西安交通大學

《Python 程序设计综合训练》

实验报告

姓　　名：	
学　　号：	
班　　级：	
日　　期：	

目　录

1 实验内容

通过本次 Python 实验，我个人选择编写有关办公软件的小工具，以通过调用个人电脑中的 Microsoft Office Word 2019 的相关开发组件，实现单个 Word 文件转 PDF 文件、多个 Word 文件批量转 PDF 文件、PDF 文件合并、统计 Word 文档页码数、提取总目录并导出目录文件等功能。通过个人编写的 Word 助手，实现对于平时学习生活中 Word 相关处理需求的满足，并且通过对于 Python 实验将编程与实际生活、学习需求的结合，做出真实有效的工具，解决于 Word 有关的一系列办公问题，方便实际生活应用，做到真正的编程利于生活。

此次工具编写过程中主要实现，如下的功能：

1、单个 Word 文件转 PDF 文件

2、多个 Word 文件批量转 PDF 文件

3、PDF 文件合并

4、统计 Word 文档页码数

5、提取总目录并导出目录文件

项目最终实现一个具有前端交互界面的软件工具，命名为“Word 助手”，该工具将借助个人电脑自带的 Word 2019 进行相关功能的具体实现。整个项目使用 python 语言编写，将分为前端页面、后端功能实现两部分进行编写，最终实现整个项目。

2 实验过程及结果

2.1 前端用户交互页面

2.1.1 前端使用技术的简介

本次项目的开发过程中，前端用户交互页面主要是使用 Qt 完成相应的编写与实现。Qt 是一个跨平台的 C++ 开发库，主要用来开发图形用户界面（Graphical User Interface，GUI）程序，当然也可以开发不带界面的命令行（Command User Interface，CUI）程序。Qt 还存在 Python、Ruby、Perl 等脚本语言的绑定，也就是说可以使用脚本语言开发基于 Qt 的程序。同时，Qt 支持的操作系统有很多，例如通用操作系统 Windows、Linux、Unix 等等。在此项目开发过程中使用的是 Windows 10 作为开发平台，能够保证 Qt 的平台支持性。同时，Qt 的简单、易编写，并且具有很多开源的项目、讲解文档与教程，可以很方便地进行参考、查阅、学习，使得本次项目开发前端很容易实现完成。

2.1.2 前端的代码结构

在此次项目过程中，前端主要是分为四个部分，分别为：listWindow.ui 、 mainWindow.ui 、 pageWindow.ui 、 transformWindow.ui 这四部分分别对应于 Word 助手软件的主要四个功能界面，分别是提取目录界面、主界面、页码界面、Word 至 PDF 的转换界面。

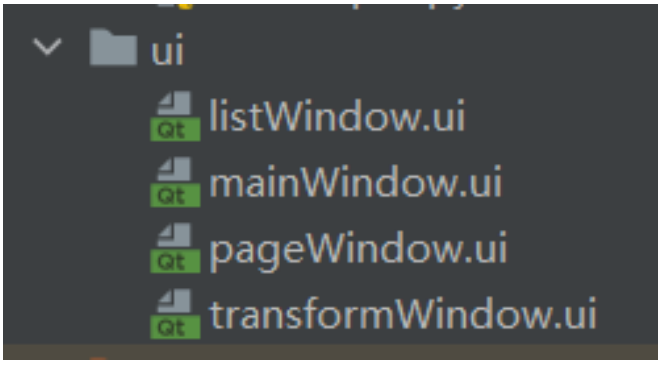


图 1 Word 助手的前端代码结构展示

2.1.3 主界面的实现与效果

图 2 主界面效果图

1、顶部功能框

```
<property name="geometry">
 <rect>
  <x>0</x>
  <y>0</y>
  <width>792</width>
  <height>18</height>
 </rect>
</property>
```

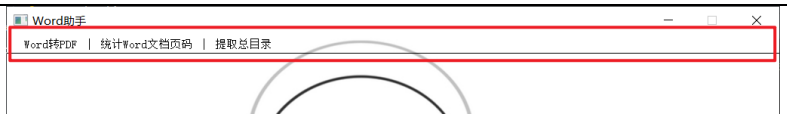


图 3 顶部功能框指示图

2、“Word 转 PDF”、“统计 Word 文档页码”、“提取总目录”三个功能按钮

```
</widget>
<action name="actionWord_PDF">
 <property name="text">
  <string>Word 转 PDF</string>
 </property>
 <property name="toolTip">
  <string><html><head/><body><p>Word 转
PDF</p></body></html></string>
 </property>
</action>
<action name="action_Word">
 <property name="text">
  <string>统计 Word 文档页码</string>
 </property>
 <property name="toolTip">
  <string><html><head/><body><p>统计 Word 文档的总页码
</p></body></html></string>
 </property>
</action>
<action name="action_list">
 <property name="text">
  <string>提取总目录</string>
 </property>
 <property name="toolTip">
  <string><html><head/><body><p>提取 Wrod 文档总目录
</p></body></html></string>
 </property>
</action>
lt;/widget>
```

图 4 “Word 转 PDF”、“统计 Word 文档页码”、“提取总目录”按钮指示图

3、界面背景

```
<widget class="QWidget" name="centralwidget"/>
<widget class="QMenuBar" name="menubar">
 <property name="geometry">
  <rect>
```

```
    <x>0</x>
    <y>0</y>
    <width>792</width>
    <height>572</height>
   </rect>
  </property>
</widget>
```

图 5 主界面背景效果图（左） 主界面原图（右）

4、功能按钮与后端动作、页面跳转的设置

```
<widget class="QStatusBar" name="statusbar"/>
<widget class="QToolBar" name="toolBar">
 <property name="windowTitle">
  <string>toolBar</string>
 </property>
 <attribute name="toolBarArea">
  <enum>TopToolBarArea</enum>
 </attribute>
 <attribute name="toolBarBreak">
  <bool>false</bool>
 </attribute>
 <addaction name="actionWord_PDF"/>
 <addaction name="separator"/>
 <addaction name="action_Word"/>
 <addaction name="separator"/>
 <addaction name="action_list"/>
</widget>
```

2.1.4 提取目录界面的实现与效果

该提取目录界面被划分成两个主要的部分，上三分之二为“源”，下三分之一为“结果”部分。

实现效果如下：

图 6 提取目录界面最终效果图

具体主要部分的编写与实现效果如下：

1、首先设置“源”界面大小

```
<property name="geometry">
 <rect>
  <x>0</x>
  <y>0</y>
  <width>800</width>
  <height>588</height>
 </rect>
</property>
```

对应的效果为下图中的红色框线部分：

图 7 源界面大小代码效果图

2、编写提取总目录按钮

```
<property name="windowTitle">
 <string>提取总目录</string>
</property>
```

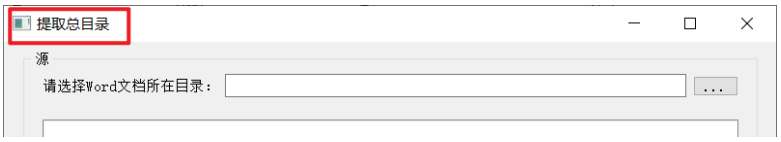


图 8 提取总目录标题效果图

3、源界面提取框

```
<property name="geometry">
 <rect>
  <x>20</x>
  <y>10</y>
  <width>761</width>
  <height>431</height>
 </rect>
</property>
```

图 9 源界面提取框效果图

4、包含页码选择框

```
<item>
 <widget class="QCheckBox" name="checkBox">
  <property name="text">
   <string>包含页码      </string>
  </property>
 </widget>
```

```
7.  </item>
```

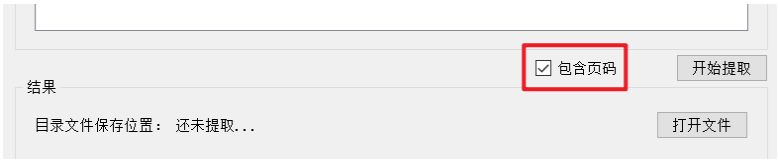


图 10 包含页码选择框效果图

5、开始提取按钮

```
1.  <item>
2.   <widget class="QPushButton" name="executeButton">
3.    <property name="text">
4.     <string>开始提取</string>
5.    </property>
6.   </widget>
7.  </item>
```

图 11 开始提取按钮效果图

6、“源”、“请选择 Word 文档所在目录：”标识

```
1.  <property name="title">
2.   <string>源</string>
3.  </property>
4.  <item>
5.   <widget class="QLabel" name="label">
6.    <property name="text">
7.     <string>请选择 Word 文档所在目录：</string>
8.    </property>
9.   </widget>
10. </item>
```

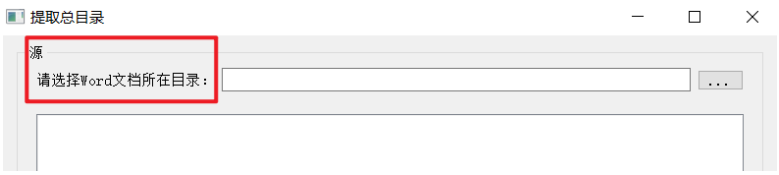


图 12 “源”、“请选择 Word 文档所在目录：”标识指示图

8、目录选择框

```
1.  <property name="geometry">
2.   <rect>
3.    <x>20</x>
```

```
    <y>443</y>
    <width>761</width>
    <height>31</height>
   </rect>
  </property>
```

图 13 目录选择框图

9、结果框及标题

```
<property name="geometry">
 <rect>
  <x>20</x>
  <y>470</y>
  <width>761</width>
  <height>81</height>
 </rect>
</property>
<property name="title">
 <string>结果</string>
</property>
```

图 14 结果框图

10、“目录文件保存位置：”框体

```
<item>
 <widget class="QLabel" name="label_2">
  <property name="text">
   <string>目录文件保存位置：</string>
  </property>
 </widget>
</item>
```

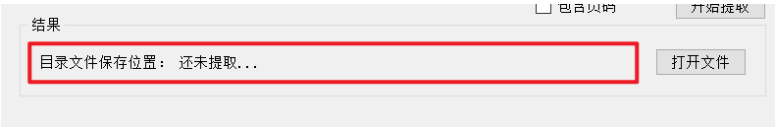


图 15 “目录文件保存位置”框体示意图

11、“打开文件”按钮设置

```
<item>
 <widget class="QPushButton" name="openButton">
  <property name="text">
   <string>打开文件</string>
  </property>
 </widget>
</item>
```

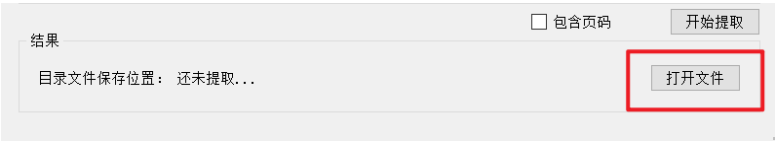


图 16 打开文件按钮指示图

12、“还未提取...”提示语句

```
<item>
 <widget class="QLabel" name="listfile">
  <property name="text">
   <string>还未提取...</string>
  </property>
 </widget>
</item>
```

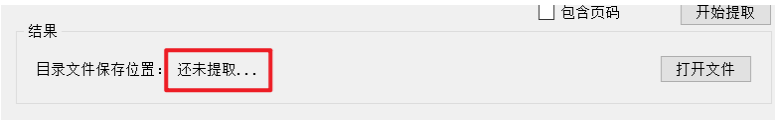


图 17 “还未提取...”提示语句

2.1.5 统计页码界面的实现与效果

1、界面及标题

```
<widget class="QMainWindow" name="PageWindow">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>792</width>
   <height>676</height>
  </rect>
 </property>
 <property name="windowTitle">
  <string>统计 Word 文档页码</string>
 </property>
```

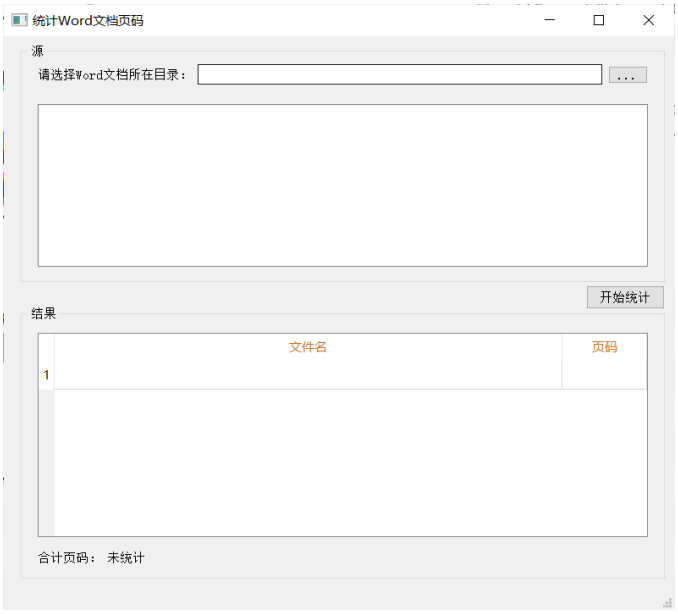


图 18 统计页码界面

2、“请选择 Word 文档所在目录：”及目录框设置

```
<widget class="QWidget" name="horizontalLayoutWidget">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>20</y>
   <width>721</width>
   <height>31</height>
  </rect>
 </property>
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
```

```
    <widget class="QLabel" name="label">
     <property name="text">
      <string>请选择 Word 文档所在目录：</string>
     </property>
    </widget>
   </item>
```

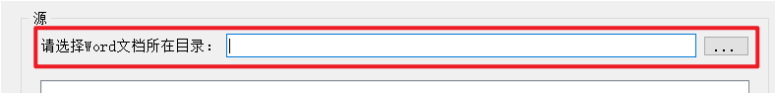


图 19 “请选择 Word 文档所在目录：”及目录框示意图

3、源界面显示框设置

```
<widget class="QListWidget" name="listword">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>70</y>
   <width>721</width>
   <height>192</height>
  </rect>
 </property>
</widget>
```

图 20 源界面显示框

4、结果框设置

```
<property name="title">
 <string>结果</string>
</property>
<widget class="QTableWidget" name="pagetable">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>30</y>
```

```
    <width>721</width>
    <height>241</height>
   </rect>
  </property>
  <property name="rowCount">
   <number>1</number>
  </property>
  <attribute name="horizontalHeaderDefaultSectionSize">
   <number>101</number>
  </attribute>
  <row/>
  <column>
   <property name="text">
    <string>文件名</string>
   </property>
  </column>
  <column>
   <property name="text">
    <string>页码</string>
   </property>
  </column>
  <item row="0" column="0">
   <property name="text">
    <string/>
   </property>
  </item>
```

图 21 结果框

5、结果框文字颜色设置

```
 <property name="background">
  <brush brushstyle="NoBrush">
   <color alpha="255">
    <red>255</red>
    <green>0</green>
    <blue>0</blue>
```

```
    </color>
   </brush>
  </property>
```

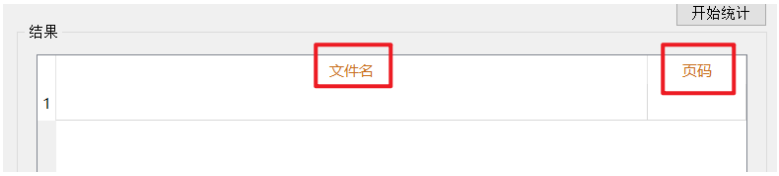


图 22 结果框红色文字效果图

6、合计页码及“未统计”提醒

```
<item>
 <widget class="QLabel" name="label_2">
  <property name="text">
   <string>合计页码：</string>
  </property>
 </widget>
</item>
<item>
 <widget class="QLabel" name="totalpage">
  <property name="text">
   <string>未统计</string>
  </property>
 </widget>
</item>
```

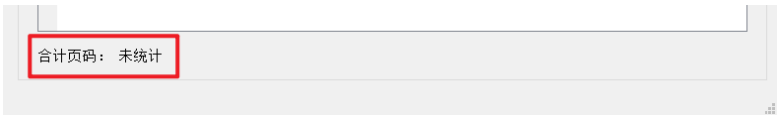


图 23 合计页码及“未统计”提醒示意图

7、开始统计按钮

```
<item>
 <widget class="QPushButton" name="executeButton">
  <property name="text">
   <string>开始统计</string>
  </property>
 </widget>
</item>
```

图 24 开始统计按钮指示图

2.1.6 转换界面的实现与效果

1、总界面大小及标题设置

```
<property name="geometry">
 <rect>
  <x>0</x>
  <y>0</y>
  <width>801</width>
  <height>648</height>
 </rect>
</property>
<property name="windowTitle">
 <string>Word 转 PDF</string>
</property>
```

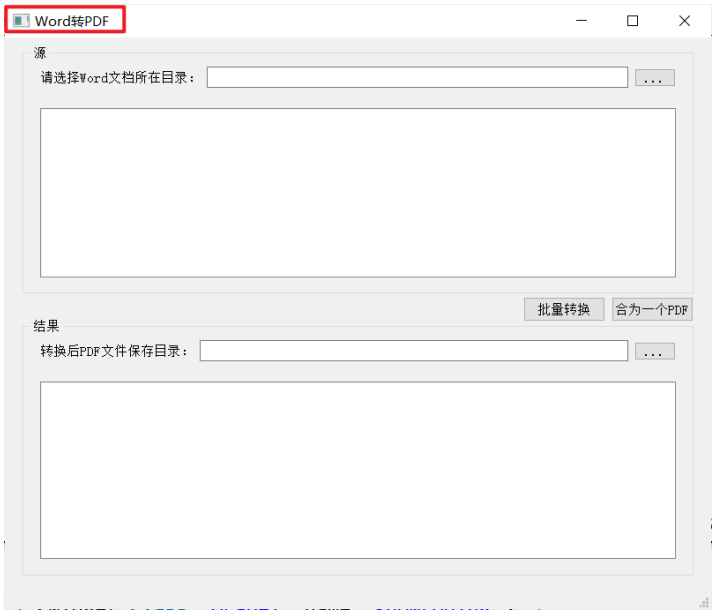


图 25 转换界面总界面及标题示意图

2、转换进度显示

```
<item>
 <widget class="QLabel" name="showLoding">
  <property name="enabled">
   <bool>true</bool>
  </property>
  <property name="text">
   <string>转换进度</string>
  </property>
 </widget>
</item>
```

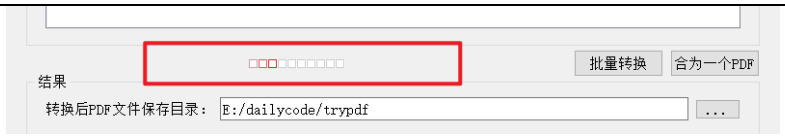


图 26 转换进度显示示意图

3、批量转换按钮

```
<item>
 <widget class="QPushButton" name="multipleExecute">
  <property name="text">
   <string>批量转换</string>
  </property>
 </widget>
</item>
```

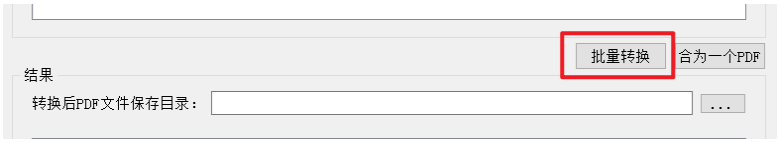


图 27 批量转换按钮示意图

4、合为一个 PDF 按钮

```
<item>
 <widget class="QPushButton" name="singleExecute">
  <property name="text">
   <string>合为一个 PDF</string>
  </property>
 </widget>
</item>
```

图 28 合为一个 PDF 按钮示意图

5、源文档目录选择框

```
<property name="geometry">
 <rect>
  <x>20</x>
  <y>20</y>
  <width>721</width>
  <height>31</height>
 </rect>
</property>
<layout class="QHBoxLayout" name="horizontalLayout">
 <item>
```

```
<widget class="QLabel" name="label">
 <property name="text">
  <string>请选择 Word 文档所在目录：</string>
 </property>
</widget>
```

图 29 源文档目录选择框示意图

6、输出文件框

```
<property name="geometry">
 <rect>
  <x>20</x>
  <y>20</y>
  <width>721</width>
  <height>31</height>
 </rect>
</property>
<layout class="QHBoxLayout" name="horizontalLayout_4">
 <item>
  <widget class="QLabel" name="label_3">
   <property name="text">
    <string>转换后 PDF 文件保存目录：</string>
   </property>
  </widget>
 </item>
```

图 30 输出文件框示意图

2.2 后端处理程序

本次项目的后端程序 Python 语言编写，根据不同页面分为了 listWindow 、mainWindow 、pageWindow 、transformWindow ，分别对应于四个不同的界面，即对应于目录界面、主界面、页码页面、转换页面，同时还编写了 wordhelper 文件作为程序的启动页面。

此外，为了方便程序的编写，还将对应的重点功能实现函数单独书写，分别编写了 common 、mergepdf 、wordtopdf 三个文件，分别实现文件的导入读取、

PDF 文件合并、word 转换为 pdf

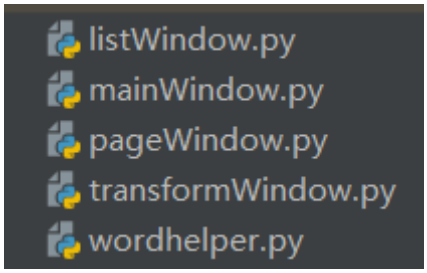


图 31 后端主要代码结构示意图

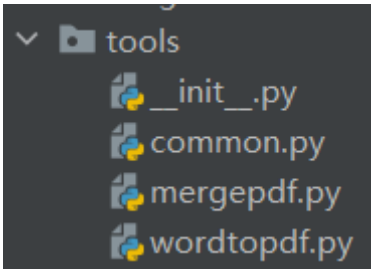


图 32 后端工具代码示意图

2.2.1 common 文件

该文件主要是实现指定目录下文件的读取已经后续输出、转换、显示规则的定义。

1、显示规则的定义

```
def indexSort(elem):
    a = re.findall(r"第\d*章", elem)
    if a == []:  # 不存在数字时
        return float("inf")  # 返回一个正无穷的数，表示最大
    else:
        return int(a[0][1:-1])
```

2、获取指定目录下的文件

该部分中，相关的主要变量为：

```
filepath：要遍历的目录
filelist_out：输出文件列表
file_ext：文件的扩展名，默认为任何类型的文件
```

实现代码：

```
def getfilenames(filepath='', filelist_out=[], file_ext='all'):
    # 遍历 filepath 下的文件
    for filename in os.listdir(filepath):
        fi_d = os.path.join(filepath, filename)
        if file_ext == '.doc':  # 遍历 Word 文档文件
            if os.path.splitext(fi_d)[1] in ['.doc', '.docx']:
                filelist_out.append(fi_d)  # 添加到路径列表中
        else:
            if file_ext == 'all':  # 遍历全部文件
```

```
                filelist_out.append(fi_d)  # 添加到路径列表中
            elif os.path.splitext(fi_d)[1] == file_ext:
                filelist_out.append(fi_d)  # 添加到路径列表中
            else:
                pass
    filelist_out.sort(key=indexSort)  # 对列表进行排序
    return filelist_out  # 返回文件完整路径列表
```

2.2.2 mergepdf 文件

该部分主要实现 PDF 文件的合并。通过遍历目录下的所有 pdf 将其合并输出到一个 pdf 文件中，返回数字（将按该数字排序）输出的 pdf 文件默认带书签，书签名为之前的文件名。默认情况下原始文件的书签也会导入到输出的 PDF 文件中。

实现函数代码：

```
def mergefiles(path, output_filename, import_bookmarks=False):
    merger = PdfMerger()
    filelist = common.getfilenames(filepath=path, filelist_out=[], file_ext='.pdf')  # 获取要合并的 PDF 文件
    if len(filelist) == 0:
        print("当前目录及子目录下不存在 pdf 文件")
        sys.exit()
    for filename in filelist:
        f = codecs.open(filename, 'rb')  # 使用 codecs 的 open()方法打开文件时，会自动转换为内部 Unicode 编码
        file_rd = PdfReader(f)
        short_filename = os.path.basename(os.path.splitext(filename)[0])
        if file_rd.is_encrypted == True:
            print('不支持的加密文件：%s' % (filename))
            continue
        merger.append(file_rd, outline_item=short_filename, import_outline=import_bookmarks)
        f.close()  # 关闭文件对象
    out_filename = os.path.join(os.path.abspath(path), output_filename)  # 将文件名和路径连接为一个完整路径
    merger.write(out_filename)
    merger.close()
```

2.2.3 wordtopdf 文件

该文件主要包含，多个文件 Word 转换为 PDF、统计页码、提取目录、Word 转换为 PDF 并提取页码、获取大纲，每一个函数为了保证能够很好地运行，都写了异常处理，

1、文件中两个全局变量

```
totalPages = 0  # 记录总页数的全局变量
returnlist = []  # 保存文件列表的全局变量
```

2、多个文件 Word 转换为 PDF 函数编写实现

```
def wordtopdf(filelist, targetpath):
    valueList = []
    try:
        pythoncom.CoInitialize()  # 调用线程初始化 COM 库，解决调用 Word 2019 时出现“尚未调用 CoInitialize”错误的问题
        gencache.EnsureModule('{00020905-0000-0000-C000-000000000046}', 0, 8, 7)
        # 开始转换
        w = Dispatch("Word.application")
        for fullfilename in filelist:
            temp = fullfilename.split('\\')
            path = temp[0]
            softfilename = os.path.splitext(temp[1])
            filename = temp[1]
            os.chdir(path)
            doc = os.path.abspath(filename)
            # filename, ext = os.path.splitext(doc)
            os.chdir(targetpath)
            pdfname = softfilename[0] + ".pdf"
            output = os.path.abspath(pdfname)
            pdf_name = output

            # 文档路径需要为绝对路径，因为 Word 启动后当前路径不是调用脚本时的当前路径。
            try:
                doc = w.Documents.Open(doc, ReadOnly=1)
                doc.ExportAsFixedFormat(output, constants.wdExportFormatPDF, \
                                        Item=constants.wdExportDocumentWithMarkup,
                                        CreateBookmarks=constants.wdExportCreateHeadingBookmarks)
            except Exception as e:
                print(e)
            if os.path.isfile(pdf_name):
                valueList.append(pdf_name)
            else:
                print('转换失败！')
                return False
        w.Quit(constants.wdDoNotSaveChanges)
```

```
        return valueList
    except TypeError as e:
        print('出错了！')
        print(e)
        return -1
```

3、Word 转换为 PDF 并提取页码

```
def wordtopdf1(filelist):
    # global totalPages  # 全局变量
    totalPages = 0
    valueList = []
    try:
        pythoncom.CoInitialize()  # 调用线程初始化 COM 库，解决调用 Word 2019 时出现“尚未调用 CoInitialize”错误的问题
        gencache.EnsureModule('{00020905-0000-0000-C000-000000000046}', 0, 8, 7)
        # 开始转换
        w = Dispatch("Word.Application")
        for fullfilename in filelist:
            temp = fullfilename.split('\\')
            path = temp[0]
            filename = temp[1]
            os.chdir(path)
            doc = os.path.abspath(filename)
            filename, ext = os.path.splitext(doc)
            output = filename + '.pdf'
            a = os.path.join(path, "pdf")
            pdf_name = output

            # 文档路径需要为绝对路径，因为 Word 启动后当前路径不是调用脚本时的当前路径。
            try:
                doc = w.Documents.Open(doc, ReadOnly=1)
                doc.ExportAsFixedFormat(output, constants.wdExportFormatPDF, \
                                        Item=constants.wdExportDocumentWithMarkup,
                                        CreateBookmarks=constants.wdExportCreateHeadingBookmarks)
            except Exception as e:
                print(e)
            if os.path.isfile(pdf_name):
                # 获取页码
                pages = getPdfPageNum(pdf_name)  # 获取页码
                valueList.append([fullfilename, str(pages)])
```

```
                totalPages += pages  # 累加页码
                os.remove(pdf_name)  # 删除生成的 PDF 文件
            else:
                print('转换失败！')
                return False
        w.Quit(constants.wdDoNotSaveChanges)
        return totalPages, valueList
    except TypeError as e:
        print('出错了！')
        print(e)
        return -1
```

4、统计页码

```
def getPdfPageNum(path):
    with open(path, "rb") as file:
        doc = PdfReader(file)
        pagecount = len(doc.pages)
    return pagecount
```

5、提取目录

```
def getPdfOutlines(pdfpath, listpath, isList):
    print("提取目录")
    with open(pdfpath, "rb") as file:
        doc = PdfReader(file)
        outlines = doc._get_outline()  # 获取大纲
        global returnlist  # 全局变量，保存大纲的列表
        returnlist = []  # 创建一个空列表
        mylist = getOutline(outlines, isList)  # 递归获取大纲
        w = DispatchEx("Word.Application")  # 创建 Word 文档应用程序对象
        w.Visible = 1
        w.DisplayAlerts = 0
        doc1 = w.Documents.Add()  # 添加一个 Word 文档对象
        range1 = doc1.Range(0, 0)
        for item in mylist:  # 通过循环将获取的目录列表插入到 Word 文档对象中
            range1.InsertAfter(item)
        outpath = os.path.join(listpath, 'list.docx')  # 连接 Word 文档路径

        doc1.SaveAs(outpath)  # 保存文件
        doc1.Close()  # 关闭 Word 文档对象
        w.Quit()  # 退出 Word 文档应用程序对象
    return outpath
```

6、获取大纲

```
def getOutline(obj, isList):
```

```
    global returnlist
    for o in obj:
        if type(o).__name__ == 'Destination':
            # mypage = getRealPage(doc, pagecount, o.get('/Page').idnum)
            if isList:  # 包括页码
                returnlist.append(o.get('/Title') + "\t\t" + str(o.get('/Page') + 1) + "\n")
            else:  # 不包括页码
                returnlist.append(o.get('/Title') + "\n")
        elif type(o).__name__ == 'list':
            getOutline(o, isList)
    return returnlist
```

2.2.4 listWindow 文件

该文件主要是对应于获取目录界面的相关功能，是一个 ui_listwindow 的类的 Python 文件，类中包含一些方法和属性。

1、创建 listWindow 类，赋值一系列属性

```
def setupUi(self, ListWindow):
    ListWindow.setObjectName("ListWindow")
    ListWindow.resize(800, 588)
    self.centralwidget = QtWidgets.QWidget(ListWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.horizontalLayoutWidget_3 = QtWidgets.QWidget(self.centralwidget)
    self.horizontalLayoutWidget_3.setGeometry(QtCore.QRect(20, 443, 761, 31))
    self.horizontalLayoutWidget_3.setObjectName("horizontalLayoutWidget_3")
    self.horizontalLayout_3 = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget_3)
    self.horizontalLayout_3.setContentsMargins(0, 0, 0, 0)
    self.horizontalLayout_3.setObjectName("horizontalLayout_3")
    spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
    self.horizontalLayout_3.addItem(spacerItem)
    self.checkBox = QtWidgets.QCheckBox(self.horizontalLayoutWidget_3)
    self.checkBox.setObjectName("checkBox")
    self.horizontalLayout_3.addWidget(self.checkBox)
    self.executeButton = QtWidgets.QPushButton(self.horizontalLayoutWidget_3)
    self.executeButton.setObjectName("executeButton")
    self.horizontalLayout_3.addWidget(self.executeButton)
```

```
20.     self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
21.     self.groupBox.setGeometry(QtCore.QRect(20, 10, 761, 431))
22.     self.groupBox.setObjectName("groupBox")
23.     self.horizontalLayoutWidget = QtWidgets.QWidget(self.groupBox)
24.     self.horizontalLayoutWidget.setGeometry(QtCore.QRect(20, 20, 721,
     31))
25.     self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget
    ")
26.     self.horizontalLayout = QtWidgets.QHBoxLayout(self.horizontalLayo
    utWidget)
27.     self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
28.     self.horizontalLayout.setObjectName("horizontalLayout")
29.     self.label = QtWidgets.QLabel(self.horizontalLayoutWidget)
30.     self.label.setObjectName("label")
31.     self.horizontalLayout.addWidget(self.label)
32.     self.sourcepath = QtWidgets.QLineEdit(self.horizontalLayoutWidget
    )
33.     self.sourcepath.setObjectName("sourcepath")
34.     self.horizontalLayout.addWidget(self.sourcepath)
35.     self.browseButton = QtWidgets.QToolButton(self.horizontalLayoutWi
    dget)
36.     self.browseButton.setObjectName("browseButton")
37.     self.horizontalLayout.addWidget(self.browseButton)
38.     self.listword = QtWidgets.QListWidget(self.groupBox)
39.     self.listword.setGeometry(QtCore.QRect(20, 70, 721, 341))
40.     self.listword.setObjectName("listword")
41.     self.groupBox_2 = QtWidgets.QGroupBox(self.centralwidget)
42.     self.groupBox_2.setGeometry(QtCore.QRect(20, 470, 761, 81))
43.     self.groupBox_2.setObjectName("groupBox_2")
44.     self.horizontalLayoutWidget_2 = QtWidgets.QWidget(self.groupBox_2
    )
45.     self.horizontalLayoutWidget_2.setGeometry(QtCore.QRect(20, 30, 72
    1, 31))
46.     self.horizontalLayoutWidget_2.setObjectName("horizontalLayoutWidg
    et_2")
47.     self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.horizontalLa
    youtWidget_2)
48.     self.horizontalLayout_2.setContentsMargins(0, 0, 0, 0)
49.     self.horizontalLayout_2.setObjectName("horizontalLayout_2")
50.     self.label_2 = QtWidgets.QLabel(self.horizontalLayoutWidget_2)
51.     self.label_2.setObjectName("label_2")
52.     self.horizontalLayout_2.addWidget(self.label_2)
53.     self.listfile = QtWidgets.QLabel(self.horizontalLayoutWidget_2)
54.     self.listfile.setObjectName("listfile")
55.     self.horizontalLayout_2.addWidget(self.listfile)
```

```
56.     spacerItem1 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy
    .Expanding, QtWidgets.QSizePolicy.Minimum)
57.     self.horizontalLayout_2.addItem(spacerItem1)
58.     self.openButton = QtWidgets.QPushButton(self.horizontalLayoutWidg
    et_2)
59.     self.openButton.setObjectName("openButton")
60.     self.horizontalLayout_2.addWidget(self.openButton)
61.     ListWindow.setCentralWidget(self.centralwidget)
62.     self.menubar = QtWidgets.QMenuBar(ListWindow)
63.     self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 18))
64.     self.menubar.setObjectName("menubar")
65.     ListWindow.setMenuBar(self.menubar)
66.     self.statusbar = QtWidgets.QStatusBar(ListWindow)
67.     self.statusbar.setObjectName("statusbar")
68.     ListWindow.setStatusBar(self.statusbar)
69.
70.     self.retranslateUi(ListWindow)
71.     QtCore.QMetaObject.connectSlotsByName(ListWindow)
```

2、将类的内容传递到前端

```
1.  def retranslateUi(self, ListWindow):
2.    _translate = QtCore.QCoreApplication.translate
3.    ListWindow.setWindowTitle(_translate("ListWindow", "提取总目录"))
4.    self.checkBox.setText(_translate("ListWindow", "包含页码       "))
5.    self.executeButton.setText(_translate("ListWindow", "开始提取"))
6.    self.groupBox.setTitle(_translate("ListWindow", "源"))
7.    self.label.setText(_translate("ListWindow", "请选择 Word 文档所在目
    录："))
8.    self.browseButton.setText(_translate("ListWindow", "..."))
9.    self.groupBox_2.setTitle(_translate("ListWindow", "结果"))
10.   self.label_2.setText(_translate("ListWindow", "目录文件保存位置：
    "))
11.   self.listfile.setText(_translate("ListWindow", "还未提取..."))
12.   self.openButton.setText(_translate("ListWindow", "打开文件"))
```

2.2.5 mainWindow 文件

该文件主要是对应于获取目录界面的相关功能，是一个 ui_mainwindow 的类的 Python 文件，类中包含一些方法和属性。

1、创建 mainWindow 类，赋值一系列属性

```
1.  def setupUi(self, MainWindow):
2.      MainWindow.setObjectName("MainWindow")
3.      MainWindow.resize(792, 572)
4.      self.centralwidget = QtWidgets.QWidget(MainWindow)
5.      self.centralwidget.setObjectName("centralwidget")
```

```
6.      MainWindow.setCentralWidget(self.centralwidget)
7.      self.menubar = QtWidgets.QMenuBar(MainWindow)
8.      self.menubar.setGeometry(QtCore.QRect(0, 0, 792, 18))
9.      self.menubar.setObjectName("menubar")
10.     MainWindow.setMenuBar(self.menubar)
11.     self.statusbar = QtWidgets.QStatusBar(MainWindow)
12.     self.statusbar.setObjectName("statusbar")
13.     MainWindow.setStatusBar(self.statusbar)
14.     self.toolBar = QtWidgets.QToolBar(MainWindow)
15.     self.toolBar.setObjectName("toolBar")
16.     MainWindow.addToolBar(QtCore.Qt.TopToolBarArea, self.toolBar)
17.     self.actionWord_PDF = QtWidgets.QAction(MainWindow)
18.     self.actionWord_PDF.setObjectName("actionWord_PDF")
19.     self.action_Word = QtWidgets.QAction(MainWindow)
20.     self.action_Word.setObjectName("action_Word")
21.     self.action_list = QtWidgets.QAction(MainWindow)
22.     self.action_list.setObjectName("action_list")
23.     self.toolBar.addAction(self.actionWord_PDF)
24.     self.toolBar.addSeparator()
25.     self.toolBar.addAction(self.action_Word)
26.     self.toolBar.addSeparator()
27.     self.toolBar.addAction(self.action_list)
28.     self.retranslateUi(MainWindow)
29.     QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

2、将类的内容传递到前端

```
1.  def retranslateUi(self, MainWindow):
2.      _translate = QtCore.QCoreApplication.translate
3.      MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
4.      self.toolBar.setWindowTitle(_translate("MainWindow", "toolBar"))
5.      self.actionWord_PDF.setText(_translate("MainWindow", "Word 转
    PDF"))
6.      self.actionWord_PDF.setToolTip(_translate("MainWindow", "<html><h
    ead/><body><p>Word 转 PDF</p></body></html>"))
7.      self.action_Word.setText(_translate("MainWindow", "统计 Word 文档页
    码"))
8.      self.action_Word.setToolTip(_translate("MainWindow", "<html><head
    /><body><p>统计 Word 文档的总页码</p></body></html>"))
9.      self.action_list.setText(_translate("MainWindow", "提取总目录"))
10.     self.action_list.setToolTip(_translate("MainWindow", "<html><head
    /><body><p>提取 Wrod 文档总目录</p></body></html>"))
```

2.2.6 wordhelper 文件

1、主窗体类初始化

```
def __init__(self):
    super(MyMainWindow, self).__init__()
    self.setupUi(self)
    self.setGeometry(100, 100, 1024, 600)
    self.setWindowTitle('Word 助手')  # 设置窗体的标题
    # 设置窗体背景
    palette = QtGui.QPalette()  # 创建调色板类的对象
    # 设置窗体背景自适应
    palette.setBrush(self.backgroundRole(), QBrush(
        QPixmap("image/bg.png").scaled(self.size(), QtCore.Qt.IgnoreAspectRatio, QtCore.Qt.SmoothTransformation)))
    self.setPalette(palette)
    self.setAutoFillBackground(True)  # 设置自动填充背景
    self.setFixedSize(1024, 600)  # 禁止显示最大化按钮及调整窗体大小
```

2、Word 转 PDF 模块

2.1、类初始化

```
def __init__(self):
    super(TransformWindow, self).__init__()
    self.setupUi(self)
    self.showLoding.setText("")  # 设置显示转换进度标签不显示内容
    self.showLoding.setMinimumWidth(100)  # 设置 Label 标签的最小宽度
    self.multipleExecute.clicked.connect(self.multipleExecuteClick)  # 批量转换按钮绑定槽函数
    self.singleExecute.clicked.connect(self.singleExecuteClick)  # 合为一个 PDF 按钮绑定槽函数
    self.sourcebrowseButton.clicked.connect(self.sourcebrowseClick)  # 选择源文件夹按钮绑定槽函数
    self.targetbrowseButton.clicked.connect(self.targetbrowseClick)  # 选择目标文件夹按钮绑定槽函数
    self.listpdf.itemDoubleClicked.connect(self.itemdoubleClick)  # 为列表项的双击事件绑定槽函数
```

2.2、自定义打开子窗体的方法

```
def open(self):
    self.__init__()
    self.show()
```

2.3、单击浏览源文件夹按钮所触发的方法

```
def sourcebrowseClick(self):  # 单击浏览源文件夹按钮所触发的方法
    # 打开选择文件夹的对话框
```

```
    dir_path = QFileDialog.getExistingDirectory(self, "请选择源文件目录", r"E:\learn\test\doc")
    if dir_path == "":  # 处理没有选择路径的情况，这里为直接返回
        return
    self.sourcepath.setText(dir_path)  # 将获取到的文件夹路径添加到文本框控件中
    self.listword.clear()  # 清空列表
    global filelist  # 定义全局变量
    filelist = common.getfilenames(dir_path, [], '.doc')  # 获取 Word 文档路径
    self.listword.addItems(filelist)  # 将获取到的 Word 文件路径添加到列表控件中
```

2.4、单击浏览目标文件夹按钮所触发的方法

```
def targetbrowseClick(self):  # 单击浏览目标文件夹按钮所触发的方法
    dir_path = QFileDialog.getExistingDirectory(self, "请选择目标文件目录", r"E:\learn\test\pdf")
    self.targetpath.setText(dir_path)
```

2.5、处理双击列表项触发的方法

```
def itemdoubleClick(self, item):  # 处理双击列表项触发的方法
    if os.path.exists(item.text()):
        os.startfile(item.text())  # 打开文件
    else:
        QMessageBox.information(self, "温馨提示：", "不是有效的文件名！", QMessageBox.Yes)
```

2.6、批量转换按钮触发的方法

```
def multipleExecuteClick(self):
    # 判断是否选择了源文件，如果没有选择则弹出提示框告知
    if self.listword.count() == 0:
        QMessageBox.information(self, "温馨提示：", "没有要转换的 Word 文档！", QMessageBox.Yes)
        return
    targetpath = self.targetpath.text()  # 获取目标文件夹
    # 判断是否选择了目标文件，如果没有选择则弹出提示框告知
    if not os.path.exists(targetpath):
        QMessageBox.information(self, "温馨提示：", "请选择正确的目标路径！", QMessageBox.Yes)
        return
    self.listpdf.clear()  # 清空结果列表
    self.showLoding.setMovie(self.gif)  # 设置 gif 图片
    self.gif.start()  # 启动图片，实现等待 gif 图片的显示
    _thread.start_new_thread(self.mExecute, ())  # 开启新线程执行批量转 PDF
```

2.7、实现批量 Word 转 PDF 操作的方法

```
def mExecute(self):
    targetpath = self.targetpath.text()  # 获取目标文件夹
    valueList = wordtopdf.wordtopdf(filelist, targetpath)  # 实现将 Word 文档批量转换为 PDF
    if (valueList != -1):
        self.showLoding.clear()  # 清除进度条
        self.listpdf.addItems(valueList)  # 将转换后的 PDF 路径显示在目标列表中
```

2.8、合为一个 PDF 按钮所触发的方法

```
def singleExecuteClick(self):
    # 判断是否选择了源文件，如果没有选择则弹出提示框告知
    if self.listword.count() == 0:
        QMessageBox.information(self, "温馨提示：", "没有要转换的 Word 文档！", QMessageBox.Yes)
        return
    # 判断是否选择了目标文件夹，如果没有选择则弹出提示框告知
    if not os.path.exists(self.targetpath.text()):
        QMessageBox.information(self, "温馨提示：", "请选择正确的目标路径！", QMessageBox.Yes)
        return
    self.listpdf.clear()  # 清空结果列表
    self.showLoding.setMovie(self.gif)  # 设置 gif 图片
    self.gif.start()  # 启动图片，实现等待 gif 图片的显示
    _thread.start_new_thread(self.sExecute, ())  # 开启新线程执行多个 Word 合为一个 PDF
```

2.9、实现合为一个 PDF 文件操作的方法

```
def sExecute(self):
    targetpath = self.targetpath.text()  # 获取目标路径
    valueList = wordtopdf.wordtopdf(filelist, targetpath)  # 将多个 Word 文档转换为 PDF 文件
    if (valueList != -1):
        mergepdf.mergefiles(targetpath, 'merged.pdf', True)  # 将多个 PDF 文件合并为一个 PDF 文件
        self.showLoding.clear()  # 清除进度条
        temp = [os.path.join(targetpath, 'merged.pdf')]  # 组合 PDF 文件路径
        self.listpdf.addItems(temp)  # 将 PDF 文件路径显示到结果列表中
        for file in valueList:  # 遍历临时生成的 PDF 文件列表
            os.remove(file)  # 删除 PDF 文件
```

3、统计 Word 文档页码模块

3.1 类初始化

```
def __init__(self):
```

```
        super(PageWindow, self).__init__()
        self.setupUi(self)
        self.pagetable.setColumnWidth(0, 600)  # 设置第一列的宽度
        self.pagetable.setColumnWidth(1, 100)  # 设置第二列的宽度
        self.pagetable.setStyleSheet("background-color: lightblue"
                                     "(spread:pad,stop:0.823 rgba(255, 25
5, 255, 204), stop:1 rgba(255, 255, 255, 204));"
                                     "selection-background-
color:lightblue;")
        headItem = self.pagetable.horizontalHeaderItem(0)  # 获得水平方向表
头的 Item 对象
        headItem.setBackground(QColor(0, 60, 10))  # 设置单元格背景颜色
        headItem.setForeground(QColor(200, 111, 30))  # 设置文字颜色
        headItem = self.pagetable.horizontalHeaderItem(1)  # 获得水平方向表
头的 Item 对象
        headItem.setBackground(QColor(0, 60, 10))  # 设置单元格背景颜色
        headItem.setForeground(QColor(200, 111, 30))  # 设置文字颜色
        self.pagetable.setEditTriggers(QTableWidget.NoEditTriggers)
        self.pagetable.setSelectionBehavior(QTableWidget.SelectRows)
        self.pagetable.setSelectionMode(QTableWidget.SingleSelection)
        self.pagetable.setAlternatingRowColors(True)
        self.totalpage.setMinimumWidth(100)  # 设置 Label 标签的最小宽度

        self.browseButton.clicked.connect(self.sourcebrowseClick)  # 选择
源路径
        self.executeButton.clicked.connect(self.executeClick)  # 开始统计
按钮的事件绑定
```

3.2 自定义打开子窗体的方法

```
def open(self):
    self.__init__()
    self.show()
```

3.3 单击浏览源文件夹按钮所触发的方法

```
def sourcebrowseClick(self):
    dir_path = QFileDialog.getExistingDirectory(self, "请选择源文件目录
", r"E:\learn\test\doc")
    if dir_path != "":  # 判断已经选择了源文件目录
        self.sourcepath.setText(dir_path)
        self.listword.clear()  # 清空列表
        global filelist
        filelist = common.getfilenames(dir_path, [], '.doc')  # 获取
Word 文档
        self.listword.addItems(filelist)
```

3.4 开始统计按钮的自定义事件

```
def executeClick(self):  # 开始统计按钮的自定义事件
```

```
    if self.listword.count() == 0:
        QMessageBox.information(self, "温馨提示：", "没有要统计页码的Word 文档！", QMessageBox.Yes)
        return
    self.totalpage.setText("")
    self.totalpage.setMovie(self.gif)  # 设置 gif 图片
    self.label_2.setText("正在统计：")
    self.gif.start()  # 启动图片，实现等待 gif 图片的显示
    _thread.start_new_thread(self.execute, ())  # 开启新线程执行统计页码
```

3.5 统计页码

```
def execute(self):
    valueList = []
    valueList = wordtopdf.wordtopdf1(filelist)
    # if valueList != []:
    #     self.totalpage.clear()  # 转换完毕就将等待 gif 图片清理掉
    totalPages = str(valueList[0])  # 总页数
    self.label_2.setText("合计页码：")
    self.totalpage.setText(totalPages)  # 显示统计出来的页码
    print("行数：", len(valueList[1]))
    self.pagetable.setRowCount(len(valueList[1]))  # 指定行数
    resultList = valueList[1]  # 获取统计结果
    for i in range(self.pagetable.rowCount()):
        for j in range(self.pagetable.columnCount()):
            content = resultList[i][j]  # 获取一个单元格的内容
            newItem = QTableWidgetItem(content)  # 转换为一个单元格对象
            self.pagetable.setItem(i, j, newItem)  # 显示在单元格中
```

4、提取总目录模块

4.1、类初始化

```
def __init__(self):
    super(ListWindow, self).__init__()
    self.setupUi(self)
    self.browseButton.clicked.connect(self.sourcebrowseClick)  # 选择源路径
    self.executeButton.clicked.connect(self.getListClick)  # 按钮事件绑定
    self.openButton.clicked.connect(self.openButtonClick)  # 为打开文件按钮绑定事件
```

4.2、自定义打开子窗体的方法

```
def open(self):
    self.__init__()
    self.show()
```

4.3、单击浏览源文件夹按钮所触发的方法

```
def sourcebrowseClick(self):
    dir_path = QFileDialog.getExistingDirectory(self, "请选择源文件目录", r"E:\learn\test\doc")
    if dir_path != "":  # 判断已经选择了源文件目录
        self.sourcepath.setText(dir_path)
        self.listword.clear()  # 清空列表
        global filelist
        filelist = common.getfilenames(dir_path, [], '.doc')  # 获取 Word 文档
        self.listword.addItems(filelist)
```

4.4、子窗体自定义事件

```
def getListClick(self):
    if self.listword.count() == 0:
        QMessageBox.information(self, "温馨提示：", "没有要提取目录的 Word 文档！", QMessageBox.Yes)
        return
    self.listfile.setText("")
    self.listfile.setMovie(self.gif)  # 设置 gif 图片
    self.gif.start()  # 启动图片，实现等待 gif 图片的显示
    _thread.start_new_thread(self.getList, ())  # 开启新线程执行统计页码
```

4.5、提取目录

```
def getList(self):
    sourcepath = self.sourcepath.text()  # 获取源路径
    if not os.path.exists(sourcepath):  # 判断是否选择了源目录
        QMessageBox.information(self, "温馨提示：", "请先选择 Word 文档所在的文件夹！", QMessageBox.Yes)
        return
    targetpath = os.path.join(sourcepath, "pdf")  # 根据源路径生成目标目录
    if not os.path.exists(targetpath):  # 判断目录是否存在，不存在则创建
        os.makedirs(targetpath)  # 创建目录
    valueList = wordtopdf.wordtopdf(filelist, targetpath)
    if (valueList != -1):
        mergepdf.mergefiles(targetpath, 'merged.pdf', True)  # 合并 PDF
        temp = [os.path.join(targetpath, 'merged.pdf')]  # 生成合并后的 PDF 文件的路径
        for file in valueList:  # 遍历临时生成的 PDF 文件列表
            os.remove(file)  # 删除 PDF 文件
        isList = self.checkBox.isChecked()  # 指定是否带目录
```

```
            resultvalue = wordtopdf.getPdfOutlines(temp[0], targetpath, i
    sList)  # 提取目录
            os.remove(temp[0])  # 删除合并后的 PDF 文件
            if valueList != []:
                self.listfile.clear()  # 转换完毕就将等待 gif 图片清理掉
            self.listfile.setText(resultvalue)  # 将生成的目录文件路径显示到
    页面中
```

4.6、自定义打开子窗体的方法

```
def open(self):
    self.__init__()
    self.show()
```

4.7、打开文件按钮触发的事件函数

```
def openButtonClick(self):
    if self.listfile.text() == "还未提取...":
        QMessageBox.information(self, "温馨提示：", "还没有提取目录，请
    先单击【开始提取】按钮！", QMessageBox.Yes)
    else:
        os.startfile(self.listfile.text())  # 打开文件
```

5、主函数

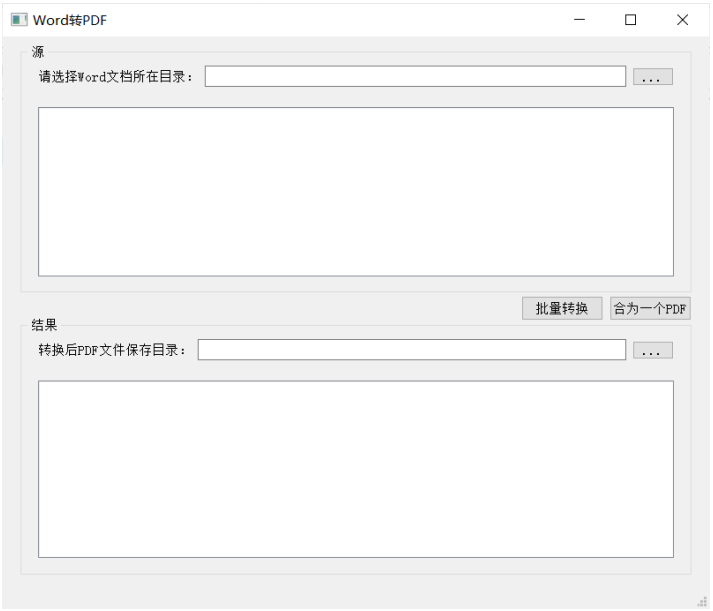
```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    main = MyMainWindow()
    qmovie = QtGui.QMovie('image/loding.gif')

    transformWindow = TransformWindow()
    transformWindow.gif = qmovie
    main.actionWord_PDF.triggered.connect(transformWindow.open)

    pagewindow = PageWindow()
    pagewindow.gif = qmovie
    main.action_Word.triggered.connect(pagewindow.open)

    listwindow = ListWindow()
    listwindow.gif = qmovie
    main.action_list.triggered.connect(listwindow.open)

    main.show()
    sys.exit(app.exec_())
```

2.3 程序使用展示

2.3.1 主界面

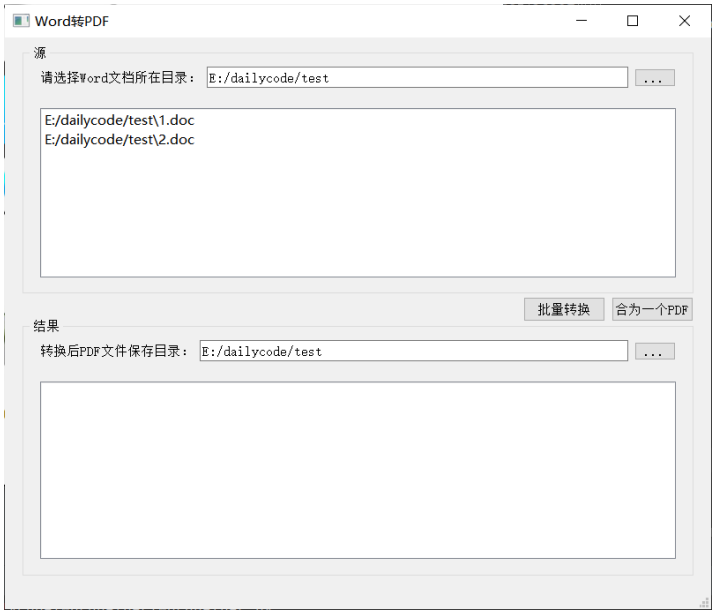
2.3.2 Word 转 PDF

1、Word 转 PDF 界面

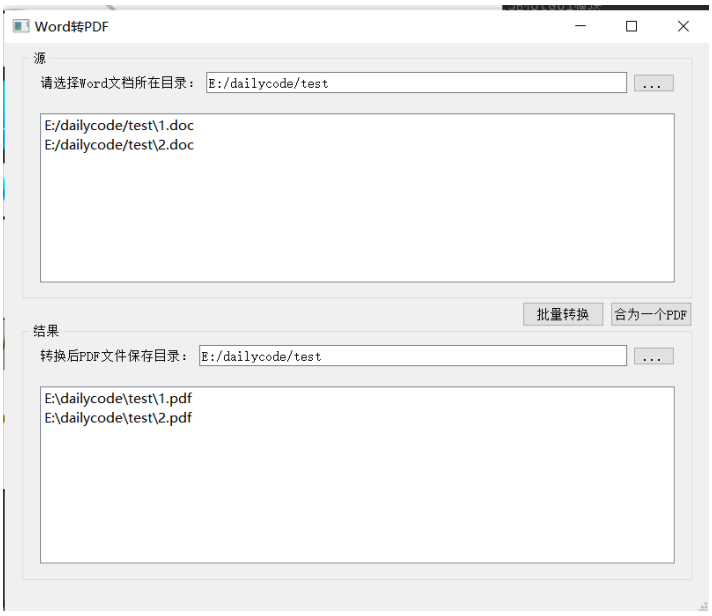


2、批量转换

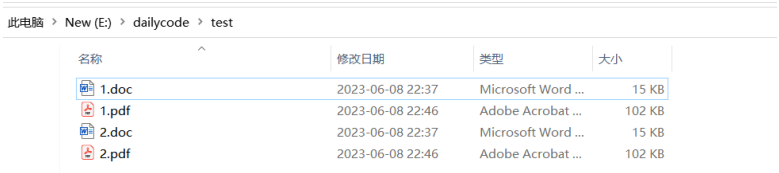
2.1 选择文件所在的文件夹以及 PDF 文件输出文件夹



2.2、点击批量转换，等待转换完成

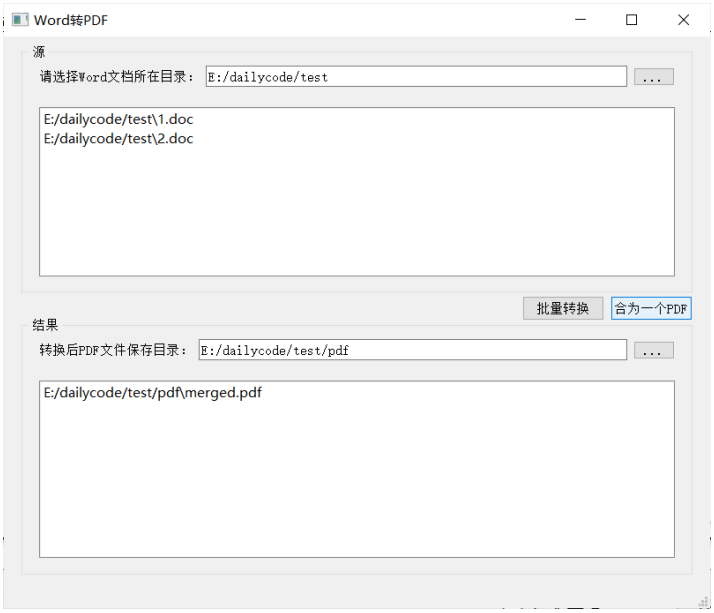


2.3、输出文件夹验证

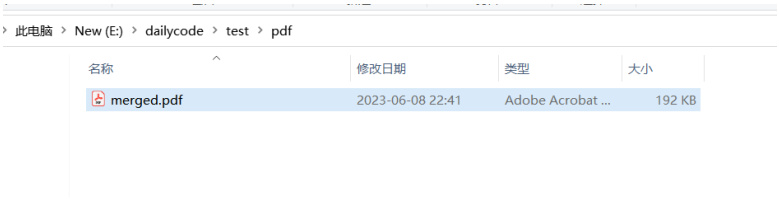


3、合为一个 PDF

3.1、点击合为一个 PDF 按钮，等待完成



3.2、输出文件夹验证

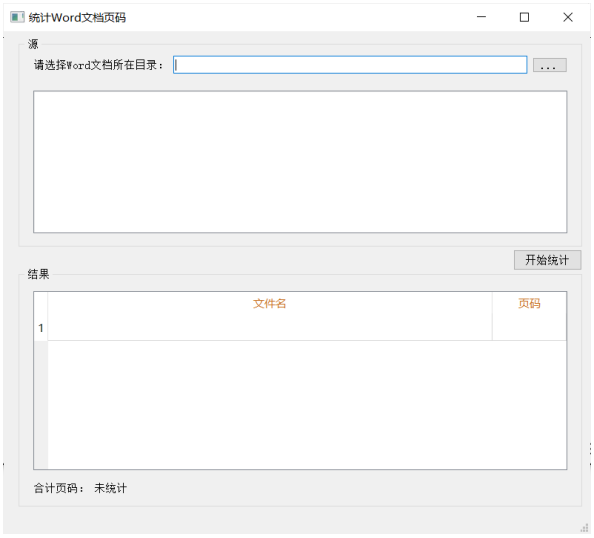


3.3、文件检验

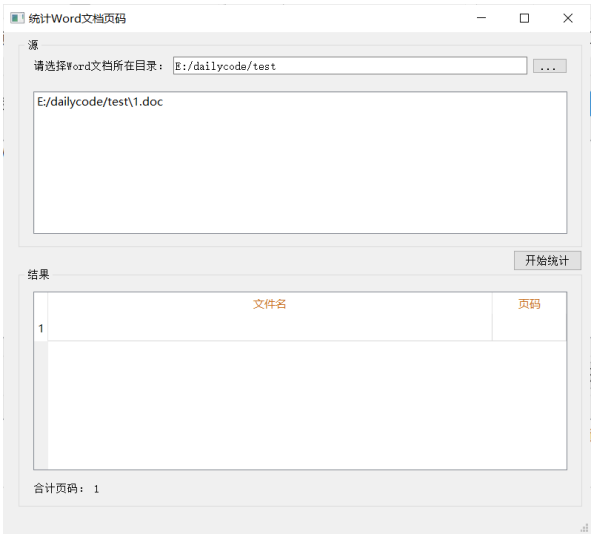
两个文件合成了一个 PDF，内容如下：（左为第一页，右为第二页）

2.3.3 统计 Word 文档页码

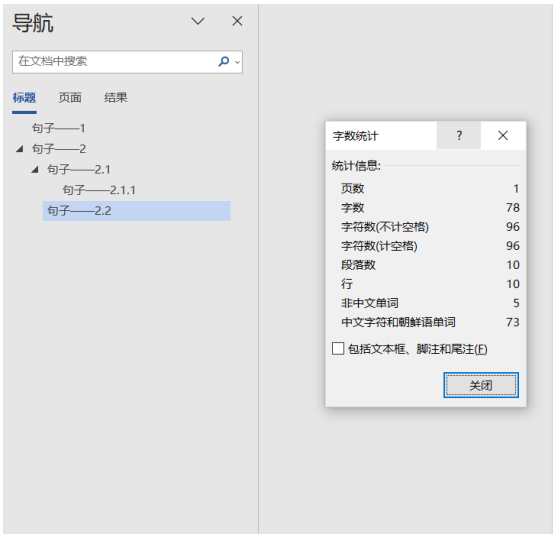
1、统计 Word 文档页码界面



2、点击开始统计按钮，等待完成



3、源文档对比



·句子——1

一人撑伞两人行，从此烟雨落京城。

·句子——2

假装看不见，余光千百遍。

·句子——2.1

初见乍然，久处亦怦然。

·句子——2.1.1

山水一程，三生有幸。

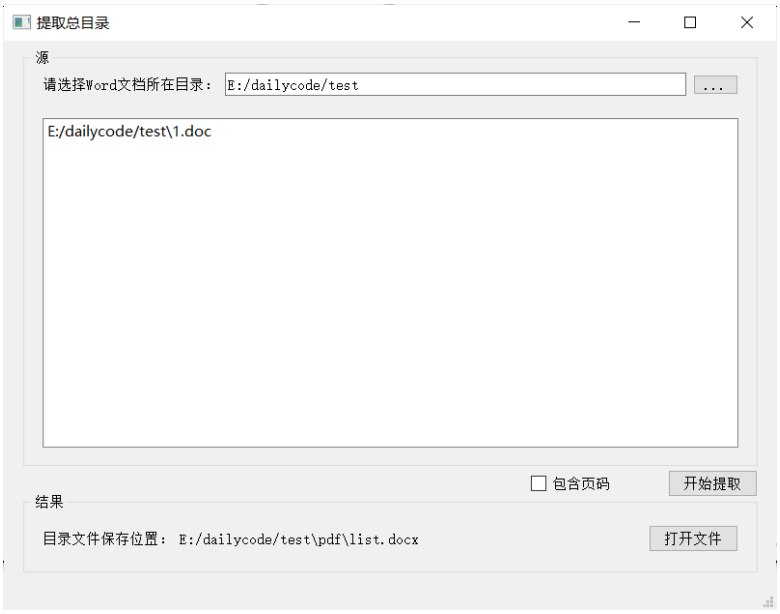
·句子——2.2

我见众人皆草木，唯你是青山。

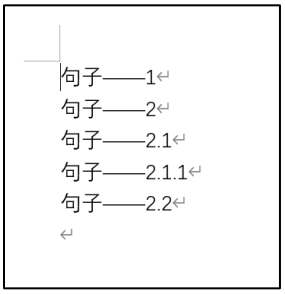
2.3.4 提取总目录

1、提取总目录界面

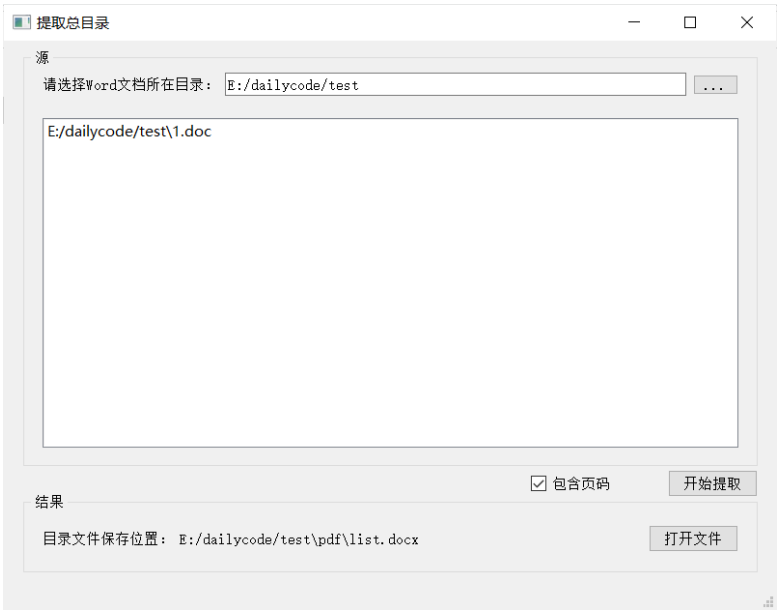
2、包含页码提取完成



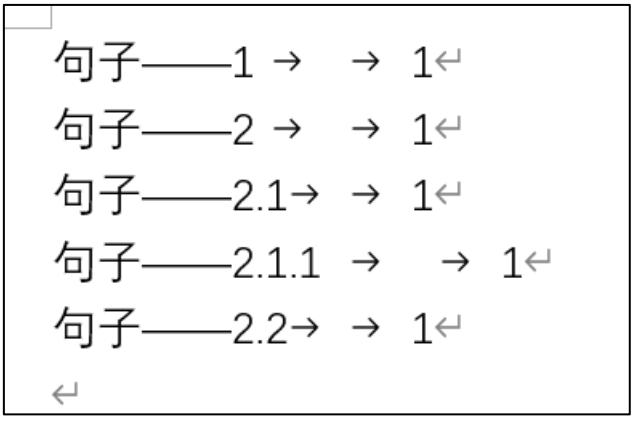
list.docx 文件内容



3、不包含页码



list.docx 文件内容：



3 实验过程中遇到的问题和解决过程

3.1 前端实现方式的选择

对于前端问题，由于之前并不怎么接触前端，所以一直不知道如何下手去做，最终，跟着相似的软件工具开发教程选择了对应的 Qt 作为前端开发工具，有根据教程对应的后端使用 Python 的 PyQt5 工具包，最终完成编写。

3.2 前端界面的设计

对于前端的设计，由于之前从未接触过前端的相关的代码，只是在控制台输出内容。本次工具的编写，首次选择前端规范化进行编写。如何将复杂的信息架构化并将其呈现出来是一个挑战。这需要在页面上进行良好的排版，使得页面有个清晰的结构，能够帮助用户快速定位信息。展示页面设计不仅需要美观、吸引人，同时也要在用户舒适度、易用性和可用性方面做出平衡。为此追求美而导致的华而不实，反而会让用户感到不舒适，难以实现重点信息的正确

传达。在综合各方面的因素之后，参考了现有的一些办公工具（如 WPS）的界面以后，最终将前端设计为当前的样式。

3.3 前端代码编写

前端对应的 Qt 之前没有接触过，对应的代码语言不熟悉，编写起来很困难。由于本身课程时间较为充裕，个人跟着 B 站相关教学视频一点点学习了如何编写 Qt 前端界面。虽然过程很艰难，但是最终完成了整个前端实现。

3.4 代码冗余

后端代码冗余主要体现在 WordtoPDF 部分、相应的排序规则、PDF 合并的相关代码重复编写上。为了保证代码的简洁明了，经过多次提取，形成了 tools 文件夹，将对应的通用工具性代码单独编写，方便调用。

3.5 结构混乱

结构混乱主要是因为之前没有个人单独编写过相应的工具，代码量比较大而且过程中出现了很多问题，再加上相应的编写周期较长，没能很好的将所有的代码整合到一块。多次尝试后，选择将工具结构提前规划好，同时使用相应的注释，尽可能将所有代码都添加上注释，方便时断时续地编写。此外，个人还找到了相应的教学视频，虽然功能存在差异，但是大致相同，编程过程中主要是跟着教学视频走，更加保证了不会出错。

3.6 Word 接口调用版本匹配

由于本工具是在调用 Word 2019 的开发工具，通过 Word 的相应功能实现的。但是由于对应的 Word 功能较为完善、环境封闭、版本越高越难以调用。虽然本工具是个人跟随教学视频完成的，但是对应的教学视频为 Word 2007 版本，与当前工具所对应的 Word 2019 并不相同。加上，网络上对于此处功能的教学较少，所以卡了很长时间。最终，经过多次尝试，通过以下代码成功调用 Word 2019。

```
gencache.EnsureModule('{00020905-0000-0000-C000-000000000046}', 0, 8, 7)
```

3.7 后端功能与前端按钮的对应

由于前后端是分别编写的，相应的前后端功能对应是一个大问题，在对应的教学视频以及个人的不断尝试下，最终解决这一问题。（详细请见 2.2 中将类的内容传递到前端部分的代码）

4 个人感悟

在此次 Python 实验过程中，老师给出了两种学习方式：跟教学视频，修改教学代码即可；个人选择项目进行编写开发。由于跟教学视频本身较为简单，所以个人一开始也选择相应的完成方式。

但是，由于在课程过程中外出实习一个月而且这一个月实习内容正好也是与 Python 有关的大数据项目，本身以为会因为自己在学校接触过会有部分基础，但是面对自己的项目、平时实习学习内容，自己束手无措。自己没有想象中的到达实习岗位后代码能力可以、设计能力不错，恰恰相反，自己就是一个新人，啥都不会。在经过几天的啥也不会之后，个人觉得需要练习一下个人的代码能力，经过思考，舍弃了已经开始的 Pyhton 实验报告，选择自己完成一个 Python 工具的开发。由于自己什么也不会，最终选择在跟着相应教学视频的前提下完成这一工具的编写。虽然这一工具距离我所学习的教学视频所实现的工具还是有很大差距，自己也在开发过程中盲目抄写了教学视频中的部分代码，例如有关线程、类设计、前端编写等内容的代码。虽然整个过程存在无数的瑕疵，但是最终勉强完成了一个工具的编写。对于相应的课程视频我也进行了相应的学习，只是没有很好地完成实验报告。

此次 Python 综合训练选择自行开发的原因并非是因为自己能力有多强，更多的是实习时看到了差距而且个人性格有现成代码就去复制粘贴，课程教学给了参考代码，会被不自觉地去复制。但是，我个人跟着学习的视频并没有给代码，只是看着视频一步步编码。虽然可能没有什么不同，但是个人感觉自己亲自一个字母一个字母地编写更加练习能力，或许也只是练一练码字能力，但是确实想从本次实验中学习到有关 Python、开发等的知识，而非一味地停留于文字、理论，希望自己的实际编写能力也有所提升。

总之，本次 Python 综合训练实验课程，从老师的教学视频学习到了很多东西，也从个人实际编写内容中学到了东西，此次 Python 课程收获满满。