クエリパラメータと文字列の検証

FastAPI ではパラメータの追加情報とバリデーションを宣言することができます。

以下のアプリケーションを例にしてみましょう:

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

クエリパラメータ `q` は `Optional[str]` 型で、 `None` を許容する `str` 型を意味しており、デフォルトは `None` です。そのため、FastAPIはそれが必須ではないと理解します。

!!! note "備考" FastAPIは、 `q` はデフォルト値が `=None` であるため、必須ではないと理解します。

`Optional[str]` における `Optional` はFastAPIには利用されませんが、エディターによるより良いサポートとエラー検出を可能にします。

バリデーションの追加

`q` はオプショナルですが、もし値が渡されてきた場合には、**50文字を超えないこと**を強制してみましょう。

`Query` のインポート

そのために、まずは `fastapi` から `Query` をインポートします:

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

デフォルト値として `Query` を使用

パラメータのデフォルト値として使用し、パラメータ `max_length` を50に設定します:

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

デフォルト値 `None` を `Query(None)` に置き換える必要があるので、 `Query` の最初の引数はデフォルト値を定義するのと同じです。

なので:

```
q: Optional[str] = Query(None)
```

...を以下と同じようにパラメータをオプションにします:

```
q: Optional[str] = None
```

しかし、これはクエリパラメータとして明示的に宣言しています。

!!! info "情報" FastAPIは以下の部分を気にすることを覚えておいてください:

```Python
= None
```

もしくは:

```Python
= Query(None)
```

そして、 `None` を利用することでクエリパラメータが必須ではないと検知します。

`Optional` の部分は、エディターによるより良いサポートを可能にします。

そして、さらに多くのパラメータを `Query` に渡すことができます。この場合、文字列に適用される、 `max_length` パラメータを指定します。

```Python
q: str = Query(None, max_length=50)
```

これにより、データを検証し、データが有効でない場合は明確なエラーを表示し、OpenAPIスキーマの *path operation* にパラメータを記載します。

バリデーションをさらに追加する

パラメータ `min_length` も追加することができます:

```Python
{!../../../docs_src/query_params_str_validations/tutorial003.py!}
```

正規表現の追加

パラメータが一致するべき正規表現を定義することができます:

```Python
{!../../../docs_src/query_params_str_validations/tutorial004.py!}
```

この特定の正規表現は受け取ったパラメータの値をチェックします:

- `^`: は、これ以降の文字で始まり、これより以前には文字はありません。
- `fixedquery`: は、正確な `fixedquery` を持っています.
- `$`: で終わる場合、 `fixedquery` 以降には文字はありません.

もしこれらすべての **正規表現**のアイデアについて迷っていても、心配しないでください。多くの人にとって難しい話題です。正規表現を必要としなくても、まだ、多くのことができます。

しかし、あなたがそれらを必要とし、学ぶときにはすでに、 **FastAPI**で直接それらを使用することができます。

デフォルト値

第一引数に `None` を渡して、デフォルト値として使用するのと同じように、他の値を渡すこともできます。

クエリパラメータ `q` の `min_length` を `3` とし、デフォルト値を `fixedquery` としてみましょう:

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

!!! note "備考" デフォルト値を指定すると、パラメータは任意になります。

必須にする

これ以上、バリデーションやメタデータを宣言する必要のない場合は、デフォルト値を指定しないだけでクエリパラメータ `q` を必須にすることができます。以下のように:

```
q: str
```

以下の代わりに:

```
q: Optional[str] = None
```

現在は以下の例のように `Query` で宣言しています:

```
q: Optional[str] = Query(None, min_length=3)
```

そのため、 `Query` を使用して必須の値を宣言する必要がある場合は、第一引数に `...` を使用することができます:

```
{!../../../docs_src/query_params_str_validations/tutorial006.py!}
```

!!! info "情報" これまで `...` を見たことがない方へ: これは特殊な単一値です。Pythonの一部であり、"Ellipsis"と呼ばれています。

これは **FastAPI** にこのパラメータが必須であることを知らせます。

クエリパラメータのリスト / 複数の値

クエリパラメータを明示的に `Query` で宣言した場合、値のリストを受け取るように宣言したり、複数の値を受け取るように宣言したりすることもできます。

例えば、URL内に複数回出現するクエリパラメータ `q` を宣言するには以下のように書きます:

```
{!../../../docs_src/query_params_str_validations/tutorial011.py!}
```

そしてURLは以下です:

```
http://localhost:8000/items/?q=foo&q=bar
```

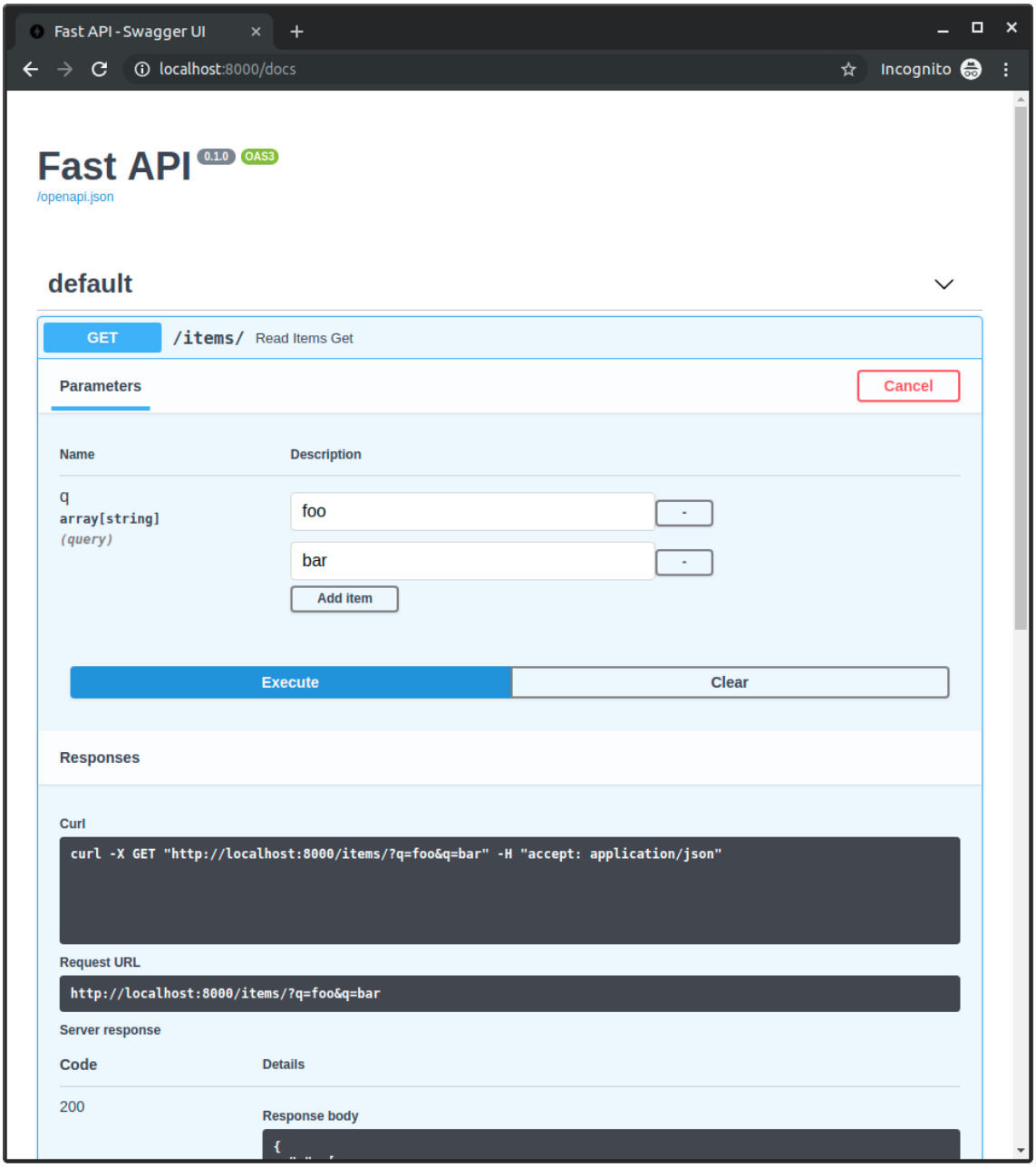
複数の *クエリパラメータ* の値 `q` （ `foo` と `bar` ）を *path operation関数* 内で *関数パラメータ* `q` としてPythonの `list` を受け取ることになります。

そのため、このURLのレスポンスは以下のようになります:

```
{
  "q": [
    "foo",
    "bar"
  ]
}
```

!!! tip "豆知識" 上述の例のように、 `list` 型のクエリパラメータを宣言するには明示的に `Query` を使用する必要があります。そうしない場合、リクエストボディと解釈されます。

対話的APIドキュメントは複数の値を許可するために自動的に更新されます。



デフォルト値を持つ、クエリパラメータのリスト / 複数の値

また、値が指定されていない場合はデフォルトの `list` を定義することもできます。

```
{!../../../docs_src/query_params_str_validations/tutorial012.py!}
```

以下のURLを開くと:

```
http://localhost:8000/items/
```

`q` のデフォルトは: `["foo", "bar"]` となり、レスポンスは以下のようになります:

```
{
  "q": [
    "foo",
    "bar"
  ]
}
```

`list` を使う

`List[str]` の代わりに直接 `list` を使うこともできます:

```
{!../../../docs_src/query_params_str_validations/tutorial013.py!}
```

!!! note "備考" この場合、FastAPIはリストの内容をチェックしないことを覚えておいてください。

例えば`List[int]`はリストの内容が整数であるかどうかをチェックします(そして、文書化します)。しかし`list`だけではそうしません。

より多くのメタデータを宣言する

パラメータに関する情報をさらに追加することができます。

その情報は、生成されたOpenAPIに含まれ、ドキュメントのユーザーインターフェースや外部のツールで使用されます。

!!! note "備考" ツールによってOpenAPIのサポートのレベルが異なる可能性があることを覚えておいてください。

その中には、宣言されたすべての追加情報が表示されていないものもあるかもしれませんが、ほとんどの場合、不足している機能はすでに開発の計画がされています。

`title` を追加できます:

```
{!../../../docs_src/query_params_str_validations/tutorial007.py!}
```

`description` を追加できます:

```
{!../../../docs_src/query_params_str_validations/tutorial008.py!}
```

エイリアスパラメータ

パラメータに `item-query` を指定するとします.

以下のような感じです:

```
http://127.0.0.1:8000/items/?item-query=foobaritems
```

しかし、 `item-query` は有効なPythonの変数名ではありません。

最も近いのは `item_query` でしょう。

しかし、どうしても `item-query` と正確に一致している必要があるとします...

それならば、 `alias` を宣言することができます。エイリアスはパラメータの値を見つけるのに使用されます:

```
{!../../../docs_src/query_params_str_validations/tutorial009.py!}
```

非推奨パラメータ

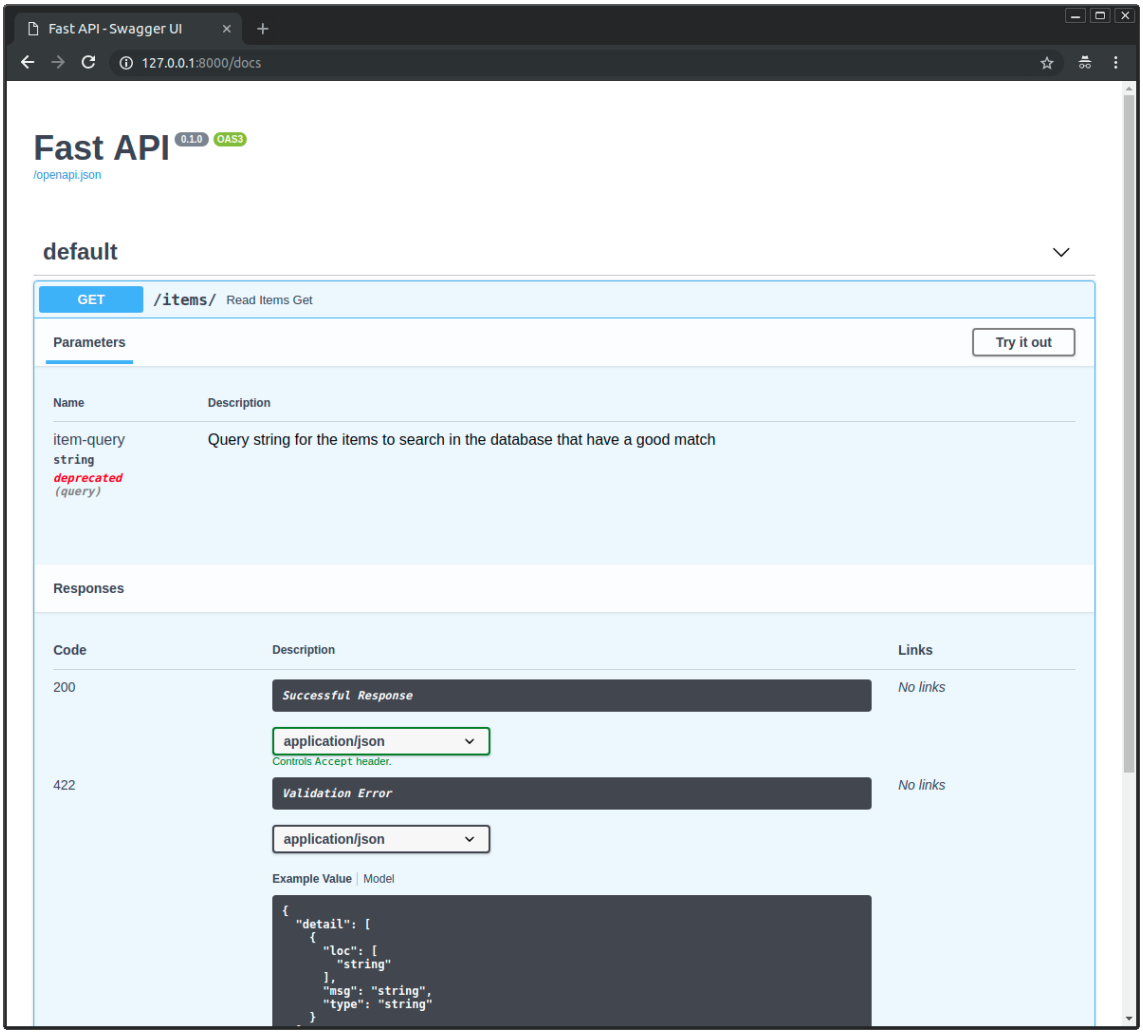
さて、このパラメータが気に入らなくなったとしましょう

それを使っているクライアントがいるので、しばらくは残しておく必要がありますが、ドキュメントには非推奨と明記しておきたいです。

その場合、 `Query` にパラメータ `deprecated=True` を渡します:

```
{!../../../docs_src/query_params_str_validations/tutorial010.py!}
```

ドキュメントは以下のようになります:



まとめ

パラメータに追加のバリデーションとメタデータを宣言することができます。

一般的なバリデーションとメタデータ:

- `alias`
- `title`
- `description`
- `deprecated`

文字列のためのバリデーション:

- `min_length`
- `max_length`
- `regex`

この例では、 `str` の値のバリデーションを宣言する方法を見てきました。

数値のような他の型のバリデーションを宣言する方法は次の章を参照してください。