第一步

最简单的 FastAPI 文件可能像下面这样：

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

将其复制到 `main.py` 文件中。

运行实时服务器：

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
<span style="color: green;">INFO</span>:     Started reloader process [28720]
<span style="color: green;">INFO</span>:     Started server process [28722]
<span style="color: green;">INFO</span>:     Waiting for application startup.
<span style="color: green;">INFO</span>:     Application startup complete.
```

!!! note `uvicorn main:app` 命令含义如下:

```
* `main`: `main.py` 文件（一个 Python「模块」）。
* `app`: 在 `main.py` 文件中通过 `app = FastAPI()` 创建的对象。
* `--reload`: 让服务器在更新代码后重新启动。仅在开发时使用该选项。
```

在输出中，会有一行信息像下面这样：

```
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

该行显示了你的应用在本机所提供服务的 URL 地址。

查看

打开浏览器访问 http://127.0.0.1:8000。

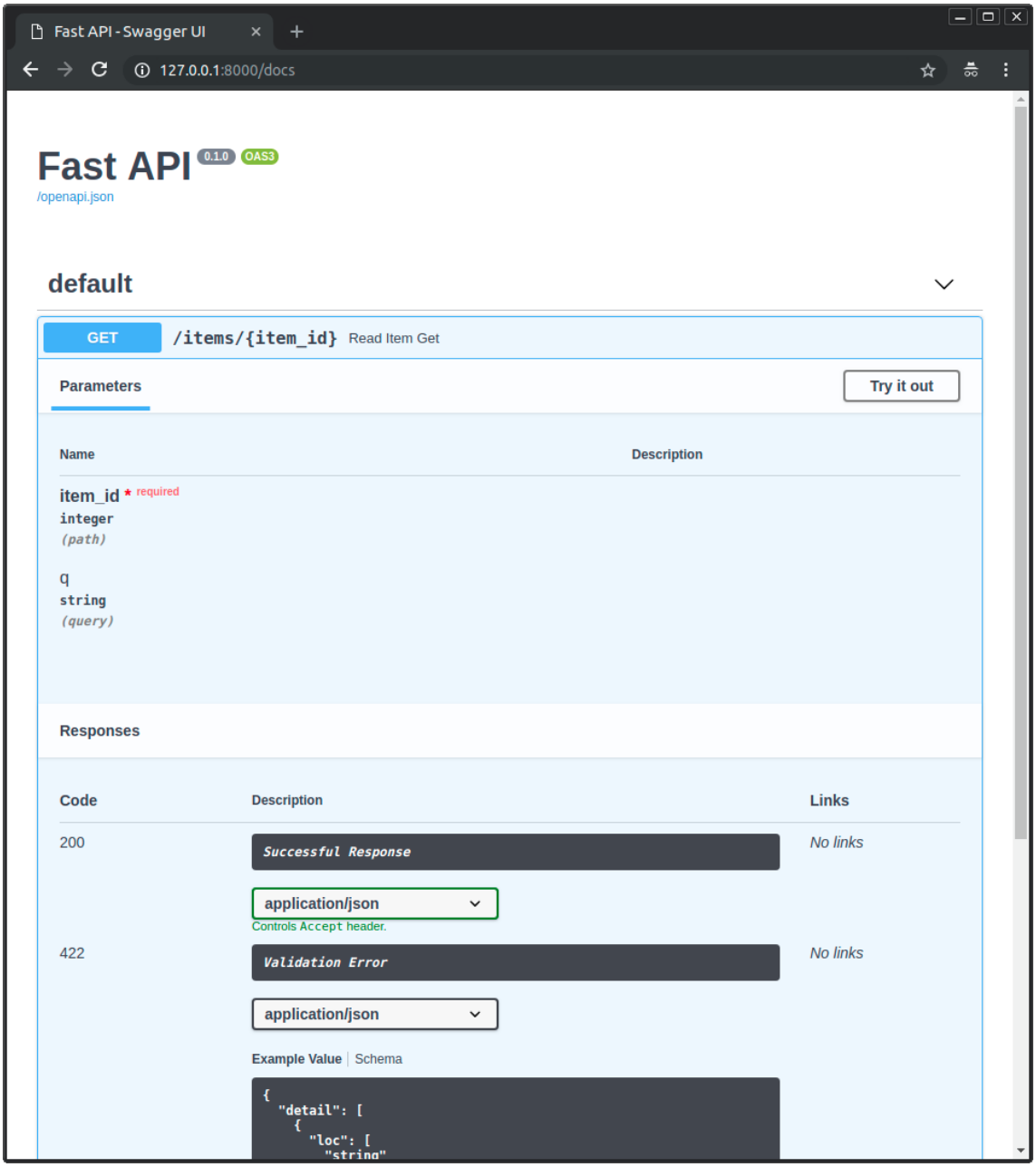
你将看到如下的 JSON 响应：

```
{"message": "Hello World"}
```

交互式 API 文档

跳转到 http://127.0.0.1:8000/docs。

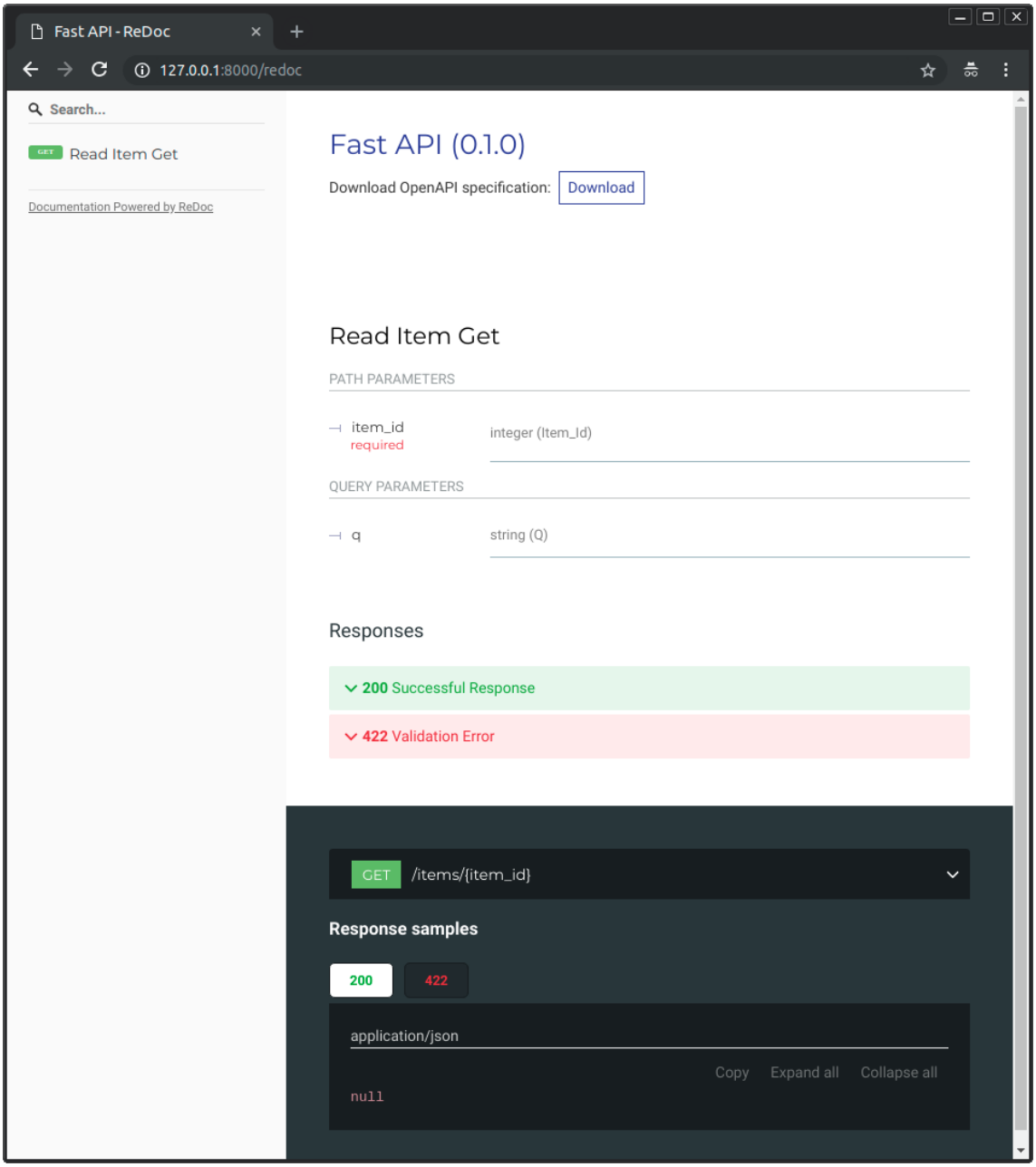
你将会看到自动生成的交互式 API 文档（由 Swagger UI 提供）：



可选的 API 文档

前往 http://127.0.0.1:8000/redoc。

你将会看到可选的自动生成文档 （由 ReDoc 提供）：



OpenAPI

FastAPI 使用定义 API 的 **OpenAPI** 标准将你的所有 API 转换成「模式」。

「模式」

「模式」是对事物的一种定义或描述。它并非具体的实现代码，而只是抽象的描述。

API「模式」

在这种场景下，OpenAPI 是一种规定如何定义 API 模式的规范。

定义的 OpenAPI 模式将包括你的 API 路径，以及它们可能使用的参数等等。

数据「模式」

「模式」这个术语也可能指的是某些数据比如 JSON 的结构。

在这种情况下，它可以表示 JSON 的属性及其具有的数据类型，等等。

OpenAPI 和 JSON Schema

OpenAPI 为你的 API 定义 API 模式。该模式中包含了你的 API 发送和接收的数据的定义（或称为「模式」），这些定义通过 JSON 数据模式标准 **JSON Schema** 所生成。

查看 `openapi.json`

如果你对原始的 OpenAPI 模式长什么样子感到好奇，其实它只是一个自动生成的包含了所有 API 描述的 JSON。

你可以直接在：http://127.0.0.1:8000/openapi.json 看到它。

它将显示以如下内容开头的 JSON：

```
{
    "openapi": "3.0.2",
    "info": {
        "title": "FastAPI",
        "version": "0.1.0"
    },
    "paths": {
        "/items/": {
            "get": {
                "responses": {
                    "200": {
                        "description": "Successful Response",
                        "content": {
                            "application/json": {


...
```

OpenAPI 的用途

驱动 FastAPI 内置的 2 个交互式文档系统的正是 OpenAPI 模式。

并且还有数十种替代方案，它们全部都基于 OpenAPI。你可以轻松地将这些替代方案中的任何一种添加到使用 **FastAPI** 构建的应用程序中。

你还可以使用它自动生成与你的 API 进行通信的客户端代码。例如 web 前端，移动端或物联网嵌入程序。

分步概括

步骤 1：导入 `FastAPI`

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

`FastAPI` 是一个为你的 API 提供了所有功能的 Python 类。

!!! note "技术细节" `FastAPI` 是直接从 `Starlette` 继承的类。

你可以通过 `FastAPI` 使用所有的 `Starlette` 的功能。

步骤 2：创建一个 `FastAPI`「实例」

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

这里的变量 `app` 会是 `FastAPI` 类的一个「实例」。

这个实例将是创建你所有 API 的主要交互对象。

这个 `app` 同样在如下命令中被 `uvicorn` 所引用：

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

如果你像下面这样创建应用：

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

将代码放入 `main.py` 文件中，然后你可以像下面这样运行 `uvicorn`：

```
$ uvicorn main:my_awesome_api --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

步骤 3：创建一个*路径操作*

路径

这里的「路径」指的是 URL 中从第一个 `/` 起的后半部分。

所以，在一个这样的 URL 中：

```
https://example.com/items/foo
```

...路径会是：

```
/items/foo
```

!!! info 「路径」也通常被称为「端点」或「路由」。

开发 API 时，「路径」是用来分离「关注点」和「资源」的主要手段。

操作

这里的「操作」指的是一种 HTTP「方法」。

下列之一：

- `POST`
- `GET`
- `PUT`
- `DELETE`

...以及更少见的几种：

- `OPTIONS`
- `HEAD`
- `PATCH`
- `TRACE`

在 HTTP 协议中，你可以使用以上的其中一种（或多种）「方法」与每个路径进行通信。

在开发 API 时，你通常使用特定的 HTTP 方法去执行特定的行为。

通常使用：

- `POST`：创建数据。
- `GET`：读取数据。
- `PUT`：更新数据。
- `DELETE`：删除数据。

因此，在 OpenAPI 中，每一个 HTTP 方法都被称为「操作」。

我们也打算称呼它们为「操作」。

定义一个*路径操作装饰器*

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

`@app.get("/")` 告诉 **FastAPI** 在它下方的函数负责处理如下访问请求：

- 请求路径为 `/`
- 使用 `get` 操作

!!! info "`@decorator` Info" `@something` 语法在 Python 中被称为「装饰器」。

像一顶漂亮的装饰帽一样，将它放在一个函数的上方（我猜测这个术语的命名就是这么来的）。

装饰器接收位于其下方的函数并且用它完成一些工作。

在我们的例子中，这个装饰器告诉 `**FastAPI**` 位于其下方的函数对应着**路径** `/` 加上 `get` **操作**。

它是一个「**路径操作装饰器**」。

你也可以使用其他的操作：

- `@app.post()`

- `@app.put()`
- `@app.delete()`

以及更少见的：

- `@app.options()`
- `@app.head()`
- `@app.patch()`
- `@app.trace()`

!!! tip 您可以随意使用任何一个操作（HTTP方法）。

```
**FastAPI** 没有强制要求操作有任何特定的含义。

此处提供的信息仅作为指导，而不是要求。

比如，当使用 GraphQL 时通常你所有的动作都通过 `post` 一种方法执行。
```

步骤 4：定义路径操作函数

这是我们的「**路径操作函数**」：

- **路径**：是 `/`。
- **操作**：是 `get`。
- **函数**：是位于「装饰器」下方的函数（位于 `@app.get("/")` 下方）。

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

这是一个 Python 函数。

每当 **FastAPI** 接收一个使用 `GET` 方法访问 URL「`/`」的请求时这个函数会被调用。

在这个例子中，它是一个 `async` 函数。

你也可以将其定义为常规函数而不使用 `async def`：

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

!!! note 如果你不知道两者的区别，请查阅 Async: *"In a hurry?"*{.internal-link target=_blank}。

步骤 5：返回内容

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

你可以返回一个 `dict`、`list`，像 `str`、`int` 一样的单个值，等等。

你还可以返回 Pydantic 模型（稍后你将了解更多）。

还有许多其他将会自动转换为 JSON 的对象和模型（包括 ORM 对象等）。尝试下使用你最喜欢的一种，它很有可能已经被支持。

总结

- 导入 `FastAPI`。
- 创建一个 `app` 实例。
- 编写一个**路径操作装饰器**（如 `@app.get("/")`）。
- 编写一个**路径操作函数**（如上面的 `def root(): ...`）。
- 运行开发服务器（如 `uvicorn main:app --reload`）。