目录

简介 1.1

第一章 PySide6简介 1.2

第二章 窗体与控件 1.3

第三章 窗体与布局 1.4

第四章 无边框窗体 1.5

第五章 窗体间通信 1.6

第六章 国际化 1.7

第七章 打包 1.8

第八章 案例解析之PySideFrameless 1.9

PySide6 开发与实战

作者：iounce

阅读说明

本书是作者的一些经验总结，记录了PySide6开发图形界面应用程序中的常见要点，也包含一些示例讲解，希望能给大家一些启发，供参考之用。

第一章 PySide6简介

第二章 窗体与控件

第三章 窗体与布局

第四章 无边框窗体

第五章 窗体间通信

第六章 国际化

第七章 打包

第八章 案例解析之PySideFrameless

第一章 PySide6简介

*Qt*是一款C++跨平台图形用户界面应用程序开发框架，完全面向对象，易扩展，实现了组件化编程。

PySide6是*Qt*的Python封装，支持*Qt* 6.0框架，其开源协议为LGPLv3/GPLv2，具体介绍可参考官网【Qt for Python - Qt Wiki】。

在创建基于组件的图形界面程序时，通常会用到以下三个模块：

- QtCore--提供基础功能，如信号和槽，属性系统，对象模型，文件系统，序列化等；
- QtGui--拓展了QtCore的图形界面功能，包括事件，窗口和屏幕，2D绘图及图像文本等；
- QtWidgets--提供了可用的组件，包括窗体及各类控件（按钮，文本框，输入框，菜单等）。

更多组件内容可参考官网的组件说明【Qt for Python - Modules API】。

接下来，将以Windows 10下的环境为例，来介绍PySide6的具体使用。

- 环境准备：
- 操作系统： Windows 10
- Python版本： 3.10.4
- VSCode版本： 1.78.2
- 安装：
- 安装Python： 直接在【Python官网】下载指定版本3.10.4安装程序，然后双击安装即可。安装完成后，在命令行窗口输入Python，有如下结果即证明安装正确：

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

- 安装PySide6： 直接在命令行输入*pip install pyside6*安装，等待安装完成即可。可通过*pip show pyside6*查看安装后的信息：

```
Name: PySide6
Version: 6.4.0.1
Summary: Python bindings for the Qt cross-platform application and UI framework
Home-page: https://www.pyside.org
Author: Qt for Python Team
Author-email: pyside@qt-project.org
License: LGPL
Location: d:\application\python\python310\lib\site-packages
Requires: PySide6-Addons, PySide6-Essentials, shiboken6
Required-by:
```

第二章 窗体与控件

PySide6中有各类丰富的控件，与Qt中的用法基本一致，只是带有Python语言的特性。常用控件包括窗体，菜单，对话框，输入框，文本框等等，一个完整的图形界面程序依赖于各类控件来实现。

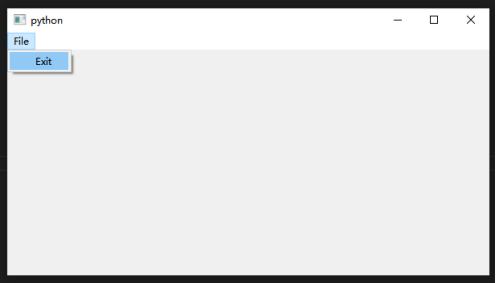
以下是一个简单的示例：

```python
import sys
from PySide6 import QtWidgets

if __name__ == "__main__":
    app = QtWidgets.QApplication([])

    win = QtWidgets.QWidget()
    win.resize(600, 300)
    win.show()

    sys.exit(app.exec())
```

首先需要引用PySide6包，然后使用其中的组件展示即可。

接下来实现一个自定义组件类，丰富界面的功能。

```python
import sys
from PySide6 import QtWidgets, QtGui

class MyWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()

        exit_action = QtGui.QAction('&Exit', self)
        exit_action.triggered.connect(self.close)

        menubar = self.menuBar()
        file_menu = menubar.addMenu('&File')
        file_menu.addAction(exit_action)

if __name__ == "__main__":
    app = QtWidgets.QApplication([])

    win = MyWindow()
    win.resize(600, 300)
    win.show()

    sys.exit(app.exec())
```

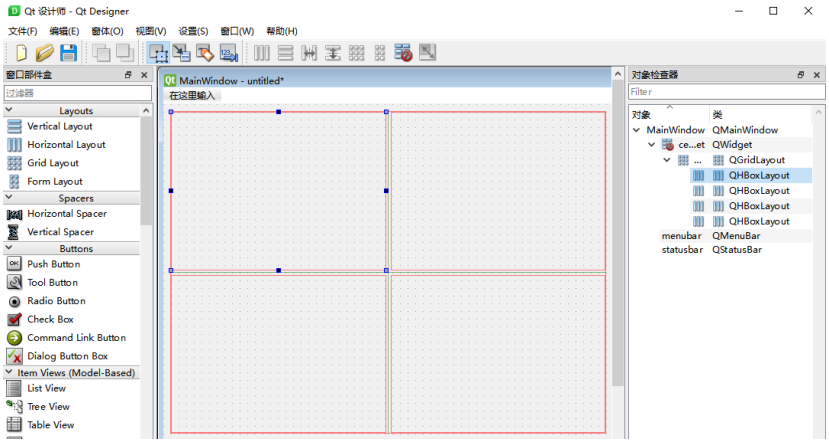


这里新建了自定义类MyWindow，继承自系统主窗体类QMainWindow，包含一些基本属性，在初始化函数**init**里面，在默认菜单栏增加了菜单项及其响应操作。

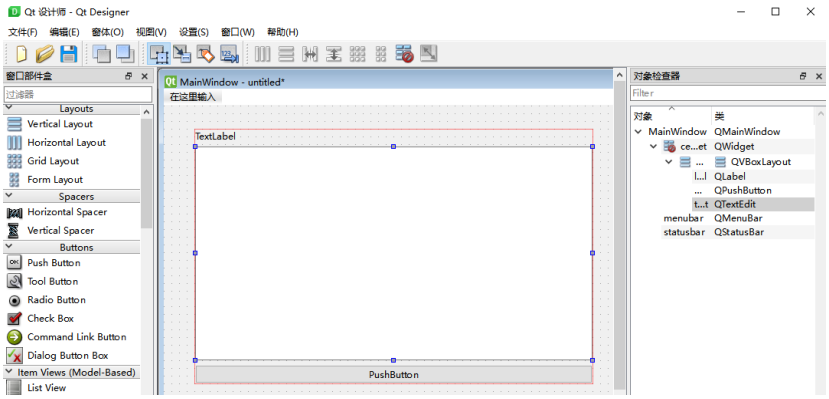
第三章 窗体与布局

PySide6中常见的布局控件有网格布局控件QGridLayout，垂直布局控件QVBoxLayout和水平布局控件QHBoxLayout。具体用法如下：

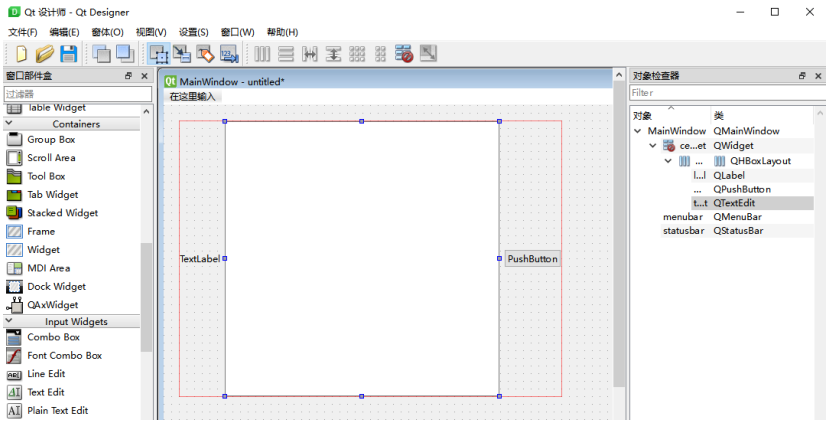
- QGridLayout：通常用于将空间分割为不同行列的单元格，可以自适应窗体大小；



- QVBoxLayout：通常用于将控件垂直对齐，即上下排列，可以自适应窗体大小；

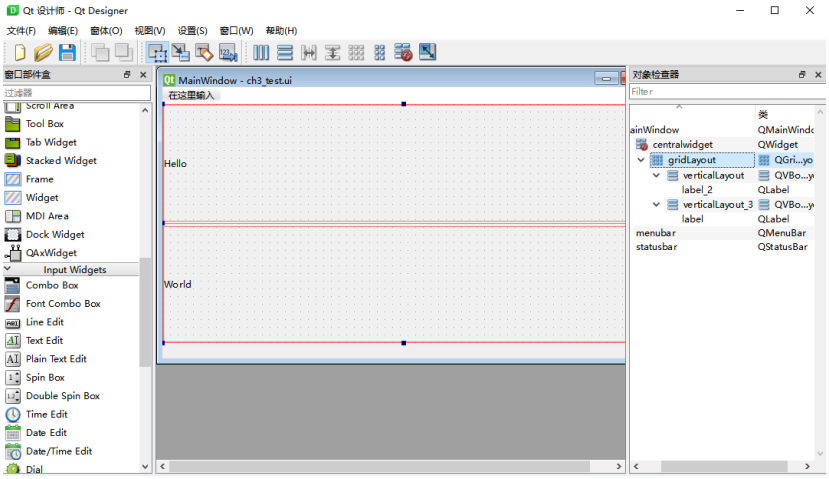


- QHBoxLayout：通常用于将控件水平对齐，即左右排列，可以自适应窗体大小。



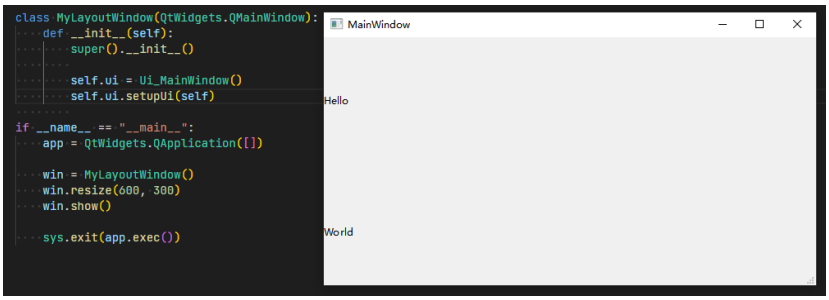
对窗体控件进行布局，可以采用安装包提供的设计工具【Qt *designer*】来编辑，或者直接使用编码的方式来实现。

- 采用设计工具实现： 打开设计工具，然后新建主窗体，增加一个网格布局控件QGridLayout，在其上面添加两个垂直布局控件QVBoxLayout，将界面分为上下两个部分。然后，分别在QVBoxLayout布局中添加QLabel控件。



  - 直接调用ui文件： 首先需要解析出ui文件中的窗体类和基类，然后实现自定义类即可：

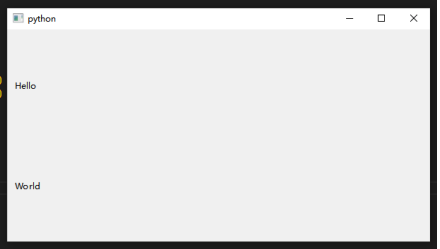
  - 调用编译后的ui文件： 首先需要使用*pyside6-uic ch3_test.ui > ch3_test.py*命令生成Python文件，然后引用界面类Ui_MainWindow即可：



- 采用直接编码实现：

  编码实现首先需要明确具体的布局，然后再编码实现，相对于设计工具而言要求更高，但是更为灵活，且不需要依赖ui文件。

```python
class MyLayoutWindow(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()

        self.grid_layout = QtWidgets.QGridLayout(self)
        hello_layout = QtWidgets.QVBoxLayout()
        world_layout = QtWidgets.QVBoxLayout()

        self.grid_layout.addLayout(hello_layout, 0, 0, 1, 2)
        self.grid_layout.addLayout(world_layout, 1, 0, 1, 2)

        hello_label = QtWidgets.QLabel(self)
        hello_label.setText("Hello")
        hello_layout.addWidget(hello_label)

        world_label = QtWidgets.QLabel(self)
        world_label.setText("World")
        world_layout.addWidget(world_label)

if __name__ == "__main__":
    app = QtWidgets.QApplication([])

    win = MyLayoutWindow()
    win.resize(600, 300)
    win.show()

    sys.exit(app.exec())
```



说明： designer.exe工具可在Python安装目录找到，参考路径：D:\Python\Python310\Lib\site-packages\PySide6\designer.exe pyside6-uic.exe工具可在Python安装目录找到，参考路径：D:\Python\Python310\Scripts\pyside6-uic.exe

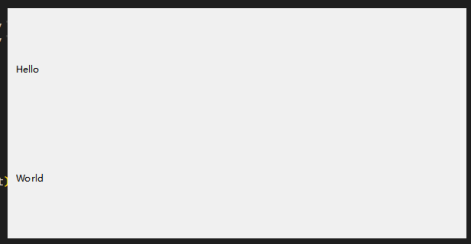
第四章 无边框窗体

PySide6中使用的是系统默认窗体以及风格，一般来说可以满足常规需要，但是很多时候，对于图形界面往往有更高的要求，譬如自定义标题栏，增加美观程度等。

对于自定义的场景，常常会使用无边框窗体。它是取消了系统窗体的默认标题栏，转而由开发者自定义。

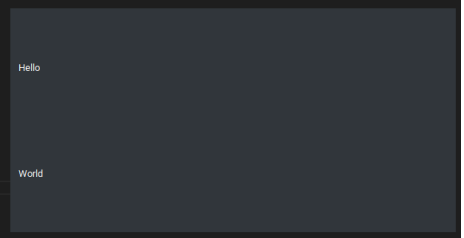
将默认窗体设置为无边框，主要通过setWindowFlags函数设置窗体标志Qt.FramelessWindowHint，示例如下：

```
54  class MyFramelessWindow(QtWidgets.QWidget):
55      def __init__(self):
56          super().__init__()
57
58          self.grid_layout = QtWidgets.QGridLayout(self)
59          hello_layout = QtWidgets.QVBoxLayout()
60          world_layout = QtWidgets.QVBoxLayout()
61
62          self.grid_layout.addLayout(hello_layout, 0, 0, 1,
63          self.grid_layout.addLayout(world_layout, 1, 0, 1,
64
65          hello_label = QtWidgets.QLabel(self)
66          hello_label.setText("Hello")
67          hello_layout.addWidget(hello_label)
68
69          world_label = QtWidgets.QLabel(self)
70          world_label.setText("World")
71          world_layout.addWidget(world_label)
72
73          self.setWindowFlags(QtCore.Qt.FramelessWindowHint)
74
75  if __name__ == "__main__":
76      app = QtWidgets.QApplication([])
77
78      win = MyFramelessWindow()
79      win.resize(600, 300)
80      win.show()
81
82      sys.exit(app.exec())
```



如果希望界面增加更多风格，或者看起来更美观，可以使用开源库【qt_material】，通过简单设置即可获取不一样的界面风格，示例如下：

```
55  class MyFramelessWindow(QtWidgets.QWidget):
56      def __init__(self):
57          super().__init__()
58
59          self.grid_layout = QtWidgets.QGridLayout(self)
60          hello_layout = QtWidgets.QVBoxLayout()
61          world_layout = QtWidgets.QVBoxLayout()
62
63          self.grid_layout.addLayout(hello_layout, 0, 0, 1, 2)
64          self.grid_layout.addLayout(world_layout, 1, 0, 1, 2)
65
66          hello_label = QtWidgets.QLabel(self)
67          hello_label.setText("Hello")
68          hello_layout.addWidget(hello_label)
69
70          world_label = QtWidgets.QLabel(self)
71          world_label.setText("World")
72          world_layout.addWidget(world_label)
73
74          self.setWindowFlags(QtCore.Qt.FramelessWindowHint)
75
76  if __name__ == "__main__":
77      app = QtWidgets.QApplication([])
78
79      qt_material.apply_stylesheet(app, 'dark_blue.xml')
80
81      win = MyFramelessWindow()
82      win.resize(600, 300)
83      win.show()
84
85      sys.exit(app.exec())
```



当然，也可以使用我们开源的【PySideFrameless】无边框窗体库，封装了更多常规操作，开箱即用，示例如下：

PySideFrameless
Theme
default theme
common_default
dark_amber
dark_blue
dark_cyan
dark_lightgreen
dark_pink
dark_purple
dark_red
dark_teal
dark_yellow
light_amber
light_blue
light_blue_500
light_cyan
light_cyan_500
light_lightgreen
light_lightgreen_500
light_orange
light_pink
light_pink_500
light_purple
light_purple_500
light_red
light_red_500
light_teal
light_teal_500
light_yellow
Theme
Language
Feedback
About
中文
English

第五章 窗体间通信

PySide6中使用多个窗体的时候，常常会涉及到窗体间的数据传递，在Qt中常用的是信号与槽，PySide6中也有类似的处理机制，通常是使用信号Signal。

在一个窗体种定义类属性的Signal信号变量，在需要的地方调用其emit函数发送信号。然后，在另外一个窗体中调用信号的connect函数，绑定此信号的处理函数，这样就可以获取此前发送的数据。

示例如下：

```
class MyMainWindow(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()

        self.grid_layout = QtWidgets.QGridLayout(self)
        hello_layout = QtWidgets.QVBoxLayout()
        world_layout = QtWidgets.QVBoxLayout()

        self.grid_layout.addLayout(hello_layout, 0, 0, 1, 2)
        self.grid_layout.addLayout(world_layout, 1, 0, 1, 2)

        hello_btn = QtWidgets.QPushButton(self)
        hello_btn.setText("Click me")
        hello_btn.clicked.connect(self.show_menu)
        hello_layout.addWidget(hello_btn)

        world_lbl = QtWidgets.QLabel(self)
        world_layout.addWidget(world_lbl)

        self.data_widget = world_lbl

    def show_menu(self):
        win = MySubWindow(self)
        win.resize(300, 200)
        win.data_signal.connect(self.on_signal)
        win.show()

    def on_signal(self, msg):
        print(msg)
        self.data_widget.setText(msg)
```

```
class MySubWindow(QtWidgets.QDialog):
    data_signal = QtCore.Signal(str)

    def __init__(self, parent):
        super().__init__(parent)

        self.signal_btn = QtWidgets.QPushButton(self)
        self.signal_btn.setText("Send message")
        self.signal_btn.clicked.connect(self.send_msg)
        self.count = 0

    def send_msg(self):
        self.count += 1
        self.data_signal.emit("Signal here: " + str(self.count))
```

```python
if __name__ == "__main__":
    app = QtWidgets.QApplication([])

    qt_material.apply_stylesheet(app, 'dark_blue.xml')

    win = MyMainWindow()
    win.resize(600, 300)
    win.show()

    sys.exit(app.exec())
```

第六章 国际化

PySide6中使用窗体的时候，会广泛使用到各种文本信息，如果需要支持多语言的话，通常会比较复杂。

PySide6中可以使用语言工具【Qt *Linguist*】来转换不同的语言。

首先获取窗体文件的中文翻译源文件即ts文件，譬如，可使用命令*pyside6-lupdate main.py -ts zh_CN.ts*获取初步的翻译源文件。

然后可以手动编辑zh_CN.ts文件，增加中文文本转换。示例如下：

zh_CN.ts - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE TS>
<TS version="2.1" language="zh_CN" sourcelanguage="en_US">
<context>
    <name>MainWindow</name>
    <message>
        <source>start</source>
        <translation>开始</translation>
    </message>
    <message>
        <source>clear</source>
        <translation>清空</translation>
    </message>

</context>
</TS>
```

接下来，拷贝一份，修改文件为en_US.ts，然后将中文手动翻译为英文。示例如下：

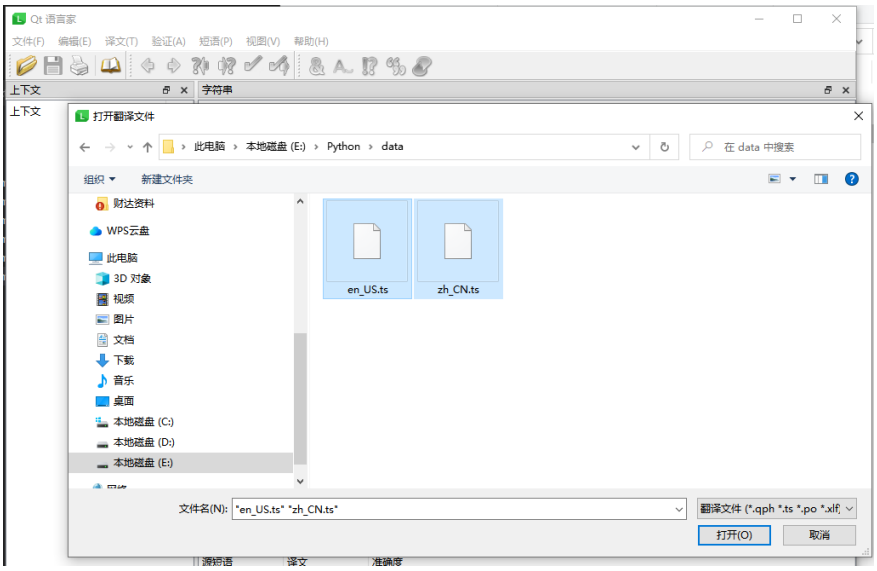
en_US.ts - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

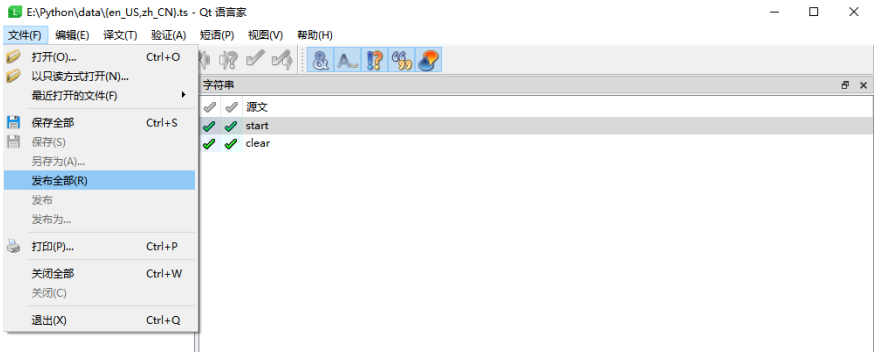
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE TS>
<TS version="2.1" language="zh_CN" sourcelanguage="en_US">
<context>
    <name>MainWindow</name>
    <message>
        <source>start</source>
        <translation>Start</translation>
    </message>
    <message>
        <source>clear</source>
        <translation>Clear</translation>
    </message>

</context>
</TS>
```

翻译源文件手动编辑完成后，就可以使用语言工具【Qt *Linguist*】将ts文件转换为目标翻译文件qm文件。

打开ts文件，可同时打开多个：



选择发布全部，就会在当前路径生成zh_CN.qm和en_US.qm文件：



最后在主程序中直接加载即可：

```python
if __name__ == "__main__":
    translator = QtCore.QTranslator()
    translator.load('zh_CN')

    app = QtWidgets.QApplication([])

    app.installTranslator(translator)

    qt_material.apply_stylesheet(app, 'dark_blue.xml')

    win = MyMainWindow()
    win.resize(600, 300)
    win.show()

    sys.exit(app.exec())

```

如果要动态切换，则需要在菜单中手动操作，切换为不同的目标语言。

说明： linguist.exe工具可在Python安装目录找到，参考路径：D:\Python\Python310\Lib\site-packages\PySide6\linguist.exe pyside6-lupdate.exe工具可在Python安装目录找到，参考路径：

D:\Python\Python310\Scripts\.pyside6-lupdate.exe

第七章 打包

PySide6窗体程序打包，一般使用Pyinstaller打包工具，它可以很方便的将Python脚本打包为可执行程序。

下面以Windows下操作为例，来作简单介绍。

首先使用命令*pip install pyinstaller*安装Pyinstaller工具，可用*pip show pyinstaller*查看安装结果：

```
Name: pyinstaller
Version: 5.1
Summary: PyInstaller bundles a Python application and all its dependencies into a single package.
Home-page: https://www.pyinstaller.org/
Author: Hartmut Goebel, Giovanni Bajo, David Vierra, David Cortesi, Martin Zibricky
Author-email:
License: GPLv2-or-later with a special exception which allows to use PyInstaller to build and distribute non-free progra
ms (including commercial ones)
Location: d:\application\python\python310\lib\site-packages
Requires: altgraph, pefile, pyinstaller-hooks-contrib, pywin32-ctypes, setuptools
Required-by: auto-py-to-exe
```

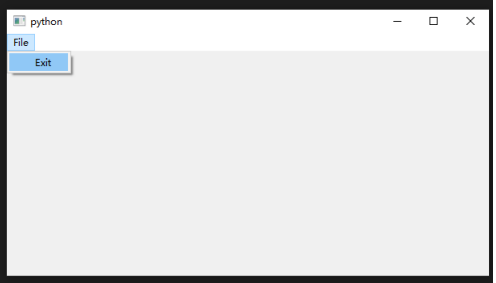
Pyinstaller工具的常用参数如下：

- -F：打包Python程序为单个可执行文件；
- -D：打包Python程序为一个文件夹；
- -i：生成图标，只适用于Windows平台；
- -n：指定打包后生成文件的名称；
- -w：禁止命令行对话框弹出。

更多其他参数，可参考Pyinstaller工具相关文档。

接下来，将如下PySide6窗体程序打包：

```python
import sys
from PySide6 import QtWidgets, QtGui, QtCore

class MyWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()

        exit_action = QtGui.QAction('&Exit', self)
        exit_action.triggered.connect(self.close)

        menubar = self.menuBar()
        file_menu = menubar.addMenu('&File')
        file_menu.addAction(exit_action)


if __name__ == "__main__":
    app = QtWidgets.QApplication([])

    win = MyWindow()
    win.resize(600, 300)
    win.show()

    sys.exit(app.exec())
```

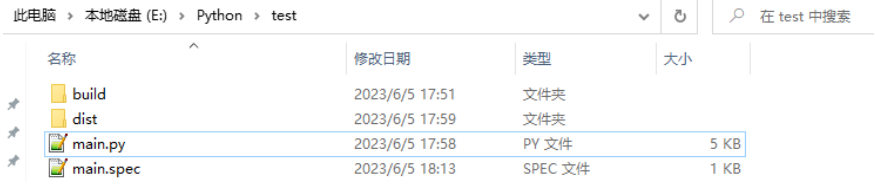


直接打包为单一可执行文件，执行如下操作： *Pyinstaller -F -w main.py -p D:\Application\Python\Python310\Lib\site-packages*

```
E:\Python\test>Pyinstaller -F -w main.py -p D:\Application\Python\Python310\Lib\site-packages
436 INFO: PyInstaller: 5.1
436 INFO: Python: 3.10.4
456 INFO: Platform: Windows-10-10.0.19042-SP0
457 INFO: wrote E:\Python\test\main.spec
490 INFO: UPX is available.
491 INFO: Extending PYTHONPATH with paths
['E:\\Python\\test', 'D:\\Application\\Python\\Python310\\Lib\\site-packages']
22 WARNING: qt_material must be imported after PySide or PyQt!
pygame 2.1.2 (SDL 2.0.18, Python 3.10.4)
Hello from the pygame community. https://www.pygame.org/contribute.html
167 WARNING: discover_hook_directories: Failed to process hook entry point 'hook-dirs = playwright._impl.__pyinstaller:g
et_hook_dirs': pkg_resources.VersionConflict: (greenlet 2.0.0 (d:\application\python\python310\lib\site-packages), Requi
rement.parse('greenlet==1.1.3'))
2051 INFO: checking Analysis
2062 INFO: checking PYZ
2067 INFO: checking PKG
2072 INFO: Bootloader D:\Application\Python\Python310\lib\site-packages\PyInstaller\bootloader\Windows-64bit\runw.exe
2072 INFO: checking EXE
```

此处加了PySide6的安装路径，便于打包程序找到相关依赖文件。

执行命令后，生成文件如下：



其中main.spec为打包配置文件，可手动修改：

```
# -*- mode: python ; coding: utf-8 -*-


block_cipher = None


a = Analysis(
    ['main.py'],
    pathex=['D:\\Application\\Python\\Python310\\Lib\\site-packages'],
    binaries=[],
    datas=[],
    hiddenimports=[],
    hookspath=[],
    hooksconfig={},
    runtime_hooks=[],
    excludes=[],
    win_no_prefer_redirects=False,
    win_private_assemblies=False,
    cipher=block_cipher,
    noarchive=False,
)
pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)

exe = EXE(
    pyz,
    a.scripts,
    a.binaries,
    a.zipfiles,
    a.datas,
    [],
    name='main',
    debug=False,
    bootloader_ignore_signals=False,
    strip=False,
    upx=True,
```

build目录为编译生成的中间文件：

此电脑 › 本地磁盘 (E:) › Python › test › build › main ›　　在 main 中搜索

名称	修改日期	类型	大小
Analysis-00.toc	2023/6/5 17:59	TOC 文件	28 KB
base_library.zip	2023/6/5 17:59	WinRAR ZIP 压缩...	813 KB
EXE-00.toc	2023/6/5 17:59	TOC 文件	20 KB
main.exe.manifest	2023/6/5 17:59	MANIFEST 文件	2 KB
main.pkg	2023/6/5 17:59	PKG 文件	30,738 KB
PKG-00.toc	2023/6/5 17:59	TOC 文件	19 KB
PYZ-00.pyz	2023/6/5 17:59	Python Zip Appli...	814 KB
PYZ-00.toc	2023/6/5 17:59	TOC 文件	10 KB
Tree-00.toc	2023/6/5 17:51	TOC 文件	102 KB
Tree-01.toc	2023/6/5 17:51	TOC 文件	10 KB
Tree-02.toc	2023/6/5 17:51	TOC 文件	1 KB
warn-main.txt	2023/6/5 17:59	TXT 文件	3 KB
xref-main.html	2023/6/5 17:59	Microsoft Edge ...	297 KB

dist目录才是最终可执行文件目录：

此电脑 › 本地磁盘 (E:) › Python › test › dist　　在 dist 中搜索

名称	修改日期	类型	大小
main.exe	2023/6/5 17:59	应用程序	31,051 KB

此时，双击dist目录下的main.exe即可打开最终打包的程序。

如果将参数-F改为-D，执行命令： *Pyinstaller -D -w main.py -p D:\Application\Python\Python310\Lib\site-packages*

```
E:\Python\test>Pyinstaller -D -w main.py -p D:\Application\Python\Python310\Lib\site-packages
421 INFO: PyInstaller: 5.1
422 INFO: Python: 3.10.4
428 INFO: Platform: Windows-10-10.0.19042-SP0
428 INFO: wrote E:\Python\test\main.spec
446 INFO: UPX is available.
447 INFO: Extending PYTHONPATH with paths
['E:\\Python\\test', 'D:\\Application\\Python\\Python310\\Lib\\site-packages']
15 WARNING: qt_material must be imported after PySide or PyQt!
pygame 2.1.2 (SDL 2.0.18, Python 3.10.4)
Hello from the pygame community. https://www.pygame.org/contribute.html
127 WARNING: discover_hook_directories: Failed to process hook entry point 'hook-dirs = playwright._impl.__pyinstaller:g
et_hook_dirs': pkg_resources.VersionConflict: (greenlet 2.0.0 (d:\application\python\python310\lib\site-packages), Requi
rement.parse('greenlet==1.1.3'))
1635 INFO: checking Analysis
1646 INFO: checking PYZ
1651 INFO: checking PKG
1652 INFO: Bootloader D:\Application\Python\Python310\lib\site-packages\PyInstaller\bootloader\Windows-64bit\runw.exe
1652 INFO: checking EXE
1653 INFO: checking COLLECT
```

这个时候，除了main.exe，还会生成其他多个依赖文件，具体如下：

此电脑 › 本地磁盘 (E:) › Python › test › dist › main ›

名称	修改日期	类型	大小
Qt6Gui.dll	2022/12/28 18:01	应用程序扩展	7,475 KB
Qt6Widgets.dll	2022/12/28 18:01	应用程序扩展	5,898 KB
Qt6Core.dll	2022/12/28 18:01	应用程序扩展	5,555 KB
opengl32sw.dll	2022/12/28 18:01	应用程序扩展	5,446 KB
Qt6Quick.dll	2022/12/28 18:01	应用程序扩展	5,019 KB
Qt6Qml.dll	2022/12/28 18:01	应用程序扩展	4,389 KB
Qt6OpenGL.dll	2022/12/28 18:01	应用程序扩展	1,880 KB
python310.dll	2023/1/4 16:03	应用程序扩展	1,466 KB
Qt6Network.dll	2022/12/28 18:01	应用程序扩展	1,375 KB
main.exe	2023/6/5 18:21	应用程序	1,145 KB
libcrypto-1_1.dll	2022/12/28 18:01	应用程序扩展	1,080 KB
ucrtbase.dll	2023/6/5 11:35	应用程序扩展	1,012 KB
base_library.zip	2023/6/5 18:21	WinRAR ZIP 压缩...	813 KB
Qt6QmlModels.dll	2022/12/28 18:01	应用程序扩展	659 KB
MSVCP140.dll	2023/1/4 16:03	应用程序扩展	551 KB
Qt6VirtualKeyboard.dll	2022/12/28 18:01	应用程序扩展	434 KB
Qt6Svg.dll	2022/12/28 18:01	应用程序扩展	351 KB
unicodedata.pyd	2023/1/4 16:03	Python Extensio...	286 KB
MSVCP140_2.dll	2023/1/4 16:03	应用程序扩展	182 KB
shiboken6.abi3.dll	2022/12/28 18:01	应用程序扩展	131 KB
_decimal.pyd	2023/1/4 16:03	Python Extensio...	102 KB
VCRUNTIME140.dll	2023/1/4 16:03	应用程序扩展	95 KB
pyside6.abi3.dll	2022/12/28 18:01	应用程序扩展	89 KB
_lzma.pyd	2023/1/4 16:03	Python Extensio...	82 KB
python3.dll	2023/1/4 16:03	应用程序扩展	61 KB
_bz2.pyd	2023/1/4 16:03	Python Extensio...	44 KB
_socket.pyd	2023/1/4 16:03	Python Extensio...	39 KB
VCRUNTIME140_1.dll	2023/1/4 16:03	应用程序扩展	37 KB
_hashlib.pyd	2023/1/4 16:03	Python Extensio...	32 KB
MSVCP140_1.dll	2023/1/4 16:03	应用程序扩展	24 KB
select.pyd	2023/1/4 16:03	Python Extensio...	22 KB
api-ms-win-core-file-l1-1-0.dll	2023/6/5 11:36	应用程序扩展	22 KB
api-ms-win-core-localization-l1-2-0.dll	2023/6/5 11:35	应用程序扩展	21 KB
api-ms-win-crt-math-l1-1-0.dll	2023/6/5 11:35	应用程序扩展	21 KB
api-ms-win-core-processthreads-l1-1...	2023/6/5 11:35	应用程序扩展	21 KB
api-ms-win-core-synch-l1-1-0.dll	2023/6/5 11:36	应用程序扩展	21 KB
api-ms-win-core-processenvironmen...	2023/6/5 11:36	应用程序扩展	20 KB

第八章 案例解析之PySideFrameless

PySideFrameless主要使用了PySide6窗体的Qt.FramelessWindowHint属性来实现无边框的效果，同时自定义了标题栏，如图标，最大化，最小化，关闭按钮，以及自定义菜单等。

除此之外，还增加了多主题，多语言的支持，只需增加业务功能即可实现相对完善的无边框应用程序。

主窗体解析

主要涉及以下文件： ui_main.ui ui_main.py main_window.py

- ui_main.ui： 主窗体的UI界面，可以使用【Qt *designer*】打开编辑，主要包括QGridLayout，QHBoxLayout，QVBoxLayout等布局的使用，合理拆分界面，保证界面伸缩而控件相对大小不变；
- ui_main.py： 使用*pyside6-uic ui_main.ui > ui_main.py*命令编译生成，将UI文件转换为对应的Python文件；
- main_window.py： 主窗体文件，使用了ui_main.py，同时添加了业务逻辑：
  - init_window()函数：通过属性Qt.FramelessWindowHint，设置窗体为无边框模式；
  - init_app_bar()函数：主要设置标题栏，自定义图标，包括最大化/最小化/还原等，还有logo，标题等；主要使用QPushButton控件，设置为扁平模式，并增加图标；图标来自于开源库qtawesome，推荐使用，非常方便；
  - init_more_menu()函数：主要用于添加下拉菜单，包括语言切换，主题，关于等；主要使用QMenu，QAction等控件。
  - init_language()函数：主要用于设置默认语言；
  - update_dynamic_widgets()函数：主要用于切换语言时手动更新控件标题；
  - proc_theme_signal()函数：主要用于获取语言窗体发送的信号，用于保存当前使用的语言；
  - get_cursor_direction()函数：主要用于获取鼠标移动的方向，便于拖动操作；
  - mousePressEvent()函数：鼠标按下的事件响应，用于获取鼠标的坐标值；
  - mouseMoveEvent()函数：鼠标移动的事件响应，用于设置窗体位置；
  - mouseReleaseEvent()函数：鼠标释放的事件响应，用于释放鼠标状态。

多主题解析

主要涉及以下文件：

- ui_theme.ui： 主窗体的UI界面，主要设计窗体布局；
- ui_theme.py： 使用*pyside6-uic ui_theme.ui > ui_theme.py*命令编译生成，将UI文件转换为对应的Python文件；
- theme_window.py： 主窗体文件，使用了ui_theme.py，同时添加了业务逻辑：

- init_window()函数：通过属性Qt.FramelessWindowHint，设置窗体为无边框模式；
- init_menu()函数：主要用于设置关闭窗体按钮及其响应；
- init_themes()函数：主要用于添加可用的主题风格，通过分组布局来实现多行多列展示。

多语言解析

主要涉及以下文件：

- zh_CN.ts： 中文翻译源文件，可以使用文本编辑器打开编辑，其中不同窗体的翻译需要在不同的 `<context></context>` 之间编辑，只能在指定窗体中使用；
- zh_CN.qm 使用语言工具【Qt *Linguist*】打开zh_CN.ts发布后生成；
- en_US.ts： 英文翻译源文件，是zh_CN.ts的英文翻译，格式相同；
- en_US.qm： 使用语言工具【Qt *Linguist*】打开en_US.ts发布后生成。

基类解析

主要涉及以下文件：

- base_window.py： 基本可以取代主窗体使用的ui_main.py文件的功能，无.ui文件的依赖；直接通过代码实现布局，所以也无法使用编辑器编辑界面：
  - init_layout()函数：通过QGridLayout，QHBoxLayout，QVBoxLayout等布局，分割界面，设计标题栏(head)，主体(body)，状态栏(tail)等功能模块；
  - init_app_bar()函数：主要用于设置标题栏，自定义图标，包括最大化/最小化/还原等，还有logo，标题等；
  - init_window()函数：通过属性Qt.FramelessWindowHint，设置窗体为无边框模式。
- base_window_demo.py： 使用base_window.py的示例程序。

说明： designer.exe工具可在Python安装目录找到，参考路径：D:\Python\Python310\Lib\site-packages\PySide6\designer.exe linguist.exe工具可在Python安装目录找到，参考路径：D:\Python\Python310\Lib\site-packages\PySide6\linguist.exe pyside6-uic.exe工具可在Python安装目录找到，参考路径：D:\Python\Python310\Scripts\pyside6-uic.exe