开发 - 贡献

首先，你最好先了解 帮助 FastAPI 及获取帮助{.internal-link target=_blank}的基本方式。

开发

如果你已经克隆了源码仓库，并且需要深入研究代码，下面是设置开发环境的指南。

通过 `venv` 管理虚拟环境

你可以使用 Python 的 `venv` 模块在一个目录中创建虚拟环境：

```
$ python -m venv env
```

这将使用 Python 程序创建一个 `./env/` 目录，然后你将能够为这个隔离的环境安装软件包。

激活虚拟环境

使用以下方法激活新环境：

=== "Linux, macOS"

```console
$ source ./env/bin/activate
```

=== "Windows PowerShell"

```console
$ .\env\Scripts\Activate.ps1
```

=== "Windows Bash"

Or if you use Bash for Windows (e.g. Git Bash):

```console
$ source ./env/Scripts/activate
```

要检查操作是否成功，运行：

=== "Linux, macOS, Windows Bash"

```
<div class="termy">

```console
$ which pip

some/directory/fastapi/env/bin/pip
```

</div>
```

=== "Windows PowerShell"

```
<div class="termy">

```console
$ Get-Command pip

some/directory/fastapi/env/bin/pip
```

</div>
```

如果显示 `pip` 程序文件位于 `env/bin/pip` 则说明激活成功。 

!!! tip 每一次你在该环境下使用 `pip` 安装了新软件包时，请再次激活该环境。

这样可以确保你在使用由该软件包安装的终端程序（如 `flit`）时使用的是当前虚拟环境中的程序，而不是其他的可能是全局安装的程序。

Flit

FastAPI 使用 Flit 来构建、打包和发布项目。

如上所述激活环境后，安装 `flit`：

```
$ pip install flit

---> 100%
```

现在重新激活环境，以确保你正在使用的是刚刚安装的 `flit`（而不是全局环境的）。

然后使用 `flit` 来安装开发依赖：

=== "Linux, macOS"

```
<div class="termy">

```console
```
```

```
$ flit install --deps develop --symlink

---> 100%
```

```

=== "Windows"

```
If you are on Windows, use `--pth-file` instead of `--symlink`:

```console
$ flit install --deps develop --pth-file

---> 100%
```

```

这将在虚拟环境中安装所有依赖和本地版本的 FastAPI。

**使用本地 FastAPI**

如果你创建一个导入并使用 FastAPI 的 Python 文件，然后使用虚拟环境中的 Python 运行它，它将使用你本地的 FastAPI 源码。

并且如果你更改该本地 FastAPI 的源码，由于它是通过 `--symlink`（或 Windows 上的 `--pth-file`）安装的，当你再次运行那个 Python 文件，它将使用你刚刚编辑过的最新版本的 FastAPI。

这样，你不必再去重新"安装"你的本地版本即可测试所有更改。

## 格式化

你可以运行下面的脚本来格式化和清理所有代码：

```
$ bash scripts/format.sh
```

它还会自动对所有导入代码进行整理。

为了使整理正确进行，你需要在当前环境中安装本地的 FastAPI，即在运行上述段落中的命令时添加 `--symlink`（或 Windows 上的 `--pth-file`）。

## 格式化导入

还有另一个脚本可以格式化所有导入，并确保你没有未使用的导入代码：

```
$ bash scripts/format-imports.sh
```

由于它依次运行了多个命令，并修改和还原了许多文件，所以运行时间会更长一些，因此经常地使用 `scripts/format.sh` 然后仅在提交前执行 `scripts/format-imports.sh` 会更好一些。
```

文档

首先，请确保按上述步骤设置好环境，这将安装所有需要的依赖。

文档使用 MkDocs 生成。

并且在 `./scripts/docs.py` 中还有适用的额外工具/脚本来处理翻译。

!!! tip 你不需要去了解 `./scripts/docs.py` 中的代码，只需在命令行中使用它即可。

所有文档均在 `./docs/en/` 目录中以 Markdown 文件格式保存。

许多的教程章节里包含有代码块。

在大多数情况下，这些代码块是可以直接运行的真实完整的应用程序。

实际上，这些代码块不是写在 Markdown 文件内的，它们是位于 `./docs_src/` 目录中的 Python 文件。

生成站点时，这些 Python 文件会被包含/注入到文档中。

用于测试的文档

大多数的测试实际上都是针对文档中的示例源文件运行的。

这有助于确保：

- 文档始终是最新的。
- 文档示例可以直接运行。
- 绝大多数特性既在文档中得以阐述，又通过测试覆盖进行保障。

在本地开发期间，有一个脚本可以实时重载地构建站点并用来检查所做的任何更改：

```
$ python ./scripts/docs.py live

<span style="color: green;">[INFO]</span> Serving on http://127.0.0.1:8008
<span style="color: green;">[INFO]</span> Start watching changes
<span style="color: green;">[INFO]</span> Start detecting changes
```

它将在 `http://127.0.0.1:8008` 提供对文档的访问。

这样，你可以编辑文档/源文件并实时查看更改。

Typer CLI （可选）

本指引向你展示了如何直接用 `python` 程序运行 `./scripts/docs.py` 中的脚本。

但你也可以使用 Typer CLI，而且在安装了补全功能后，你将可以在终端中对命令进行自动补全。

如果你打算安装 Typer CLI，可以使用以下命令安装自动补全功能：

```
$ typer --install-completion

zsh completion installed in /home/user/.bashrc.
Completion will take effect once you restart the terminal.
```

应用和文档同时运行

如果你使用以下方式运行示例程序：

```
$ uvicorn tutorial001:app --reload

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

由于 Uvicorn 默认使用 `8000` 端口，因此运行在 `8008` 端口上的文档不会与之冲突。

翻译

非常感谢你能够参与文档的翻译！这项工作需要社区的帮助才能完成。 🌎 🚀

以下是参与帮助翻译的步骤。

建议和指南

- 在当前 已有的 pull requests 中查找你使用的语言，添加要求修改或同意合并的评审意见。

!!! tip 你可以为已有的 pull requests 添加包含修改建议的评论。

详情可查看关于 `<a href="https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-request-reviews" class="external-link" target="_blank">添加 pull request 评审意见</a>` 以同意合并或要求修改的文档。

- 在 issues 中查找是否有对你所用语言所进行的协作翻译。
- 每翻译一个页面新增一个 pull request。这将使其他人更容易对其进行评审。

对于我（译注：作者使用西班牙语和英语）不懂的语言，我将在等待其他人评审翻译之后将其合并。

- 你还可以查看是否有你所用语言的翻译，并对其进行评审，这将帮助我了解翻译是否正确以及能否将其合并。
- 使用相同的 Python 示例并且仅翻译文档中的文本。无需进行任何其他更改示例也能正常工作。
- 使用相同的图片、文件名以及链接地址。无需进行任何其他调整来让它们兼容。
- 你可以从 ISO 639-1 代码列表 表中查找你想要翻译语言的两位字母代码。

已有的语言

假设你想将某个页面翻译成已经翻译了一些页面的语言，例如西班牙语。

对于西班牙语来说，它的两位字母代码是 `es`。所以西班牙语翻译的目录位于 `docs/es/`。

!!! tip 主要（"官方"）语言是英语，位于 `docs/en/` 目录。

现在为西班牙语文档运行实时服务器：

```
// Use the command "live" and pass the language code as a CLI argument
$ python ./scripts/docs.py live es

<span style="color: green;">[INFO]</span> Serving on http://127.0.0.1:8008
```

```
<span style="color: green;">[INFO]</span> Start watching changes
<span style="color: green;">[INFO]</span> Start detecting changes
```

现在你可以访问 http://127.0.0.1:8008 实时查看你所做的更改。

如果你查看 FastAPI 的线上文档网站，会看到每种语言都有所有页面。但是某些页面并未被翻译并且会有一处关于缺少翻译的提示。

但是当你像上面这样在本地运行文档时，你只会看到已经翻译的页面。

现在假设你要为 Features{.internal-link target=_blank} 章节添加翻译。

- 复制下面的文件：

```
docs/en/docs/features.md
```

- 粘贴到你想要翻译语言目录的相同位置，比如：

```
docs/es/docs/features.md
```

!!! tip 注意路径和文件名的唯一变化是语言代码，从 `en` 更改为 `es` 。

- 现在打开位于英语文档目录下的 MkDocs 配置文件：

```
docs/en/docs/mkdocs.yml
```

- 在配置文件中找到 `docs/features.md` 所在的位置。结果像这样：

```yaml
site_name: FastAPI
# More stuff
nav:
- FastAPI: index.md
- Languages:
  - en: /
  - es: /es/
- features.md
```

- 打开你正在编辑的语言目录中的 MkDocs 配置文件，例如：

```
docs/es/docs/mkdocs.yml
```

- 将其添加到与英语文档完全相同的位置，例如：

```yaml
site_name: FastAPI
# More stuff
nav:
- FastAPI: index.md
- Languages:
  - en: /
  - es: /es/
- features.md
```

如果配置文件中还有其他条目，请确保你所翻译的新条目和它们之间的顺序与英文版本完全相同。

打开浏览器，现在你将看到文档展示了你所加入的新章节。 

现在，你可以将它全部翻译完并在保存文件后进行预览。

新语言

假设你想要为尚未有任何页面被翻译的语言添加翻译。

假设你想要添加克里奥尔语翻译，而且文档中还没有该语言的翻译。

点击上面提到的链接，可以查到"克里奥尔语"的代码为 `ht` 。

下一步是运行脚本以生成新的翻译目录：

```
// Use the command new-lang, pass the language code as a CLI argument
$ python ./scripts/docs.py new-lang ht

Successfully initialized: docs/ht
Updating ht
Updating en
```

现在，你可以在编辑器中查看新创建的目录 `docs/ht/` 。

!!! tip 在添加实际的翻译之前，仅以此创建首个 pull request 来设定新语言的配置。

这样当你在翻译第一个页面时，其他人可以帮助翻译其他页面。 

首先翻译文档主页 `docs/ht/index.md` 。

然后，你可以根据上面的"已有语言"的指引继续进行翻译。

不支持的新语言

如果在运行实时服务器脚本时收到关于不支持该语言的错误，类似于：

```
 raise TemplateNotFound(template)
jinja2.exceptions.TemplateNotFound: partials/language/xx.html
```

这意味着文档的主题不支持该语言（在这种例子中，编造的语言代码是 `xx` ）。

但是别担心，你可以将主题语言设置为英语，然后翻译文档的内容。

如果你需要这么做，编辑新语言目录下的 `mkdocs.yml` ，它将有类似下面的内容：

```
site_name: FastAPI
# More stuff
theme:
  # More stuff
  language: xx
```

将其中的 language 项从 `xx` （你的语言代码）更改为 `en` 。

然后，你就可以再次启动实时服务器了。

预览结果

当你通过 `live` 命令使用 `./scripts/docs.py` 中的脚本时，该脚本仅展示当前语言已有的文件和翻译。

但是当你完成翻译后，你可以像在线上展示一样测试所有内容。

为此，首先构建所有文档：

```
// Use the command "build-all", this will take a bit
$ python ./scripts/docs.py build-all

Updating es
Updating en
Building docs for: en
Building docs for: es
Successfully built docs for: es
Copying en index.md to README.md
```

这将在 `./docs_build/` 目录中为每一种语言生成全部的文档。还包括添加所有缺少翻译的文件，并带有一条"此文件还没有翻译"的提醒。但是你不需要对该目录执行任何操作。

然后，它针对每种语言构建独立的 MkDocs 站点，将它们组合在一起，并在 `./site/` 目录中生成最终的输出。

然后你可以使用命令 `serve` 来运行生成的站点：

```
// Use the command "serve" after running "build-all"
$ python ./scripts/docs.py serve

Warning: this is a very simple server. For development, use mkdocs serve instead.
This is here only to preview a site with translations already built.
Make sure you run the build-all command first.
Serving at: http://127.0.0.1:8008
```

测试

你可以在本地运行下面的脚本来测试所有代码并生成 HTML 格式的覆盖率报告：

```
$ bash scripts/test-cov-html.sh
```

该命令生成了一个 `./htmlcov/` 目录，如果你在浏览器中打开 `./htmlcov/index.html` 文件，你可以交互式地浏览被测试所覆盖的代码区块，并注意是否缺少了任何区块。