PYINSTALLER的用途以及实现方式

在这个章节将会包含构建PyInstaller的基础理念（或想法）。这些想法在所有平台上都可实现。可选项以及特殊情况都写在下面，在Using PyInstaller章节中。

PyInstaller可以阅读你所编写的python代码，它会分析并找出这些代码执行所需的模组和库文件。然后便会收集一份这些文件，包括默认执行的Python解释器！然后把这些东西与代码放进一个单独的文件夹中，当然也可以生成一个单独的可执行文件。

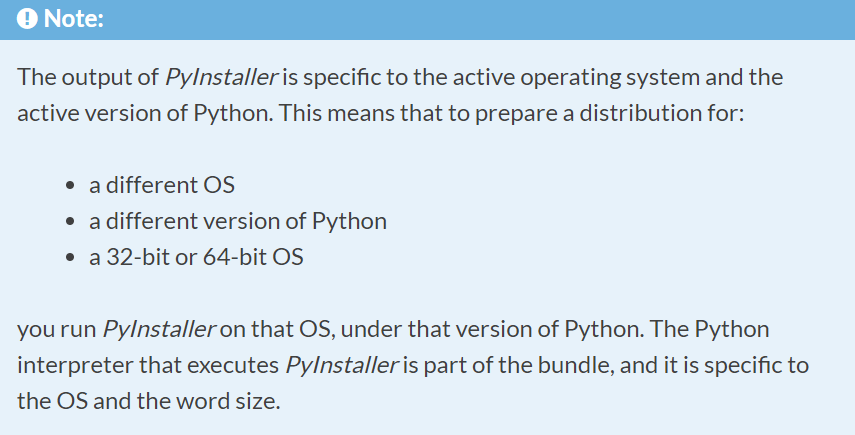
对于大多数程序而言，这些工作都可以用一句非常短的命令完成：

pyinstaller myscript.py

或添加一些选项，比如说要生成一个窗口应用（windowed application）并压缩成一个可执行文件：

pyinstaller -onfile -windowed myscript.py

然后你便可将这些可执行文件分发给其他人，而其他人在使用的过程中不需要考虑脚本依赖库等问题。（可能你所用的库文件不被允许安装在正式工作环境中）。因为对于你的使用者而言，打包的应用是完全独立的。别说代码中所涉及的依赖库，甚至连Python都不需要安装。



Note

…

分析：找到你得程序所需的文件(Analysis:Finding the Files Your Program Needs)

如果你得代码需要一些模组或库才能运行？（我们称之为依赖）。当代码量较多，或者不是一人完成，这种情况确实经常发生。为了找到这些依赖的模组与库文件，PyInstaller首先会找到你代码中所有import开头的语句，然后不过循环这种方法直到获取到你代码所用模组的列表。（译者注：虽然官方文档这么讲，但不要以为自己就可以不写Pyinstaller的配置文件，代码里写了`from elasticsearch import Elasticsearch`，pyinstaller并没有把对应的ES库添加进去）

PyInstaller可以理解（阅读）Python包经常所用的`egg`的分发格式。如果你得代码中引入模组通过`egg`的方式，PyInstaller也是可以添加这颗蛋的。

PyInstaller同样也熟悉当下一些主流的Python包，比如GUI（图形用户界面）的Qt(通过PyQt或PtSide引入)，WxPython，TkInter，Django和其他包。如果想了解详细的列表，请查看 Supported Packages章节。

当然也会有一些代码引用模组的方法不能被PyInstaller所探测出，比如说用\_\_import\_\_()函数或者在运行的时候操纵sys.path（在执行时修改python的系统路径导入一些平时不在默认路径下的模组或包）值。如果有这样的情况，下面的内容或许能帮到你：

你可以通过pyinstaller命令行添加额外的文件。

你可以通过pyinstaller命令行添加额外的引入路径。

你可以编辑由PyInstaller在第一次运行时创建的myscript.spec文件。在这个spec文件中你可以告诉PyInstaller那些你的代码中所需要的核心模组。

你可以编写`hook`文件来告知PyInstaller关于你代码中的隐式引用(hidden imports)。如果你编写的`hook`文件其他的的PyInstaller用户也能用得上，那么欢迎你能共享这份文件。

如果你的程序依赖连接一些特定的数据文件，那你也可以把这些数据文件一并与代码进行打包。同样，这的操作可以通过修改spec文件，详细的内容请查看 Using Spec Files章节。

为了在程序运行的时候能定位所需的文件，你所编写的程序得能在任何情况下获取它自己本身的路径，相关的内容可以查看 Run-Time Information章节。

PyInstaller本身并不包含在当前操作系统下应该存在的任何库文件。比如说在Linux操作系统下，PyInstaller并不捆绑任何来自/lib或/usr/lib下的文件，当然不管linux的什么分发系统，你总能找到找到这些文件。

将打包内容创建到一个文件夹中(Bundling to One Folder)

当你运行PyInstaller来打包myscript.py，默认的情况下会创建出一个名叫myscript的文件夹。这个文件夹会包含所有代码的依赖内容，并且还有一个名为myscript的可执行文件，当然在windows系统下会叫myscript.exe。

然后你就可以把这个文件夹压缩成zip或其他压缩包然后发给需求程序的用户。他们只需要解压就可以使用。（译者注：官网说得非常简单，实际上还需要考虑诸如c语言编译库的版本等问题，比如你在centos7上打包的python代码，是不能直接在centos6上使用，需要更新使用主机的glibc库）

将代码打包成单独一个文件夹还有另外一个好处，就是当你需要修改你的代码，切并不修改代码所依赖内容，那么你可以把更新的内容直接升级到myscript可执行文件中。通常来说，这个可执行文件的体积远远小于整个项目的体积。不过如果你的代码需要引入更多依赖内容，或者所依赖的内容需要进行升级，那么你需要重新进行打包工作。

不过任何事物都有利弊，只创建一个文件夹的话势必会造成这个文件夹中包含了大量各种图标的文件，而大多数程序的使用者并不是专家，有可能不小心误操作移动或者删除某些文件，而因此导致程序无法正常运行。

麻雀虽小，五脏俱全(How the One-Folder Program Works)

一个打包后的程序通常使用PyInstaller的引导程序（bootloader）来开始它的工作。这是文件夹中myscript可执行文件的核心。

PyInstaller bootloader是一个在多平台下都能干活的二进制可执行文件。当使用者启动你的（打包后的）程序，就是bootloader施展拳脚的时候。它会创建一个临时的Python环境，让Python的解释器找到对应的依赖包然后进行工作，而工作环境就是这个名叫myscript的文件夹。

当然这只是PyInstaller如何让打包程序工作的简介，如果你需要更详细的内容，请看后面的The Bootstrap Process in Detail 章节。

极简主义，你懂吗？(Bundling to One File)

什么，连一个文件夹你都觉得复杂？只想要一个可执行文件，这样不就把小白误删组件的可能都避免了。幸运地是，PyInstaller确实可以做到这件事。

把代码打包成一个孤零零的可执行文件，让程序回归了工具的本质。是蛮不错的，不过问题就是，你需要额外创建一些相关文件，比如README。并且相比打包成文件夹而言，单独一个可执行文件的执行速度会相对慢一些。

在你尝试将代码打包成一个单独的文件之前，一定要确保打包成文件夹的情况下程序也能正常运行，因为后者明显更容易在程序运行出现问题时进行诊断，然后，别发呆了，赶快去改BUG啊！

单一的可执行文件是如何工作的？(How the One-File Program Works)

即便体积缩小到一个单独的文件，bootloader也仍然是其核心。没有文件夹，就在系统指定的缓存目录下创建一个文件夹，这个文件夹一般会被命名为\_MEIXXXXX,一长串XXXX通常是一些无意义的随机数字。

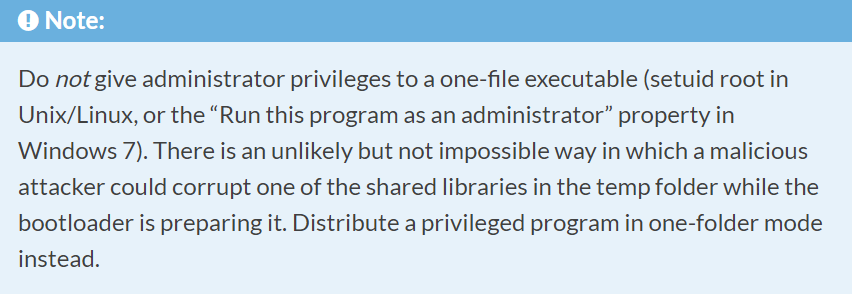
这个单一的文件，会包含代码所需要的Python模组的植入压缩包，同样任何非Python支持的文件也会被压缩。（比如.so文件）然后bootloader把这些内容解压后放入上文所讲的临时文件夹中。这也是单一可执行文件执行速度会略微的原因。

在完成解压的工作之后，bootloader处理后续任务不会与打包成文件夹的情况有什么不同。当程序完成工作，bootloader会自动清理掉缓存文件夹。

这里需要注意，在linux以及相关系统环境下，可能在挂在/tmp文件夹的时候设置`no-execution`选项，这个选项会对Pyinstaller产生冲突，所以一定要避免这个情况。

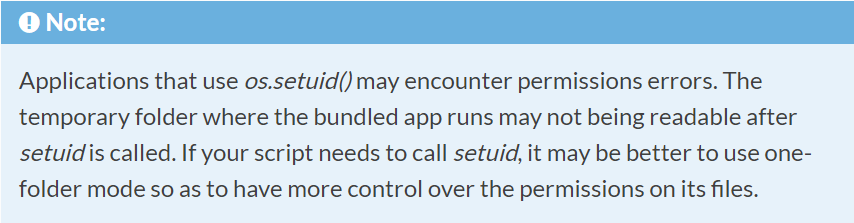
因为一个程序创建的临时目录需要一个唯一的名称，由此你才能同时运行相同程序多次，而且他们之间也是相互独立的。独立到他们不会共享缓存文件夹中的任何内容。所以你需要考虑当程序的使用环境，若是一个存储空间很珍贵的环境(expensive in disk space)，就不适合把程序打包成单一可执行文件。

还有一点，如果程序自己崩溃掉，或者被强制终结（比如unix下的kill -9，windows任务管理器杀程序以及Mac OS下的`Force Quit`），缓存目录可并不会被删除。所以如果你的程序经常崩溃（需要调试），那么不推荐你过早打包程序。



Note

不要给单独的可执行文件授予这些权限:unix/linux下的setuid root和windows下的`以管理员运行该程序`这样的系统管理员几I别的权限。因为这样有可能（原文对此可能性的描述是unlikely but not impossible）在bootloader做准备的时候，一些恶意程序借此对临时目录中的共享库文件进行攻击与破坏(corrupt)。



一些使用os.setuid()的程序可能会遇到权限问题而无法工作。造成这个问题的原因是当setuid被调用时，打包应用产生的缓存文件夹却处于不可读的状态。如果你得代码需要调用setuid，那最好不要将代码打包成单独可执行文件。

使用控制台窗口(Using a Console Window)

默认情况下，bootloader会创建一个命令行控制台（在linux和macos系统下为终端，在windows下为命令窗口）。这个窗口用于给Python解释器完成标准输入输出的工作(standard input and output)。比如你代码中的print和input()就会在这里完成它们的工作。同样的，代码运行过程中抛出的异常以及日志信息也会出现在这个控制台中。

对于Windows和MacOS系统下的PyInstaller而言，是有一个选项可以关闭这个命令窗口。这样bootloader启动的Python将处于标准输入输出未定向的状态。当你得程序是以GI，即图形界面的方式获取程序回复与诊断的情况下可以开启这个选项。

隐藏源代码(Hiding the Source Code)

打包后的程序是不包含任何源代码的，但是，PyInstaller会打包Python编译文件，也就是.pyc文件。而这些文件是可以被反编译后泄露你代码的逻辑。

所以如果你希望彻底隐藏你的代码，一个可行的方法就是使用Cython这个模块来编译。使用Cython可以把Python代码完全编译成机器语言。然后记得使用PyInstaller将Cython C模块打包即可。

额外提一句，在PyInstaller命令行中，Python字节代码可用AES256配合特定的密钥来进行加密。当然这种加密方式是可以被解密的，不过仍然不失为一种有效的手段。