Python と東海と私

Atsushi Odagiri

2024-11-16

Agenda

はじめに

Python と私

Python Packaging

まとめ

お前誰よ

- Atsushi Odagiri
- 2000 年〜2004 年 豊橋技術科学大学
- 2005 年 名古屋で就職
- 2016 年〜 Open Collector
- Python は大学時代に 1.5 くらいのころから

Python と私

- aodag 青春グラフィティ後半 東海編
- 前半の静岡編は PyCon mini 静岡でやる予定だったのにね (´・ω・`)

2000 年 ハッカーになろう

- 当時豊橋の大学生
- How To Become A Hacker (和訳：ハッカーになろう) を読んで Python に触れる
  - 当時のバージョンでは学ぶべき言語は Python, Perl, C, Lisp となっていた
  - 後のバージョンで Java が加わり Go にも言及している

2001年 - 2002年 Zopeとかコミュニティとか

- 当時 Python といえば Zope という勢いがあった
- Japan Zope Users Group(JZUG)
- 名古屋のコミュニティ参加
- Nagoya Zope Users Group(NZUG)

2004 年 - 2005 年 就職と秘密兵器

- 名古屋で就職
- Python の仕事？ ありません
- Java とか PHP、あと .NET
- こっそりインストールした Python でデータ生成やテキスト加工とかこっそりやる
  - スクリプトとして正しい使い方

2006 年 Python コミュニティへ

- 一回目の転職もこのころ
- Python Developers Camp 2006 富士
- その年忘年会を各地で同時開催 名古屋で幹事をやる

2007 年 東海 Python Workshop

- Python Developers Camp 2007 Winter 志賀高原
- 東海 Python Workshop 01
  - 30 人くらいは来てくれた記憶
  - 地元発表者の不足
  - 地元コミュニティが必要だ

2008 年 Python 東海

- 2008 年に開始した... と思う
- 割とすぐに東京に行くことになってしまい、早々に主催を交代
- 主催はどんどん変わっていいです

名古屋で勉強会とか

- FLOSS 桜山
- メンバーがかぶりがちな勉強会主催者がよく来るので日程調節

Python パッケージングのおさらいをしよう

python パッケージをインストールしてみましょう

Python パッケージをインストール

- Python Packaging は PEP で定義されるようになっています
- PEP と Python 本体の仕様だけを信じてインストールしてみましょう
- python で実行するとうっかり pip を呼んでしまいそうなので UNIX コマンドを使います

パッケージをインストールして使えるまで

- pypi で配布物のダウンロード URL を取得
- 配布物をダウンロード
- 配布物をインストール先に展開

どこからダウンロードする？

PEP 503, 691 にあります

- PyPI の project データ
- Simple Repository

Simple Repository

PEP 503 にしたがって PyPI のプロジェクトデータを確認します。

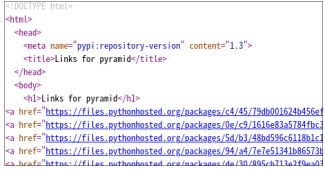
- https://pypi.org/simple/project/ で情報を取得

```
$ curl https://pypi.org/simple/pyramid/
```

HTML が返ってきます。この HTML をパースしてもよいのですが...

Links for pyramid

pyramid-1.0a1.tar.gz
pyramid-1.0a2.tar.gz
pyramid-1.0a3.tar.gz
pyramid-1.0a4.tar.gz
pyramid-1.0a5.tar.gz
pyramid-1.0a6.tar.gz



JSON API

PEP691 には JSON API が定義されていて ACCEPT ヘッダでレスポンスが変わることとなっています。

- https://pypi.org/simple/project/ で情報を JSON で取得
- application/vnd.pypi.simple.v1+json ACCEPT ヘッダで JSON を受け取る
- 情報が多いのでファイル名だけ jq で抜き取ってみます

```
$ curl https://pypi.org/simple/pyramid/ \
    -H "ACCEPT: application/vnd.pypi.simple.v1+json" \
        | jq -c ".files[].filename"
```

PyPIから取得できた情報

```
"pyramid-1.0a1.tar.gz"
"pyramid-1.0a2.tar.gz"
...
"pyramid-2.0.1-py3-none-any.whl"
"pyramid-2.0.1.tar.gz"
"pyramid-2.0.2-py3-none-any.whl"
"pyramid-2.0.2.tar.gz"
```

いっぱいありますね

どれをダウンロードしよう？

例えば pyramid-2.0.2-py3-none-any.whl を PEP 427 にしたがって解釈すると以下のような情報が入っています。

- pyramid 配布物の名前
- 2.0.2 配布物のバージョン
- py3 対応している python バージョン
- none バイナリの形式
- any プラットフォーム OS やリンクしている libc などのアーキテクチャ
- whl 配布物の形式

ダウンロード

いったんは最新バージョンで wheel ファイルをダウンロードしてみましょう。

```
$ curl https://pypi.org/simple/pyramid/ \
    -H "ACCEPT: application/vnd.pypi.simple.v1+json" \
        | jq -c '.files[]
        | select(
        .filename == "pyramid-2.0.2-py3-none-any.whl").url'
```

https://files.pythonhosted.org/packages/db/41/a[..]af/pyramid-2.0.2-py3-none-any.whl

ダウンロードした配布物のフォーマット

- sdist pyproject.toml を含むアーカイブ (zip, tar) ファイル
- bdist wheel ソースコードとメタデータを含む ZIP ファイル

pyramid-2.0.2-py3-none-any.whl は wheel ファイルフォーマットの配布物

wheel ファイルの中身

とりあえず ZIP ファイルなので unzip で中を見てみましょう

```
$ unzip -t pyramid-2.0.2-py3-none-any.whl
Archive:  pyramid-2.0.2-py3-none-any.whl
    testing: pyramid/__init__.py        OK
    testing: pyramid/asset.py           OK
    testing: pyramid/authentication.py    OK
        ...
    testing: pyramid/scripts/ptweens.py    OK
    testing: pyramid/scripts/pviews.py    OK
    testing: pyramid-2.0.2.dist-info/LICENSE.txt    OK
    testing: pyramid-2.0.2.dist-info/METADATA    OK
    testing: pyramid-2.0.2.dist-info/WHEEL    OK
    testing: pyramid-2.0.2.dist-info/entry_points.txt    OK
    testing: pyramid-2.0.2.dist-info/top_level.txt    OK
    testing: pyramid-2.0.2.dist-info/RECORD    OK
```

dist-info

PEP427 によると project.dist-info ディレクトリにはソースコード以外の配布物の情報が入っている

- パッケージメタデータ METADATA
- wheel に関する情報 WHEEL
- インストールデータベース (要はファイル一覧) RECORD
- ライセンスファイルやその他色々 (setuptools 由来のものとかまだ入ってる)

パッケージメタデータを読む理由

- パッケージメタデータから依存ライブラリの情報を取得
  - 依存関係のグラフを作成してバージョン指定などを解決（難しいので今日は割愛）

さあインストールだ

- どのディレクトリに？
- site-packages の特定方法
- system site-packages vs user site-packages

```
$ python3 -c 'print(__import__("site").getsitepackages())'
['/usr/local/lib/python3.12/dist-packages',
 '/usr/lib/python3/dist-packages',
 '/usr/lib/python3.12/dist-packages']
$ python3 -c 'print(__import__("site").getusersitepackages())'
/home/aodag/.local/lib/python3.12/site-packages
```

EXTERNAL-MANAGED

- OS パッケージで管理してる site-packages にインストールしてはだめ (EXTERNALLY-MANAGED) PEP 668

Debian の例

```
$ cat /usr/lib/python3.12/EXTERNALLY-MANAGED
[externally-managed]
Error=To install Python packages system-wide, try apt install
 python3-xyz, where xyz is the package you are trying to
 install.
...
 See /usr/share/doc/python3.12/README.venv for more information.
```

インストールの手順

- WHEEL ファイルの中身を確認 purelib or platlib
- RECORD ファイルでインストールするファイルをチェック PEP 376
- site-packages 以下にコピー

ライブラリが purelib か platlib か確認

WHEEL ファイルの Root-Is-Purelib を確認する

```
$ rg Root-Is-Purelib pyramid-2.0.2.dist-info/WHEEL
3:Root-Is-Purelib: true
```

wheel の中身を site-packages に展開する

```
$ python3 -m venv .venv
$ .venv/bin/python -c 'print(__import__("sysconfig")\
                                .get_path("purelib"))'
/home/aodag/[...]/lib/python3.12/site-packages
$ mv * \
  $(.venv/bin/python -c 'print(__import__("sysconfig")\
                                .get_path("purelib"))')
```

残りの pyramid の依存ライブラリも同様にしてインストールしましょう。

メタデータ中の依存関係

```
Requires-Dist: hupper >=1.5
...
Requires-Dist: webob >=1.8.3
Requires-Dist: zope.deprecation >=3.5.0
Requires-Dist: zope.interface >=3.8.0
Provides-Extra: docs
Requires-Dist: Sphinx >=3.0.0 ; extra == 'docs'
Requires-Dist: docutils ; extra == 'docs'
...
Provides-Extra: testing
Requires-Dist: webtest >=1.3.1 ; extra == 'testing'
...
```

依存関係記述の形式

- パッケージ名
- バージョン
- 特定の実行環境で必要になる依存関係
  - os_name
  - sys_platform
  - ...

extras って？

- 追加機能とかで必要になる依存ライブラリなど
- テストやドキュメントのビルドなど
- 通常利用では必要でない依存ライブラリを別グループで管理しているもの

wheel のファイル名再び

C 拡張が入ってる wheel のファイル名は複雑

- zope.interface-7.1.1-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_
- pillow-11.0.0-cp313-cp313t-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
- PyQt6-6.7.1-cp38-abi3-manylinux_2_28_x86_64.whl

Python ABI とプラットフォーム

OS とリンクしてる libc と Python ビルド ABI と...

- cp313 vs cp313t
  - cp313 は CPython 3.13 という意味
  - GIL Free オプションが有効になってる場合は cp313t と t サフィックスが付く
- abi3
  - Python3 であればマイナーバージョンが異なっても利用できる C 拡張 API だけで作られているもの
- manylinux
  - Linux はディストリビューションがたくさんある
  - manylinux という最小高分母的な仮定のディストロをプラットフォームとしている
  - 主な違いは glibc のバージョンなので manylinux2014 以降は直接 glibc バージョンを指定するようになった

linux 以外のプラットフォーム

Windows と Mac について

- Windows
  - 過去に python バージョンごとにどのコンパイラバージョンでどの LIBC をリンクするかという知見が必要だった
  - 現在は LIBC が universal CRT に固定されたようなのであまり気にしていないと思われます (Python3.5 くらいから)
- Mac はよくわからないです（ごめんね）
  - 概ね OS のバージョンで下位互換になっているはずです。
  - M? な ARM 系 Mac の場合は macosx_11_0_arm64 など arm64 となっているものを使いましょう。

wheel 形式のパッケージインストール完了！

じゃあ sdist の場合は？

sdist から wheel を作る

- PyPI に wheel をアップロードするだけであれば好きな方法でいい
  - その wheel がどう作られたのかはインストールで問題にならない
  - 手作業でソースコードとメタデータファイルを zip で固めて正しいファイル名をつければインストールできます
- sdist を配布するのであればユーザーが build することになる
  - C 拡張が入っていたり別の言語からのコンパイルだったり
  - ツールの準備などなど
  - build するための手順はいくらでも複雑になる

sdist をインストールする

- 実際ビルドするのは各種のビルドバックエンド
- PEP517 が決めるのはビルドバックエンドの呼び出し方
- PEP518 でビルドバックエンドの指定方法が決められている

PEP517 ビルドの流れ

- sdist をダウンロードした後に
- ビルド専用の venv を作成
- venv 内に pyproject.toml に記述されているビルドバックエンドを準備
- ビルドバックエンドを実行して wheel を作成
- 作成された wheel をインストール

sdist から wheel を作ろう

ダウンロードして展開

```
$ wget $(curl https://pypi.org/simple/pyramid/ \
    -H "ACCEPT: application/vnd.pypi.simple.v1+json" \
  | jq -r -c '.files[]
              |
              select(.filename == "pyramid-2.0.2.tar.gz").url')
$ tar xvf pyramid-2.0.2.tar.gz
$ cd pyramid-2.0.2
```

pyproject.toml から build-system を確認

```
$ rg -A 5 build-system pyramid-2.0.2/pyproject.toml
1:[build-system]
2-requires = ["setuptools", "wheel"]
3-build-backend = "setuptools.build_meta"
4-
5-[tool.black]
6-line-length = 79
```

ビルド環境を作る

build-system.requires に列挙されていた setuptools と wheel をインストールします。
(ここでの wheel は wheel という名前のツールのこと)

```
$ python3 -m venv pyramid-2.0.2-build-venv
$ ./pyramid-2.0.2-build-venv/bin/pip install setuptools wheel
```

wheel を作る！

PEP517 のとおりにビルドバックエンドを import して build_wheel 関数を呼びます。
引数で指定した dist ディレクトリに wheel が作成されます。

```
$ ../pyramid-2.0.2-build-venv/bin/python \
    -c 'print("build result:",
                __import__("setuptools.build_meta")
                    .build_meta.build_wheel("dist"))'
...
build result: pyramid-2.0.2-py3-none-any.whl
```

できあがった wheel は PyPI からダウンロードした wheel と同じようにインストールできます。

足りてないもの

バージョンロックの方法がまだありません

- いろんな環境でインストールするときにバージョンを固定しておきたい
  - 本番環境、開発環境、デモ環境....

各種ツールのバージョンロック

- pip で requirements.txt を管理
  - 直接依存と間接依存を区別するのが面倒
  - 更新する方法とか気をつけないと...
- constraints の併用
  - 直接依存と間接依存を区別
  - 更新する方法は煩雑 (ツールでサポートとかしてるわけじゃない)
  - 依存グラフを復元できない
- piptool
  - 依存関係の情報をコメントで追記
- poetry や flit など (多分 uv あたりも)
  - ツールごとに独自の形式
  - ダウンロード URL やファイルハッシュ、依存関係などの追加情報を多く含む

ロックファイル

PEP 751 – A file format to record Python dependencies for installation reproducibility

- 議論中
- extras group や dependency graph の情報など、poetry.lock のなどと同様の情報
- pylock.toml というファイル名になる予定

まとめ

- 名古屋懐かしい
- PEP どおりにやればちゃんとインストールできる
- でも構成管理とかを考えるとまだ足りない？
- ロックファイル早く標準化してほしいですね

参考文献

- Packaging PEPs
  - PEP 376 – Database of Installed Python Distributions
  - PEP 425 – Compatibility Tags for Built Distributions
  - PEP 427 – The Wheel Binary Package Format 1.0
  - PEP 440 – Version Identification and Dependency Specification
  - PEP 496 – Environment Markers
  - PEP 508 – Dependency specification for Python Software Packages
  - PEP 517 – A build-system independent format for source trees
  - PEP 518 – Specifying Minimum Build System Requirements for Python Projects
  - PEP 668 – Marking Python base environments as “externally managed”
  - PEP 691 – JSON-based Simple API for Python Package Indexes
  - PEP 751 – A file format to record Python dependencies for installation reproducibility
- Python Packaging User Guide
- sysconfig — Provide access to Python’s configuration information
- Support for new cp313t GIL-free ABI variant