# 为什么 Python 开发人员应该使用 Pipenv

2018/07/09 · 基础知识 · Pipenv

原文出处： Lacey Williams Henschel, Jeff Triplett 译文出处：linux中国–MjSeven

只用了一年， Pipenv 就变成了管理软件包依赖关系的 Python 官方推荐资源。

Pipenv 是由 Kenneth Reitz 在一年多前创建的“面向开发者而生的 Python 开发工作流”，它已经成为管理软件包依赖关系的 Python 官方推荐资源。但是对于它解决了什么问题，以及它如何比使用 `pip` 和 `requirements.txt` 文件的标准工作流更有用处，这两点仍然存在困惑。在本月的 Python 专栏中，我们将填补这些空白。

# Python 包安装简史

为了理解 Pipenv 所解决的问题，看一看 Python 包管理如何发展十分有用的。

让我们回到第一个 Python 版本，这时我们有了 Python，但是没有干净的方法来安装软件包。

然后有了 Easy Install，这是一个可以相对容易地安装其他 Python 包的软件包，但它也带来了一个问题：卸载不需要的包并不容易。

pip 登场，绝大多数 Python 用户都熟悉它。`pip` 可以让我们安装和卸载包。我们可以指定版本，运行 `pip freeze > requirements.txt` 来输出一个已安装包列表到一个文本文件，还可以用相同的文本文件配合 `pip install -r requirements.txt` 来安装一个应用程序需要的所有包。

因此我们需要一种方法来实现这一点。随之而来的是虚拟环境，它使我们能够为我们开发的每个应用程序创建一个小型的、隔离的环境。我们已经看到了许多管理虚拟环境的工具：virtualenv、 venv、 virtualenvwrapper、 pyenv、 pyenv-virtualenv、 pyenv-virtualenvwrapper 等等。它们都可以很好地使用 `pip` 和 `requirements.txt` 文件。

# 新方法：Pipenv

Pipenv 旨在解决几个问题：

首先，需要 `pip` 库来安装包，外加一个用于创建虚拟环境的库，以及用于管理虚拟环境的库，再有与这些库相关的所有命令。这些都需要管理。Pipenv 附带包管理和虚拟环境支持，因此你可以使用一个工具来安装、卸载、跟踪和记录依赖性，并创建、使用和组织你的虚拟环境。当你使用它启动一个项目时，如果你还没有使用虚拟环境的话，Pipenv 将自动为该项目创建一个虚拟环境。

Pipenv 通过放弃 `requirements.txt` 规范转而将其移动到一个名为 Pipfile 的新文档中来完成这种依赖管理。当你使用 Pipenv 安装一个库时，项目的 `Pipfile` 会自动更新安装细节，包括版本信息，还有可能的 Git 仓库位置、文件路径和其他信息。

其次，Pipenv 希望能更容易地管理复杂的相互依赖关系。你的应用程序可能依赖于某个特定版本的库，而那个库可能依赖于另一个特定版本的库，这些依赖关系如海龟般堆叠起来。当你的应用程序使用的两个库有冲突的依赖关系时，你的情况会变得很艰难。Pipenv 希望通过在一个名为 `Pipfile.lock` 的文件中跟踪应用程序相互依赖关系树来减轻这种痛苦。`Pipfile.lock` 还会验证生产中是否使用了正确版本的依赖关系。

另外，当多个开发人员在开发一个项目时，Pipenv 很方便。通过 `pip` 工作流，凯西可能会安装一个库，并花两天时间使用该库实现一个新功能。当凯西提交更改时，他可能会忘记运行 `pip freeze` 来更新 `requirements.txt` 文件。第二天，杰米拉取凯西的改变，测试就突然失败了。这样会花费好一会儿才能意识到问题是在 `requirements.txt` 文件中缺少相关库，而杰米尚未在虚拟环境中安装这些文件。

因为 Pipenv 会在安装时自动记录依赖性，如果杰米和凯西使用了 Pipenv，`Pipfile` 会自动更新并包含在凯西的提交中。这样杰米和凯西就可以节省时间并更快地运送他们的产品。

最后，将 Pipenv 推荐给在你项目上工作的其他人，因为它使用标准化的方式来安装项目依赖项和开发和测试的需求。使用 `pip` 工作流和 `requirements.txt` 文件意味着你可能只有一个 `requirements.txt` 文件，或针对不同环境的多个 `requirements.txt` 文件。例如，你的同事可能不清楚他们是否应该在他们的笔记本电脑上运行项目时是运行 `dev.txt` 还是 `local.txt`。当两个相似的 `requirements.txt` 文件彼此不同步时它也会造成混淆：`local.txt` 是否过时了，还是真的应该与 `dev.txt` 不同？多个 `requirements.txt` 文件需要更多的上下文和文档，以使其他人能够按照预期正确安装依赖关系。这个工作流程有可能会混淆同时并增加你的维护负担。

使用 Pipenv，它会生成 `Pipfile`，通过为你管理对不同环境的依赖关系，可以避免这些问题。该命令将安装主项目依赖项：

Python

```
pipenv install
```

添加 `--dev` 标志将安装开发/测试的 `requirements.txt`：

Python

使用 Pipenv 还有其他好处：它具有更好的安全特性，以易于理解的格式绘制你的依赖关系，无缝处理 `.env` 文件，并且可以在一个文件中自动处理开发与生产环境的不同依赖关系。你可以在文档中阅读更多内容。

# 使用 Pipenv

使用 Pipenv 的基础知识在官方 Python 包管理教程管理应用程序依赖关系部分中详细介绍。要安装 Pipenv，使用 `pip`：

```Python
pip install pipenv
```

要安装在项目中使用的包，请更改为项目的目录。然后安装一个包（我们将使用 Django 作为例子），运行：

```Python
pipenv install django
```

你会看到一些输出，表明 Pipenv 正在为你的项目创建一个 `Pipfile`。

如果你还没有使用虚拟环境，你还会看到 Pipenv 的一些输出，说明它正在为你创建一个虚拟环境。

然后，你将看到你在安装包时常看到的输出。

为了生成 `Pipfile.lock` 文件，运行：

```Python
pipenv lock
```

你也可以使用 Pipenv 运行 Python 脚本。运行名为 `hello.py` 的上层 Python 脚本：

```Python
pipenv run python hello.py
```

你将在控制台中看到预期结果。

启动一个 shell，运行：

```Python
pipenv shell
```

如果你想将当前使用 `requirements.txt` 文件的项目转换为使用 Pipenv，请安装 Pipenv 并运行：

```Python
pipenv install requirements.txt
```

这将创建一个 Pipfile 并安装指定的 `requirements.txt`。考虑一下升级你的项目！