PDM- 基于 PEP 582 的 python 包管理

作者：bingoct
原文链接：https://ld246.com/article/1649917988078
来源网站：链滴

预备知识：pythonpath、pyvenv 、PEP 582。

官网 https://pdm.fming.dev/

简易使用说明

1. 安装　pipx install pdm，添加至$PATH，bash配置pdm 自动补全pdm completion bash | sudo ee /etc/bash_completion.d/_pdm

2. pdm config 查看全局

开启pdm缓存：pdm config install.cache on

配置pypi_url源：pdm config pypi https://mirrors.tencent.com/pypi/simple/

删除配置：pdm config --delete foo bar

3. 初始化python项目：　pdm init，在项目目录下生成pyproject.toml、.pdm.toml等文件

切换python版本：pdm use pythonx.x

环境查看：pdm info

```
PDM version:        1.14.0
Python Interpreter: $HOME/.local/pipx/venvs/pdm/bin/python (3.8)
Project Root:       $HOME/py-program
Project Packages:   $HOME/__pypackages__/3.8
```

4. 包管理，在项目目录下

- 导入requirements.txt环境配置：　pdm import -f {file}
- 安装包:　pdm add <pkg>
- 更新包：　pdm update <pkg>[--dry-run] 显示差异，并不实际安装[--save-minimum] 保持小依赖版本的包
- 卸载包：　pdm remove <pkg>
- 导出包：
  - requirements.txt　pdm export -o requirements.txt
  - setup.py　pdm export -f setuppy -o setup.py
- 查看本地包：　pdm list
- 搜索包：　pdm search

5. 运行

- 方案1　pdm run python main.py
- 方案2(流程执行推荐) 修改pyproject.toml文件，设置pdm脚本

```
[tool.pdm.scripts]
# 支持shell命令
echo = {shell = "echo 'hello'"}
# 执行python文件
start = "python main.py"
# 支持参数指定的写法，调用的是shell脚本
start = {shell = [
```

```
    "python",
    "main.py",
    # Important comment here about always using port 54321
    "-xxx", "yyy"
]}
# 上面等效于
start.shell = "python main.py -xxx yyy"
# 前置命令
pre_start = "{{ Run BEFORE the `start` script }}"
# 后置命令
post_start = "{{ Run AFTER the `start` script }}"
```

使用pdm start执行对应的start脚本。pre_start、post_start会在start前后执行，可以简单的编排任
。
使用pdm run --list可以查看配置的pdm脚本

- 方案3 （命令行推荐），修改当前的 PYTHONPATH

```
$ pdm --pep582
if [ -n "$PYTHONPATH" ]; then
    export PYTHONPATH='xxxx':$PYTHONPATH
else
    export PYTHONPATH='xxx'
fi
```

PDM原理

运行机制

- 选择python解释器，pdm会先读取 .pdm.toml文件中的python解释器路径。
- 添加PYTHONPATH python -m site看到当前环境的**PYTHONPATH**。

pdm在当前项目下，创建__pypackages__目录，管理当前添加的包。pdm run 命令或tool.pdm.script
执行两个步骤：

1. 在执行命令前，插入 **pypackages** 目录到**PYTHONPATH** 中
2. 在执行命令后，删除 **PYTHONPATH** 中的**pypackages** 目录

```
# 默认PYTHONPATH
python -m site
sys.path = [
'$HOME/py-program',
'/usr/lib/python38.zip',
'/usr/lib/python3.8',
'/usr/lib/python3.8/lib-dynload',
'$HOME/.local/pipx/venvs/pdm/lib/python3.8/site-packages',

]
```

pdm run后的PYTHONPATH

pdm run python -m site
sys.path = [
'$HOME/py-program',
'/usr/lib/python38.zip',
'/usr/lib/python3.8',
'/usr/lib/python3.8/lib-dynload',
'$HOME/py-program/**pypackages**/3.8/lib',
'$HOME/.local/pipx/venvs/pdm/lib/python3.8/site-packages',
]

缓存机制

pdm利用缓存机制实现了PEP582。

1. PDM的PYTHON解释器只是软链接，并没有重新下载。

```bash
$ file $HOME/.local/pipx/venvs/pdm/bin/python
$HOME/.local/pipx/venvs/pdm/bin/python: symbolic link to python3
```

2. pypackages目录下添加的包指向缓存目录cache_dir。pdm config cache_die <__cache_path> 改缓存目录$ pdm config install.cache_method [symlink | pth] 修改连接策略

默认是symlink，以软链接的方式链接包目录pth，当 Python 在遍历已知的库文件目录过程中，如果现有.pth 文件，就会将文件中所记录的路径加入到 sys.path 设置中，于是.pth 文件说指明的库也就以被 Python 运行环境找到了。

全局和局部

pdm配置分为全局和局部。命令，使用 -g参数，读取全局的参数；没有-g参数，默认读取本地项目的pyproject.toml。

pdm list -g 查看全局已安装的包。

pdm init -g 使用全局配置初始化项目

使用 pdm config auto_global true ，当本地目录没有 pyproject.toml，pdm 使用全局的 pyproject toml。

对于pdm config，默认打开的全局配置；使用 -l/--local 时，指定本地项目。

-p <project_path>，强制使用路径项目配置。

```
# 比如修改其他项目的局部配置，不仅需要加 `-p/--project` 还要加 `-l/--local` 选项
pdm config -l -p <project_path> pypi.url http://pypi.douban.com/simple
```

vscode使用

确保vscode选择的interpret和 .pdm.toml配置的相同

根据缓存机制，只需要在setting.json中添加"python.envFile": "${workspaceFolder}/.env"。
.env文件中写入当前lib路径即可，PYTHONPATH=__pypackages__/x.x/lib。

对比conda使用体验

pdm注重的是项目，在项目管理不同的包。conda注重的虚拟环境，通过切换环境管理不同的包。

项目和环境不是一回事：项目打包应该维护一个最小的包；环境是提供一个通用的运行时。

pdm优点

- pdm config语法简洁;
- pdm.sript方便CLI批处理;
- 解释器和包都是链接形式，磁盘占用资源少;

pdm缺点

- anaconda有navigatorUI界面，方便图形操作（当然miniconda更和我口味
- 在不同项目之间复用环境得反复导出，使用-p也不太方便。

原文链接：PDM- 基于 PEP 582 的 python 包管理