

パッケージを配ろう

Atsushi Odagiri

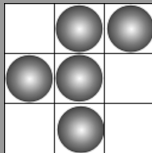
2024-04-25

Outline

- 1 パッケージを配ろう
- 2 パッケージを配るということ
- 3 パッケージを配るための PEP
- 4 参考文献

お前誰よ

- Atsushi Odagiri
- Open Collector
- Python は 1.5 くらいのところから



ケソオール

パッケージエコシステム

- 作る
- 配る
- 使う

パッケージを配るということ

- 広く一般に向けて配る
- 狭い範囲で限られた利用のために配る

広く一般に向けて pypi で配る

- PyPA ツールのデフォルト
- `tween` でアップロード
- `pip` がダウンロードしてインストール

狭い範囲で限られた利用のために配る

- マイクロサービスのそれぞれで使うようなライブラリ
- 特殊なパッチをあてたローカルフレーバーライブラリ

狭い範囲で配る

- 社内ネットワークや VPN の中で
- k8s や vpc の中で
- 範囲内の IP アドレスにだけ
- 認証をつけたい

httplib.server でのお手軽 repository

- ダウンロードできるリンクがあればいいので http モジュールでサーバーを起動するだけ
- wheel ファイルのあるディレクトリで実行

```
python3 -m pip download pyramid  
python3 -m http.server
```

Directory listing for /

- [hupper-1.12.1-py3-none-any.whl](#)
- [PasteDeploy-3.1.0-py3-none-any.whl](#)
- [plaster-1.1.2-py2.py3-none-any.whl](#)
- [plaster_pastedeploy-1.0.1-py2.py3-none-any.whl](#)
- [pyramid-2.0.2-py3-none-any.whl](#)
- [setuptools-69.5.1-py3-none-any.whl](#)
- [translationstring-1.4-py2.py3-none-any.whl](#)
- [venusian-3.1.0-py3-none-any.whl](#)
- [WebOb-1.8.7-py2.py3-none-any.whl](#)
- [zope.deprecation-5.0-py3-none-any.whl](#)
- [zope.interface-6.3-cp311-cp311-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl](#)

URL 指定でインストール

- pip は URL 指定で直接インストールできる
- 正確なファイル名を知らないといけない
- wheel はプラットフォームなどの情報を含んでいる

```
pip install \  
http://localhost:8000/pyramid-2.0.2-py3-none-any.whl
```

複雑な wheel ファイル名

- oh...

zope.interface-6.4

-cp311

-cp311

-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.
.whl

find-links

- `find-links` で指定した場所から探してもらう

```
pip install -f http://localhost:8000 zope.interface
```

no-index

- 場合によっては pypi への接続も制限される環境
- 全てをお手軽 repository から取得するなら no-index も使うようにしてみよう
- no-index pypi などの index を見にいかない
- find-url 指定したページからダウンロード URL をスクレーピング

index は必要？

- pip を直接使うなら `find-url` でもいいかも？
- メタデータを取得するのに配布物をダウンロードするという効率の悪さはある
- `poetry source add` で使えるのは `simple repository`
 - pip だと `--index-url` で指定するものに相当

独自の pypi を立てたい!

- PyPI 自体のソースコードは公開されている
 - <https://github.com/pypi/warehouse>
 - インフラ構築保守など手間もかかる
- devpi
 - <https://github.com/devpi/devpi>
 - PyPI へのプロキシやプロジェクトごとの名前空間設定など多機能
 - それなりにインフラ構築保守の手間がかかる
- `http.server` くらいに簡単に立ち上って欲しいところ

パッケージを配るための PEP

- PEP 458 – Secure PyPI downloads with signed repository metadata
- PEP 480 – Surviving a Compromise of PyPI: End-to-end signing of packages
- PEP 503 – Simple Repository API
- PEP 592 – Adding “Yank” Support to the Simple API
- PEP 629 – Versioning PyPI’s Simple API
- PEP 658 – Serve Distribution Metadata in the Simple Repository API
- PEP 691 – JSON-based Simple API for Python Package Indexes
- PEP 700 – Additional Fields for the Simple API for Package Indexes
- PEP 714 – Rename dist-info-metadata in the Simple API

Simple Repository

representation

- HTML PEP503
- JSON PEP691

バージョン

- 1.0 PEP503/PEP691
- 1.1 PEP700
- PEP714 メタデータフィールドの取り扱いについての修正
 - warehouse の実装で間違えがあったらしい

PyPI の Simple Repository

- <https://pypi.org/simple/> とても大きいのでアクセス注意！

実装方針

- 標準ライブラリでいこう
 - Batteries Included!
- 1 ファイルデプロイ
- DB などを使わず起動するだけで使える

標準ライブラリで web アプリケーションを書く

- json
- wsgiref

project list

- ホストしているプロジェクト (ほぼパッケージの意味) を一覧で出すだけ
- v1.0 のプロジェクトに関する情報は `name` のみ

使うライブラリ

- これだけ!
- 100% 標準ライブラリのみ!

```
import argparse
import hashlib
import itertools
import json
import operator
import pathlib
import re
import zipfile

from typing import TypedDict, NotRequired, Iterable
from wsgiref.types import WSGIApplication, WSGIEnvironment, St
from wsgiref.simple_server import make_server
```

Meta

- simple repository に関する情報
- バージョン

```
Meta = TypedDict(  
    "Meta",  
    {  
        "api-version": str,  
    },  
)
```

project detail

- プロジェクト (パッケージ) ごとのダウンロード可能なファイル一覧
- ファイルの URL やパッケージメタデータなど

project file の typing

- 事前に確認可能なパッケージメタデータ
- ダウンロードに必要な情報 URL やハッシュ

```
ProjectFile = TypedDict(  
    "ProjectFile",  
    {  
        "filename": str,  
        "url": str,  
        "hashes": dict[str, str],  
        "requires-python": NotRequired[str],  
        "dist-info-metadata": NotRequired[str],  
        "gpg-sig": NotRequired[bool],  
        "yanked": NotRequired[bool],  
    },  
)
```

project detail の typing

- project file の一覧が主な情報

```
ProjectDetail = TypedDict(  
    "ProjectDetail",  
    {  
        "name": str,  
        "files": list[ProjectFile],  
        "meta": Meta,  
    },  
)
```

project list の typing

```
Project = TypedDict("Project", {"name": str})
ProjectList = TypedDict(
    "ProjectList",
    {
        "meta": Meta,
        "projects": list[Project],
    },
)
```

wheel ファイルを探し出す

- pathlib でできちゃうね!

```
wheelhouse.glob("*.whl")
```

wheel ファイル名から情報を取得

- wheel ファイルのファイル名は形式が決まっている
 - PEP 491 The Wheel Binary Package Format 1.9
 - `{distribution}-{version}(-{build tag})?-{python tag}-{abi tag}-{platform tag}.whl`.

wheel ファイル名から情報を取得

- 今回欲しいのは `distiribution`
- `"-"` で `split` して最初の1つ

```
def extract_dist_name(wh: pathlib.Path) -> str:  
    return wh.name.split("-", 1)[0]
```

プロジェクト名を正規化

- PEP 503 で正式に正規化方法が定義されている
- アルファベットは全て小文字
- 記号は - に正規化
- 例: `zope.interface` -> `zope-interface`

```
def normalize(name: str) -> str:  
    return re.sub(r"[_\.]+", "-", name).lower()
```

metadata

- METADATA を wheel から取り出す
- wheel は zip ファイル
- METADATA の場所は決まっている
 - PEP 491 The Wheel Binary Package Format 1.9
 - {distribution}-{version}.dist-info/ contains metadata.

```
def get_metadata(whl: pathlib.Path):  
    parts = whl.name.split("-")  
    dist_name, version = parts[0], parts[1]  
    metadata_path = f"{dist_name}-{version}.dist-info/METADATA"  
    with zipfile.ZipFile(whl) as zf:  
        with zf.open(metadata_path) as metadata:  
            return metadata.read()
```


全部まとめて wheel ファイルの情報を取得

- プロジェクト名をキーにしてメタデータと wheel ファイルパスをグループピング

```
def load_wheels(
    wheelhouse: pathlib.Path,
) -> itertools.groupby[str, tuple[str, bytes, pathlib.Path]]:
    wheels = itertools.groupby(
        (
            (normalize(extract_dist_name(w)), get_metadata(w),
             for w in wheelhouse.glob("*.whl"))
        ),
        key=operator.itemgetter(0),
    )
    return wheels
```

プロジェクトごとにファイル情報をまとめる

- プロジェクト名、メタデータ、wheel ファイルパスをもとに JSON データを作成

```
def load_projects(
    wheelhouse: pathlib.Path,
) -> tuple[ProjectList, dict[str, ProjectDetail]]:
    meta = Meta({"api-version": "1.0"})

    project_list = ProjectList(
        {
            "meta": meta,
            "projects": [],
        }
    )

    project_details: dict[str, ProjectDetail] = {}
    for project_name, files in load_wheels(wheelhouse):
        project = ProjectDetail({"name": project_name, "files":
```

wsgi アプリケーション:project list

```
class ProjectListApp:
    def __init__(self, project_list: ProjectList) -> None:
        self.project_list = project_list

    def __call__(
        self, environ: WSGIEnvironment, start_response: StartResponse
    ) -> Iterable[bytes]:
        start_response(
            "200 OK", [("Content-Type", "application/vnd.pypi.project-list+json")]
        )
        return [json.dumps(self.project_list).encode("utf-8")]
```

wsgi アプリケーション:project detail

```
class ProjectDetailApp:
    def __init__(self, project_details: dict[str, ProjectDetail]) -> None:
        self.project_details = project_details

    def __call__(
        self, environ: WSGIEnvironment, start_response: StartResponse
    ) -> Iterable[bytes]:
        project_name = environ["wsgiorg.routing_args"][1]["project_name"]
        if project_name not in self.project_details:
            start_response(
                "404 Not Found",
                [("Content-Type", "application/vnd.pypi.simple.v1+json")]
            )
            return [b""]
        start_response(
            "200 OK", [("Content-Type", "application/vnd.pypi.simple.v1+json")]
        )
```

wsgi アプリケーション: ダウンロード

- wheel ファイルの中身をレスポンスボディにする
- wheel の content-type は特に決まってないので application/octet-stream にする
- ブラウザでアクセスしたときにダウンロードになるよう Content-Disposition をつける

```
class WheelDownloadApp:
    def __init__(self, wheelhouse: pathlib.Path) -> None:
        self.wheelhouse = wheelhouse

    def __call__(
        self, environ: WSGIEnvironment, start_response: StartResponse
    ) -> Iterable[bytes]:
        file_name: str = environ["wsgiorg.routing_args"][1]["wsgiorg.routing_args"]
        p = self.wheelhouse / file_name
        if not p.exists():
            start_response(
                "404 Not Found",
```

WSGI アプリケーションのルーティング

- wsgi.org のドキュメントにルーティングのサンプルがあるので流用
- https://wsgi.readthedocs.io/en/latest/specifications/routing_args.html

```
class RegexDispatch(object):
    def __init__(self, patterns: list[tuple[re.Pattern, WSGIAp
        self.patterns = patterns

    def __call__(
        self, environ: WSGIEnvironment, start_response: StartF
    ) -> Iterable[bytes]:
        script_name: str = environ.get("SCRIPT_NAME", "")
        path_info: str = environ.get("PATH_INFO", "")
        for regex, application in self.patterns:
            match = regex.match(path_info)
            if not match:
                continue
            extra_path_info = path_info[match.end():]
```

simple repository の機能

- / project list
- /{project} project detail
- 実際に wheel ファイルをダウンロードする URL
 - 今回は /{project}/files/{wheel} にします

```
def make_app(wheelhouse: pathlib.Path) -> WSGIApplication:
    (project_list, project_details) = load_projects(wheelhouse)
    app: WSGIApplication = RegexDispatch(
        [
            (re.compile(r"^/$"), ProjectListApp(project_list)),
            (
                re.compile(r"^/(?P<project_name>\w+)$"),
                ProjectDetailApp(project_details),
            ),
            (
                re.compile(
                    r"^/(?P<project_name>\w+)/files/(?P<wheel_
```

さあ!wsgi アプリケーションを立ち上げよう!

- 重要なのは wheel ファイルを置いてある wheelhouse ディレクトリ
- host, port は web アプリケーションとして必要な情報

```
def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("wheelhouse", type=pathlib.Path)
    parser.add_argument("--host", type=str, default="0.0.0.0")
    parser.add_argument("--port", type=int, default=8000)
    args = parser.parse_args()
    app = make_app(args.wheelhouse)
    httpd = make_server(args.host, args.port, app)
    httpd.serve_forever()

if __name__ == "__main__":
    main()
```


pip から使う

- project list 呼ばれてないかも？

```
$ pip install pyramid --index-url=http://localhost:8000/
```

The Update Framework

- TUF

参考文献

- PyPA Simple Repository API, <https://packaging.python.org/en/latest/specifications/simple-repository-api/>
- The Update Framework, <https://theupdateframework.io/>