

1. FastAPI 介绍和安装

FastAPI 是一个基于 Python 3.6+ 版本的异步 WEB 应用框架，使用 Python 类型注解构建 web API。它的主要特点如下：

1. **高性能**：与 Node JS 和 Go 相当。
2. **编码快**：将开发功能的速度提高 2~3 倍。
3. **Bug少**：减少大约 40% 的由开发人员导致的错误。
4. **直观**：强大的编辑器支持，可智能感知和补全代码。
5. **简单**：易于学习和使用，减少文档阅读时间。
6. **简短**：尽量减少代码重复。
7. **健壮**：获得可用于生产的代码，具有自动交互文档。
8. **基于标准**：基于 OpenAPI 和 JSON Schema。

✍ 基于 API 的开放标准 OpenAPI 就是之前被称为 Swagger 的标准。

首先，我们来安装 FastAPI 及其依赖：

```
pip install fastapi==0.115.0
```

在 `fastapi` 的依赖中，有三个核心依赖：

1. [Pydantic](#)
2. [Starlette](#)
3. [Uvicorn](#)

FastAPI 使用 `pydantic` 处理所有数据验证、数据序列化以及基于 JSON Schema 的自动模型文档。

Starlette 是一个轻量级的 ASGI 框架 / 工具包，非常适合用 Python 构建异步 Web 服务。FastAPI 就是基于 Starlette 扩展而来的，FastAPI 提供的 `Request` 请求报文更是直接使用了 Starlette 的 `Request`。

✍ FastAPI 基于 Starlette 和 Pydantic 做了很多封装，简化了我们的编码工作。

Uvicorn 是一个轻量级的 ASGI 服务器，基于 `uvloop` 和 `httptools` 实现，运行速度极快。我们使用 Uvicorn 来运行 FastAPI 构建的 Web 应用。

ASGI 被称为[异步服务器网关接口](#)，和 WSGI 一样，二者都是为 Python 语言定义的 Web 服务器和 Web 应用之间的通用接口。

ASGI 向后兼容了 WSGI，可以认为是 WSGI 的扩展，并且提供异步特性和 WebSocket 支持。

总的来说，[FastAPI 框架](#)集众框架之所长，不仅具有 Flask / Django 框架的 Web 核心功能，并且兼具异步特点，具有同步和异步两种运行模式。

有关 FastAPI 在设计之初从其他 Web 框架中受到的启发详见官网原文。

<https://fastapi.tiangolo.com/alternatives/>

2. 第一个 FastAPI 应用

2.1 创建并运行一个 web 应用

下面，我们创建一个 main.py 文件，内容如下：

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def index():
    return {"msg": "Hello World!"}

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

FastAPI 可以同时兼容同步和异步两种运行模式，异步 API 使用 `async/await` 关键字。

首先，实例化了一个 `FastAPI` 对象 `app`，然后使用 `@app.get()` 装饰器注册了两个处理 GET 请求的视图函数 `index` 和 `get_item`：

1. `index`: 处理发往 / 的 GET 请求。
2. `get_item`: 访问集合资源 `/items` 中的单个元素。

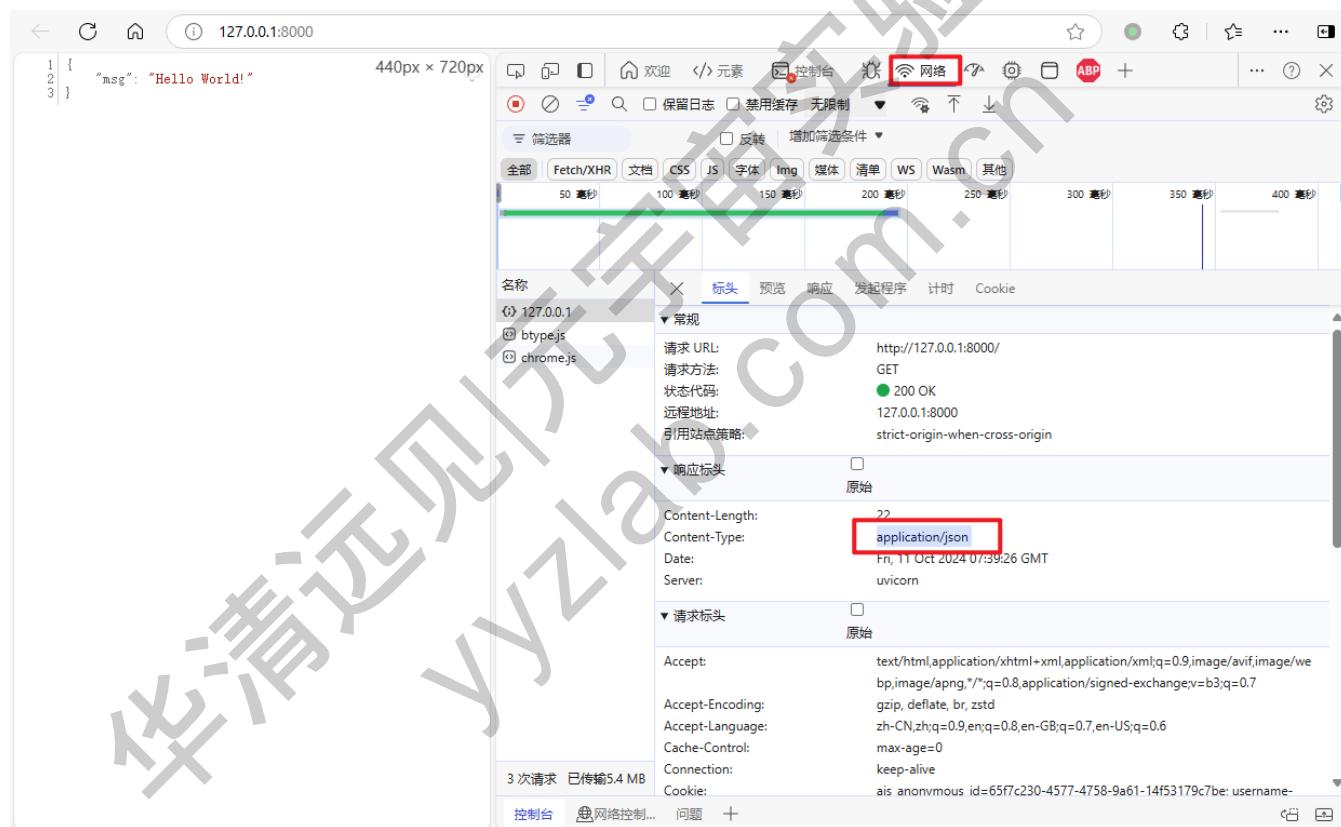
下面，我们来通过 Uvicorn 来运行上述应用：

```
uvicorn main:app --reload
```

其中 `reload` 参数可以在代码更新后自动重启服务，在开发时方便我们随时验证 API。命令行运行结果如下：

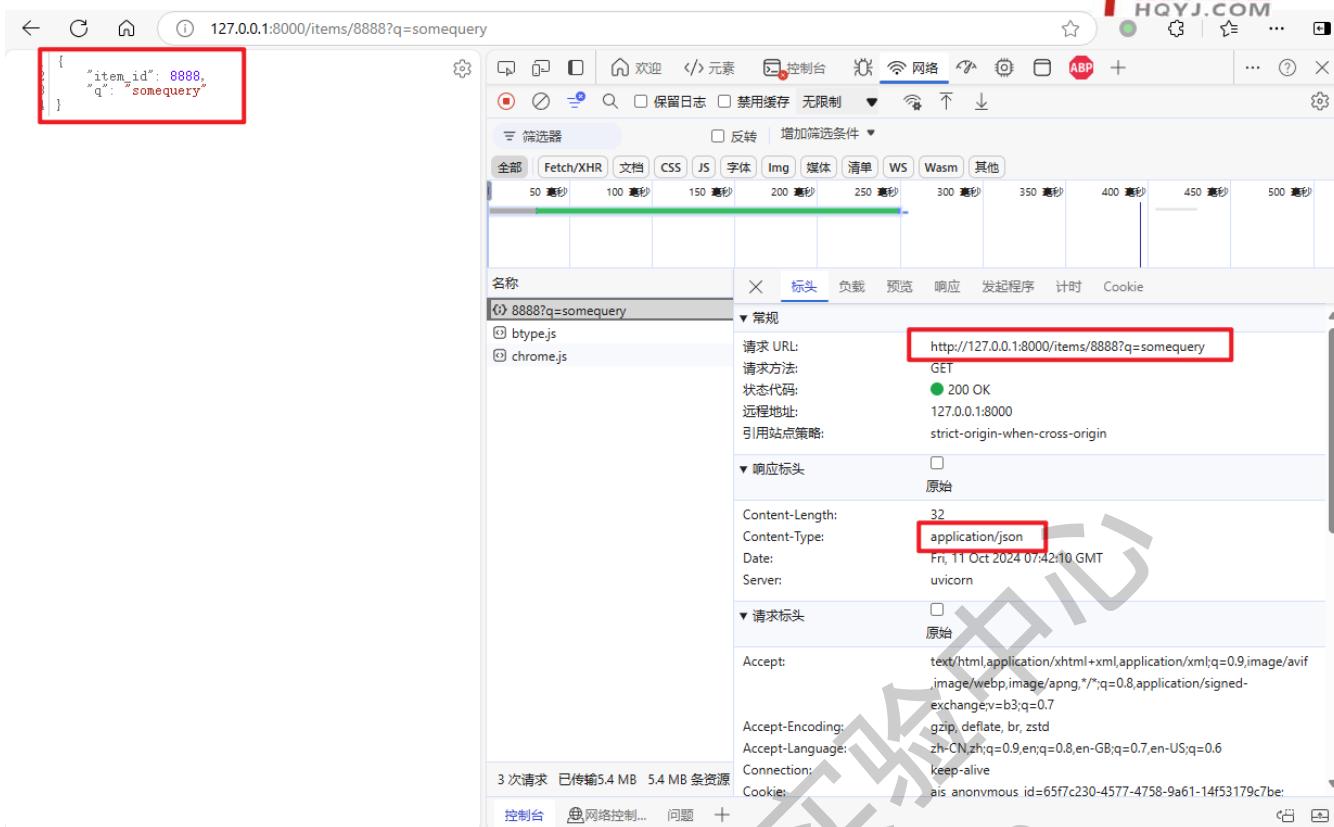
```
○ (AI_env) (base) root@2bf30626209a:~/workdir# uvicorn api:app --reload
INFO:     Will watch for changes in these directories: ['/root/workdir']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [3261] using WatchFiles
INFO:     Started server process [3263]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

下面，我们访问命令行输出提示中的 URL 地址（F12 打开检查，在网络处刷新该页面）：



视图函数 `index` 返回的字典被自动转换为了 JSON 格式的字符串，并作为 `json` 类型的响应发送给浏览器。

下面，我们通过浏览器访问 `http://127.0.0.1:8000/items/8888?q=somequery` 地址，触发视图函数 `get_item` 的执行。



观察返回的 JSON 响应发现：URL 地址中的 8888 被解析为地址参数 `item_id`，并根据类型注解自动转换为 `int` 类型；查询参数 `?q=somequery` 被解析为关键字参数 `q`，并获得字符串类型的值 'somequery'。

2.2 API 文档

FastAPI 为我们自动生成了两种形式的 API 文档：

1. 交互式 API 文档
2. 备用 API 文档

访问网址 `http://127.0.0.1:8000/docs`，打开交互式 API 文档：

FastAPI 0.1.0 OAS 3.1

/openapi.json

default

GET / Index

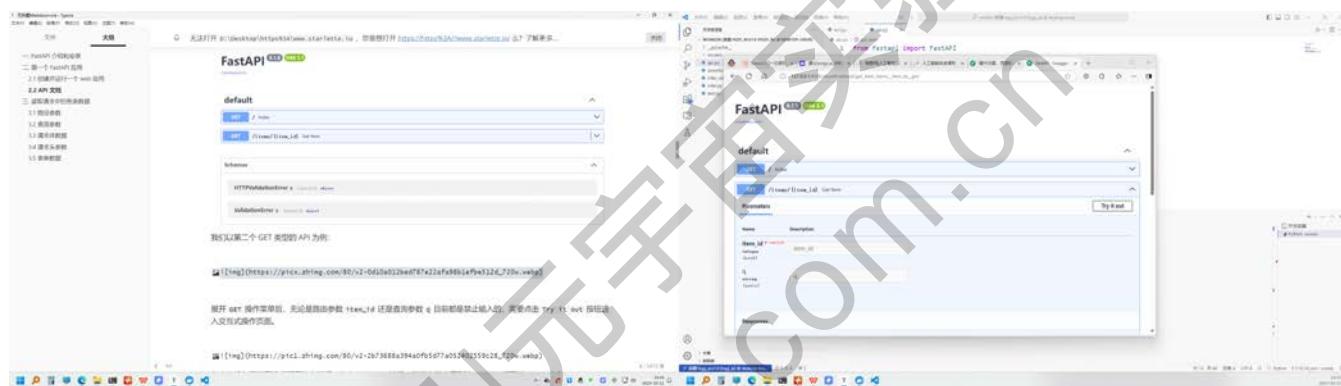
GET /items/{item_id} Get Item

Schemas

HTTPValidationError > Expand all object

ValidationError > Expand all object

我们以第二个 GET 类型的 API 为例：



展开 `GET` 操作菜单后，无论是路由参数 `item_id` 还是查询参数 `q` 目前都是禁止输入的，需要点击 `Try it out` 按钮进入交互式操作页面。

The screenshot shows a REST API documentation page for a GET request to '/items/{item_id}'. The 'Parameters' section contains two fields: 'item_id' (integer, path) with value '6666' and 'q' (string, query) with value 'hello'. A red box highlights these parameters. Below the parameters is a 'Servers' section with a dropdown set to '/'. A large red arrow points from the 'Execute' button at the bottom to the 'Servers' dropdown. The 'Responses' section is partially visible.

在文本输入框填写请求所需的参数后，点击 Execute 按钮，即可在下方看到该 API 接口返回的 HTTP 响应。

The screenshot shows the results of the executed API call. It includes:

- Curl:** curl -X 'GET' \ 'http://127.0.0.1:8000/items/6666?q=hello' \ -H 'accept: application/json'
- Request URL:** http://127.0.0.1:8000/items/6666?q=hello
- Server response:**

| Code | Details |
|------|--|
| 200 | Response body <pre>{ "item_id": 6666, "q": "hello" }</pre> Response headers <pre>content-length: 28 content-type: application/json date: Fri, 11 Oct 2024 08:07:13 GMT server: uvicorn</pre> |
- Responses:**

| Code | Description | Links |
|------|---------------------|----------|
| 200 | Successful Response | No links |

除了上述的交互式 API 文档，我们也可以访问 <http://127.0.0.1:8000/redoc> 查看 FlaskAPI 应用的备用文档。

FastAPI (0.1.0)

Download OpenAPI specification: [Download](#)

Index

Responses

> 200 Successful Response

Get Item

PATH PARAMETERS

→ item_id required integer (Item Id)

Content type
application/json

null

Copy

200 422

3. 读取请求中的各类数据

3.1 路径参数

FastAPI 使用格式化字符串中的 {} 声明路径参数，即 URL 变量。比如上文用到的 `get_item` 视图函数对应的 URL 规则中就使用了路径变量 `item_id`。

```
@app.get("/items/{item_id}")
async def get_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

URL 变量 `item_id` 的值会作为参数 `item_id` 传递到视图函数 `get_item()`。

此外，参数 `item_id` 使用了标准的 Python 类型注解，标注类型为 `int`；FastAPI 将利用类型检查自动完成请求解析，将 `item_id` 转换为整型。

如果类型转换失败，将返回 422 错误响应：

The screenshot shows a browser developer tools Network tab with a failed request for `http://127.0.0.1:8000/items/hello`. The status code is `422 Unprocessable Entity`. The response body is a JSON object containing an error detail:

```
{
  "detail": [
    {
      "type": "int_parsing",
      "loc": [ "path", "item_id" ],
      "msg": "Input should be a valid integer, unable to parse string as an integer",
      "input": "Hello"
    }
  ]
}
```

字符串 `Hello` 无法被转换为整型，因此类型校验失败，返回的错误消息也清晰地指出了这一点，还是非常友好的。

最后，我们需要了解一下路由的匹配顺序：

路由匹配是按顺序进行的，这意味着如果某个静态的 URL 刚好可以匹配到另一个动态的 URL 规则，为了能够正确触发静态 URL 规则对应的视图函数，应确保该 URL 在动态 URL 规则之前声明。

比如：一个固定 URL 的 `/users/me` 获取当前用户的数据，还有一个动态路由 `/users/{user_id}` 获取指定 ID 的用户数据。

我们需要确保 `/users/me` 路径在 `/users/{user_id}` 路径之前已声明，否则 `/users/{user_id}` 也将匹配 `users/me`。

```
@app.get("/users/me")
async def get_user_me():
    return {"user_id": "current_user info..."}  
  

@app.get("/users/{user_id}")
async def get_user(user_id: str):
    return {"user_id": user_id}
```

测试运行结果：

`get_user_me` 被触发：

127.0.0.1:8000/users/me

```

1 "user_id": "current_user info..."
2
3

```

get_user 被触发：

127.0.0.1:8000/users/1008

```

1 {
2   "user_id": "1008"
3 }

```

3.2 查询参数

当视图函数声明不属于 URL 路径参数的其他参数时，FastAPI 将自动解析为 Query 查询参数。

查询参数即 URL 地址中 ? 之后的一系列用 & 分隔的 key-value 键值对。

比如，下面用于处理集合资源访问的视图函数就定义了两个查询参数：

```

@app.get("/items/")
async def get_items(skip: int = 0, limit: int = 10):
    return {"items": [], "skip": skip, "limit": limit}

```

查询参数作为 URL 的一部分，默认的类型也为字符串，因此需要借助类型注解转换为 int 类型，FastAPI 将依据注解的类型来验证传入的参数。

访问链接 <http://127.0.0.1:8000/items/?skip=20&limit=15>，可以看到查询参数被正确解析到了视图函数的关键字参数中。

```

1 {
2   "items": [],
3   "skip": 20,
4   "limit": 15
5 }

```

此外，因为我们在定义视图函数时，为查询参数 skip 和 limit 指定了默认值，因此查询参数将变为可选的，缺省时将使用默认值。

```

1 {
2   "items": [],
3   "skip": 0,
4   "limit": 10
5 }

```

当定义一个必须指定的查询参数时，就不能再为这个参数定义任意的默认值。

```
@app.get("/items/{item_id}")
async def get_item(item_id: int, q: str):
    return {"item_id": item_id, "q": q}
```

当未指定查询参数 `q` 时，将收到 FastAPI 返回的错误响应，提示我们 `q` 为必须的查询参数。

The screenshot shows a browser developer tools Network tab with a failed API request. The response body is a JSON object with a 'detail' field containing an array of errors. One error is highlighted with a red box, showing a 'missing' type error at path 'query.q'. The response status code is 422 Unprocessable Entity.

最后，有关查询参数的类型转换，我们来补充一个示例：

```
@app.get("/items/{item_id}")
async def get_item(item_id: int, q: str = None, short: bool = False):
    item = {"item_id": item_id}
    if q:
        item.update({'q': q})
    item.update({'short': short})
    return item
```

其中，查询参数 `short` 是一个布尔类型的变量，默认值为 `False`。在交互式文档中，我们可以清晰地看到 `short` 和 `q` 两个查询参数都是可选的，并且 `short` 变量类型为布尔型。

GET /items/{item_id} Get Item

Parameters

| Name | Description |
|--|------------------------------------|
| item_id <small>* required</small> <small>integer (path)</small> | <input type="text" value="10086"/> |
| q <small>string (query)</small> | <input type="text" value="q"/> |
| short <small>boolean (query)</small> | <input type="text" value="true"/> |

Servers

These operation-level options override the global server options.

Responses

Execute

字段 `short` 的下拉选项有 3 个值：缺省、`true` 以及 `false`。当选择 `true` 时返回的响应如下：

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/items/10086?short=true' \
-H 'accept: application/json'
```

Request URL

`http://127.0.0.1:8000/items/10086?short=true`

Server response

| Code | Details |
|------|---|
| 200 | Response body <pre>{ "item_id": 10086, "short": true }</pre> <p>Download</p> Response headers <pre>content-length: 30 content-type: application/json date: Fri, 11 Oct 2024 10:13:39 GMT server: unicorn</pre> |

Responses

| Code | Description | Links |
|------|---------------------|----------|
| 200 | Successful Response | No links |

在进行布尔型类转换时，对于 `1` `yes` `True` 这样的值，也会自动转换为 `true`；相反对 `0` `no` `False` 会自动转换为 `false`。

The figure consists of four vertically stacked browser windows, each displaying a JSON response. The top window shows the URL 127.0.0.1:8000/items/10009?short=no. The JSON content is:

```

1 {
2     "item_id": 10009,
3     "short": false
4 }

```

The second window shows the URL 127.0.0.1:8000/items/10009?short=yes. The JSON content is:

```

1 {
2     "item_id": 10009,
3     "short": true
4 }

```

The third window shows the URL 127.0.0.1:8000/items/10009?short=0. The JSON content is:

```

1 {
2     "item_id": 10009,
3     "short": false
4 }

```

The fourth window shows the URL 127.0.0.1:8000/items/10009?short=1. The JSON content is:

```

1 {
2     "item_id": 10009,
3     "short": true
4 }

```

3.3 请求体数据

定义请求体需要使用 `pydantic` 模型，不能通过 GET 请求发送请求体。发送请求体时必须通过以下方法：

1. POST
2. PUT
3. DELETE
4. PATCH

下面，我们使用 `pydantic` 模块提供的 `BaseModel` 创建一个数据模型类：

```

from pydantic import BaseModel

class Item(BaseModel):
    name: str
    description: str = None
    width: float
    height: float

```

和查询参数类似，模型类中定义的属性字段如果不是必需字段的话，可以设定默认值；否则该字段就是必须的。

接下来，我们只需要在视图函数的参数列表中，将参数的类型注解为模型类 `Item` 即可：

```
from fastapi import FastAPI

app = FastAPI()

@app.post("/items/")
async def create_item(item: Item):
    return item

@app.put("/items/{item_id}")
async def modify_item(item_id: int, item: Item):
    return {'item_id': item_id, **item.dict()}
```

FastAPI 将按照 JSON 类型的响应读取请求体中的数据，并按照 `Item` 类中属性的类型注解验证数据类型，验证失败时返回错误字段的位置以及原因。

数据模型 `Item` 的 JSON Schema 将成为 OpenAPI 生成模式的一部分，用于交互式文档：



处理 PUT 请求的视图函数 `modify_item` 则不仅包含 JSON 格式的请求体，还定义了一个路径参数 `item_id`。

The screenshot shows a FastAPI documentation interface for a `PUT /items/{item_id}` endpoint. The `item_id` parameter is highlighted with a red box, indicating it's a path parameter. The request body schema is set to `application/json`, and an example value is provided as a JSON object:

```
{  
    "name": "string",  
    "description": "string",  
    "width": 0,  
    "height": 0  
}
```

FastAPI 在解析以上三种类型的参数时，遵循以下规则：

1. 如果 `path` 中声明了相应的变量，则解析为路径参数；
2. 如果参数是单一类型 `int` `float` `str` `bool` 等，将被解析为查询参数；
3. 如果参数类型为 pydantic 的数据模型类，将被解析为 JSON 格式的请求体数据；

下面是在交互式文档中使用 `PUT` 类型的 API 完成的请求、响应测试。

PUT /items/{item_id} Modify Item

Parameters

Name Description

item_id * required integer (path) 10086

Request body required application/json

```
{  
    "name": "hello",  
    "description": "test",  
    "width": 640,  
    "height": 640  
}
```

Servers

These operation-level options override the global server options.

/

Execute Clear

Responses

Curl

```
curl -X 'PUT' \
'http://127.0.0.1:8000/items/10086' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"name": "hello",
"description": "test",
"width": 640,
"height": 640
}'
```



Request URL

<http://127.0.0.1:8000/items/10086>

Server response

| Code | Details |
|------|---|
| 200 | <p>Response body</p> <pre>{ "item_id": 10086, "name": "hello", "description": "test", "width": 640, "height": 640 }</pre> <p>Copy Download</p> <p>Response headers</p> <pre>content-length: 82 content-type: application/json date: Fri, 11 Oct 2024 10:30:09 GMT server: uvicorn</pre> |

[Download](#)

3.4 请求头参数

在视图函数的参数列表中定义请求头参数，需要导入 `fastapi` 模块提供的参数类 `Header`：

```
from fastapi import FastAPI, Header
from pydantic import BaseModel

app = FastAPI()

@app.get("/items/")
def get_items(user_agent: str = Header(None)):
    return {'User-Agent': user_agent}
```

参数 `user_agent` 被定义为一个请求头参数，默认值为 `None`。



`Header` 是 `Path`、`Query` 和 `Cookie` 的兄弟类，他们均继承自 `Param` 类。

因为 Python 在定义变量名称时使用数字、字母以及下划线，因此在定义包含连字符 - 的请求头字段时，应使用 _ 代替 - 。

HTTP 表头并不区分大小写，我们可以使用 python 风格的变量命名 `user_agent`，而不必写为 `User-Agent` 的形式。

下面是使用 Microsoft Edge 浏览器发送向 `/items/` 发送的 GTE 请求响应页面：

The screenshot shows the Microsoft Edge DevTools Network tab. A red box highlights the User-Agent header in the request section. The header value is "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36 Edg/130.0.0.0".

| 名称 | 值 |
|--------------------------|---|
| <code>items/</code> | |
| <code>btype.js</code> | |
| <code>chrome.js</code> | |
| <code>User-Agent:</code> | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36 Edg/130.0.0.0 |

视图函数中的变量 `user_agent` 成功读取到了请求头中的字段 `User-Agent` 的值。

当我们需要禁用请求头字段的“下划线”到“连字符”的自动转换时，可以将 `Header` 的参数 `convert_underscores` 设为 `False` 。

```
@app.get("/items/")
def get_items(user_agent: str = Header(None,
convert_underscores=False)):
    return {'User-Agent': user_agent}
```

禁用之后，变量 `user_agent` 读取到的就不再是浏览器设置的 `User-Agent` 请求头字段，而是来自我们自定义的请求头字段 `user_agent`。

The screenshot shows a browser developer tools interface with the Network tab selected. A red arrow points from the top-left towards the User-Agent field in the Headers section. The Headers section shows various HTTP headers, and the User-Agent header is explicitly highlighted with a red box. Its value is listed as "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36 Edg/130.0.0.0".

因为请求头尚未设置 `user_agent` 字段，因此将使用 `Header` 指定的默认值 `None`。下面，我们使用交互式 API 添加自定义的 HTTP 头部字段 `user_agent`：

The screenshot shows the FastAPI documentation for a `/items/` endpoint. The `user_agent` parameter is highlighted with a red box. The `Curl` section contains a command line example with `-H "user-agent: hello"`. The `Request URL` field shows `http://127.0.0.1:8000/items/`.

⚠ 注意一些 HTTP 代理和服务器禁止使用带下划线的 HTTP 标头。

3.5 表单数据

当视图函数需要接收表单字段而非 JSON 时，需要明确使用 `Form` 类，否则参数将被解析为查询参数或 JSON 主体。

```
from fastapi import FastAPI, Form

app = FastAPI()

@app.post("/login/")
async def login(username: str = Form('admin'), password: str = Form()):
    return {"username": username, "password": password}
```

HTML 表单在将数据提交到服务器时，会对数据进行特殊编码处理，常用的编码类型为媒体类型：

`application/x-www-form-urlencoded`

当包含文件上传字段时，将被编码为 `multipart/form-data` 类型。

下面，我们在交互式文档中测试上述 POST 请求类型的 API：

The screenshot shows a browser-based API testing tool for a FastAPI application. The URL is 127.0.0.1:8000/docs#/default/login_login_post. The interface is titled 'default' and shows a 'POST /login/' operation for 'Login'. Under 'Parameters', it says 'No parameters'. In the 'Request body' section, there are two fields: 'username' (type string, value admin) and 'password' (type string, value 123456). A dropdown menu labeled 'application/x-www-form-urlencoded' is highlighted with a red box. Below the request body, there's a 'Servers' section with a dropdown set to '/'. At the bottom is a large blue 'Execute' button.

其中用户名使用了实例化 `Form` 字段时提供的默认值 `admin`，同时我们会看到 FastAPI 已经为我们设定了正确的请求体的编码类型。

执行上述 HTTP 请求的响应如下：

The screenshot shows a browser interface with a blue header bar containing 'Execute' and 'Clear' buttons. Below the header is a 'Responses' section. Under 'Responses', there is a 'Curl' block containing a command-line example for a POST request to 'http://127.0.0.1:8000/login/' with headers '-H 'Content-Type: application/json'' and '-H 'Content-Type: application/x-www-form-urlencoded'' (the second line is highlighted with a red box). Below this is a 'Request URL' field with the value 'http://127.0.0.1:8000/login/'. The 'Server response' section shows a 'Code' tab selected, displaying '200'. Under 'Response body', there is a JSON object: { "username": "admin", "password": "123456" }. Under 'Response headers', there is a list: content-length: 40, content-type: application/json, date: Fri, 11 Oct 2024 10:43:24 GMT, server: uvicorn.

视图函数 `login` 处理表单类型的提交，返回的字典将作为 JSON 格式的响应主体返回。