PythonOCC 环境配置

安装conda

推荐安装miniconda，较为轻量级，下面是下载地址，我们以windows平台为例

```
# miniconda3 下载地址
https://docs.conda.io/en/latest/miniconda.html
https://repo.anaconda.com/miniconda/Miniconda3-latest-Windows-x86_64.exe
https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

下载得到 Miniconda3-latest-Windows-x86_64.exe 安装即可；

完成后检查虚拟环境目录

```
conda config --show
#重点检查 envs_dirs 一栏 是否在用户隐藏文件中  若是则需要修改环境路径
conda config --add envs_dirs newdir # 增加环境路径
conda config --remove envs_dirs newdir  # 删除环境路径
#然后在如下位置找到文件     .condarc
C:\Users\username\.condarc
#在该文件中添加
envs_dirs:
  - E:\miniconda3\envs #新的环境保存位置
#然后查看路径
conda env list
```

创建PYOCC环境

注意版本推荐7.7.0

```
# OCC 7.5.1
conda create --name=pyoccenv python=3.7
source activate pyoccenv
conda install -c conda-forge pythonocc-core=7.5.1
# OCC 7.7.0
conda create --name=pyoccenv python=3.9
activate pyoccenv
conda install -c conda-forge pythonocc-core=7.7.0

#环境配置完成后如有需要可以克隆一个分支环境
conda create -n pyoccenv_pyside --clone pyoccenv
```

完成后 需要安装一个pyside界面或者pyqt 来使用pyocc库的样例代码，这里我们推荐安装官方的pyside

```
#pyside环境
activate pyoccenv_pyside
pip install pyside2 -i https://pypi.douban.com/simple/
#pyqt环境
activate pyoccenv_pyqt
pip install PyQt5-tools -i https://pypi.douban.com/simple
```

样例代码

首先看pyside是否安装成功，建立文件 pyside_test.py 执行

```
#pyside_test.py
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import (
    QApplication,
    QLabel
)
# Create a Qt application
app = QApplication(sys.argv)
# Create a Label and show it
label = QLabel("Hello World")
label.show()
# Enter Qt application main loop
app.exec_()
sys.exit()
```

再看pyocc是否能够运行 建立文件 pyocc_test.py

```python
#pyocc_test.py
#第一个程序，在显示框中显示两个点
from OCC.Display.SimpleGui import init_display
from OCC.Core.gp import gp_Pnt

if __name__ == '__main__':
    display, start_display, add_menu, add_function_to_menu = init_display()
    P0=gp_Pnt(0,0,1)
    P1 =gp_Pnt(0, 30, 20)
    display.DisplayShape(P0)
    display.DisplayShape(P1)
    start_display()
```

编译器

编译器推荐 vscode 或 pycharm ， vscode配置可能较为麻烦，pycharm功能强大易用，且使用社区版即可，这里使用pycharm；

```
# pycharm 下载地址
https://www.jetbrains.com/pycharm/download/
https://www.jetbrains.com/pycharm/download/download-thanks.html?
platform=windows&code=PCC
```

配合QT窗体开发

这里介绍将pyocc 结合 pyside 和 pyqt 两种代码，风格相同，细节但有区别

首先使用qt-designer建立一个window.ui文件作为界面

```
<!-- window.ui    注意这里面的两个关键字  button  hLayout  后台代码要用到  -->
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>526</width>
    <height>801</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>标题栏</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPlainTextEdit" name="textEdit">
    <property name="geometry">
     <rect>
      <x>60</x>
      <y>20</y>
      <width>401</width>
      <height>161</height>
```

```
      </rect>
     </property>
    </widget>
    <widget class="QPushButton" name="button">
     <property name="geometry">
      <rect>
       <x>60</x>
       <y>190</y>
       <width>191</width>
       <height>71</height>
      </rect>
     </property>
     <property name="text">
      <string>显示BOX</string>
     </property>
    </widget>
    <widget class="QWidget" name="horizontalLayoutWidget">
     <property name="geometry">
      <rect>
       <x>60</x>
       <y>280</y>
       <width>401</width>
       <height>201</height>
      </rect>
     </property>
     <layout class="QHBoxLayout" name="hLayout"/>
    </widget>
   <widget class="QMenuBar" name="menubar">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>0</y>
      <width>526</width>
      <height>23</height>
     </rect>
    </property>
   </widget>
   <widget class="QStatusBar" name="statusbar"/>
  </widget>
  <resources/>
  <connections/>
</ui>
```

运行部分代码采用类将ui部分逻辑包起来

首先看在pyside2环境下的情况

```
#pyocc_pyside2_ui.py
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.Aspect import  Aspect_GFM_VER

from OCC.Display.backend import load_backend
load_backend("qt-pyside2")          #这里必须 填写关键字  qt-pyside2 qt-pyqt5  wx
根据环境选一个
from OCC.Display import qtDisplay
```

```
from PySide2.QtWidgets import QApplication, QMessageBox, QDialog, QMainWindow
from PySide2.QtUiTools import QUiLoader
from PySide2.QtCore import QFile

#记得继承 QMainWindow 或者 QDialog
class Stats(QMainWindow):

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

        # 从文件中加载UI定义    这是加载的一种备用方式

        # qfile_stats = QFile("../ui/window_demo.ui")
        # qfile_stats.open(QFile.ReadOnly)
        # # qfile_stats.close()
        # self.ui = QUiLoader().load(qfile_stats)

        # 这里是直接加载的方式
        # 从 UI 定义中动态 创建一个相应的窗口对象
        # 注意：里面的控件对象也成为窗口对象的属性了
        # 比如 self.ui.button , self.ui.textEdit
        self.ui = QUiLoader().load('./window_demo.ui')

        #给按钮绑定建立BOX的动作
        self.ui.button.clicked.connect(self.displayBOX)

        self.canvas = qtDisplay.qtViewer3d(self)     #这里初始化occ针对qt的显示
        self.ui.hLayout.addWidget(self.canvas)       #关键代码 将occ针对qt的显示 加载
到控件
        self.canvas.resize(200, 200)
        self.canvas.InitDriver()         # canva的驱动,设置驱动后，才能成功display
        self.display = self.canvas._display
        self.display.set_bg_gradient_color([206, 215, 222], [128, 128, 128],
Aspect_GFM_VER)# 设置背景渐变色
        self.display.display_triedron()   # display black trihedron 轴标

    #创建box模型
    def displayBOX(self):
        a_box = BRepPrimAPI_MakeBox(10.0, 20.0, 30.0).Shape()
        self.ais_box = self.display.DisplayShape(a_box)[0]
        self.display.FitAll()


app = QApplication([])
stats = Stats()
stats.ui.show()
app.exec_()
```

再来看在pyqt5环境下的情况

```
#pyocc_pyqt5_ui.py
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.Aspect import  Aspect_GFM_VER

from PyQt5.QtWidgets import QApplication, QMessageBox, QDialog, QMainWindow
# from PyQt5.QtUiTools import QUiLoader
from PyQt5 import uic       #这里和pyside2不同  官方的pyside命名是比较正规的
```

```
from PyQt5.QtCore import QFile

from OCC.Display.backend import load_backend
load_backend("qt-pyqt5")    #这里必须 填写关键字  qt-pyside2 qt-pyqt5  wx   根据环境选一个
from OCC.Display import qtDisplay

# from pyocc_ui_demo import ShowShape

class Stats(QDialog):

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

        # 从文件中加载UI定义 这是加载的一种备用方式
        # qfile_stats = QFile("./window_demo.ui")
        # qfile_stats.open(QFile.ReadOnly)
        # qfile_stats.close()
        # self.ui = QUiLoader().load(qfile_stats)

        # 这里是直接加载的方式
        # 从 UI 定义中动态 创建一个相应的窗口对象
        # 注意：里面的控件对象也成为窗口对象的属性了
        # 比如 self.ui.button , self.ui.textEdit
        self.ui = uic.loadUi('./window_demo.ui')

        #给按钮绑定建立BOX的动作
        self.ui.button.clicked.connect(self.displayBOX)

        self.canvas = qtDisplay.qtViewer3d(self)    #这里初始化occ针对qt的显示 qtviewer3d
        self.ui.hLayout.addWidget(self.canvas)      #关键代码 将occ针对qt的显示 加载到控件
        self.canvas.resize(200, 200)
        self.canvas.InitDriver()
        self.display = self.canvas._display
        self.display.set_bg_gradient_color([206, 215, 222], [128, 128, 128], Aspect_GFM_VER)# 设置背景渐变色
        self.display.display_triedron()   # display black trihedron 轴标

    #OCC创建box模型
    def displayBOX(self):
        a_box = BRepPrimAPI_MakeBox(10.0, 20.0, 30.0).Shape()
        self.ais_box = self.display.DisplayShape(a_box)[0]
        self.display.FitAll()

app = QApplication([])
stats = Stats()
stats.ui.show()
# ShowShape()
app.exec_()
```