1. StickyNotes
   1. DetailBox.py

from imports import \*

class FormDialog(QDialog):  # type: ignore

    # 插入或修改数据的表单

    def \_\_init\_\_(self, item, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Detail")

        self.setWindowFlags(Qt.Drawer)

        # Remove the '?' button by setting the window flags

        self.setWindowFlags(

            self.windowFlags() & ~Qt.WindowContextHelpButtonHint

            | Qt.WindowCloseButtonHint

        )

        self.item = item

        # 文字和输入框

        self.id\_Label = QLabel("Id:", self)  # type: ignore

        self.id\_LineEdit = QLineEdit(self)  # type: ignore

        self.id\_Layout = QHBoxLayout()

        self.id\_Layout.addWidget(self.id\_Label, alignment=Qt.AlignLeft)

        self.id\_Layout.addWidget(self.id\_LineEdit, alignment=Qt.AlignRight)

        self.title\_Label = QLabel("Title:", self)

        self.title\_LineEdit = QLineEdit(self)

        self.title\_Layout = QHBoxLayout()

        self.title\_Layout.addWidget(self.title\_Label, alignment=Qt.AlignLeft)

        self.title\_Layout.addWidget(self.title\_LineEdit, alignment=Qt.AlignRight)

        self.describe\_Label = QLabel("Describe:", self)

        self.describe\_LineEdit = QLineEdit(self)

        self.describe\_Layout = QHBoxLayout()

        self.describe\_Layout.addWidget(self.describe\_Label, alignment=Qt.AlignLeft)

        self.describe\_Layout.addWidget(self.describe\_LineEdit, alignment=Qt.AlignRight)

        self.important\_Label = QLabel("Important:", self)

        self.important\_LineEdit = QLineEdit(self)

        self.important\_Layout = QHBoxLayout()

        self.important\_Layout.addWidget(self.important\_Label, alignment=Qt.AlignLeft)

        self.important\_Layout.addWidget(

            self.important\_LineEdit, alignment=Qt.AlignRight

        )

        self.start\_time\_Label = QLabel("Start\_time:", self)

        self.start\_time\_LineEdit = QLineEdit(self)

        self.start\_time\_Layout = QHBoxLayout()

        self.start\_time\_Layout.addWidget(self.start\_time\_Label, alignment=Qt.AlignLeft)

        self.start\_time\_Layout.addWidget(

            self.start\_time\_LineEdit, alignment=Qt.AlignRight

        )

        self.end\_time\_Label = QLabel("End\_time:", self)

        self.end\_time\_LineEdit = QLineEdit(self)

        self.end\_time\_Layout = QHBoxLayout()

        self.end\_time\_Layout.addWidget(self.end\_time\_Label, alignment=Qt.AlignLeft)

        self.end\_time\_Layout.addWidget(self.end\_time\_LineEdit, alignment=Qt.AlignRight)

        self.file\_Label = QLabel("File:", self)

        self.file\_LineEdit = QLineEdit(self)

        self.file\_Layout = QHBoxLayout()

        self.file\_Layout.addWidget(self.file\_Label, alignment=Qt.AlignLeft)

        self.file\_Layout.addWidget(self.file\_LineEdit, alignment=Qt.AlignRight)

        self.submitButton = QPushButton("Submit", self)

        self.submitButton.clicked.connect(self.submit\_form)

        # 修改时显示原先数据

        if self.item != None:

            self.id\_LineEdit.setText(str(self.item.id))

            self.title\_LineEdit.setText(self.item.title)

            self.describe\_LineEdit.setText(self.item.describe)

            self.important\_LineEdit.setText(str(self.item.important))

            self.start\_time\_LineEdit.setText(self.item.start\_time)

            self.end\_time\_LineEdit.setText(self.item.end\_time)

            self.file\_LineEdit.setText(self.item.file)

        layout = QVBoxLayout()

        layout.addLayout(self.id\_Layout)

        layout.addLayout(self.title\_Layout)

        layout.addLayout(self.describe\_Layout)

        layout.addLayout(self.important\_Layout)

        layout.addLayout(self.start\_time\_Layout)

        layout.addLayout(self.end\_time\_Layout)

        layout.addLayout(self.file\_Layout)

        layout.addWidget(self.submitButton)

        self.setLayout(layout)

    def submit\_form(self):

        self.form\_data = {

            "id": self.id\_LineEdit.text(),

            "title": self.title\_LineEdit.text(),

            "describe": self.describe\_LineEdit.text(),

            "important": self.important\_LineEdit.text(),

            "start\_time": self.start\_time\_LineEdit.text(),

            "end\_time": self.end\_time\_LineEdit.text(),

            "file": self.file\_LineEdit.text(),

        }

        # 类型检查、空值判断

        try:

            self.form\_data["id"] = eval(self.form\_data["id"])

            # 同时做了空值判断和类型检查

        except:

            QMessageBox.warning(self, "Event error", "Id must be integer!!!")

        if self.form\_data["important"] == "":

            self.form\_data["important"] = "null"  # 空值判断

        else:

            try:

                self.form\_data["important"] = eval(

                    self.form\_data["important"]

                )  # 类型检查

            except:

                QMessageBox.warning(self, "Event error", "Important must be integer!!!")

        # 空值判断

        if self.form\_data["title"] == "":

            self.form\_data["title"] = "null"

        if self.form\_data["describe"] == "":

            self.form\_data["describe"] = "null"

        if self.form\_data["start\_time"] == "":

            self.form\_data["start\_time"] = "null"

        if self.form\_data["end\_time"] == "":

            self.form\_data["end\_time"] = "null"

        if self.form\_data["file"] == "":

            self.form\_data["file"] = "null"

        self.accept()

    def get\_form\_data(self):

        return self.form\_data

class DetailBox(QDialog):

    # 一个event对应一个detailbox

    def \_\_init\_\_(self, db, item, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Sticky Notes Event Detail")

        self.setWindowFlags(Qt.Drawer)

        self.db = db

        self.item = item

        # Remove the '?' button by setting the window flags

        self.setWindowFlags(

            self.windowFlags() & ~Qt.WindowContextHelpButtonHint

            | Qt.WindowCloseButtonHint

        )

        self.label = QLabel(str(self.item))

        self.okButton = QPushButton("OK")

        self.okButton.clicked.connect(self.accept)

        self.updateButton = QPushButton("Update")

        self.updateButton.clicked.connect(self.detailUpdate)

        # Horizontal layout for the buttons

        buttonLayout = QHBoxLayout()

        buttonLayout.addWidget(self.updateButton)

        buttonLayout.addWidget(self.okButton)

        layout = QVBoxLayout()

        layout.addWidget(self.label)

        layout.addLayout(buttonLayout)

        self.setLayout(layout)

    def detailUpdate(self):

        formDialog = FormDialog(self.item)

        if formDialog.exec\_() == QDialog.Accepted:

            form\_data = formDialog.get\_form\_data()

            self.db.Update(

                self.item.id,

                id=form\_data["id"],

                title=form\_data["title"],

                describe=form\_data["describe"],

                important=form\_data["important"],

                start\_time=form\_data["start\_time"],

                end\_time=form\_data["end\_time"],

                file=form\_data["file"],

            )

            self.item = self.db.Select(form\_data["id"])

            self.label.setText(str(self.item))

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    def show\_silent\_message():

        db = Database()

        msg = DetailBox(db, db.Select(1))

        msg.exec\_()

    show\_silent\_message()

    sys.exit(app.exec\_())

* 1. Imports.py

import os

import sys

import winreg as reg

from PyQt5.QtCore import \*

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from PyQt5.QtSql import \*

from Ui import \*

from Sql import \*

from LineDelegate import \*

from DetailBox import \*

from Startup import \*

* 1. LineDelegate.py

from imports import \*

class LineDelegate(QStyledItemDelegate):  # type: ignore

    # 自定义信号，传递项目索引

    rightClicked = pyqtSignal(int)  # type: ignore

    leftClicked = pyqtSignal(int)  # type: ignore

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

    def paint(self, painter, option, index):

        painter.save()

        # 设置字体大小

        self.font = QFont()  # type: ignore

        self.font.setPointSize(14)  # 设置字体大小为14

        painter.setFont(self.font)

        # 设置文本颜色

        painter.setPen(option.palette.color(QPalette.Text))  # type: ignore

        super().paint(painter, option, index)

        painter.restore()

    def sizeHint(self, option, index):

        size = super().sizeHint(option, index)

        size.setHeight(size.height() + 10)  # 增加高度以适应分隔线

        return size

    def createEditor(self, parent, option, index):

        # 返回None，表示不允许编辑

        return None

    def editorEvent(self, event, model, option, index):

        if event.type() == QEvent.MouseButtonPress and event.button() == Qt.RightButton:  # type: ignore

            self.rightClicked.emit(index.row())  # 发出右键点击信号

            return True

        elif (

            event.type() == QEvent.MouseButtonPress and event.button() == Qt.LeftButton  # type: ignore

        ):

            self.leftClicked.emit(index.row())  # 发出左键点击信号

            return True

        return super().editorEvent(event, model, option, index)

* 1. Main

from imports import \*

plugin\_path = os.path.join(

    os.path.dirname(sys.executable), "Lib", "site-packages", "PyQt5", "Qt", "plugins"

)

os.environ["QT\_QPA\_PLATFORM\_PLUGIN\_PATH"] = plugin\_path

# 设置为exe即将存储的目录路径或克隆下来的项目路径

directory = "D:\college\Project\StickyNotes"

Startup.create\_bat\_file(directory)

class Main(QMainWindow):  # type: ignore

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # 使用生成的界面类创建界面对象

        self.ui = Ui()

        self.ui.setupUi(self)  # 设置界面

        # 可以在这里连接信号和槽，设置界面初始化逻辑、初始化变量等

        self.setWindowTitle("Sticky Notes")

        self.is\_on\_top = False  # 是否置顶

        self.is\_startup = Startup.is\_in\_startup(

            directory + "\StickyNotes.bat"

        )  # 是否开机启动

        self.colored\_dates = []  # 选中item的日期

        self.item = None  # 选中的item

        # event栏

        self.model = QStandardItemModel()  # 创建一个 QStandardItemModel

        self.ui.listView.setModel(self.model)  # 设置为QListView的模型

        self.delegate = LineDelegate()  # 设置自定义委托

        self.ui.listView.setItemDelegate(self.delegate)

        self.delegate.rightClicked.connect(self.showListViewMenu)  # 设置右键点击事件

        self.delegate.leftClicked.connect(self.colorSelectedDate)  # 设置左键点击事件

        self.db = Database()  # 连接数据库

        for i in self.db.SelectALL():

            self.model.appendRow(i)  # 添加项目

        # 时钟

        self.ui.lcdNumber.setDigitCount(19)  # 设置显示的位数，包括日期和时间

        self.timer = QTimer(self)  # 创建一个定时器，每秒更新一次时间显示

        self.timer.timeout.connect(self.updateTime)  # 设置时钟更新

        self.timer.start(1000)  # 每隔1000毫秒（1秒）更新一次

    def updateTime(self):

        # 获取当前日期和时间

        currentDateTime = QDateTime.currentDateTime()

        # 将日期和时间格式化为字符串

        displayText = currentDateTime.toString("yyyy-MM-dd hh:mm:ss")

        # 在 QLCDNumber 上显示时间

        self.ui.lcdNumber.display(displayText)

    def clearPreviousFormats(self):

        if self.item != None:

            self.item = None

        if len(self.colored\_dates) != 0:

            for date in self.colored\_dates:

                self.ui.calendarWidget.setDateTextFormat(date, QTextCharFormat())

            self.colored\_dates = []

    def colorSelectedDate(self, index):

        # Clear previous formats

        self.clearPreviousFormats()

        self.item = self.model.item(index)

        # 给对应的日期上色

        fmt = QTextCharFormat()

        fmt.setBackground(QBrush(QColor("yellow")))

        date\_format = "yyyy-MM-dd hh:mm:ss"

        start\_time = self.item.start\_time

        end\_time = self.item.end\_time

        start\_time = QDateTime.fromString(start\_time, date\_format)

        end\_time = QDateTime.fromString(end\_time, date\_format)

        start\_date = start\_time.date()

        end\_date = end\_time.date()

        if not start\_date.isNull():

            current\_date = start\_date

            while True:

                self.ui.calendarWidget.setDateTextFormat(current\_date, fmt)

                self.colored\_dates.append(current\_date)

                current\_date = current\_date.addDays(1)

                if current\_date > end\_date:

                    break

    def showListViewMenu(self, index):

        self.colorSelectedDate(index)

        self.item = self.model.item(index)

        # 在右键点击时显示上下文菜单

        menu = QMenu(self)

        action1 = menu.addAction("detail")

        action2 = menu.addAction("file")

        action3 = menu.addAction("delete")

        # 显示菜单并获取所选的操作

        action = menu.exec\_(self.cursor().pos())

        # 根据所选的操作执行相应的逻辑

        if action == action1:

            DetailBox(self.db, self.item).exec\_()  # type: ignore

            self.model.clear()

            for i in self.db.SelectALL():

                self.model.appendRow(i)  # 添加项目

        elif action == action2:

            files = self.item.file

            files = files.split("; ")

            for file in files:

                if file == "":

                    QMessageBox.warning(self, "文件打开出错", "文件为空")

                    return

                try:

                    os.startfile(file)  # 在 Windows 上打开文件

                except:

                    QMessageBox.warning(self, "文件打开出错", "暂不支持这种类型的文件")

        elif action == action3:

            self.db.Delete(self.item.id)

            self.model.clear()

            for i in self.db.SelectALL():

                self.model.appendRow(i)  # 添加项目

    def eventFilter(self, source, event):

        if (

            event.type() == event.MouseButtonPress

            and source is self.ui.listView.viewport()

        ):

            index = self.ui.listView.indexAt(event.pos())

            if not index.isValid():

                self.clearPreviousFormats()

                if event.button() == Qt.LeftButton:

                    self.handle\_left\_click(event)

                elif event.button() == Qt.RightButton:

                    self.handle\_right\_click(event)

                return True

        return super().eventFilter(source, event)

    def handle\_left\_click(self, event):

        pass

    def handle\_right\_click(self, event):

        # 在右键点击时显示上下文菜单

        menu = QMenu(self)

        action1 = menu.addAction("insert")

        action2 = menu.addAction("topping √" if self.is\_on\_top else "topping")

        action3 = menu.addAction("startup √" if self.is\_startup else "startup")

        action4 = menu.addAction("about")

        action5 = menu.addAction("exit")

        # 显示菜单并获取所选的操作

        action = menu.exec\_(self.cursor().pos())

        # 根据所选的操作执行相应的逻辑

        if action == action1:

            formDialog = FormDialog(None)  # type: ignore

            if formDialog.exec\_() == QDialog.Accepted:

                form\_data = formDialog.get\_form\_data()

                if not isinstance(form\_data["id"], int):

                    return

                elif form\_data["important"] != "null" and not isinstance(

                    form\_data["important"], int

                ):

                    return

                new\_item = Entity(

                    id=form\_data["id"],

                    title=form\_data["title"],

                    describe=form\_data["describe"],

                    important=form\_data["important"],

                    start\_time=form\_data["start\_time"],

                    end\_time=form\_data["end\_time"],

                    file=form\_data["file"],

                )

                if self.db.Insert(new\_item):

                    self.model.clear()

                    for i in self.db.SelectALL():

                        self.model.appendRow(i)  # 添加项目

                else:

                    QMessageBox.warning(

                        self, "Insert new event error", "Id was repeat!!!"

                    )

        elif action == action2:

            if self.is\_on\_top:

                self.setWindowFlags(self.windowFlags() & ~Qt.WindowStaysOnTopHint)

            else:

                self.setWindowFlags(self.windowFlags() | Qt.WindowStaysOnTopHint)

            self.show()  # Update the window

            self.is\_on\_top = not self.is\_on\_top  # Toggle the state

        elif action == action3:

            if self.is\_startup:

                Startup.remove\_from\_startup(directory + "\StickyNotes.bat")

            else:

                Startup.add\_to\_startup(directory + "\StickyNotes.bat")

            self.is\_startup = not self.is\_startup  # Toggle the state

        elif action == action4:

            self.msg\_window = QWidget()

            self.msg\_window.setWindowTitle("About")

            self.msg\_window.setWindowFlags(Qt.Drawer)

            self.layout = QVBoxLayout()

            self.label = QLabel("版本: 1.0.2\n作者: cxlhyx")

            self.layout.addWidget(self.label)

            self.msg\_window.setLayout(self.layout)

            self.msg\_window.show()

        elif action == action5:

            QCoreApplication.quit()  # 退出应用

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    window = Main()

    window.ui.listView.viewport().installEventFilter(window)

    window.show()

    sys.exit(app.exec\_())

* 1. README.md

- 👋 Hi, I’m @货又星

- 👀 I’m interested in ...

- 🌱 I’m currently learning ...

- 💞 I’m looking to collaborate on ...

- 📫 How to reach me ...

  - [README 目录（持续更新中） 各种错误处理、爬虫实战及模板、百度智能云人脸识别、计算机视觉深度学习CNN图像识别与分类、PaddlePaddle自然语言处理知识图谱、GitHub、运维...](https://blog.csdn.net/muaamua/article/details/134426428?spm=1001.2014.3001.5502)

  - WeChat：1297767084

  - Email：cxlhyx1297767084@gmail.com

  - GitHub：[https://github.com/cxlhyx](https://github.com/cxlhyx)

    ![在这里插入图片描述](/qrcode.jpg)

- 😄 Pronouns: ...

- ⚡ Fun fact: ...

<!---

cxlhyx/cxlhyx is a ✨ special ✨ repository because its `README.md` (this file) appears on your GitHub profile.

You can click the Preview link to take a look at your changes.

--->

## 项目结构

```

D:.

│  Main.py                                 # 主函数，实现按键点击等功能

│  Ui.ui                                   # 使用Qt Designer设计的界面ui

│  Ui.py                                   # 根据界面ui生成的python代码

│  LineDelegate.py                         # 左侧event栏的自定义委托

│  Sql.py                                  # 数据：数据库增删改查、event实体

│  DetailBox.py                            # event详情盒：查看event、更新event

│  Startup.py                              # 开机自启动

│  imports.py                              # 导入需要的库

│  requirements.txt                        # 项目依赖

│  README.md

│  Database.db                             # sqlite数据库

│  StickyNotes.exe                         # 打包成的exe

│  StickyNotes.exe - 快捷方式.lnk           # exe的快捷方式，用户可能需要自己更改

│  qrcode.jpg                              # 作者wx

│

└─\_\_pycache\_\_                              # 运行缓存

        DetailBox.cpython-39.pyc

        imports.cpython-39.pyc

        LineDelegate.cpython-39.pyc

        Sql.cpython-39.pyc

        Startup.cpython-39.pyc

        Ui.cpython-39.pyc

```

## 使用

### 抓取

```

git clone git@github.com:cxlhyx/StickyNotes.git

```

### 安装依赖

```

pip install -r requirements.txt

```

### 运行

```

python Main.py

```

## 注意

- event的日期格式必须为"\*\*yyyy-MM-dd hh:mm:ss\*\*"，否则右侧日历表将无法对选中的event日期进行涂色。

- event的file为\*\*绝对路径\*\*。

  对于每个文件可以用**\*\*双引号\*\***也可以不用，对于多个文件中间需要用"; "隔开（**\*\*英文分号和空格\*\***），最后一个文件时不能再用"; "。

- 建议不要直接使用项目的exe，以免bug。先修改Main.py中第8行的directory = "D:\college\Project\StickyNotes"，改为exe即将存储的路径或直接改为项目克隆的路径，接着在克隆下来的目录下使用以下命令即可编译生成exe文件。StickyNotes.exe 将被生成在 dist 目录中。PyInstaller 还会创建 build 目录和 MyApp.spec 文件。build 目录包含构建过程中使用的中间文件，而 .spec 文件是构建过程的配置文件。如果不希望保留 build 目录和 .spec 文件，可以手动删除它们。

```

pip install pyinstaller

pyinstaller -F -w --name StickyNotes Main.py

```

**## release**

v1.0.0：2024-06-29 基础功能

v1.0.1：2024-06-29 新增打开多个文件的功能

v1.0.2: 2024-07-05 修复了开机自动启动event列表数据不显示的问题

**## 欢迎微信、邮箱、GitHub提出建议或问题**

* 1. Requirements.txt

PyQt5

* 1. Sql.py

from imports import \*

class Entity(QStandardItem):  # type: ignore

    def \_\_init\_\_(

        self,

        id,

        title=None,

        describe=None,

        important=-1,

        start\_time=None,

        end\_time=None,

        create\_time=None,

        file=None,

    ):

        super().\_\_init\_\_(str(id) + " " + str(title))

        assert id != None, "Id was needed!!!"

        if create\_time == None:

            # 获取当前日期和时间

            currentDateTime = QDateTime.currentDateTime()  # type: ignore

            # 将日期和时间格式化为字符串

            create\_time = currentDateTime.toString("yyyy-MM-dd hh:mm:ss")

        self.id = id

        self.title = title

        self.describe = describe

        self.important = important

        self.start\_time = start\_time

        self.end\_time = end\_time

        self.create\_time = create\_time

        self.file = file

    def \_\_str\_\_(self):

        return (

            "{"

            + "\n    id: "

            + str(self.id)

            + ",\n    title: "

            + self.title

            + ",\n    describe: "

            + self.describe

            + ",\n    important: "

            + str(self.important)

            + ",\n    start\_time: "

            + self.start\_time

            + ",\n    end\_time: "

            + self.end\_time

            + ",\n    creat\_time: "

            + self.create\_time

            + ",\n    file: "

            + self.file

            + "\n}"

        )

class Database:

    def \_\_init\_\_(self) -> None:

        # 初始化数据库连接

        self.initDatabase()

    def initDatabase(self):

        # 添加 SQLite 驱动

        db = QSqlDatabase.addDatabase("QSQLITE")  # type: ignore

        db.setDatabaseName("Database.db")  # 使用文件路径，也可以是内存中的数据库

        if not db.open():

            print("无法建立数据库连接！")

            return False

        # 创建表格

        query = QSqlQuery()  # type: ignore

        query.exec\_(

            """

            CREATE TABLE IF NOT EXISTS event

            (id INTEGER PRIMARY KEY AUTOINCREMENT,

            title TEXT,

            describe TEXT,

            important INTEGER,

            start\_time TEXT,

            end\_time TEXT,

            create\_time TEXT,

            file TEXT)

            """

        )

        """

        # 插入测试数据

        self.Insert(

            StickyNotesEntity(

                1,

                "测试",

                "用于测试的事件",

                1,

                "2024-06-25 23:00:32",

                "2024-06-25 23:00:32",

                None,

                "D:\college\Project\新建文件夹\StickyNotes.ui",

            )

        )

        """

        return True

    def Select(self, id):

        query = QSqlQuery()  # type: ignore

        query.exec\_(

            f"""

            SELECT \*

            FROM event

            WHERE id={id}

            """

        )

        if query.next():

            return Entity(

                query.value(0),

                query.value(1),

                query.value(2),

                query.value(3),

                query.value(4),

                query.value(5),

                query.value(6),

                query.value(7),

            )

        else:

            return None

    def SelectALL(self):

        # 查询数据库中的数据并输出

        query = QSqlQuery()  # type: ignore

        query.exec\_("SELECT \* FROM event")

        stickyNotesEntities = []

        while query.next():

            stickyNotesEntity = Entity(

                query.value(0),

                query.value(1),

                query.value(2),

                query.value(3),

                query.value(4),

                query.value(5),

                query.value(6),

                query.value(7),

            )

            stickyNotesEntities.append(stickyNotesEntity)

        return stickyNotesEntities

    def Insert(self, stickyNotesEntity):

        tmp = f"{stickyNotesEntity.id}"

        if stickyNotesEntity.title != "null":

            tmp += f", '{stickyNotesEntity.title}'"

        else:

            tmp += f", {stickyNotesEntity.title}"

        if stickyNotesEntity.describe != "null":

            tmp += f", '{stickyNotesEntity.describe}'"

        else:

            tmp += f", {stickyNotesEntity.describe}"

        tmp += f", {stickyNotesEntity.important}"

        if stickyNotesEntity.start\_time != "null":

            tmp += f", '{stickyNotesEntity.start\_time}'"

        else:

            tmp += f", {stickyNotesEntity.start\_time}"

        if stickyNotesEntity.end\_time != "null":

            tmp += f", '{stickyNotesEntity.end\_time}'"

        else:

            tmp += f", {stickyNotesEntity.end\_time}"

        tmp += f", '{stickyNotesEntity.create\_time}'"

        if stickyNotesEntity.file != "null":

            tmp += f", '{stickyNotesEntity.file}'"

        else:

            tmp += f", {stickyNotesEntity.file}"

        query = QSqlQuery()  # type: ignore

        return query.exec\_(

            f"""

            INSERT INTO event (id, title, describe, important, start\_time, end\_time, create\_time, file)

            VALUES ({tmp})

            """

        )

    def Delete(self, id):

        query = QSqlQuery()  # type: ignore

        return query.exec\_(

            f"""

            DELETE

            FROM event

            WHERE id={id}

            """

        )

    def Update(self, ID, \*\*kwargs):

        tmp = ""

        for key, value in kwargs.items():

            if key == "id" or key == "important" or value == "null":

                tmp += f"{key}={value},"

            else:

                tmp += f"{key}='{value}',"

        query = QSqlQuery()  # type: ignore

        return query.exec\_(

            f"""

            UPDATE event

            SET {tmp[:-1]}

            WHERE id={ID}

            """

        )

if \_\_name\_\_ == "\_\_main\_\_":

    sql = Database()

    sql.Select(231)

* 1. Startup.py

from imports import \*

class Startup:

    def add\_to\_startup(file\_path=""):

        if not file\_path:

            file\_path = os.path.realpath(\_\_file\_\_)  # type: ignore

        file\_name = os.path.basename(file\_path)  # type: ignore

        key = r"Software\Microsoft\Windows\CurrentVersion\Run"

        try:

            registry\_key = reg.OpenKey(reg.HKEY\_CURRENT\_USER, key, 0, reg.KEY\_WRITE)  # type: ignore

            reg.SetValueEx(registry\_key, file\_name, 0, reg.REG\_SZ, file\_path)  # type: ignore

            reg.CloseKey(registry\_key)  # type: ignore

            print(f"Added {file\_name} to startup")

        except Exception as e:

            print(f"Failed to add to startup: {e}")

    def remove\_from\_startup(file\_path=""):

        if not file\_path:

            file\_path = os.path.realpath(\_\_file\_\_)  # type: ignore

        file\_name = os.path.basename(file\_path)  # type: ignore

        key = r"Software\Microsoft\Windows\CurrentVersion\Run"

        try:

            registry\_key = reg.OpenKey(reg.HKEY\_CURRENT\_USER, key, 0, reg.KEY\_WRITE)  # type: ignore

            reg.DeleteValue(registry\_key, file\_name)  # type: ignore

            reg.CloseKey(registry\_key)  # type: ignore

            print(f"Removed {file\_name} from startup")

        except FileNotFoundError:

            print(f"{file\_name} not found in startup")

        except Exception as e:

            print(f"Failed to remove from startup: {e}")

    def is\_in\_startup(file\_path=""):

        if not file\_path:

            file\_path = os.path.realpath(\_\_file\_\_)  # type: ignore

        file\_name = os.path.basename(file\_path)  # type: ignore

        key = r"Software\Microsoft\Windows\CurrentVersion\Run"

        try:

            registry\_key = reg.OpenKey(reg.HKEY\_CURRENT\_USER, key, 0, reg.KEY\_READ)  # type: ignore

            value, reg\_type = reg.QueryValueEx(registry\_key, file\_name)  # type: ignore

            reg.CloseKey(registry\_key)  # type: ignore

            if value == file\_path:

                return True

            else:

                print(

                    f"Registry value does not match the expected path: {value} != {file\_path}"

                )

                return False

        except FileNotFoundError:

            print(f"Registry entry not found for: {file\_name}")

            return False

        except Exception as e:

            print(f"Failed to check registry: {e}")

            return False

    def create\_bat\_file(directory):

        # 定义批处理文件中需要包含的命令

        bat\_commands = [

            "cd " + directory,

            directory[0:2],

            'start "" "StickyNotes.exe"',

            "exit",

        ]

        with open(directory + "\StickyNotes.bat", "w") as bat\_file:

            for command in bat\_commands:

                bat\_file.write(command + "\n")

if \_\_name\_\_ == "\_\_main\_\_":

    # Pass the script path to the function

    Startup.add\_to\_startup("D:\college\Project\新建文件夹\StickyNotesMain.py")

    Startup.is\_in\_startup("D:\college\Project\新建文件夹\StickyNotesMain.py")

    # Pass the script name (or the name used to add the startup entry) to the function

    Startup.remove\_from\_startup("D:\college\Project\新建文件夹\StickyNotesMain.py")

    Startup.is\_in\_startup("D:\college\Project\新建文件夹\StickyNotesMain.py")

* 1. StickyNotes.bat

cd D:\college\Project\StickyNotes

D:

start "" "StickyNotes.exe"

exit

* 1. Ui.py

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'StickyNotes.ui'

#

# Created by: PyQt5 UI code generator 5.15.9

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again.  Do not edit this file unless you know what you are doing.

from imports import \*

class Ui(object):

    def setupUi(self, Frame):

        # 设置窗口透明度

        Frame.setWindowOpacity(0.5)

        # 设置窗口为无边框

        Frame.setWindowFlags(Qt.FramelessWindowHint | Qt.Tool)  # type: ignore

        # 设置窗口位置

        self.fixed\_position = QPoint(1160, 0)  # type: ignore # 固定位置

        Frame.move(self.fixed\_position)

        Frame.setObjectName("Frame")

        Frame.resize(761, 574)

        self.calendarWidget = QCalendarWidget(Frame)  # type: ignore

        self.calendarWidget.setGeometry(QRect(250, 0, 511, 371))  # type: ignore

        self.calendarWidget.setStyleSheet(

            "/\* 设置日历工具按钮（如上一个月、下一个月按钮）的样式 \*/\n"

            "                    QCalendarWidget QToolButton {\n"

            "                    height: ; /\* 按钮高度 \*/\n"

            "                    width: ; /\* 按钮宽度 \*/\n"

            "                    color: ; /\* 按钮文字颜色 \*/\n"

            "                    font-size: ; /\* 按钮文字大小 \*/\n"

            "                    icon-size: , ; /\* 按钮图标大小 \*/\n"

            "                    background-color: ; /\* 按钮背景颜色 \*/\n"

            "                    border: ; /\* 按钮无边框 \*/\n"

            "                    margin: ; /\* 按钮外边距 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置工具按钮悬停时的背景颜色 \*/\n"

            "                    QCalendarWidget QToolButton:hover {\n"

            "                    background-color: #87CEEB; /\* 悬停时的背景颜色 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置工具按钮菜单指示器的位置 \*/\n"

            "                    QCalendarWidget QToolButton::menu-indicator {\n"

            "                    subcontrol-position: bottom center; /\* 菜单指示器的位置 \*/\n"

            "                    subcontrol-origin: padding; /\* 菜单指示器的原点 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置年份选择框（QSpinBox）的样式 \*/\n"

            "                    QCalendarWidget QSpinBox {\n"

            "                    width: 100px; /\* 选择框宽度 \*/\n"

            "                    font-size: 18px; /\* 字体大小 \*/\n"

            "                    color: white; /\* 字体颜色 \*/\n"

            "                    background: transparent; /\* 背景透明 \*/\n"

            "                    selection-background-color: #1E90FF; /\* 选中时的背景颜色 \*/\n"

            "                    selection-color: white; /\* 选中时的字体颜色 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置年份选择框的上按钮样式 \*/\n"

            "                    QCalendarWidget QSpinBox::up-button {\n"

            "                    subcontrol-origin: border; /\* 上按钮的原点 \*/\n"

            "                    subcontrol-position: top right; /\* 上按钮的位置 \*/\n"

            "                    width: 35px; /\* 上按钮宽度 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置年份选择框的下按钮样式 \*/\n"

            "                    QCalendarWidget QSpinBox::down-button {\n"

            "                    subcontrol-origin: border; /\* 下按钮的原点 \*/\n"

            "                    subcontrol-position: bottom right; /\* 下按钮的位置 \*/\n"

            "                    width: 35px; /\* 下按钮宽度 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置年份选择框的上箭头大小 \*/\n"

            "                    QCalendarWidget QSpinBox::up-arrow {\n"

            "                    width: 20px; /\* 上箭头宽度 \*/\n"

            "                    height: 20px; /\* 上箭头高度 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置年份选择框的下箭头大小 \*/\n"

            "                    QCalendarWidget QSpinBox::down-arrow {\n"

            "                    width: 20px; /\* 下箭头宽度 \*/\n"

            "                    height: 20px; /\* 下箭头高度 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置日历小部件的交替背景颜色 \*/\n"

            "                    QCalendarWidget QWidget {\n"

            "                    alternate-background-color: ; /\* 交替背景颜色 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置日历视图中启用项的样式 \*/\n"

            "                    QCalendarWidget QAbstractItemView:enabled {\n"

            "                    font-size: 18px; /\* 字体大小 \*/\n"

            "                    color: ; /\* 字体颜色 \*/\n"

            "                    background-color: ; /\* 背景颜色 \*/\n"

            "                    selection-background-color: ; /\* 选中时的背景颜色 \*/\n"

            "                    selection-color: ; /\* 选中时的字体颜色 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置日历视图中禁用项的样式 \*/\n"

            "                    QCalendarWidget QAbstractItemView:disabled {\n"

            "                    color: ; /\* 禁用项的字体颜色 \*/\n"

            "                    }\n"

            "\n"

            "                    /\* 设置日历导航栏的背景颜色 \*/\n"

            "                    QCalendarWidget QWidget#qt\_calendar\_navigationbar {\n"

            "                    background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:0, stop:0\n"

            "                    rgba(0, 0, 0, 255), stop:1 rgba(255, 255, 255, 255)); /\* 导航栏背景颜色 \*/\n"

            "                    }\n"

            "                "

        )

        self.calendarWidget.setFirstDayOfWeek(Qt.Sunday)  # type: ignore

        self.calendarWidget.setObjectName("calendarWidget")

        self.lcdNumber = QLCDNumber(Frame)  # type: ignore

        self.lcdNumber.setGeometry(QRect(240, 370, 521, 211))  # type: ignore

        self.lcdNumber.setObjectName("lcdNumber")

        self.listView = QListView(Frame)  # type: ignore

        self.listView.setGeometry(QRect(0, 0, 251, 581))  # type: ignore

        self.listView.setObjectName("listView")

        self.retranslateUi(Frame)

        QMetaObject.connectSlotsByName(Frame)  # type: ignore

    def retranslateUi(self, Frame):

        \_translate = QCoreApplication.translate  # type: ignore

        Frame.setWindowTitle(\_translate("Frame", "Frame"))

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QApplication(sys.argv)  # type: ignore

    Frame = QFrame()  # type: ignore

    ui = Ui()

    ui.setupUi(Frame)

    Frame.show()

    sys.exit(app.exec\_())

* 1. Ui.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

 <class>Frame</class>

 <widget class="QFrame" name="Frame">

  <property name="geometry">

   <rect>

    <x>0</x>

    <y>0</y>

    <width>761</width>

    <height>574</height>

   </rect>

  </property>

  <property name="windowTitle">

   <string>Frame</string>

  </property>

  <widget class="QCalendarWidget" name="calendarWidget">

   <property name="geometry">

    <rect>

     <x>250</x>

     <y>0</y>

     <width>511</width>

     <height>371</height>

    </rect>

   </property>

   <property name="styleSheet">

    <string notr="true">/\* 设置日历工具按钮（如上一个月、下一个月按钮）的样式 \*/

                    QCalendarWidget QToolButton {

                    height: ; /\* 按钮高度 \*/

                    width: ; /\* 按钮宽度 \*/

                    color: ; /\* 按钮文字颜色 \*/

                    font-size: ; /\* 按钮文字大小 \*/

                    icon-size: , ; /\* 按钮图标大小 \*/

                    background-color: ; /\* 按钮背景颜色 \*/

                    border: ; /\* 按钮无边框 \*/

                    margin: ; /\* 按钮外边距 \*/

                    }

                    /\* 设置工具按钮悬停时的背景颜色 \*/

                    QCalendarWidget QToolButton:hover {

                    background-color: #87CEEB; /\* 悬停时的背景颜色 \*/

                    }

                    /\* 设置工具按钮菜单指示器的位置 \*/

                    QCalendarWidget QToolButton::menu-indicator {

                    subcontrol-position: bottom center; /\* 菜单指示器的位置 \*/

                    subcontrol-origin: padding; /\* 菜单指示器的原点 \*/

                    }

                    /\* 设置年份选择框（QSpinBox）的样式 \*/

                    QCalendarWidget QSpinBox {

                    width: 100px; /\* 选择框宽度 \*/

                    font-size: 18px; /\* 字体大小 \*/

                    color: white; /\* 字体颜色 \*/

                    background: transparent; /\* 背景透明 \*/

                    selection-background-color: #1E90FF; /\* 选中时的背景颜色 \*/

                    selection-color: white; /\* 选中时的字体颜色 \*/

                    }

                    /\* 设置年份选择框的上按钮样式 \*/

                    QCalendarWidget QSpinBox::up-button {

                    subcontrol-origin: border; /\* 上按钮的原点 \*/

                    subcontrol-position: top right; /\* 上按钮的位置 \*/

                    width: 35px; /\* 上按钮宽度 \*/

                    }

                    /\* 设置年份选择框的下按钮样式 \*/

                    QCalendarWidget QSpinBox::down-button {

                    subcontrol-origin: border; /\* 下按钮的原点 \*/

                    subcontrol-position: bottom right; /\* 下按钮的位置 \*/

                    width: 35px; /\* 下按钮宽度 \*/

                    }

                    /\* 设置年份选择框的上箭头大小 \*/

                    QCalendarWidget QSpinBox::up-arrow {

                    width: 20px; /\* 上箭头宽度 \*/

                    height: 20px; /\* 上箭头高度 \*/

                    }

                    /\* 设置年份选择框的下箭头大小 \*/

                    QCalendarWidget QSpinBox::down-arrow {

                    width: 20px; /\* 下箭头宽度 \*/

                    height: 20px; /\* 下箭头高度 \*/

                    }

                    /\* 设置日历小部件的交替背景颜色 \*/

                    QCalendarWidget QWidget {

                    alternate-background-color: ; /\* 交替背景颜色 \*/

                    }

                    /\* 设置日历视图中启用项的样式 \*/

                    QCalendarWidget QAbstractItemView:enabled {

                    font-size: 18px; /\* 字体大小 \*/

                    color: ; /\* 字体颜色 \*/

                    background-color: ; /\* 背景颜色 \*/

                    selection-background-color: ; /\* 选中时的背景颜色 \*/

                    selection-color: ; /\* 选中时的字体颜色 \*/

                    }

                    /\* 设置日历视图中禁用项的样式 \*/

                    QCalendarWidget QAbstractItemView:disabled {

                    color: ; /\* 禁用项的字体颜色 \*/

                    }

                    /\* 设置日历导航栏的背景颜色 \*/

                    QCalendarWidget QWidget#qt\_calendar\_navigationbar {

                    background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:0, stop:0

                    rgba(0, 0, 0, 255), stop:1 rgba(255, 255, 255, 255)); /\* 导航栏背景颜色 \*/

                    }

                </string>

   </property>

   <property name="firstDayOfWeek">

    <enum>Qt::Sunday</enum>

   </property>

  </widget>

  <widget class="QLCDNumber" name="lcdNumber">

   <property name="geometry">

    <rect>

     <x>250</x>

     <y>370</y>

     <width>511</width>

     <height>211</height>

    </rect>

   </property>

  </widget>

  <widget class="QListView" name="listView">

   <property name="geometry">

    <rect>

     <x>0</x>

     <y>0</y>

     <width>251</width>

     <height>581</height>

    </rect>

   </property>

  </widget>

 </widget>

 <resources/>

 <connections/>

</ui>