src/path_params/tutorial005.py!}
```

クライアントは以下の様なJSONレスポンスを得ます:

```
{
  "model_name": "alexnet",
  "message": "Deep Learning FTW!"
}
```

パスを含んだパスパラメータ

パス `/files/{file_path}` となる *path operation* を持っているとしましょう。

ただし、 `home/johndoe/myfile.txt` のようなパスを含んだ `file_path` が必要です。

したがって、URLは `/files/home/johndoe/myfile.txt` の様になります。

OpenAPIサポート

OpenAPIはテストや定義が困難なシナリオにつながる可能性があるため、内部に*パス*を含む*パスパラメータ*の宣言をサポートしていません。

それにも関わらず、Starletteの内部ツールのひとつを使用することで、**FastAPI**はそれが実現できます。

そして、パラメータがパスを含むべきであることを明示的にドキュメントに追加することなく、機能します。

パス変換

Starletteのオプションを直接使用することで、以下のURLの様な*パス*を含んだ、*パスパラメータ*の宣言ができます:

```
/files/{file_path:path}
```

この場合、パラメータ名は `file_path` です。そして、最後の部分 `:path` はパラメータがいかなるパスにもマッチすることを示します。

したがって、以下の様に使用できます:

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

!!! tip "豆知識" 最初のスラッシュ (`/`)が付いている `/home/johndoe/myfile.txt` をパラメータが含んでいる必要があります。

この場合、URLは `files` と `home` の間にダブルスラッシュ (`//`) のある、`/files//home/johndoe/myfile.txt` になります。

まとめ

簡潔で、本質的で、標準的なPythonの型宣言を使用することで、**FastAPI**は以下を行います:

- エディターサポート: エラーチェック、自動補完、など
- データ「解析」
- データバリデーション
- APIアノテーションと自動ドキュメント生成

そしてこれはたった一度宣言するだけです。

これは恐らく、(パフォーマンスを除いて) 他のフレームワークと比較したときの、**FastAPI**の主な目に見える利点です。