路径参数

你可以使用与 Python 格式化字符串相同的语法来声明路径"参数"或"变量":

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

路径参数 `item_id` 的值将作为参数 `item_id` 传递给你的函数。

所以，如果你运行示例并访问 http://127.0.0.1:8000/items/foo，将会看到如下响应：

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

有类型的路径参数

你可以使用标准的 Python 类型标注为函数中的路径参数声明类型。

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

在这个例子中，`item_id` 被声明为 `int` 类型。

!!! check 这将为你的函数提供编辑器支持，包括错误检查、代码补全等等。

数据转换

如果你运行示例并打开浏览器访问 http://127.0.0.1:8000/items/3，将得到如下响应：

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

!!! check 注意函数接收（并返回）的值为 3，是一个 Python `int` 值，而不是字符串 `"3"`。

所以，`**FastAPI**` 通过上面的类型声明提供了对请求的自动`<abbr title="将来自 HTTP 请求中的字符串转换为 Python 数据类型">"解析"</abbr>`。

数据校验

但如果你通过浏览器访问 http://127.0.0.1:8000/items/foo，你会看到一个清晰可读的 HTTP 错误：

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

```
    ]
}
```

因为路径参数 `item_id` 传入的值为 `"foo"`，它不是一个 `int`。

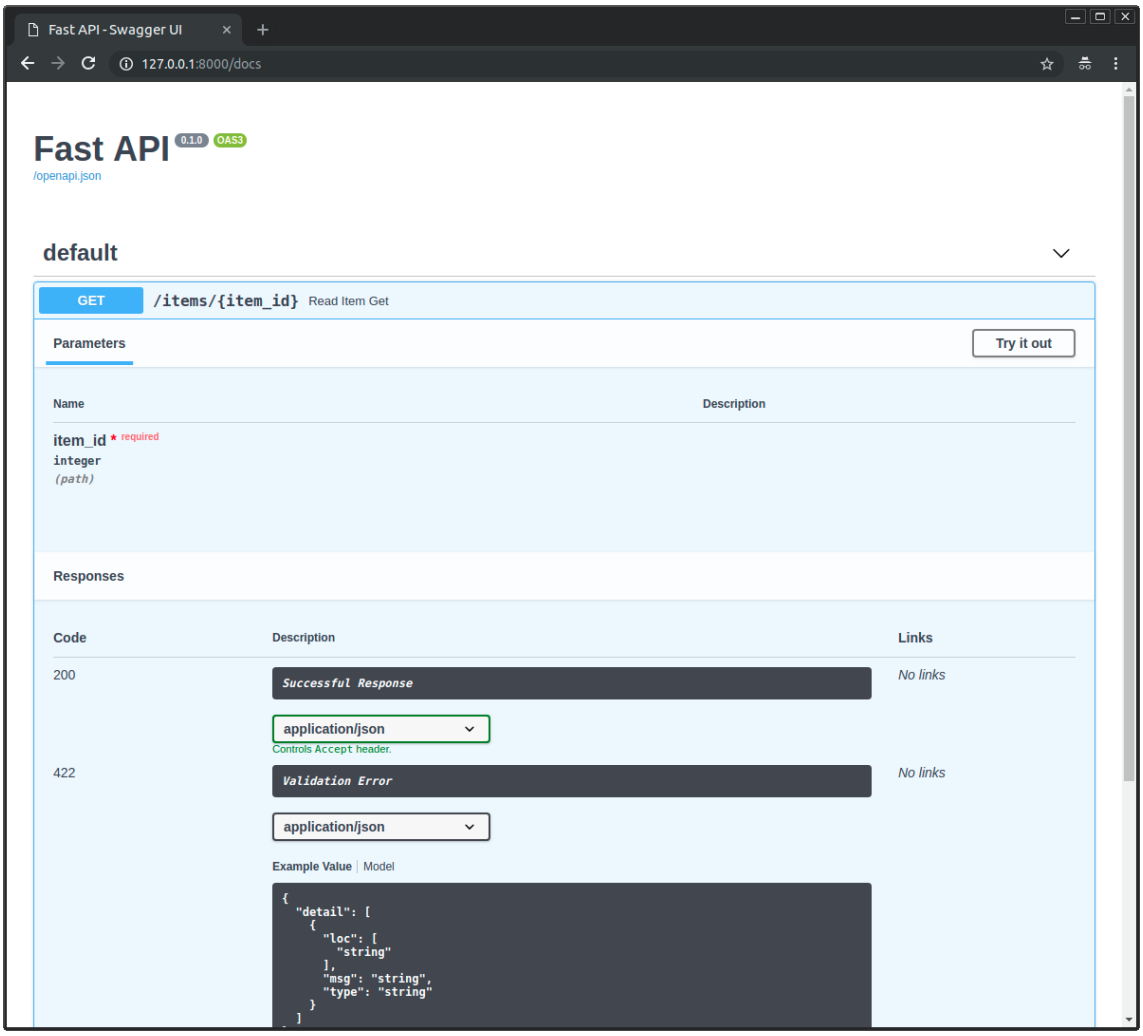
如果你提供的是 `float` 而非整数也会出现同样的错误，比如： http://127.0.0.1:8000/items/4.2

!!! check 所以，通过同样的 Python 类型声明，**FastAPI** 提供了数据校验功能。

> 注意上面的错误同样清楚地指出了校验未通过的具体原因。
>
> 在开发和调试与你的 `API` 进行交互的代码时，这非常有用。

文档

当你打开浏览器访问 http://127.0.0.1:8000/docs，你将看到自动生成的交互式 API 文档：



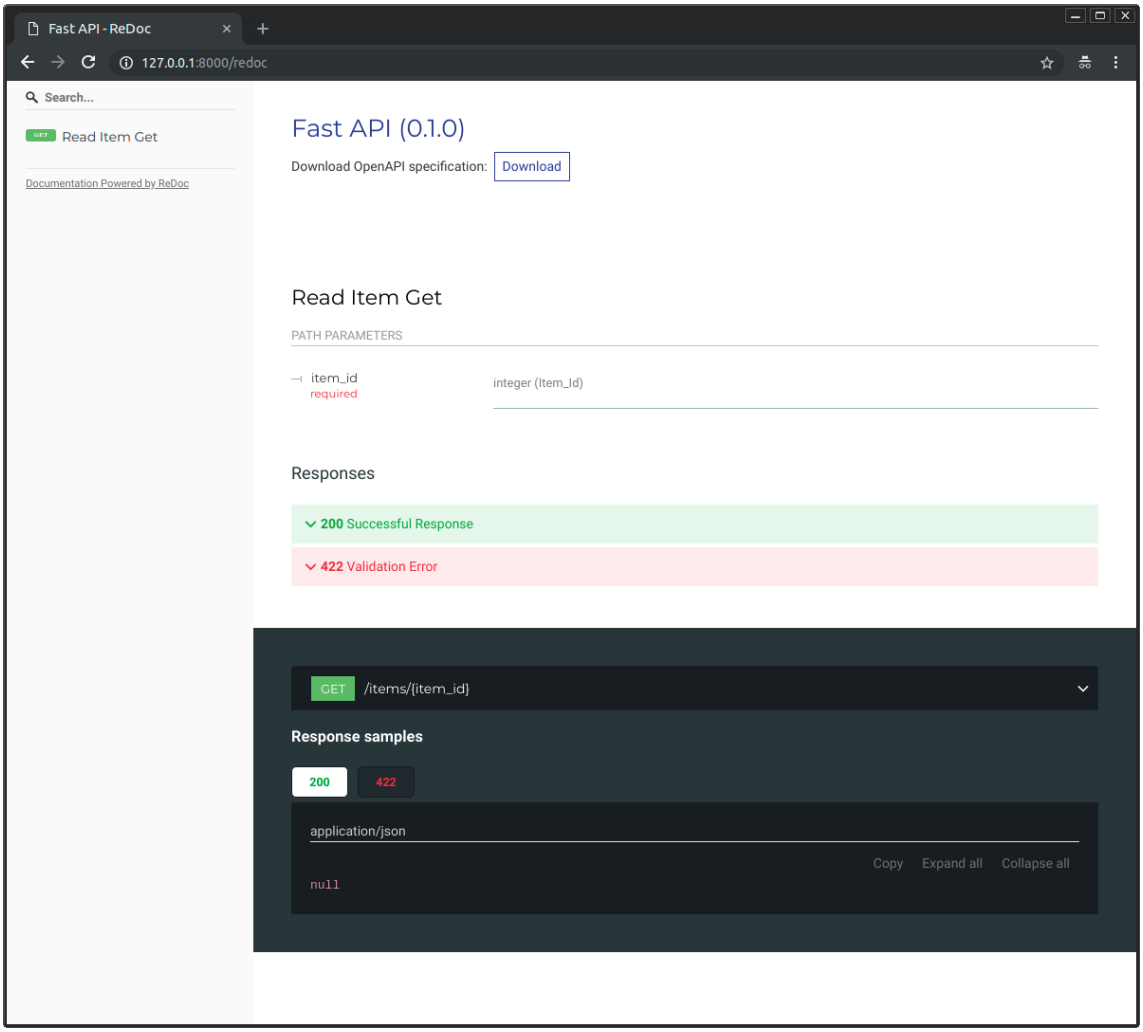
!!! check 再一次，还是通过相同的 Python 类型声明，**FastAPI** 为你提供了自动生成的交互式文档（集成 Swagger UI）。

注意这里的路径参数被声明为一个整数。

基于标准的好处：可选文档

由于生成的 API 模式来自于 OpenAPI 标准，所以有很多工具与其兼容。

正因如此，**FastAPI** 内置了一个可选的 API 文档（使用 Redoc）：



同样的，还有很多其他兼容的工具，包括适用于多种语言的代码生成工具。

Pydantic

所有的数据校验都由 Pydantic 在幕后完成，所以你可以从它所有的优点中受益。并且你知道它在这方面非常胜任。

你可以使用同样的类型声明来声明 `str`、`float`、`bool` 以及许多其他的复合数据类型。

本教程的下一章节将探讨其中的一些内容。

顺序很重要

在创建*路径操作*时，你会发现有些情况下路径是固定的。

比如 `/users/me`，我们假设它用来获取关于当前用户的数据.

然后，你还可以使用路径 `/users/{user_id}` 来通过用户 ID 获取关于特定用户的数据。

由于*路径操作*是按顺序依次运行的，你需要确保路径 `/users/me` 声明在路径 `/users/{user_id}` 之前：

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

否则， `/users/{user_id}` 的路径还将与 `/users/me` 相匹配，"认为"自己正在接收一个值为 `"me"` 的 `user_id` 参数。

预设值

如果你有一个接收路径参数的路径操作，但你希望预先设定可能的有效参数值，则可以使用标准的 Python `Enum` 类型。

创建一个 `Enum` 类

导入 `Enum` 并创建一个继承自 `str` 和 `Enum` 的子类。

通过从 `str` 继承，API 文档将能够知道这些值必须为 `string` 类型并且能够正确地展示出来。

然后创建具有固定值的类属性，这些固定值将是可用的有效值：

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

!!! info 枚举（或 enums）从 3.4 版本起在 Python 中可用。

!!! tip 如果你想知道，"AlexNet"、"ResNet" 和 "LeNet" 只是机器学习中的模型名称。

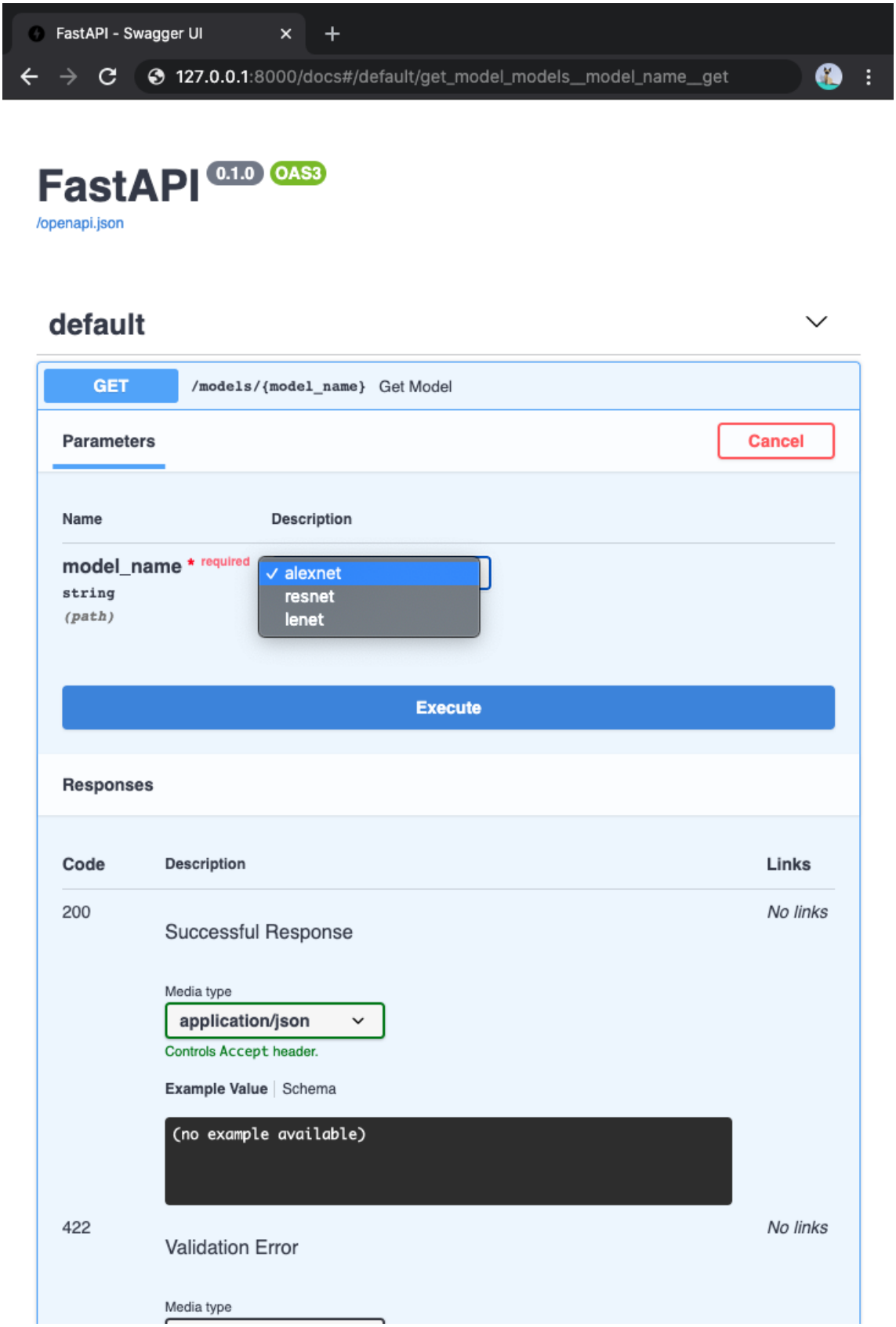
声明*路径参数*

然后使用你定义的枚举类（ `ModelName` ）创建一个带有类型标注的*路径参数*：

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

查看文档

因为已经指定了*路径参数*的可用值，所以交互式文档可以恰当地展示它们：



使用 Python *枚举类型*

*路径参数*的值将是一个*枚举成员*。

比较*枚举成员*

你可以将它与你创建的枚举类 `ModelName` 中的*枚举成员*进行比较：

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

获取*枚举值*

你可以使用 `model_name.value` 或通常来说 `your_enum_member.value` 来获取实际的值（在这个例子中为 `str`）：

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

!!! tip 你也可以通过 `ModelName.lenet.value` 来获取值 `"lenet"`。

返回*枚举成员*

你可以从*路径操作*中返回*枚举成员*，即使嵌套在 JSON 结构中（例如一个 `dict` 中）。

在返回给客户端之前，它们将被转换为对应的值：

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

包含路径的路径参数

假设你有一个*路径操作*，它的路径为 `/files/{file_path}`。

但是你需要 `file_path` 自身也包含*路径*，比如 `home/johndoe/myfile.txt`。

因此，该文件的URL将类似于这样：`/files/home/johndoe/myfile.txt`。

OpenAPI 支持

OpenAPI 不支持任何方式去声明*路径参数*以在其内部包含*路径*，因为这可能会导致难以测试和定义的情况出现。

不过，你仍然可以通过 Starlette 的一个内部工具在 **FastAPI** 中实现它。

而且文档依旧可以使用，但是不会添加任何该参数应包含路径的说明。

路径转换器

你可以使用直接来自 Starlette 的选项来声明一个包含*路径*的*路径参数*：

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

在这种情况下，参数的名称为 `file_path`，结尾部分的 `:path` 说明该参数应匹配任意的*路径*。

因此，你可以这样使用它：

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

!!! tip 你可能会需要参数包含 `/home/johndoe/myfile.txt`，以斜杠（`/`）开头。

在这种情况下，`URL` 将会是 `/files//home/johndoe/myfile.txt`，在`files` 和 `home` 之间有一个双斜杠（`//`）。

总结

使用 **FastAPI**，通过简短、直观和标准的 Python 类型声明，你将获得：

- 编辑器支持：错误检查，代码补全等
- 数据 "解析"
- 数据校验
- API 标注和自动生成的文档

而且你只需要声明一次即可。

这可能是 **FastAPI** 与其他框架相比主要的明显优势（除了原始性能以外）。