用 Poetry 创建并发布 Python 包

johnfraney Python程序员 今天

如果您曾经使用setup.py脚本发布过一个python包，那么您可能会发现编写脚本来发布包比编写包本身要困难。

Python 开发人员认识到了这一点，并且有一些工具采用了一种更现代的方法来构建包。Poetry_和 Flit 是构建Python 包的两种流行工具。

因为我使用了Poetry作为一个Python依赖性管理工具，所以我决定玩转它的包管理功能。

为了获得使用Poetry发布包的实践经验，我最近发布了Flake8 Markdown，一个用 Flake8 在Markdown文件中lint Python 代码的工具。

本文将介绍我对Flake8 Markdown的pyproject.toml和codebase所做的更改，以使其可以在 PyPI（python包索引）上发布。

> 信息 已发布版本请查看Flake8 Markdown 仓库。

创建一个包

用 Poetry 创建一个包, 如果 Poetry 已安装，是会有帮助的。要做到这一点，请遵循 Poetry 安装说明。现在，用Poetry 创建一个包，我们将运行 poetry new，并提供包含该包的目录的名称:

```
$ poetry new flake8-markdown
Created package flake8-markdown in flake8-markdown
```

我们收到邮件了! 不管怎样，我们有一个包裹 (package，双关语---译者注)。让我们打开它:

```
flake8-markdown/
├── flake8_markdown/
│   └── __init__.py
├── tests/
│   ├── __init__.py
│   └── test_flake8_markdown.py
├── pyproject.toml
└── README.rst
```

开箱即用，Poetry为我们提供了一个简单的包结构和 pyproject.toml 文件:

- 0.1.0的包版本
- Python 的最小版本(在我的例子中是^3.7)
- pytest 支持和一个单元测试

> 信息
> 关于 pyproject.toml的信息和背景，见 PEP 518。

如果我们将这个包按原样发布在 Python包索引(PyPI)上，那么这个包看起来就不怎么样，所以让我们让它看起来像值得安装的东西。

自定义包

以下是初始 pyproject.toml文件:

```
[tool.poetry]
name = "flake8-markdown"
version = "0.1.0"
description = ""
authors = ["John Franey <johnfraney@gmail.com>"]

[tool.poetry.dependencies]
python = "^3.7"

[tool.poetry.dev-dependencies]
pytest = "^3.0"

[build-system]
requires = ["poetry>=0.12"]
build-backend = "poetry.masonry.api"
```

以下是完成的 pyproject.toml:

```toml
[tool.poetry]
name = "flake8-markdown"
version = "0.1.1"
description = "Lints Python code blocks in Markdown files using flake8"
authors = ["John Franey <johnfraney@gmail.com>"]
# New attributes
license = "MIT"
readme = "README.md"
homepage = "https://github.com/johnfraney/flake8-markdown"
repository = "https://github.com/johnfraney/flake8-markdown"
keywords = ["flake8", "markdown", "lint"]
classifiers = [
    "Environment :: Console",
    "Framework :: Flake8",
    "Operating System :: OS Independent",
    "Topic :: Software Development :: Documentation",
    "Topic :: Software Development :: Libraries :: Python Modules",
    "Topic :: Software Development :: Quality Assurance",
]
include = [
    "LICENSE",
]

[tool.poetry.dependencies]
# Updated Python version
python = "^3.6"
# New dependency
flake8 = "^3.7"

[tool.poetry.dev-dependencies]
pytest = "^3.0"

# New scripts
[tool.poetry.scripts]
flake8-markdown = 'flake8_markdown:main'

[build-system]
requires = ["poetry>=0.12"]
build-backend = "poetry.masonry.api"
```

> 信息
> 有关自定义Poetry包的可用部分的完整列表，请参阅 Poetry's pyproject.toml 文档。 让我们从上到下逐步进行更改。

version

Version 描述了一个包的当前版本。我尝试遵循语义版本控制，所以当更新向后不兼容时很明显。

> 注
> 一旦一个包的版本存在于pypi上，就不可能使用相同的版本号上传不同的代码。在发布Pypi上的版本之前，请确保使用 TestPyPI 进行测试更改。

description

这个简短的描述出现在 pip search 和 PyPI搜索结果 中。

license

许可证出现在包的 PyPI 页面的“Meta”部分。

我用 Github 的 choosealicense.com 找到了一个合适的许可证，并选定 MIT。Poetry 还列出了其他常见的许可证 和推荐的记号。

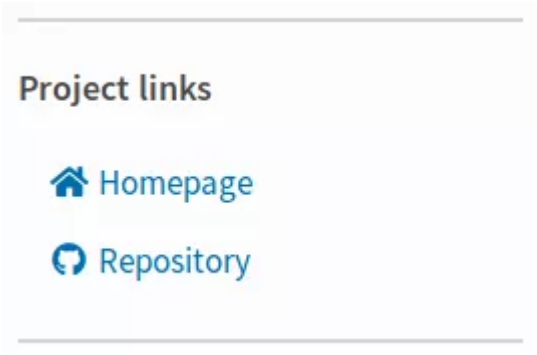
readme

Poetry创建了一个 README.rst 文件，但我更喜欢用 Markdown 编写文档。 README 的内容在 PyPI 上显示为项目描述，因此很好地解释为什么以及如何使用你的包。 我的 README 包括：

- 盾牌（因为它们看起来很漂亮）
- 引言
- 安装
- 使用
- 行为准则
- 历史记录（大致遵循保留一个变更日志）

homepage and repository

这些出现在包的 PyPI 页面的 Project 链接部分：



如果你的包有一个单独的文档网站，那么还有一个 documentation 属性。

keywords

这些出现在 PyPI 页面的 Meta 部分：

classifiers

“Trove分类器”作为 PyPI 包的类别，可以用来过滤 PyPI 上的包。它们出现在 PyPI 页面的“分类器”部分：

完整列表见 PyPI 分类器页面。

[tool.poetry.dependencies]

“依赖项”部分展示了Poetry的一个最佳特征。

当使用普通的 pip 或 Pipenv管理包时，通常在两个位置指定包：dev和部署依赖项的requirements.txt或Pipfile，以及运行时/安装依赖项的setup.py。

Poetry对所有依赖项使用pyproject.toml，这简化了依赖项管理。

这部分包括两个更改：

1. 将最低的 Python 版本从3.7更新到3.6，以获得更广泛的兼容性
2. 添加flake8作为依赖项

[tool.poetry.scripts]

Scripts 是“安装包时将被安装的脚本或可执行文件”。换句话说，开发人员可以在这里根据函数创建CLI命令。

脚本的形式为：

```
script_name = '{package_name}:{function_name}'
```

Flake8 markdown — 包含你的惊喜 — 有一个名为flake8 markdown的CLI命令：

```
flake8-markdown = 'flake8_markdown:main'
```

在安装 flake8-markdown 包之后, 运行 flake8-markdown 将从 flake8_markdown/__init__.py调用 main() 函数。

> 信息 要使包成为一个可运行模块，比如python -m flake8-markdown，它需要一个 __main__.py 模块。在Flake8 Markdown中， __main__.py 文件导入并运行与上述脚本相同的 main() 函数。

发布包

TestPyPI

TestPyPI 是“一个独立的Python包索引实例，它允许您在不影响实际索引的情况下尝试分发工具和进程”。将包上传到TestPyPI 并从那里安装，可以帮助包维护人员避免发送损坏的包版本。
让我们看看如何将包上载到TestPyPI。

> 注
> 在将包上载到测试包索引之前，您需要注册一个TestPyPI 帐户。

首先，构建包：

```
$ poetry build
```

接下来，将测试PyPI添加为备用包存储库：

```
$ poetry config repositories.testpypi https://test.pypi.org/simple
```

现在，把包发布到TestPyPI：

```
$ poetry publish -r testpypi

Publishing flake8-markdown (0.1.1) to testpypi
Username:
Password:

 - Uploading flake8-markdown-0.1.1.tar.gz 100%
 - Uploading flake8_markdown-0.1.1-py3-none-any.whl 100%
```

最后，通过在 testpypi.pypi.org 上查看包并在独立的虚拟环境中安装测试版本，验证包的外观和工作是否符合预期：

```
pip install --index-url https://test.pypi.org/simple/ flake8-markdown
```

PyPI

如果这个包在测试PyPI上看起来很好，并且正常启动，那么发布到PyPI就非常简单：

```
poetry publish
```

> 注
> 在将包上传到包索引之前，你需要注册一个PyPI帐户 。此帐户与TestPyPI上的任何帐户都是独立的。

总结

PEP 517 打开了Poetry等工具的大门，提供了一种对开发人员友好的构建 Python 包的方式。因此，用Poetry创建和发布一个包是一种直接的、无需费力的体验。构建一个包就像编写代码和向pyproject.toml文件中添加各部分一样简单。

很愉快，我决定写一首关于它的诗：

> 浮动Python代码 用Poetry集结 云端之旅

Poetry 诗歌。怎么样?

> 英文原文：https://johnfraney.ca/posts/2019/05/28/create-publish-python-package-poetry/
> 译者：青书