PyInstaller 简述

pyinstaller 自身打包的流程：读取编写好的 python 脚本，分析其中调用的模块和库，然后收集这些文件的副本(包括 Python 的解释器)。最后把副本与脚本,可执行文件等放在一个文件夹中，或者可选的封装在一个可执行文件中，和编译成真正的机器码完全是两回事。打包后的好处是在运行者的机器上不用安装 python 和你的脚本依赖的库。在 Linux 操作系统下，它主要用的 binutil 工具包里面的 ldd 和 objdump 命令。pyinstaller 作为 python 打包使用最多的库，其兼容性无疑是极佳的。但需要注意的是，PyInstaller 打包的执行文件，只能在和打包机器系统同样的环境下。也就是说，不具备可移植性，若需要在不同系统上运行，就必须针对该平台进行打包。

PyInstaller 安装

```
pip install pyinstaller
```

或使用国内镜像

```
pip install pyinstaller -i https://pypi.douban.com/simple/
```

若安装失败，可到：https://www.lfd.uci.edu/~gohlke/pythonlibs/
下载编译好的 whl 文件：

PyJWT-1.7.1-py2.py3-none-any.whl
pyinstaller_hooks_contrib-2022.2-py2.py3-none-any.whl
pyinstaller-4.10-py3-none-win_amd64.whl
pyinstaller-4.10-py3-none-win32.whl
PyInstaller-3.6-py2.py3-none-any.whl
pyimgur-0.6.0-py2.py3-none-any.whl
pyimagej-1.1.1-py3-none-any.whl

然后执行安装 `pip install PyInstaller-3.6-py2.py3-none-any.whl`

测试是否安装成功

```
PS E:\workspace\private\Note\3_script\python> pyinstaller
usage: pyinstaller [-h] [-v] [-D] [-F] [--specpath DIR] [-n NAME]
                   [--add-data <SRC;DEST or SRC:DEST>]
                   [--add-binary <SRC;DEST or SRC:DEST>] [-p DIR]
                   [--hidden-import MODULENAME]
                   [--collect-submodules MODULENAME]
                   [--collect-data MODULENAME] [--collect-binaries MODULENAME]
                   [--collect-all MODULENAME] [--copy-metadata PACKAGENAME]
                   [--recursive-copy-metadata PACKAGENAME]
                   [--additional-hooks-dir HOOKSPATH]
                   [--runtime-hook RUNTIME_HOOKS] [--exclude-module EXCLUDES]
                   [--key KEY] [--splash IMAGE_FILE]
                   [-d {all,imports,bootloader,noarchive}]
                   [--python-option PYTHON_OPTION] [-s] [--noupx]
                   [--upx-exclude FILE] [-c] [-w]
                   [-i <FILE.ico or FILE.exe,ID or FILE.icns or "NONE">]
                   [--disable-windowed-traceback] [--version-file FILE]
                   [-m <FILE or XML>] [--no-embed-manifest] [-r RESOURCE]
                   [--uac-admin] [--uac-uiaccess] [--win-private-assemblies]
                   [--win-no-prefer-redirects]
                   [--osx-bundle-identifier BUNDLE_IDENTIFIER]
                   [--target-architecture ARCH] [--codesign-identity IDENTITY]
                   [--osx-entitlements-file FILENAME] [--runtime-tmpdir PATH]
                   [--bootloader-ignore-signals] [--distpath DIR]
                   [--workpath WORKPATH] [-y] [--upx-dir UPX_DIR] [-a]
                   [--clean] [--log-level LEVEL]
                   scriptname [scriptname ...]
```

PyInstaller 相关参数

常用参数

选项	作用
-F, –onefile	打包一个单个文件，如果你的代码都写在一个.py 文件的话，可以用这个，如果是多个.py 文件就尽量别用
-D, –onedir	（默认选项）打包多个文件，在 dist 中生成很多依赖文件，适合以框架形式编写工具代码，我个人比较推荐这样，代码易于维护
-K, –tk	在部署时包含 TCL/TK
-a, –ascii	不包含编码.在支持 Unicode 的 python 版本上默认包含

选项	作用
	所有的编码.
-d, –debug	产生 debug 版本的可执行文件
-w, –windowed, –noconsole	使用 Windows 子系统执行.当程序启动的时候不会打开命令行(只对 Windows 有效)
-c, –nowindowed, –console	（默认选项）使用控制台子系统执行(只对 Windows 有效)pyinstaller -c xxxx.pypyinstaller xxxx.py –console
-s, –strip	可执行文件和共享库将 run through strip.注意 Cygwin 的 strip 往往使普通的 win32 Dll 无法使用.
-X, –upx	如果有 UPX 安装(执行 Configure.py 时检测),会压缩执行文件(Windows 系统中的 DLL 也会)(参见 note)
-o DIR, –out=DIR	指定 spec 文件的生成目录,如果没有指定,而且当前目录是 PyInstaller 的根目录,会自动创建一个用于输出(spec 和生成的可执行文件)的目录.如果没有指定,而当前目录不是 PyInstaller 的根目录,则会输出到当前的目录下.
-p DIR, –path=DIR	设置导入路径(和使用 PYTHONPATH 效果相似).可以用路径分割符(Windows 使用分号,Linux 使用冒号)分割,指定多个目录.也可以使用多个-p 参数来设置多个导入路径，让 pyinstaller 自己去找程序需要的资源
–icon=<FILE.ICO>	将 file.ico 添加为可执行文件的资源(只对 Windows 系统有效)，改变程序的图标 pyinstaller -i ico 路径 xxxxx.py
–icon=<FILE.EXE,N>	将 file.exe 的第 n 个图标添加为可执行文件的资源(只对 Windows 系统有效)
-v FILE, –version=FILE	将 verfile 作为可执行文件的版本资源(只对 Windows 系统有效)
-n NAME, –name=NAME	可选的项目(产生的 spec 的)名字.如果省略,第一个脚本的主文件名将作为 spec 的名字

图标资源可通过阿里巴巴矢量图库获取：http://www.iconfont.cn/ ，提供一个 png,jpeg 等图片格式转 ico 各种大小的网站：https://lvwenhan.com/convertico/

详细参数

通过 `pyinstaller -h` 查看并整理说明

1. 通用参数

参数名	描述	说明
-h	显示帮助	无
-v	显示版本号	无
–distpath	生成文件放在哪里	默认：当前目录的 dist 文件夹内
–workpath	生成过程中的中间文件放在哪里	默认：当前目录的 build 文件夹内
-y	如果 dist 文件夹内已经存在生成文件，则不询问用户，直接覆盖	默认：询问是否覆盖
–upx-dir UPX_DIR	指定 upx 工具的目录	默认：execution path
-a	不包含 unicode 支持	默认：尽可能支持 unicode
–clean	在本次编译开始时，清空上一次编译生成的各种文件	默认：不清除
–log-level LEVEL	控制编译时 pyi 打印的信息	一共有 6 个等级，由低到高分别为 TRACE DEBUG INFO(默认) WARN ERROR CRITICAL。也就是默认清空下，不打印 TRACE 和 DEBUG 信息

2. 与生成结果有关的参数

参数名	描述	说明
-D	生成 one-folder 的程序（默认）	生成结果是一个目录，各种第三方依赖、资源和 exe 同时存储在该目录
-F	生成 one-file 的程序	生成结果是一个 exe 文件，所有的第三方依赖、资源和代码均被打包进该 exe 内
–specpath	指定.spec 文件的存储路径	默认：当前目录
-n	生成的.exe 文件和.spec 的文件名	默认：用户脚本的名称，即 main.py 和 main.spec

3. 指定打包资源/代码

参数名	描述	说明
–add-data	打包额外资源	用法：pyinstaller main.py –add-data=src;dest。windows 以;分割，linux 以:分割
–add-binary	打包额外的代码	用法：同–add-data。与–add-data 不同的是，用 binary 添加的文件，pyi 会分析它引用的文件并把它们一同添加进来
-p	指定额外的 import 路径，类似于使用 PYTHONPATH	参见 PYTHONPATH
–hidden-import	打包额外 py 库	pyi 在分析过程中，有些 import 没有正确分析出来，运行时会报 import error，这时可以使用该参数
–additional-hooks-dir	指定用户的 hook 目录	hook 用法参见其他，系统 hook 在 PyInstaller
–runtime-hook	指定用户 runtime-hook	如果设置了此参数，则 runtime-hook 会在运行 main.py 之前被运行
–exclude-	需要排除的 module	pyi 会分析出很多相互关联的库，但

参数名	描述	说明
module		是某些库对用户来说是没用的，可以用这个参数排除这些库，有助于减少生成文件的大小
–key	pyi 会存储字节码，指定加密字节码的 key	16 位的字符串

4. 生成参数

参数名	描述	说明
-d	执行生成的 main.exe 时，会输出 pyi 的一些 log，有助于查错	默认：不输出 pyi 的 log
-s	优化符号表	原文明确表示不建议在 windows 上使用
–noupx	强制不使用 upx	默认：尽可能使用。

5. 其他

参数名	描述	说明
–runtime-tmpdir	指定运行时的临时目录	默认：使用系统临时目录

6. Windows 和 Mac 特有的参数

参数名	描述	说明
-c	显示命令行窗口	与-w 相反，默认含有此参数
-w	不显示命令行窗口	编写 GUI 程序时使用此参数有用。
-i	为 main.exe 指定图标	pyinstaller -i beauty.ico main.py

7. Windows 特有的参数

参数名	描述	说明
–version-file	添加版本信息文件	pyinstaller –version-file ver.txt
-m, –manifest	添加 manifest 文件	pyinstaller -m main.manifest

参数名	描述	说明
-r RESOURCE	请参考原文	
–uac-admin	请参考原文	
–uac-uiaccess	请参考原文	

PyInstaller 打包流程

基本使用

若需将 xxx.py 文件打包，只需在终端执行：

```
pyinstaller xxx.py
```

需要打包一个单独的.py 文件时

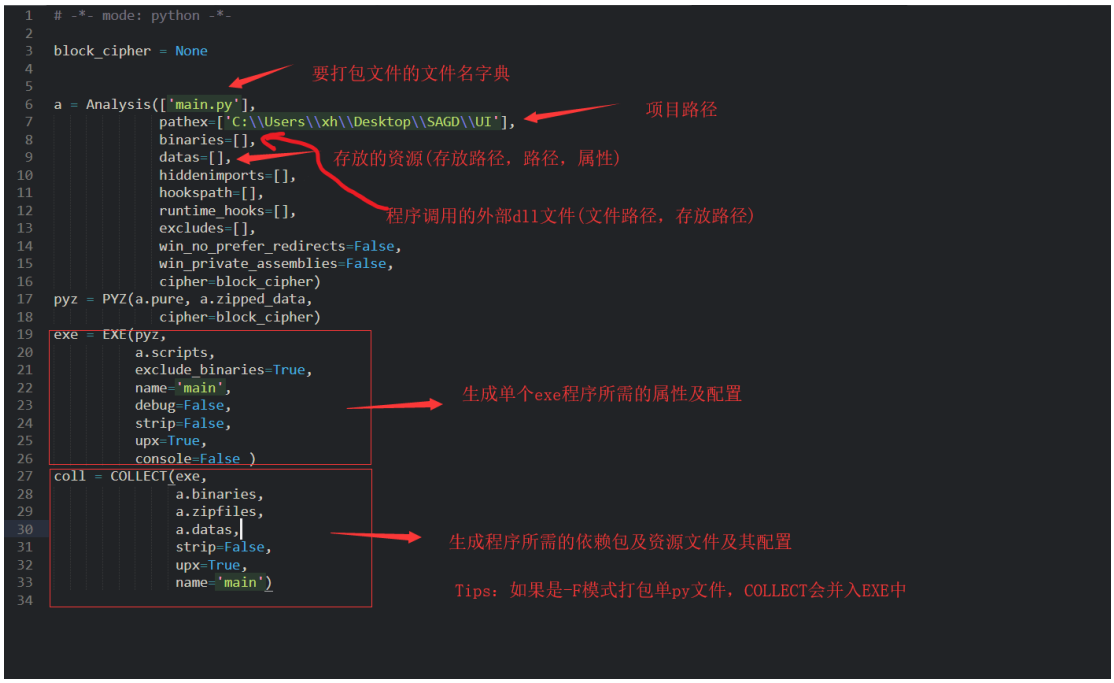
```
pyinstaller -F main.py |  | # 带控制台
pyinstaller -w -F main.py | # 不带控制台，直接以 Windows 窗体打开
pyinstaller -w -F main.py -i hello.ico --version-file version_info.txt # 添加图标和版本信息
```

需要打包多 py 文件的项目时

```
pyinstaller -w -D main.py | # 不带控制台，直接以 windows 窗体打开
```

注：终端需切换至 xxx.py 文件所在目录下。 ### 高级操作 #### 1. 生成 spec 文件
输入 `pyi-makespec -w main.py` 回车，`-w` 是为了关闭命令提示窗

打开生成在当前目录下的 main.spec 文件




```
# -*- mode: python -*-

block_cipher = None


a = Analysis(['main.py'],
             pathex=['C:\\Users\\xh\\Desktop\\SAGD\\UI'],
             binaries=[],
             datas=[],
             hiddenimports=[],
             hookspath=[],
             runtime_hooks=[],
             excludes=[],
             win_no_prefer_redirects=False,
             win_private_assemblies=False,
             cipher=block_cipher)
pyz = PYZ(a.pure, a.zipped_data,
             cipher=block_cipher)
exe = EXE(pyz,
          a.scripts,
          exclude_binaries=True,
          name='main',
          debug=False,
          strip=False,
          upx=True,
          console=False )
coll = COLLECT(exe,
               a.binaries,
               a.zipfiles,
               a.datas,
               strip=False,
               upx=True,
               name='main')
```

2. 添加项目配置信息

父字段	子字段	说明
Analysis		用于定义 python 源文件，包括搜索路径和源文件名等。
	[]	在打包多文件的项目中，需把每个 py 文件的路径添加到 Analysis 的第一个字段里，PS：和 main.py 同级目录下的 py 文件可以不用添加
	pathex	项目根路径；
	binaries	动态库；pyd,dll 文件放入到 binaries 元组中，第一个参数路径可以使用通配符，第二个参数为打包后的目录相对路径
	datas	数据文件，包括图片字体等；
	scripts	在 Analysis 中定义的源文件；
	pure	python 模块；
	zipfiles	zip 格式的依赖文件，一般是 egg 格式的库文件。
	scripts	在 Analysis 中定义的源文件；
PYZ		将 python 文件压缩打包，包含程序运行需要的所有依

父字段	子字段	说明
		赖，输入一般是 Analysis.pure。
EXE		打包生成 exe 文件，根据上面两项生成。EXE 子任务包括 Analysis 的所有 5 个输出项以及程序运行所需的一些配置文件和动态库。配置文件和动态库通过 TOC 格式来配置，格式为(name, path, typecode)，即[('解压后地址', '文件地址', '类型参数')]
	typecode	EXTENSION：python 扩展库；PYSOURCE：python 脚本；PYMODULE；PYZ；PKG；BINARY：动态库；DATA：数据文件；OPTION。
COLLECT		用来构建最终的生成目录，可以复制其他子任务生成的结果，并拷贝到指定目录，形成最终的打包结果，COLLECT 也可以没有。

图片等资源文件的添加与封装

第一种：当程序中的图片地址为相对路径时，直接把图片目录放入打包后的文件中或通过 spec 配置。但当图片被用户删除或路径改变，程序无法获取对应资源

```
datas=[("..\\lib3rd", "lib3rd"), ("..\\libutils", "libutils"), ("..\\product", "product"), ("..\\docxcompose", "docxcompose")],
```

第二种：把图片等资源封装到 exe 中，但前提要求是程序中的资源路径都得改为绝对路径，因为打包为单 exe 文件中，分析运行文件时利用 Tree 函数形成一个文件目录树，运行时将这些文件 释放至 `C:\Users\USERNAME\AppDataLocal` 下的一个临时文件夹 XXXXX 内。程序自身引用此目录的绝对路径来获得所需的文件。

第一步：先将程序中的相对路径更改为绝对路径

第二步：在 Spec 文件中添加参数，Tree 函数将所有文件均添加为'DATA'型数据，其结构为：

typecode 使用示例

```
exe = EXE(pyz,
      a.scripts,
      a.binaries,
      a.zipfiles,
      a.datas,
      [('logging.conf', 'src/logging.conf', 'DATA')],
      [('clr.pyd', 'C:\\Python27\DLLs\\clr.pyd', 'EXTENSION'),
      ('Python.Runtime.dll', 'C:\\Python27\DLLs\\Python.Runtime.dll', 'BINARY'),
      ('Python.Runtime.pdb', 'C:\\Python27\DLLs\\Python.Runtime.pdb', 'BINARY'),
      ('n3kAdrtB.dll', 'src\\n3kAdrtB.dll', 'BINARY') ],
      name=os.path.join('build\\pyi.win32\\wgClient', 'wgClient.exe'),
      debug=False,
      strip=None,
      upx=True,
      console=True )
```

3. 执行打包命令

```
pyinstaller main.spec --noconfirm
```

命令可选项包括：

–upx-dir，

–distpath，

–noconfirm，

–ascii。

4. 查看目录结构，运行应用

打包完成后目录多了 **bulid**，**dist** 目录，**dist** 目录下存放着最终可发布的打包目录，双击 `main.exe` 运行

5. 总结

1. 能 from…import…就尽量用这个
2. 所有的路径都必须使用 `\\` 作为分隔符，且路径不能带有中文
3. spec 中可进行打包参数的编辑，在之后可以不用再加参数直接 pyinstaller
4. 第一次打包时建议用-c 模式（默认），让程序以控制台子系统执行，方便调试，无错误后用-F 或-w 进行打包
5. 添加 os.system(‘pause’)，表示程序暂停，按任意键继续。。。

PyQt 打包过程中遇到的问题

1. `RecursionError: maximum recursion depth exceeded` 或 `IndexError：tuple index out of range`
   这是出现在 pyinstaller 打包时的问题，中文意思是超出最大递归深度，解决方法：退至低版本的 python 可以得到解决，原因是兼容性问题

2. 双击程序报错 `Failed to excute Script main`
   本质上还是模块缺失了，建议使用-c 模式重新打包调试，找到缺失的模块 pip install 即可解决

3. 文件打包后过大的问题
   这个问题本质上其实是没办法解决的，因为 pyinstaller 不是编译，而是将 py 程序与相应的库打包成可执行文件，大小和速度是和你调用的库的大小与速度成正比的，所以在写 python 程序的过程中尽量不要使用 import，而是用 from...import...来减少调用。如果是 import 的话，在打包的时候，会将整个包都打包到 exe 里面，没有意义的增大了工具的大小！

4. 防止反编译的问题
   按上述步骤打包的程序是容易被反编译为 pyc 然后破译出 main 入口的源码的，为了安全起见，可以通过工具把 py 转为 pyd 来调用后打包

5. 打包报错 `upx is not availible.`
   到官网 https://upx.github.io 下载 UPX，根据自己系统位数，选择相应版本就行。 然后解压缩，得到 upx.exe 这个文件，找到当时安装 Python 位置的文件夹，将 upx.exe 拷贝到 scripts 文件夹中。

6. ctypes.cdll.LoadLibrary 报错 OSError: [WinError 126] 找不到指定的模块。
   首先检查调用的 dll 库(32/64)与 python 版本(32/64)是否对应，其次检查 LoadLibrary 所加载的 dll 是否依赖其他 dll，所依赖的 dll 是否在当前路径下或 path 环境变量中，排除以上场景后将当前使用的目录添加到环境变量中，如：

```
import os

path = os.path.realpath(os.path.join(os.getcwd(), "lib3rd/vzsdk"))
os.environ['path'] += f';{path}'
```

7. 打包/运行报错 `no module named xxx`
   确保对应 module 已安装的前提下，添加打包参数 `pyinstaller --hidden-import xxx`，或在 spec 文件中添加对应的项目

8. 代码调试正常，但打包后提示 `FileNotFoundError:[Error 2] No such file or directory`
   检查报错信息，是否是第三方库使用的文件，如果是的话需要创建本地

目录并提供文件副本，然后执行对应第三方模块设置动作，如果不是检查本身文件路径是否正确（路径中尽量不要使用中文）

常见国内镜像地址

> 清华：https://pypi.tuna.tsinghua.edu.cn/simple
> 阿里云：http://mirrors.aliyun.com/pypi/simple/
> 中国科技大学 https://pypi.mirrors.ustc.edu.cn/simple/
> 华中理工大学：http://pypi.hustunique.com/
> 山东理工大学：http://pypi.sdutlinux.org/
> 豆瓣：http://pypi.douban.com/simple/

临时使用

可以在使用 pip 的时候加参数：`-i https://pypi.tuna.tsinghua.edu.cn/simple`

永久修改

Linux 下，修改 `~/.pip/pip.conf` (没有就创建一个文件夹及文件。文件夹要加“.”，表示是隐藏文件夹)
windows 下，直接在 `%userprofile%` 目录中创建一个 pip 目录，如：`C:\Users\xx\pip`，新建文件 pip.ini。

```
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
[install]
trusted-host=mirrors.aliyun.com
```

更多

github:https://github.com/pyinstaller/pyinstaller
官方 doc：https://pyinstaller.readthedocs.io/en/stable/