23-24春程设大作业——晨坝笔记本项目作业报告

组号: 91 / 组名: 刘老师班第四组

组员: 孙坝 (2100018731) / 郑晨峰 (2300094820)

1．程序简介

**晨坝笔记本（ChainBat Note, alpha1）**是用Qt框架开发的基于.html和纯文本格式的文本编辑器。中文名称取项目组两位成员名字各一字，英文名ChainBat是其谐音。该软件提供了最常见和基础的文本编辑体验，可以用于日常笔记、备忘录和文档编写。支持多窗口，主要功能位于4个菜单栏下。

2. 功能介绍

菜单栏下主要功能包括：

文件管理：新建、打开、保存、另存为、删除5类文件操作，以及.txt和.html格式的转换行为。

文本格式：基于.html设置简单的文本格式，字体、字号、颜色、样式（斜体、粗体、下划线、删除线）各一工具栏，另有快捷功能（显示/隐藏所有工具栏、清除格式）。

模式切换：切换编辑器的模式，有2种，编辑模式和查看模式。

辅助功能：包括3种，分别是最近打开的文件、全文查找和替换、”Chain A Bat”随机修改文本玩笑功能。

功能的快捷键在菜单里给出。

2-1. 文件管理（File）

New: 创建新的临时文档，在新的窗口中显示。

Open: 打开已有的文件，支持.html和.txt格式，在新窗口中显示文件内容。

Save: 保存当前文档，如果是临时文档则执行另存为。

Save As: 另存为，将当前文档保存为新文件，并转到新文件的编辑上。

Delete: 快捷删除从目录里选择的若干个文件。

格式说明: 支持.txt/.html，优先支持.html格式，如果需要可以打开或保存为.txt格式。带Format的文件内容存为.txt时，存储内容是清除全部Format后的纯文本。（因此编辑器上的内容有无Format是不影响.txt实现的，总是把文本视为有Format，除非在刚刚打开.txt文件还没有修改Format时，这个有无Format的信息在底部状态栏左二显示。）

2-2. 文本格式（Format）

编辑器自动支持基本的剪贴板交互和撤销回退编辑操作。

右侧有4个控制Format的工具栏可以修改框选文本的格式。

Font: 自选字体。

Size: 自选字号（Point Size）。

Color: 自选颜色。

Style: 自选有无样式，即斜体、粗体、下划线和删除线。

菜单内4个按钮控制对应工具栏的显隐，还有3个快捷按钮。

Show/Hide All Docks: 显示或隐藏所有格式工具栏。

Clear Formatting: 清除选中文本的所有格式为默认值。

2-3. 模式切换（Mode）

Write Mode: 切换到编辑模式，允许用户编辑文本。

View Mode: 切换到查看模式，只读，不允许编辑。

模式信息在底部状态栏左侧显示。

2-4. 辅助功能（Auxilliary）

Recent Files: 显示并全局维护最近打开的文件列表，方便快速打开。最多可加入15项。

Find: 在全文查找字符串，并加下划虚线显示。

Replace: 在全文先查找再替换字符串。

Chain A Bat: 打开ChainBat玩笑对话框，Chain A Bat的意思是就是把一种Bat效果应用给当前窗口的全文。Bat有五种，KuXuanBat（随机改全文Format）、YouMoBat（随机插入和打乱空格和符号）、ChouXiangBat（Base64）、TangShiBat（文本换成Emoji）、QiMoBat（使编辑器崩溃）。

最后，主窗口包含标题信息和状态栏，用于显示当前编辑的文件名。

3. 项目各模块与类设计细节

一共五个类：MainWindow（主要窗口）、WindowManager（全局窗口管理器）、FileManager（文件操作执行器）、TextEditor（编辑操作执行器）、ChainBatDialog（玩笑功能弹窗）。

3-1. MainWindow类

项目的主窗口；定义每个编辑窗口（窗口之间互相平行）的组件，包含顶部菜单栏、右侧格式工具栏和底部状态栏；负责与大部分专门功能的交互逻辑。

成员：

// 基本组件指针

FileManager \*fileManager; // 文件管理器指针

TextEditor \*textEditor; // 文本编辑器指针

QStatusBar \*statusBar; // 状态栏指针

QLabel \*modeLabel; // 状态栏模式标签

QLabel \*formatStatusLabel; // 状态栏格式状态标签

// 标题显示参数

QString currentFileName; // 此文件名

bool isModified = false; // 文本已修改标志

// 辅助功能菜单指针及参数

QMenu \*auxiliaryMenu; // 辅助功能菜单

QAction \*recentFilesSubMenuAction; // 最近文件子菜单的展开按钮

QMenu \*recentFilesSubMenu; // 最近文件子菜单

// 格式菜单、工具栏和动作

QMenu \*formatMenu; // 格式菜单

QDockWidget \*fontDock; // 字体工具栏

...

QAction \*toggleFontDockAction;

...

// 文件动作指针

QAction \*newAct;

...

// 辅助功能动作指针

QAction \*editModeAction; // 切换编辑模式动作

...

3-1-1. 初始化函数

explicit MainWindow(QWidget \*parent = nullptr, const QString &fileName = "");

void createActions(); // 创建各种动作（文件打开、模式切换等）

void createMenus(); // 创建菜单（并绑定动作按钮）

void createAuxiliaryMenu(); // 创建辅助功能菜单内容

// etc

以构造函数为主函数，依次调用下面各个初始化函数。构造函数逻辑包括三部分：

A. 窗口创建时设置基本组件和初始化各个标志。

B. 实现打开文件功能的一部分逻辑（前一半在MainWindow::openFile函数里），因为窗口是由新建文件或打开文件操作唤出的。

C. 和窗口总管理类WindowManager交互，因为分配了新窗口。

3-1-2. 文件操作

包含几个槽函数。

void newFile(); // 新建文件槽函数

void openFile(); // 打开文件槽函数

void saveFile(); // 保存文件槽函数

void saveFileAs(); // 另存为文件槽函数

void deleteFile(); // 删除文件槽函数

其中openFile和saveFileAs要与WindowManager交互，因为新增或修改了窗口的分配。

一些细节包括：

文件操作函数都要更新标题显示的文字。

saveFile会更新isModified参数。

saveFileAs涉及生成一个最小可用的缺省文件名。

deleteFile会删除文件，但不关闭窗口。

3-1-3. 格式操作

4个Format控制工具(Dock)在初始化里创建，然后用槽函数绑到Format菜单上。

void createFormatDockWidgets(); // 定义4个工具栏的外形和结构

void toggleFontDock(); // 显示/隐藏字体工具栏槽函数

void toggleSizeDock(); // 显示/隐藏大小工具栏槽函数

void toggleColorDock(); // 显示/隐藏颜色工具栏槽函数

void toggleStyleDock(); // 显示/隐藏样式工具栏槽函数

快捷操作在菜单初始化函数定义。

void createFormatMenu(); // 创建格式菜单内容

由于操作比较低级繁多，这些动作都没有在类里声明槽函数，而是直接用lambda定义再connect绑定，比如：

connect(sizeSpinBox, QOverload<int>::of(&QSpinBox::valueChanged), [=](int size) {

textEditor->setFontPointSize(size);

updateFormatStatus("Yes");

});

这里FontDock存在更新会覆盖其它值的BUG，暂未找到修正方法。

3-1-4. 模式操作

只有两个动作，绑定了给TextEditor类的槽函数，设置只读与否。

editModeAction = new QAction(tr("Edit Mode"), this); // 设置为编辑模式动作

connect(editModeAction, &QAction::triggered, textEditor, &TextEditor::setEditMode);

editModeAction->setShortcut(QKeySequence("Ctrl+W"));

3-1-5. 辅助功能操作

三个功能各交给一个不同的类完成主要功能。

最近访问文件通过openFile函数上交给WindowManager处理了一部分，因为最近访问文件表需要全局维护和更新到全部窗口。本类只实现窗口内的更新/打开表的操作。

void MainWindow::openFile() {

...

// 更新最近打开列表

WindowManager::instance().addRecentFile(fileName);

...

}

void openRecentFile(); // 在辅助功能最近文件子菜单打开最近文件

void updateRecentFilesMenu(); // 更新最近文件子菜单

查找/替换是用本类的槽函数实现大部分逻辑，中间调用了TextEditor的方法去做高亮和替换。

void findText(); // 查找文本槽函数

void replaceText(); // 替换文本槽函数

玩笑功能只用槽函数新建一个ChainBatDialog实例，在后者自己的类里面完成独立的功能。

void MainWindow::openChainBatDialog() {

ChainBatDialog \*dialog = new ChainBatDialog(this);

dialog->exec();

delete dialog;

}

3-1-6. 状态栏和标题信息维护

各自有一个更新函数，在其它方法里反复调用。

void updateWindowTitle(); // 更新窗口标题显示, 与isModified值有关, 需要响应键入事件

void updateFormatStatus(const QString &status); // 更新Format状态显示, 在文件操作里更新

void updateModeLabel(const QString &mode); // 更新编辑状态槽函数, 与TextEditor绑定了

3-1-7. 关闭和销毁

关闭事件进行了自定义，一是反馈给WindowManager，二是弹出确认保存窗口。

~MainWindow();

void closeEvent(QCloseEvent \*event) override;

3-2. WindowManager类

全局单例类，负责管理所有打开的文件窗口，确保每个文件只有一个窗口实例，并提供统一的窗口管理接口。使用了QMap记录有名文件窗口，使用QSet记录无名临时文件窗口。此外还维护一些全局值：1) 维护最近打开文件的列表；2) 维护上次保存文件的目录。

static WindowManager& instance();

void createWindow(const QString &fileName = "");

bool hasWindowRecord(const QString &fileName) const;

void addWindowRecord(const QString &fileName, MainWindow \*window);

void removeWindowRecord(const QString &fileNam, MainWindow \*window);

bool isFirstLaunch = true;

QMap<QString, MainWindow\*> windows; // 有名文件的窗口列表; 键: 文件名; 值: 窗口指针

QSet<MainWindow\*> anonymousFileWindows; // 新建无名文件的窗口列表

QString lastSaveDir = ""; // 上次保存的目录

QStringList recentFiles; // 最近打开的文件列表

添加新窗口时，检查文件名是否已存在于QMap中，避免重复打开。关闭窗口时，从QMap或QSet中移除对应的窗口记录。

3-3. FileManager类

负责文件创建、打开、保存和删除的具体处理。较基类改动较少。

bool createFile(const QString &fileName); // 创建文件

bool openFile(const QString &fileName); // 打开文件

bool saveFile(const QString &fileName, const QString &content); // 保存文件

bool deleteFile(const QString &fileName); // 删除文件

QString getFileContent() const; // 获取文件内容

QString getCurrentFile() const; // 获取当前文件名

3-4. TextEditor类

继承自QTextEdit，提供文本编辑和显示功能，基本没有新增改动，只加入了控制模式切换槽函数。

public slots:

void setEditMode(); // 设置编辑模式

void setViewMode(); // 设置阅览模式

signals:

void modeChanged(const QString &mode); // 模式更改信号

private:

QString currentMode; // 当前模式

3-5. ChainBatDialog类

ChainBatDialog类是一个简单的小游戏界面，通过QDialog显示，提供多个“Bat”功能选项。使用QVBoxLayout和QGroupBox组织界面组件。根据用户选择的Bat功能，随机选择一个并应用到当前文档。

// 界面组件

QGroupBox \*chainGroupBox;

QGroupBox \*batGroupBox;

QTextEdit \*descriptionText;

// 功能按钮

QPushButton \*chainBatButton;

// Bat 功能选择

QCheckBox \*chaosBatCheck;

...

private slots:

void chainBatClicked();

在最后的槽函数里用条件分支实现全部Bat的逻辑，其中透过指针访问了MainWindow和TextEditor。

4. 小组成员分工情况

本项目由两位成员共同完成：

郑晨峰(缓考)：编写了项目的初稿框架代码，负责了最终演示视频制作。

孙坝：提出项目的初步构想，完成了后续的大部分功能的开发工作。

5. 项目总结与反思

在完成晨坝笔记本项目的过程中，我们收获了许多经验，但也遇到了一些挑战，以下是我们的总结与反思：

成功之处：合理的类设计(尤其是WindowManager)和界面组件布局设计。

需要反思改进之处：

1) 字体和格式：由于对font和format的格式研究不够，导致在实现Format设置时耗时过长，还遇到了不知如何修复的unintended behavior。

2) 文件和标记语言知识不足：由于对文件操作和标记语言的知识水平有限，一些初版设想的高级功能如插入图片和锚点未能成功实现。

3) 模块化编程经验：由于小组编程经验不多，一些功能的归类和负责还是有些杂或不一致。

4) 用户体验：在交互细节方面还有提升空间，如更流畅的多窗口切换。

通过这次程设项目开发，我们不仅加深了对Qt框架和C++编程的理解，也认识到在软件开发中深入学习和不断实践的重要性。未来我们将继续提升技术水平，争取在更多的项目中取得更大的进步。