CORS（跨域资源共享）

CORS 或者「跨域资源共享」指浏览器中运行的前端拥有与后端通信的 JavaScript 代码，而后端处于与前端不同的「源」的情况。

源

源是协议（`http`，`https`）、域（`myapp.com`，`localhost`，`localhost.tiangolo.com`）以及端口（`80`、`443`、`8080`）的组合。

因此，这些都是不同的源：

- `http://localhost`
- `https://localhost`
- `http://localhost:8080`

即使它们都在 `localhost` 中，但是它们使用不同的协议或者端口，所以它们都是不同的「源」。

步骤

假设你的浏览器中有一个前端运行在 `http://localhost:8080`，并且它的 JavaScript 正在尝试与运行在 `http://localhost` 的后端通信（因为我们没有指定端口，浏览器会采用默认的端口 `80`）。

然后，浏览器会向后端发送一个 HTTP `OPTIONS` 请求，如果后端发送适当的 headers 来授权来自这个不同源（`http://localhost:8080`）的通信，浏览器将允许前端的 JavaScript 向后端发送请求。

为此，后端必须有一个「允许的源」列表。

在这种情况下，它必须包含 `http://localhost:8080`，前端才能正常工作。

通配符

也可以使用 `"*"`（一个「通配符」）声明这个列表，表示全部都是允许的。

但这仅允许某些类型的通信，不包括所有涉及凭据的内容：像 Cookies 以及那些使用 Bearer 令牌的授权 headers 等。

因此，为了一切都能正常工作，最好显式地指定允许的源。

使用 `CORSMiddleware`

你可以在 **FastAPI** 应用中使用 `CORSMiddleware` 来配置它。

- 导入 `CORSMiddleware`。
- 创建一个允许的源列表（由字符串组成）。
- 将其作为「中间件」添加到你的 **FastAPI** 应用中。

你也可以指定后端是否允许：

- 凭证（授权 headers，Cookies 等）。
- 特定的 HTTP 方法（`POST`，`PUT`）或者使用通配符 `"*"` 允许所有方法。
- 特定的 HTTP headers 或者使用通配符 `"*"` 允许所有 headers。

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

默认情况下，这个 `CORSMiddleware` 实现所使用的默认参数较为保守，所以你需要显式地启用特定的源、方法或者 headers，以便浏览器能够在跨域上下文中使用它们。

支持以下参数：

- `allow_origins` - 一个允许跨域请求的源列表。例如 `['https://example.org', 'https://www.example.org']`。你可以使用 `['*']` 允许任何源。
- `allow_origin_regex` - 一个正则表达式字符串，匹配的源允许跨域请求。例如 `'https://.*\.example\.org'`。
- `allow_methods` - 一个允许跨域请求的 HTTP 方法列表。默认为 `['GET']`。你可以使用 `['*']` 来允许所有标准方法。
- `allow_headers` - 一个允许跨域请求的 HTTP 请求头列表。默认为 `[]`。你可以使用 `['*']` 允许所有的请求头。`Accept`、`Accept-Language`、`Content-Language` 以及 `Content-Type` 请求头总是允许 CORS 请求。
- `allow_credentials` - 指示跨域请求支持 cookies。默认是 `False`。另外，允许凭证时 `allow_origins` 不能设定为 `['*']`，必须指定源。
- `expose_headers` - 指示可以被浏览器访问的响应头。默认为 `[]`。
- `max_age` - 设定浏览器缓存 CORS 响应的最长时间，单位是秒。默认为 `600`。

中间件响应两种特定类型的 HTTP 请求……

CORS 预检请求

这是些带有 `Origin` 和 `Access-Control-Request-Method` 请求头的 `OPTIONS` 请求。

在这种情况下，中间件将拦截传入的请求并进行响应，出于提供信息的目的返回一个使用了适当的 CORS headers 的 `200` 或 `400` 响应。

简单请求

任何带有 `Origin` 请求头的请求。在这种情况下，中间件将像平常一样传递请求，但是在响应中包含适当的 CORS headers。

更多信息

更多关于 CORS 的信息，请查看 Mozilla CORS 文档。

!!! note "技术细节" 你也可以使用 `from starlette.middleware.cors import CORSMiddleware`。

出于方便，`**FastAPI**` 在 `` `fastapi.middleware` `` 中为开发者提供了几个中间件。但是大多数可用的中间件都是直接来自 `Starlette`。