小黄人时间管理器

内容索引

1. exe文件夹，里面有使用Pyinstaller生成的.exe可执行文件，可以直接双击运行(双击会先弹出一个终端，可能得等一小会儿才会弹出应用程序界面)
2. src文件夹，里面只包含.py文件。
3. full_src文件，除了.py文件，里面还包括了使用QtDesigner生成的.ui和.qrc文件，resource子目录下是程序使用到的一些图片与图标。
4. 说明文档
5. 演示视频

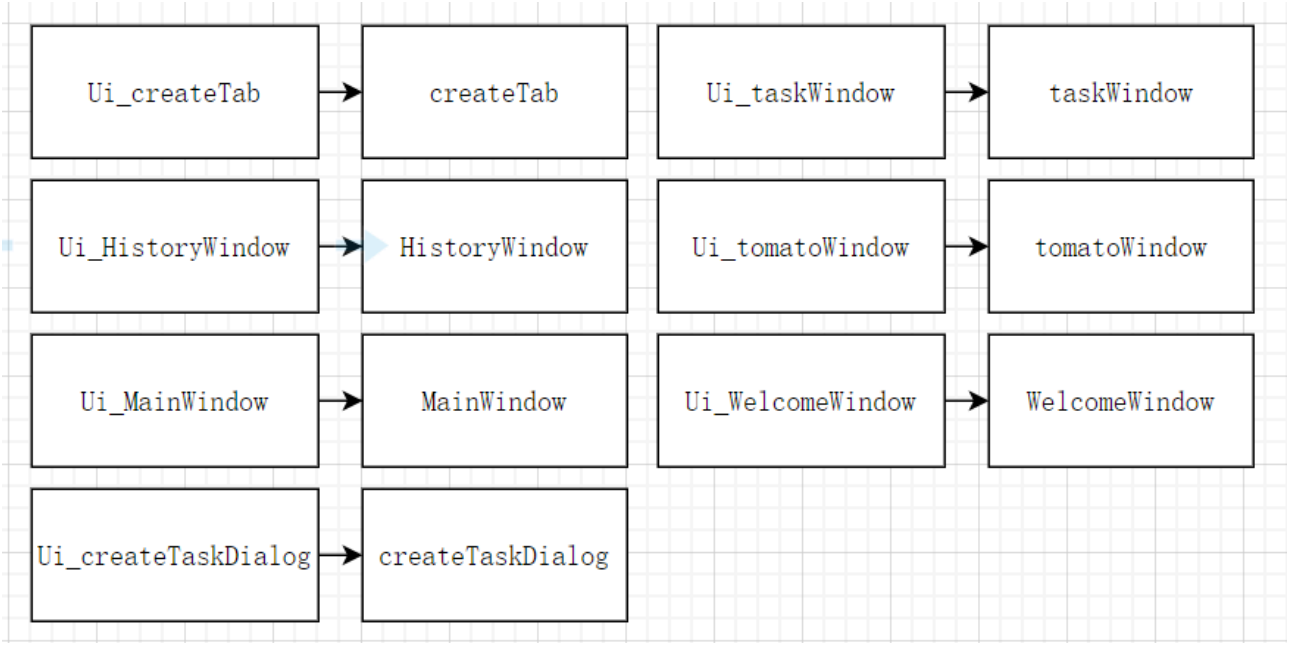
文档内容索引

1. 设计架构
2. 核心功能讲解
3. 程序使用说明书
4. 程序依赖的库与安装方法

设计架构

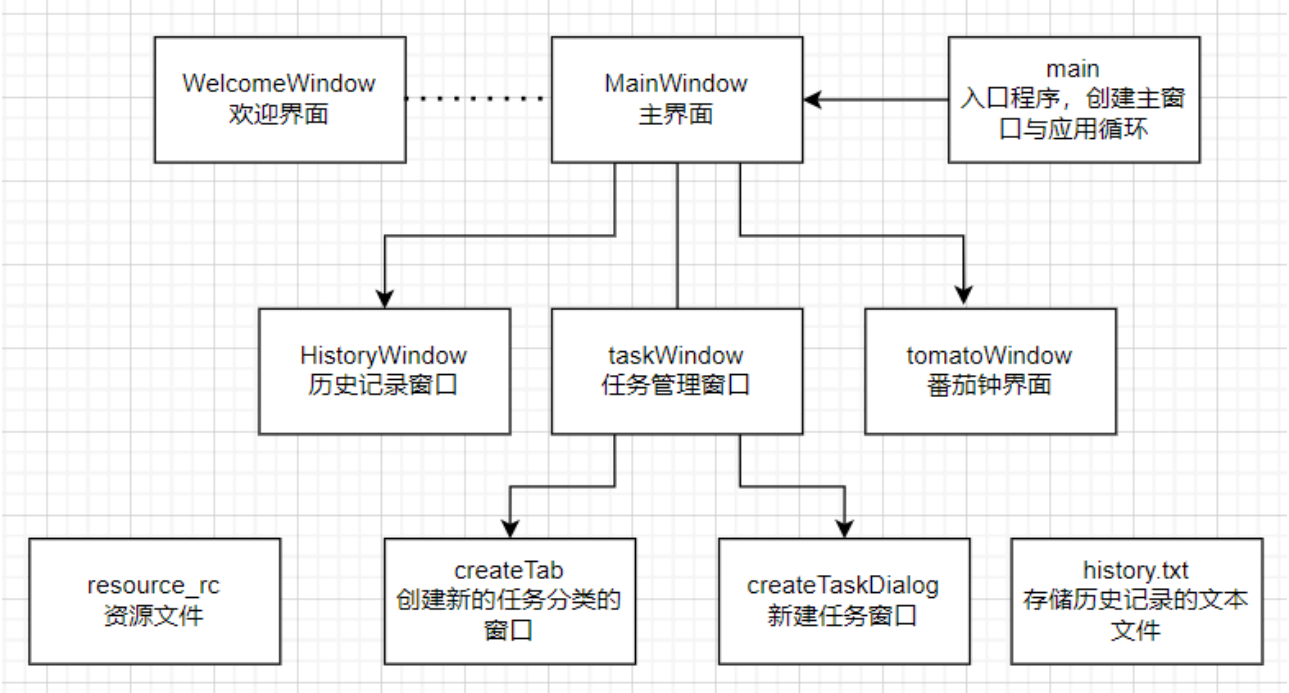
1. **开发工具与流程**
   该程序使用的IDE为Pycharm，配合PyQt5自带工具QtDesigner和PyQt自带编译工具rcc,uic。在QtDesigner中完成了部分图形界面的开发，生成图形界面.ui文件，再使用Qt自带编译工具uic将其转换为.py文件。所使用的资源文件(背景图片，图标等)在qt的.rc文件中，使用qt自带编译工具rcc将其转换为.py文件，文件名为resource_rc.py。
2. **结合QtDesigner开发细节**
   main.py是程序入口程序，负责创建主窗口与欢迎界面。以Ui*开头的文件都是由QtDesigner的.ui文件使用uic编译的.py源文件，而以Call开头的文件则是继承其对应的Ui*文件中的有关类创建的相应的窗口类并使用setupUi()函数初始化GUI，相当于Call*文件中的窗口类是对应的Ui_*文件中类的又一层封装，这样可以简化窗口类的调用。每个Ui_文件与Call*文件的对应

关系如图，箭头代表继承关系：



3. **程序组织构思**

   程序是通过多种窗口类来组织的，一个有7个窗口类，分别为createTab(创建新的任务分类窗口),HistoryWindow(查看历史记录窗口),MainWIndow(主窗口)，createTaskDialog(创建新任务窗口)，taskWindow(任务管理窗口),tomatoWindow(番茄钟窗口)，WelcomeWindow(欢迎界面窗口)，它们中有些是另一个窗口的子窗口。具体的关系见下图。



4. **单个类详解**

   每个窗口类都负责一定的功能，有子窗口类的窗口还要提供调用子窗口的接口。下面对每个窗口类及其作用与类方法作简要介绍。

- **WelcomeWindow , createTaskDialog , createTab三个窗口类**没有特别的类方法，只需要负责显示即可，因此它们只有一个**init**类方法来使用setupUi函数初始化GUI界面并使用

setWindowTile函数设置窗口标题，以及使用setGeometry函数来设置窗口大小。它们的作用分别为显示欢迎界面，显示创建任务的对话框以及显示创建新任务分类标签的对话框。对于用户在createTaskDialog，createTab窗口输入的数据的处理是交给其父窗口类taskWindow来处理的。

```
class WelcomeWindow(QMainWindow, Ui_WelcomeWindow):
    def __init__(self, parent: object = None) -> None:
        super(WelcomeWindow,self).__init__(parent)

        self.setGeometry(1000, 400, 511, 741)  # 设置窗口位置和大小
        self.setupUi(self)
        self.setWindowTitle("小黄人时间管理器")
```

- **MainWindow主窗口类**，它提供了调用三个功能窗口的接口（按钮）

```
# 信号与槽的连接,按下相应的按钮执行相应的功能
self.startTiming.clicked.connect(self.timerManage)
self.callTomatoWindowButton.clicked.connect(self.callMyTomatoWindow)
self.callTaskWIndowButton.clicked.connect(self.callMyTaskWindow)
self.readHistoryButton.clicked.connect(self.callMyHistoryWindow)
self.myTaskWindow.task_signal.connect(self.startTaskTiming)
self.endTaskButton.clicked.connect(self.endTaskTiming)
self.resetButton.clicked.connect(self.resetTimer)
```

功能的具体实现是在子窗口中处理的，但是历史记录的添加是由主窗口来处理的，因为选择开始执行任务时会跳转到主窗口计时器开始计时，点击主窗口结束任务按钮结束任务，此时主窗口会记录计时器时间，并向历史记录文本文件写入一条记录。

主窗口还有三个函数timerManage,resetTimer,timerUpdate是来负责控制计时器的，通过相应的开始/暂停按钮，归零按钮实现对计时器的控制。

- **HistoryWindow窗口类**，负责历史记录的处理，统计与显示。它读取保存历史纪录的文本文件history.txt并进行统计。它使用一个存储Task类的列表taskList来存储所有任务。

```
class Task:
    def __init__(self,name,kind,note,time,endurance=0):
        self.name = name  # 名称
        self.kind = kind  # 类型
        self.note = note  # 备注
        self.time = time  # 创建时间
        self.endurance = 0 # 执行时间
```

Task类有多个属性，分别存储一个任务的名称，种类，备注，执行时间，开始时间。

1. processHistory函数，负责从history.txt历史记录文件中读取记录并处理，处理结果存储在taskList中，同时也在dateList和kindList中存储进相应的值，这两个数据是字典类型，dateList的键为日期，而对应的值是当天完成任务的总时长，kindList的键为任务类型，对应的值是完成该类任务的累计总时长(因该函数代码较长，故不在此展示，可在代码文件CallHistoryWindow.py中查看)。
2. kindTabProcess函数，负责利用kindList中的值在按任务类别分类面板中展示统计结果
3. statisticShow函数，负责利用dateList中的值在按日期分类面板中展示统计结果。
4. plotDraw函数，负责利用kindList中的值画出饼图。
5. loadHistory函数，负责在每次打开查看历史记录窗口时更新显示面板的内容。
6. clearHstory函数，负责在按下清空历史记录按钮时清空所有历史记录。

- **tomatoWindow窗口类**，负责根据用户设定的时间来设置计时器，进行倒计时。
  1. manageTiming和updateTiming函数负责控制计时器的显示，逻辑与主窗口计时器逻辑类似。
  2. alert函数，负责在计时器到时提示用户。
  3. readTime函数负责当用户有自定义时间输入时或者没有自定义输入时读取选择的时间按钮的值，并设置计时器的值。
- **taskWindow窗口类**
  1. 提供了调用createTab和createTaskDialog子窗口的接口（按钮），通过showMyDialog和showCreateTabDialog函数来调用。
  2. process_task_input函数负责处理创建任务时用户在createTaskDialog窗口的输入
  3. process_tab_input函数负责处理创建新的任务分类时用户在createTab窗口的输入
  4. start_task函数负责任务的执行。当用户点击开始任务时，跳转到主窗口并开始计时，主窗口和任务管理窗口两个窗口之间的通信是通过发送与接受信号实现的。
  5. del_task函数负责任务的删除。
  6. init_default_creator函数和init_default_tab函数负责更新任务面板的显示内容，当创建新的任务或者新的任务分类标签时更新。

核心功能讲解

1. **开始任务与结束任务功能**：
   在任务管理窗口中选中要执行的任务并点击开始任务后，程序会跳转到主窗口，且主窗口计时器开始计时。具体代码如下：

```
def start_task(self):
    index = self.taskTab.currentIndex()  # 获取当前选中的标签页索引
    tab_name = self.taskTab.tabText(index)  # 获取当前标签页名称

    for tab in self.tabList:
        if tab.name == tab_name:
            selected_row = tab.list_widget.currentRow()  # 获取当前选中项的行号
```

```
            if selected_row != -1:  # 确保有选中的项
                selected_item = tab.list_widget.item(selected_row)  # 获取选中的
项
                task_content = selected_item.text()  # 获取选中项的文本
                self.task_signal.emit(task_content)  # 发射信号
                self.close()
            else:
                QMessageBox.warning(self, '提示', '请先选择要处理的任务')
```

程序会先获取当前选中的任务的内容，并保存在task_content中，然后从任务管理窗口发出一个信号以及将task_content送出

```
self.task_signal.emit(task_content)  # 发射信号
```

该信号会被主窗口接收，并保存task_content的值，便于结束任务时写入历史记录。结束任务的逻辑可见保存历史到文件功能。

2. **历史记录统计功能**

   历史记录的统计是在从历史记录保存文件中读取记录的同时一起完成的。历史记录的保存是在主窗口中完成，但是历史记录的处理统计是在历史记录显示窗口中完成的。

   HistoryWindow类的processHistory函数负责从文件中读取历史记录并处理。它有四个成员变量：history,taskList,dateList,kindList。作用分别为：列表taskList负责存储Task类类型的值，每一个Task类对应从历史记录中读取的一条历史记录。history为中间值，可忽略

   dateList为字典类型，键为日期，而键对应的值是该日期下执行任务的总时间。

   kindList为字典类型，键为任务类别，而键对应的值是属于该类别的所有执行的任务的总时 间。

```
def processHistory(self):
    self.history = []
    self.taskList = []
    self.dateList = dict()
    self.kindList = dict()

    file_path = "history.txt"
    # 判断文件是否存在
    if not os.path.exists(file_path):
        # 如果文件不存在，创建文件
        with open(file_path, 'w') as file:
            file.write('')  # 创建空文件

    with open("history.txt", 'r') as file:
        for line in file:
            parts = line.strip().split('    ')  # 去掉行首尾空白字符
            filtered_parts = [item.strip() for item in parts if item.strip()]
```

```
            name = filtered_parts[0].strip().split(':')[1].strip()
            kind = filtered_parts[1].strip().split(':')[1].strip()
            note = filtered_parts[2].strip().split(':')[1].strip()
            endurance = float(filtered_parts[3].strip().split(' ')[1].strip())
            time = filtered_parts[4].strip().split(':',1)[1].strip()
            if kind not in self.kindList.keys():
                self.kindList[kind] = endurance
            else:
                self.kindList[kind] += endurance
            if time.split(" ")[0] not in self.dateList.keys():
                self.dateList[time.split(" ")[0]] = endurance
            else:
                self.dateList[time.split(" ")[0]] += endurance
            new_task = Task(name,kind,time,note,endurance)
            self.taskList.append(new_task)

    # 按日期排序
    self.taskList.sort(key=lambda task: task.time)  # 按时间属性排序
```

函数从历史记录文件history.txt中依次读取一行，并提取相应的信息，同时进行统计（if else语句块)。

3. **历史记录显示功能**：

   历史记录显示面板是一个ListWidget控件，每读取一条历史记录，就往ListWidget控件里面添加一个QListWidgetItem小组件，小组件的显示内容是经过处理的任务信息。

```
def loadHistory(self):
    self.processHistory()
    # 更新列表显示
    self.showHistoryLW.clear()  # 清空已有的显示
    for task in self.taskList:
        line = f"任务名称: {task.name}  类型: {task.kind} 时间:
{task.time.strftime('%Y-%m-%d %H:%M:%S')} 执行时间: {task.endurance:1.2f} 分钟
备注: {task.note}"  # 格式化任务信息
        new_item = QListWidgetItem(line.strip())
            self.showHistoryLW.addItem(new_item)    # 添加条目
```

4. **细分的历史记录显示与统计功能**：

   前面说到，历史记录的统计与处理是在读取历史记录的同时同步完成的。统计完成后，剩下的工作就是显示，显示的逻辑和3中的相同，只是按任务分类显示时，不是通过QListWidget，而是通过QTabWidget,对于每一个不同的分类，添加一个单独的Tab来显示其总时间

```
for tab in self.kindList.keys():
    new_tab = Tab(tab)
    newTabWidget = QWidget()  # 创建 QWidget 作为标签页
    newTabWidget.setLayout(new_tab.tab_layout)  # 设置布局
    self.kindTab.addTab(newTabWidget, tab)  # 将新标签页添加到 tab
```

5. **保存历史记录到文件功能**：

历史记录保存在主目录下一个加history.txt的文本文件中，如果该文件不存在，则创建该文件，如果该文件存在，则向该文件内追加内容。历史记录的保存是在主窗口完成的，因为开始执行任务后会跳转到主窗口时钟开始计时，点击主窗口的结束任务按钮结束计时。具体代码如下：

```
def endTaskTiming(self):
    time = self.Timer.text()  # 获取当前计时器的内容
    task_time = time_calculator(time)  # 获取总任务时间
    self.resetTimer()
    self.ongoingTask += f"    时间: {task_time:.2f} 分钟"

    # 获取当前日期
    current_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  # 格式化日期

    # 追加日期信息
    self.ongoingTask += f"    日期: {current_date}"

    with open("history.txt", 'a') as f:      # 写入一条历史记录
        f.write(self.ongoingTask + "\n")
```

获取当前计时器的信息来计算任务执行的时间，追加到要写入文件的历史记录字符串ongoingTask末尾，然后打开文件并写入。

6. **创建任务分类功能**：

通过QTabWidget来为每个分类创建一个Tab来分别显示不同任务分类下的任务列表

```
def process_tab_input(self):
    newTabWidget = QWidget()
    tab_text = self.myCreateTabDialog.tabInput.text()
    new_tab = Tab(tab_text)

    self.taskTab.addTab(newTabWidget,tab_text)
    newTabWidget.setLayout(new_tab.tab_layout)
    new_tab.tab_layout.addWidget(new_tab.list_widget)
```

从createTab窗口读取用户输入的新分类的名称，然后创建一个Tab类

```
class Tab:
    def __init__(self,name):
        self.name = name  # 名称
        self.taskList = []
        self.tab_layout = QVBoxLayout()
        self.list_widget = QListWidget()
```

每个Tab类有四个成员变量，分别为名称，属于该分类的任务列表，一个布局控件tab_layout,来设置任务显示的垂直布局，一个ListWidget控件，用来控制任务条目的显示。当该类下创建新的任务时，将新任务添加进list_widget控件即可。

7. **番茄钟功能**：

   番茄钟功能的特点是，由用户选择要倒计时的时间，然后程序读取输入来设置时钟。读取输入的代码如下：

```
def readTime(self) -> int:
    if self.editTomato.value():
        value = self.editTomato.value()
        self.editTomato.setValue(0)
        return int(value)
    elif self.tomato10.isChecked():
        return 10
    elif self.tomato20.isChecked():
        return 20
    elif self.tomato30.isChecked():
        return 30
    elif self.tomato45.isChecked():
        return 45
    else:
        return 0
```

首先，手动输入框具有最大优先级，当手动输入框没有输入时，查看哪个单选框被勾选，从而该函数返回对应的时间。时钟的更新是通过QTimer类来实现的。将该QTimer的时间间隔设置为1秒发送一次timeout信号，将该信号与更新时钟的函数updateTiming绑定。即每一秒刷新一次时钟显示。

```
self.tomatoTimer = QTimer()
self.tomatoTimer.setInterval(1000)  # 每秒触发一次
self.tomatoTimer.timeout.connect(self.updateTiming)  # 连接超时信号

self.isTomatoTiming = False
self.startTomato.clicked.connect(self.manageTiming)  # 连接按钮点击信号
```

使用一个标志isTomatoTiming来指示是否正在计时。当点击开始按钮时调用函数manageTiming。该函数判断是否正在计时，如果正在计时，就更新时钟，如果没有计时，则开始计时，并设置时钟为用户选定的值。

```
def manageTiming(self):
    if not self.isTomatoTiming:
        self.isTomatoTiming = True
        self.remainingTime = self.readTime() * 60  # 转换为秒
        show_minutes = self.remainingTime // 60
        show_seconds = self.remainingTime % 60
        self.tomatoTimeBoard.setText(f'{show_minutes:02}:{show_seconds:02}')
        self.tomatoTimer.start()
    else:
        self.isTomatoTiming = False
        self.tomatoTimer.stop()
```

当开始计时以后，更新时钟的函数updateTiming也与时钟信号绑定，每秒调用一次来刷新时钟显示的值：

```
def updateTiming(self):
    if self.remainingTime > 0:
        self.remainingTime -= 1
        show_minutes = self.remainingTime // 60
        show_seconds = self.remainingTime % 60
        self.tomatoTimeBoard.setText(f'{show_minutes:02}:{show_seconds:02}')
    else:
        self.tomatoTimer.stop()
        self.alert()
        self.isTomatoTiming = False
```

如果倒计时没有到，则将当前时钟-1，更新时钟显示。如果时间到，则调用alert函数，发送时间到的提醒。

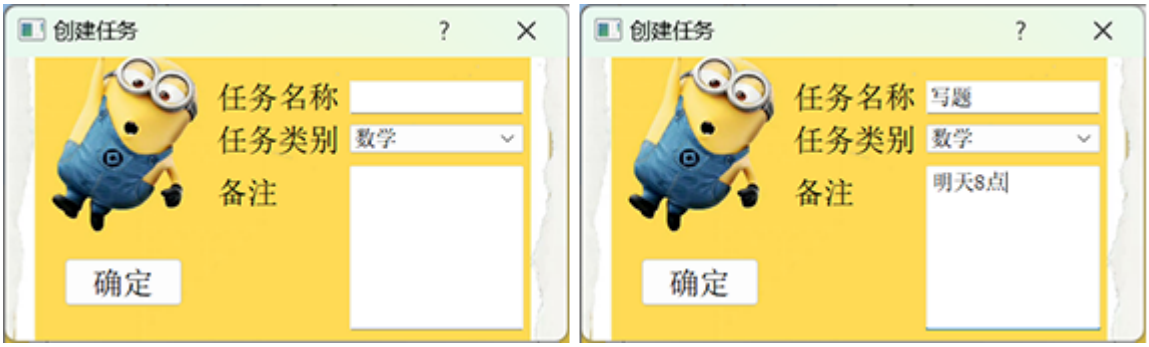
```
def alert(self):
    QMessageBox.warning(self, '提示', '时间到!')
```

8. **任务备注功能**：
   创建的每一个任务都是以Task类的形式存储的

```
class Task:
    def __init__(self,name,kind,note,time,endurance=0):
        self.name = name  # 任务名称
        self.kind = kind  # 任务分类
```

```
        self.note = note  # 任务备注
        self.time = time  # 创建时间
        self.endurance = 0 # 执行时间
```

在创建任务时可以输入任务备注,保存时输入的备注保存在Task类实例的note成员变量中。保存历史记录时备注也会被保存。显示面板上也会显示任务备注。



具体的处理逻辑是使用读取的数据来初始化一个Task类

```
input_kind = self.myDialog.taskKind.currentText()  # 获取任务分类
input_name = self.myDialog.taskName.text()  # 获取任务名称
input_note = self.myDialog.taskNote.toPlainText()  # 获取任务备注
current_time = datetime.now()  # 获取创建任务的时间

new_task = Task(input_name,input_kind,input_note,current_time)
```

9. **删除已有任务功能**:

```
def del_task(self):
    index = self.taskTab.currentIndex()  # 获取当前选中的标签页索引
    tab_name = self.taskTab.tabText(index)  # 获取当前标签页名称

    for tab in self.tabList:
        if tab.name == tab_name:
            selected_row = tab.list_widget.currentRow()  # 获取当前选中项的行号
            if selected_row != -1:  # 确保有选中的项
                # 删除选中的任务项
                tab.list_widget.takeItem(selected_row)
                QMessageBox.warning(self, '提示', '删除成功')
            else:
                QMessageBox.warning(self, '提示', '请先选择要删除的任务')
```

创建的任务显示在任务分类显示面板上，每一个任务分类对应一个tab类，

tab类里有四个成员变量,分别是任务分类的名称name，所属该任务分类下的任务列表taskList,一个垂直布局控件tab_layout，一个QListWidget控件list_widget。

```
class Tab:
    def __init__(self,name):
        self.name = name  # 名称
        self.taskList = []
        self.tab_layout = QVBoxLayout()
        self.list_widget = QListWidget()
```

这些tab类存储在taskWindow类的类成员变量tabList中，所以当要删除某项任务时，先获取当前标签页名称

```
index = self.taskTab.currentIndex()  # 获取当前选中的标签页索引
tab_name = self.taskTab.tabText(index)  # 获取当前标签页名称
```

再通过for循环遍历tabList确定任务所在的标签页，然后获取当前选中项，并将该项直接从tab的布局控件中删除即可，删除成功会显示消息。如果没有选中任务点击删除则会有提示“请先选择要删除的任务”。

```
for tab in self.tabList:
    if tab.name == tab_name:
```

```
        selected_row = tab.list_widget.currentRow()  # 获取当前选中项的行号
        if selected_row != -1:  # 确保有选中的项
            # 删除选中的任务项
            tab.list_widget.takeItem(selected_row)
            QMessageBox.warning(self, '提示', '删除成功')
        else:
            QMessageBox.warning(self, '提示', '请先选择要删除的任务')
```

程序使用说明书

1. 执行程序一开始进入欢迎界面，经过三秒欢迎界面自动关闭，显示主窗口。左图为主窗口，右图为欢迎界面。

2. 主窗口有一个计时器与附属的三个控制按钮，分别为开始/暂停按钮，结束任务按钮，归零按钮。
   开始/暂停按钮：手动开始或者暂停主界面计时器。不执行任务时，可以当作普通计时器使用。执行任务时，当进入任务界面选择要执行的任务并点击开始任务按钮时会自动跳转到主界面，此时主界面的该计时器自动开始计时，可以点击该按钮暂停任务计时。

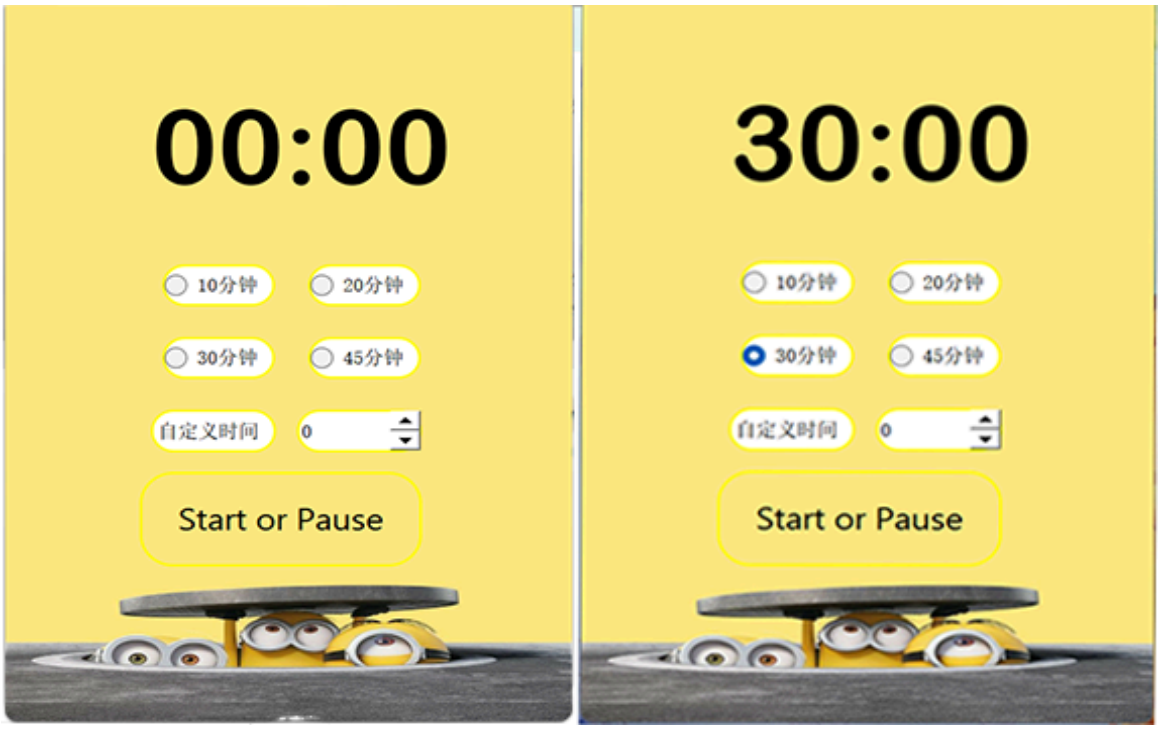
结束任务按钮：当正在执行任务时，点击该按钮则结束任务并且将计时器清零，同时写入一条任务执行记录。

归零按钮：将计时器清零。

3. 番茄钟功能

点击主界面底部的番茄钟按钮进入番茄钟功能界面。界面上有六种预设的时间，选择相应的预选框并点击开始即可进入倒计时，点击暂停按钮课暂停计时，但此时再次点击开始时时间会重置为选择的时间，意思就是没有暂停的机会！所以一定要坚持完成每个番茄钟哦~。

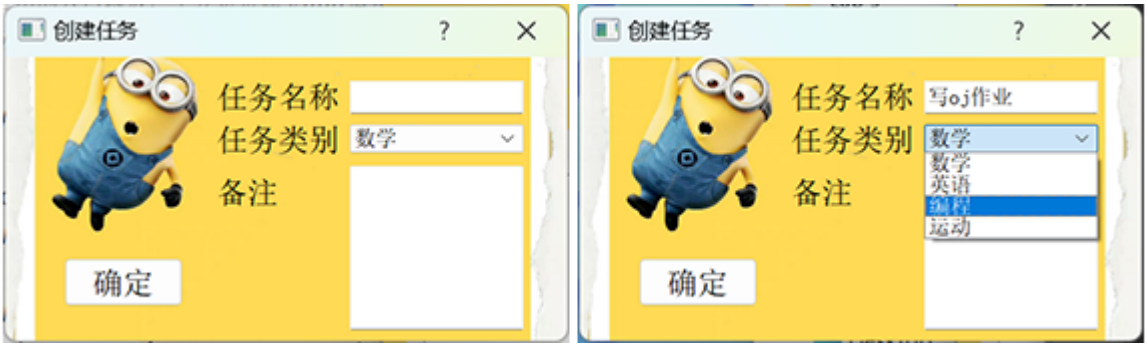
同时还有自定义时间的功能，只要在框内键盘输入或通过上下按钮选择想要的时间即可



4. 任务管理器

点击主界面底部的任务按钮即可进入任务管理器界面。在该界面可以创建，开始或删除任务，且可以创建新的任务分类。

点击创建任务按钮会弹出创建任务窗口。在该窗口中可以填写想要创建的任务的详细信息，比如任务名称，备注，还可以从已有的分类中选择一个该任务的所属分类。



点击确认后，创建任务窗口自动关闭，在任务显示分类面板上选择相应的分类，可以查看自己创建的任务。

选中一项任务然后点击删除按钮可以将该任务删除。

点击创建新的分类按钮可以创建新的任务分类

在任务展示面板上选中一项任务并点击开始任务按钮，程序会自动跳到主界面，且主界面计时器自动开始计时。可以按开始/暂停按钮来继续或者暂停任务计时。点击结束任务按钮来结束该任务的计时且将计时器归零，此时程序会自动保存该次任务的执行记录，记录可以在历史记录面板查看（稍后介绍）。

5. 查看历史记录窗口

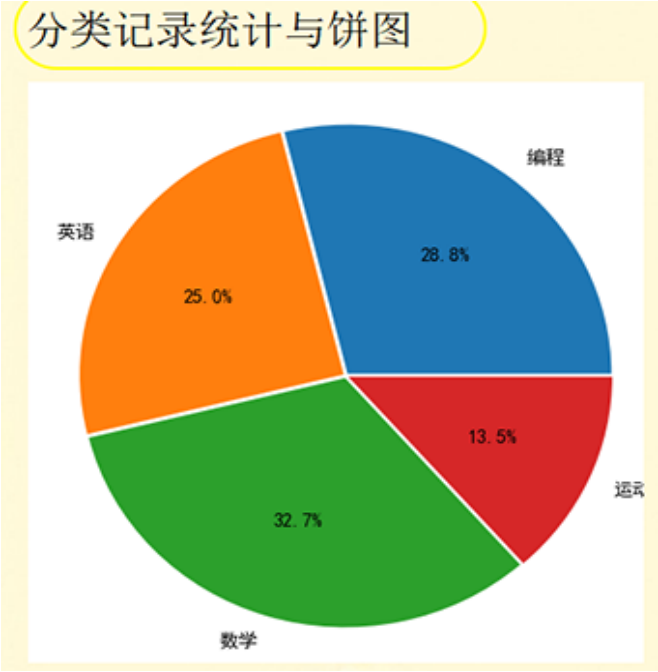
点击主界面底部的历史按钮，可以打开查看历史记录窗口。注意：只有在任务管理窗口里选择开始任务，并且在主窗口计时器下方点击结束任务按钮结束的任务才会有历史记录。

可以看到在查看历史记录窗口中有四个面板，展示历史记录的统计结果。

- 历史记录查看面板：展示所有的历史记录，按任务执行的先后顺序从上到下排列。

- 历史记录统计-按日期分类面板：该面板展示按照日期统计的任务执行情况，比如哪一天共执行任务多少分钟。
- 历史记录统计-按任务类别分类面板：该面板展示按照任务类别分类统计的任务执行情况，比如共执行数学类任务多少分钟

- 分类记录统计与饼图面板：该面板以饼图的形式展示根据上面按任务类别分类面板的统计结果。



- 清空历史记录按钮：划至界面底部，可以看到一个清空历史记录按钮，点击该按钮可以清空所有历史记录(历史记录实际上保存在一个叫history.txt文本文件中，清空历史记录就是清空该文件)

程序依赖的库与安装方法

1. Python 解释器-我的python版本为3.8.6

```
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. PyQt5 PyQt5-tools

```
# 最好不要使用PyQt6，PyQt6中废弃了PyQt5的一些函数，可能导致程序无法运行
# 安装完成后不要忘记添加环境变量
pip install PyQt5
pip install PyQt5-tools
```

详细的安装步骤可见CSDN-PyQt5的安装

3. matplotlib 图形库

```
pip install matplotlib
```

4. python内置模块datetime，sys，os无需另外安装