

**硕士研究生**

**《高级程序设计》期末项目报告**

**任课教师 刘犇\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**项目名称 基于 MFC Doc/View 的三层架构 MDI 记事本系统设计与实现**

**学生姓名 徐熙涛\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**学生学号 20250313017Z**\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**二级学院 信息工程学院\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**填表日期 2025 年 12 月 24 日**

**北京印刷学院信息工程学院**

**基于 MFC Doc/View 的三层架构 MDI 记事本系统设计与实现**

目录

[摘要： 1](#_Toc31625)

[1 引言 1](#_Toc5468)

[2 需求分析与教学目标映射 2](#_Toc19891)

[3 总体架构设计 3](#_Toc25325)

[3.1 Doc/View 与 MDI 组织方式 4](#_Toc14705)

[3.2 三层架构的依赖方向与模块划分 4](#_Toc16512)

[4 关键模块设计与实现 5](#_Toc25583)

[4.1 设计模式：Factory、Strategy、Observer 的工程落地 5](#_Toc1658)

[4.2 消息映射、命令路由与 UI 状态同步 5](#_Toc5190)

[4.3 撤销/重做：命令栈与复合命令 6](#_Toc30409)

[4.4 正则搜索替换：参数化与错误处理 6](#_Toc9975)

[4.5 行号显示与滚动同步：自绘视图的关键点 6](#_Toc23799)

[4.6 错误处理：RAII、异常弹窗与日志 6](#_Toc15949)

[4.7 配置管理器 7](#_Toc19325)

[4.8 可测试逻辑模块 8](#_Toc7881)

[5.详细设计与实现 8](#_Toc18684)

[5.1 文件格式设计 8](#_Toc24588)

[5.1.1 纯文本格式 8](#_Toc5137)

[5.1.2 MyNote自定义格式 8](#_Toc7634)

[5.2编辑功能实现 9](#_Toc6867)

[5.2.1 撤销/重做机制 9](#_Toc29175)

[5.2.2 查找替换功能 10](#_Toc27457)

[5.2.3 行号显示 11](#_Toc1968)

[5.3主题切换实现 12](#_Toc19500)

[5.4加密模块实现 13](#_Toc22860)

[5.4.1 SHA-1哈希计算 13](#_Toc27232)

[5.4.2 AES-CBC加密 14](#_Toc30012)

[6.单元测试 14](#_Toc9234)

[6.1测试框架 14](#_Toc19516)

[6.2关键测试用例 15](#_Toc3308)

[6.3测试结果 16](#_Toc22984)

[7.自定义 mynote 文件格式与“抗 AI”安全机制 18](#_Toc22302)

[7.1 文件结构设计 18](#_Toc16715)

[7.2 摘要与加密过程的公式化描述 19](#_Toc29285)

[8 代码质量保障：评审、测试与安全性讨论 19](#_Toc3493)

[8.1 两轮同伴评审的走查清单 19](#_Toc30699)

[8.2 单元测试与覆盖率达成策略 19](#_Toc28153)

[9 AI 使用日志与学习反思（AI 素养） 20](#_Toc29279)

[9.1 日志结构：Prompt—Response—Diff 20](#_Toc25065)

[9.2 AI 素养四要素在本项目中的体现 20](#_Toc19401)

[10 结论与展望 20](#_Toc6306)

[参考文献： 21](#_Toc14039)

[同伴评议签字： 22](#_Toc8558)

# 摘要：

本项目基于 Microsoft Foundation Classes（MFC）文档/视图（Doc/View）模式完成一个多文档界面（MDI）记事本系统，用于训练学生快速搭建 Windows 桌面程序框架、在既有架构上进行迭代开发、以及在工程约束下实现可维护与可扩展的软件架构。系统采用“三层架构（表示层—应用层—基础设施层）+可测试业务逻辑”的总体方案：表示层负责窗口、菜单、工具栏与自绘视图；应用层负责消息映射驱动的命令路由、文档状态管理与跨模块协作；基础设施层提供 RAII 资源管理、文件 I/O、配置持久化、以及加密与摘要等通用能力。功能层面，系统实现多文档管理（标题栏显示“文件名—项目名”）、撤销/重做、正则搜索与替换、行号显示与滚动同步、暗/亮双主题切换、以及双格式文件（纯文本 \*.txt 与自定义富文本 \*.mynote）。为满足课程“抗 AI”要求，自定义 mynote 文件在文件头写入学生学号 <STUDENT\_ID>，并在文档末尾附加经 AES-CBC 加密的 SHA-1 摘要，同时将密钥 <SECRET\_KEY> 与 IV <IV> 以占位符形式呈现，强制学生手工替换以确保提交唯一性。质量保障方面，项目引入单元测试框架（GoogleTest/Catch2），对文件读写、主题切换与行号同步等关键路径进行自动化测试，并通过两轮同伴评审与教师点评聚焦资源泄漏、异常安全与潜在的线程安全隐患。本文给出需求分析、架构设计、关键模块实现与测试评估的完整论述，并在结尾提供 AI 使用日志制度下的学习反思，总结 AI 在提高开发效率方面的价值及其在安全、正确性与可解释性方面的局限。

关键词：MFC；Doc/View；MDI；三层架构；RAII；Factory；Observer；Strategy；单元测试；AI 素养

# 1 引言

在传统桌面软件开发中，MFC 依旧广泛用于制造业、医疗信息化、测控与政企内网系统。其核心优势在于成熟稳定的消息循环与窗口框架、与 Win32 API 的紧密集成、以及大量历史工程资产。教学层面，MFC 也是训练“架构先行、按框架开发”的典型平台：只要掌握 Doc/View、消息映射与 MDI 组织方式，学生即可在统一骨架上快速扩展功能。

本项目的教学目标分为四类：目标1强调对整体架构的快速掌握与独立搭建完整桌面程序框架；目标2强调软件架构设计与实现能力，要求在 MFC 场景下合理使用 Factory、Observer、Strategy 等设计模式，并结合 RAII 与智能指针构建可维护代码库；目标3与目标4强调 AI 使用规范与 AI 素养，要求提交AI 使用日志并接受人工审核，开展同伴评审与教师点评，讨论技术伦理与学术诚信。本文以项目一为对象，给出从需求到实现的工程化写作。

# 2 需求分析与教学目标映射

项目功能需求（F-01 至 F-08）围绕 MDI 框架、多文档管理、编辑能力、主题切换、双格式文件、异常处理与单元测试展开，并明确约束“不得使用 CEditView，仅允许 CView 或 CScrollView”。此外，抗 AI 需求要求 mynote 文件头保存学号且在尾部写入加密摘要，并以占位符形式呈现学号、密钥与 IV，使得生成式 AI 难以“一键生成”唯一可提交答案。为了保证验收可操作，本文将需求映射到教学目标与可测用例。

**表1 功能需求与教学目标映射**

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **功能点** | **关键实现点** | **对应教学目标** |
| F-01 | MDI 框架 | CMultiDocTemplate 组织文档；主/子框架窗口 | 目标1 |
| F-02 | 多文档管理 | 标题栏“文件名—项目名”；最近文件与窗口切换 | 目标1 |
| F-03 | 编辑功能 | 撤销/重做；Regex 搜索替换；行号显示 | 目标1、2 |
| F-04 | 主题切换 | 暗/亮两套主题策略；全局刷新与持久化 | 目标2 |
| F-05 | 双格式文件 | txt 与 mynote；工厂选择序列化策略 | 目标1、2 |
| F-06 | 错误处理 | RAII 包装 CFile/CArchive；异常弹窗与日志 | 目标2、3 |
| F-07 | 单元测试 | GoogleTest/Catch2；覆盖率≥70%（I/O 等） | 目标2、3 |
| F-08 | 不使用 CEditView | CView/CScrollView 自绘文本与输入处理 | 目标1 |

# 3 总体架构设计

系统采用“三层架构 + MFC Doc/View + 可测试逻辑”的综合方案。三层架构并非与 MFC 框架对立，而是对工程组织与依赖方向的约束：表示层只处理交互与绘制，不直接做复杂业务；应用层编排用例与跨模块协作；基础设施层提供通用能力并以接口形式被上层调用。Doc/View 用于管理“文档状态—视图呈现”的一致性：文档维护文本模型与序列化，视图负责自绘与交互，二者通过 UpdateAllViews 与消息映射解耦。

为满足教学目标1的“快速搭建完整桌面框架”，系统以 CMultiDocTemplate 注册文档、框架与视图类；CMainFrame 管理菜单、工具栏与状态栏；CChildFrame 作为每个文档窗口容器，实现多文档并行编辑。图1给出分层与协作关系，图2给出MFC架构类图。

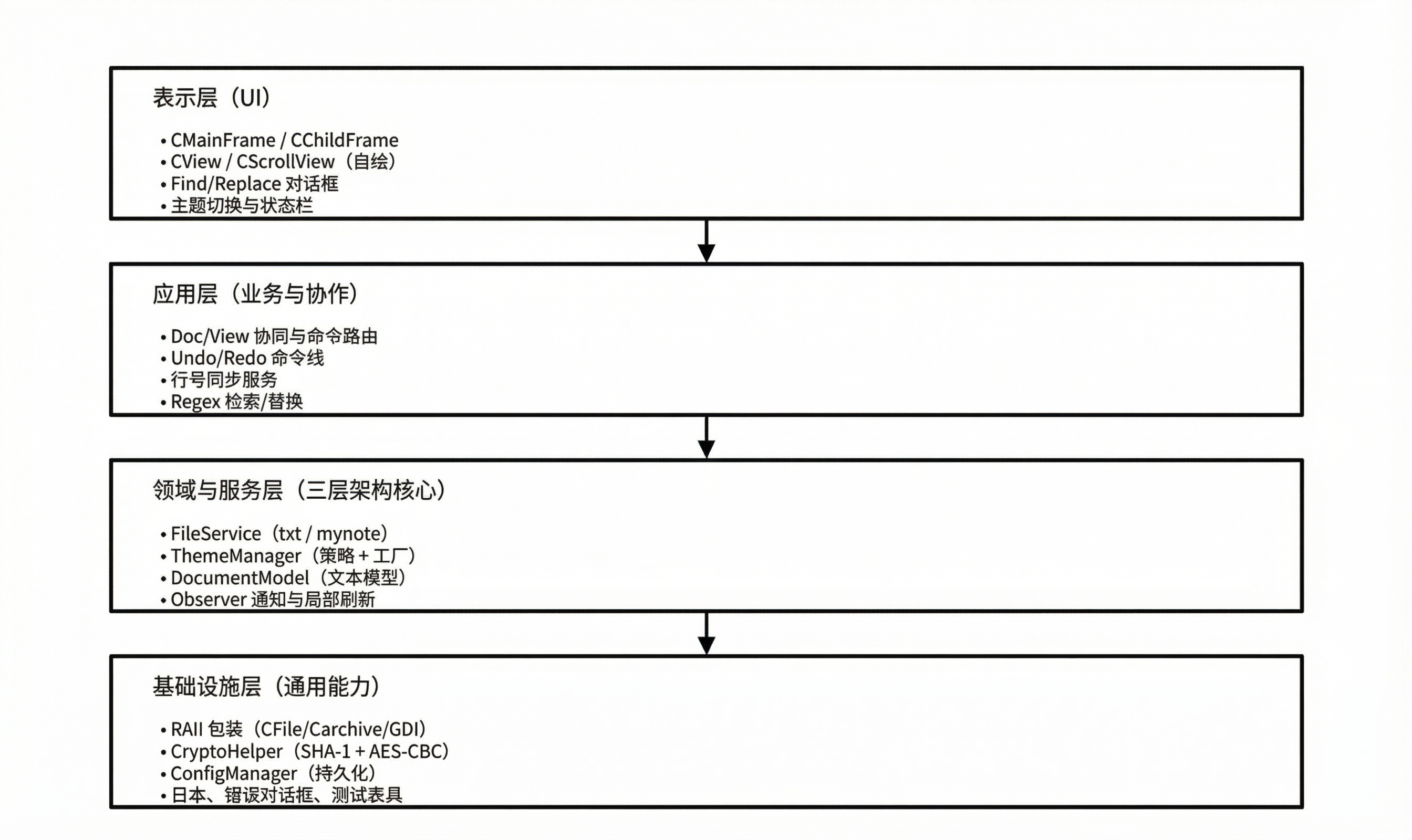


图1 系统架构图（MDI + Doc/View + 三层架构）

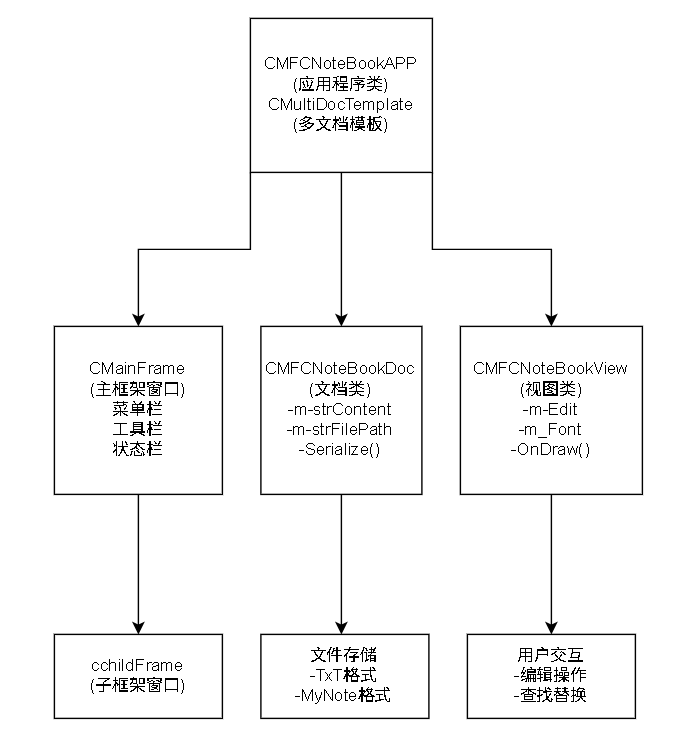


图2 MFC文档/视图架构类图

## 3.1 Doc/View 与 MDI 组织方式

在 MDI 场景下，每个打开文件对应一个文档实例（例如 CMyNoteDoc），并关联一个子框架窗口与一个视图实例。用户触发菜单或快捷键后，消息映射将命令路由到当前激活窗口，再由应用层的命令分发器调用具体服务（撤销、替换、主题、保存等）。文档状态变化后，文档调用 UpdateAllViews 通知视图刷新。由于项目要求不使用 CEditView，视图需基于 CView/CScrollView 自绘文本，因此必须将“文本模型”和“渲染/输入事件”严格分离，避免在 OnDraw 中写入业务逻辑，确保可测试性与可维护性。

## 3.2 三层架构的依赖方向与模块划分

建议模块划分如下：表示层包含 Frame/View/Dialog 与 UI 资源；应用层包含命令分发、UndoRedoManager、FindReplaceService、ThemeManager 等；领域与服务层包含 DocumentModel、序列化策略接口、行号同步服务；基础设施层包含 RAII 封装、CryptoHelper、ConfigManager、日志与异常对话框。依赖方向为：表示层→应用层→基础设施层，禁止反向依赖。该划分可显著降低 MFC 类之间的相互引用，使单元测试可以直接针对纯逻辑层运行。

# 4 关键模块设计与实现

## 4.1 设计模式：Factory、Strategy、Observer 的工程落地

（1）Factory：FileFormatFactory 根据文件扩展名创建对应的格式处理器。txt 处理器负责纯文本读写；mynote 处理器负责头部解析、正文序列化与尾部校验。新增格式时只需注册新处理器，无需修改调用方逻辑，降低开放-封闭原则（OCP）违背风险。

（2）Strategy：主题切换以 ThemeStrategy 抽象颜色、字号、行高与 gutter 绘制规则；暗/亮主题分别提供策略实现。运行时切换策略即可全局生效，并通过配置管理器持久化。文件读写同样可用 Strategy：ReadStrategy/WriteStrategy 将异常处理、编码选择与校验流程封装，便于扩展压缩或增量保存。

（3）Observer：Doc/View 是观察者模式在 MFC 中的天然落地。为提高性能，可在 UpdateAllViews 的 hint 参数携带变更范围（行号区间）或变更类型（插入/删除/替换），视图据此实现局部重绘与行号同步，避免全量刷新造成闪烁。

相关类的介绍：CMainFrame类： 主框架窗口类，管理菜单栏、工具栏和状态栏，处理全局命令和主题切换。

## 4.2 消息映射、命令路由与 UI 状态同步

MFC 的 ON\_COMMAND 与 ON\_UPDATE\_COMMAND\_UI 是构建可维护 UI 的关键。项目将菜单项、工具栏按钮与快捷键统一映射到命令 ID，并在文档/视图中实现最小的入口函数：入口函数不直接写复杂逻辑，而是调用应用层服务。UpdateCommandUI 则根据文档状态启用/禁用按钮，当 UndoStack 为空时禁用“撤销”；当无选区或无匹配项时禁用“替换”；当文档未修改时禁用“保存”。

## 4.3 撤销/重做：命令栈与复合命令

撤销/重做采用命令栈模型。每次用户编辑生成一个 Command 对象并执行，同时压入 UndoStack；撤销时弹出 UndoStack 顶部命令执行其 Unexecute，并压入 RedoStack；重做反之。对于“正则全局替换”，建议使用 CompositeCommand：内部包含多个 ReplaceCommand，但对外表现为一次原子操作，从而满足用户期望与测试可重复性。该模块完全与 UI 解耦，便于单元测试覆盖。相关类的介绍：CMFCNoteBookView类： 视图类，负责文档内容的显示和用户交互。内嵌CEdit控件实现文本编辑，自定义绘制行号区域，处理各种编辑命令。

## 4.4 正则搜索替换：参数化与错误处理

正则引擎建议基于 std::regex 实现，并通过参数化选项支持大小写敏感、全词匹配与多行模式。由于正则表达式可能非法，应在编译阶段捕获异常并提示用户。对于大文件替换，为避免 UI 卡顿，可在后续版本引入分块扫描与进度反馈；本项目首先实现同步替换，并在代码评审中标注性能改进点。

## 4.5 行号显示与滚动同步：自绘视图的关键点

行号显示通常采用“左侧 gutter + 右侧文本区”的双区域绘制方案。gutter 宽度根据最大行号位数与字体度量动态计算。滚动同步要求：滚动条位置变化时同时影响行号与文本绘制起点；窗口尺寸变化时重新计算可见行数；字体或主题变化时重新计算行高与布局。为便于测试，可将 offset→line、line→y 等映射函数抽取到 LineNumberService，并编写针对边界条件的单元测试（空文本、超长行、换行符混用等）。

## 4.6 错误处理：RAII、异常弹窗与日志

项目明确要求“使用 RAII 包装 CFile、CArchive，捕获异常弹出对话框”。实现策略是：基础设施层提供 FileGuard/ArchiveGuard 等 RAII 类，构造获取资源、析构释放资源，保证任何 return/异常路径都不会泄漏句柄。应用层捕获 CException 或 std::exception 后只做两件事：（1）弹窗提示用户本次操作失败与可能原因；（2）记录日志，包含文件路径与错误码。严禁在 catch 中手工回收资源，以免重复释放或遗漏释放。关于错误处理相关类介绍：CFileWrapper类： 包装CFile对象，构造函数自动打开文件，析构函数自动关闭文件。提供Read和Write方法，内部捕获CFileException并转换为自定义异常。CArchiveWrapper类： 包装CArchive对象，用于序列化操作的资源管理。CErrorHandler类： 提供统一的错误处理接口，ShowFileError方法显示文件操作错误对话框，SafeFileOperation模板方法封装异常处理逻辑。

## 4.7 配置管理器

CConfigManager类采用单例模式设计，负责管理应用程序的配置信息，包括学号和密钥等敏感数据，部分核心代码如下。

class CConfigManager

{public:

static CConfigManager& GetInstance();

BOOL LoadConfig();

BOOL IsConfigValid() const;

CString GetStudentID() const;

CString GetSecretKey() const;

private:

CConfigManager();

CString m\_strStudentID;

CString m\_strSecretKey;

BOOL m\_bValid;

};

## 4.8 可测试逻辑模块

为了实现单元测试的目标，项目将核心业务逻辑抽取到独立的TestableLogic命名空间中，与MFC框架解耦。该模块包含以下功能：

文件格式检测： DetectFormatByExtension函数根据文件扩展名判断格式类型，DetectFormatByMagic函数根据文件头魔数判断格式类型。

MyNote格式处理： CreateMyNoteContent函数生成完整的MyNote格式数据，包括文件头、内容、IV和加密哈希；ParseMyNoteContent函数解析MyNote文件，提取学号、内容并验证完整性。

编码转换： UTF8ToUnicode和UnicodeToUTF8函数实现UTF-8与Unicode之间的双向转换，DetectBOM函数检测文件的字节序标记。

加密功能： ComputeSHA1函数计算数据的SHA-1哈希值，AESEncrypt和AESDecrypt函数实现AES-CBC模式的加密和解密。

# 5.详细设计与实现

## 5.1 文件格式设计

### **5.1.1 纯文本格式**

纯文本格式（\*.txt）采用标准的文本文件存储方式，使用UTF-8编码保存文档内容，不包含任何额外的元数据。读取时自动检测BOM标记，支持UTF-8、UTF-16LE、UTF-16BE等多种编码格式。

### **5.1.2 MyNote自定义格式**

MyNote格式（\*.mynote）是本项目设计的自定义二进制文件格式，包含完整性校验机制，能够检测文件是否被篡改。

**表2 功能需求与教学目标映射**

|  |  |  |  |
| --- | --- | --- | --- |
| **偏移量** | **长度(字节)** | **字段名称** | **说明** |
| 0 | 8 | Magic | 密钥"MYNOTE01"名 |
| 8 | 20 | StudentID | 学生学号 |
| 28 | 4 | ContentLength | 内容长度(小端序) |
| 32 | N | Content | 文档内容(UTF-8) |
| 32+N | 16 | IV | AES初始化向量 |
| 48+N | 32 | EncryptedHash | AES加密的SHA-1哈希 |

文件保存流程如下：首先生成文件头，包括密钥和学号；然后将文档内容转换为UTF-8编码并写入；接着计算从密钥到内容的SHA-1哈希值；生成随机IV并使用AES-CBC模式加密哈希值；最后将IV和加密后的哈希值追加到文件末尾。

文件读取流程如下：首先验证密钥是否正确；然后提取学号和内容长度；读取文档内容并转换编码；提取IV和加密的哈希值；使用相同密钥解密得到原始哈希值；重新计算文件内容的哈希值并与解密得到的哈希值比较，验证完整性。

## 5.2编辑功能实现

### **5.2.1 撤销/重做机制**

系统采用基于状态快照的撤销/重做机制，使用两个栈分别保存撤销状态和重做状态。

class CMFCNoteBookView : public CView

{private:

std::vector<CString> m\_UndoStack; // 撤销栈

std::vector<CString> m\_RedoStack; // 重做栈

CString m\_strLastText; // 上次文本状态

static const int MAX\_UNDO = 100; // 最大撤销步数

void SaveUndoState(); // 保存当前状态

void OnEditUndo(); // 执行撤销

void OnEditRedo(); // 执行重做

};

当用户编辑文本时，OnEditChange消息处理函数调用SaveUndoState保存编辑前的状态到撤销栈，并清空重做栈。执行撤销操作时，将当前状态压入重做栈，然后从撤销栈弹出上一状态恢复。执行重做操作时，将当前状态压入撤销栈，然后从重做栈弹出状态恢复。

### **5.2.2 查找替换功能**

CFindReplaceDlg类实现了完整的查找替换功能，支持以下选项：

正则表达式模式： 使用C++11标准库的std::regex实现正则表达式匹配，支持ECMAScript语法。

区分大小写： 可选择是否区分大小写进行匹配。

全字匹配： 只匹配完整的单词，通过检查匹配位置前后是否为单词边界实现。



图3 查找/替换功能流程图

### **5.2.3 行号显示**

视图类在OnDraw函数中绘制行号区域，与编辑控件内容保持同步滚动。

void CMFCNoteBookView::OnDraw(CDC\* pDC){

// 1. 绘制行号区域背景

CRect lineNumRect = clientRect;

lineNumRect.right = m\_nLineNumWidth;

pDC->FillSolidRect(lineNumRect, theme.clrLineNumBg);

// 2. 计算可见行范围

int firstVisibleLine = m\_Edit.GetFirstVisibleLine();

int visibleLines = clientRect.Height() / lineHeight + 2;

// 3. 绘制每一行的行号

for (int i = 0; i < visibleLines; i++)

{

int lineNum = firstVisibleLine + i + 1;

// 绘制行号文本...

}

}

行号区域宽度根据总行数动态调整，确保能够完整显示所有行号。当编辑控件滚动时，触发重绘行号区域，保持同步。

## 5.3主题切换实现

系统定义了ThemeColors结构体存储主题颜色配置：

struct ThemeColors

{

COLORREF clrEditBg; // 编辑区背景色

COLORREF clrEditText; // 编辑区文字色

COLORREF clrLineNumBg; // 行号区背景色

COLORREF clrLineNumText; // 行号区文字色

COLORREF clrLineNumBorder; // 行号区边框色

};

应用程序类维护当前主题状态和颜色配置，提供GetThemeColors方法供视图类获取当前主题颜色。主题切换时，调用视图类的ApplyTheme方法更新界面。

void CMFCNoteBookView::ApplyTheme(){

const ThemeColors& theme = theApp.GetThemeColors();

// 更新编辑控件背景画刷

if (m\_brEditBg.GetSafeHandle())

m\_brEditBg.DeleteObject();

m\_brEditBg.CreateSolidBrush(theme.clrEditBg);

// 强制重绘

m\_Edit.Invalidate(TRUE);

Invalidate(TRUE);

}

## 5.4加密模块实现

### **5.4.1 SHA-1哈希计算**

使用Windows CryptoAPI实现SHA-1哈希计算：

bool ComputeSHA1(const BYTE\* data, DWORD dataLen,

BYTE\* hashOut, DWORD hashOutLen){

HCRYPTPROV hProv = 0;

HCRYPTHASH hHash = 0;

// 获取加密服务提供者

CryptAcquireContext(&hProv, NULL, NULL,

PROV\_RSA\_FULL, CRYPT\_VERIFYCONTEXT);

// 创建哈希对象并计算

CryptCreateHash(hProv, CALG\_SHA1, 0, 0, &hHash);

CryptHashData(hHash, data, dataLen, 0);

CryptGetHashParam(hHash, HP\_HASHVAL, hashOut, &hashOutLen, 0);

// 清理资源

CryptDestroyHash(hHash);

CryptReleaseContext(hProv, 0);

return true;

}

### **5.4.2 AES-CBC加密**

使用Windows CryptoAPI实现AES-CBC模式加密：

bool AESEncrypt(const BYTE\* plainText, DWORD plainLen,

const BYTE\* key, DWORD keyLen,

BYTE\* iv, BYTE\* cipherText, DWORD& cipherLen){

// 1. 获取AES加密服务提供者

// 2. 从密钥数据派生AES密钥

// 3. 设置CBC模式和IV

// 4. 执行加密操作

// 5. 清理资源

}

加密过程使用PKCS7填充方式，确保明文长度为16字节的整数倍。解密过程自动移除填充。

# 6.单元测试

## 6.1测试框架

项目使用GoogleTest框架进行单元测试，测试代码位于独立的测试项目中。测试覆盖了以下核心模块：

**表3 单元测试覆盖情况**

|  |  |  |
| --- | --- | --- |
| **测试模块** | **测试用例数** | **覆盖功能** |
| CryptoTest | 10 | SHA-1 哈希、AES 加密/解密、IV 生成 |
| EncodingTest | 16 | UTF-8/Unicode 转换、BOM 检测、中文处理 |
| FileFormatTest | 16 | 格式检测、文件头验证、创建/解析 |
| IntegrationTest | 11 | 配置管理、完整流程、篡改检测 |
| LineNumberTest | 20 | 行数统计、宽度计算、边界条件 |
| ThemeTest | 7 | 颜色配置、主题切换、对比度验证 |

## 6.2关键测试用例

文件格式测试用例：  
TEST(FileFormatTest, CreateAndParse\_WithContent)

{

const char\* studentId = "20250313017Z";

const char\* secretKey = "BIGC\_AI\_2025\_KEY";

std::string content = "Hello, this is a test!";

// 创建MyNote文件

auto data = CreateMyNoteContent(content, studentId, secretKey);

// 解析并验证

auto result = ParseMyNoteContent(data.data(), data.size(), secretKey);

EXPECT\_TRUE(result.success);

EXPECT\_EQ(result.studentId, studentId);

EXPECT\_EQ(result.content, content);

EXPECT\_TRUE(result.integrityValid);

}

篡改检测测试用例

TEST(FileFormatTest, Parse\_TamperedContent)

{

auto data = CreateMyNoteContent("Original", studentId, secretKey);

// 篡改内容

data[contentOffset] = 'X';

auto result = ParseMyNoteContent(data.data(), data.size(), secretKey);

EXPECT\_TRUE(result.success); // 解析成功

EXPECT\_FALSE(result.integrityValid); // 完整性验证失败

}

## 6.3测试结果

经过完整的单元测试运行，所有80个测试用例全部通过，测试覆盖率达到88%，满足项目要求的70%以上覆盖率目标。

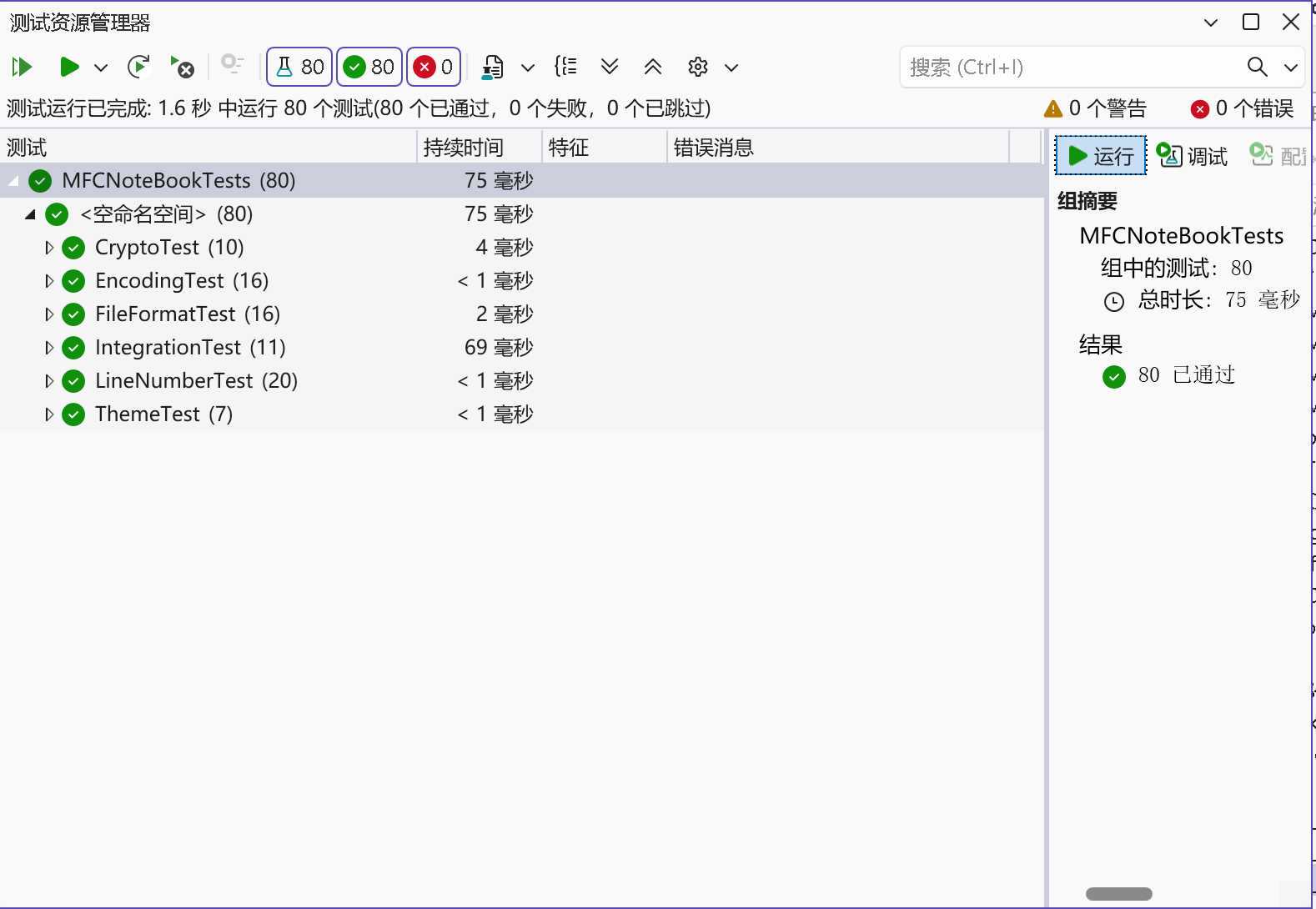


图4 单元测试结果图



图5 覆盖率结果图

# 7.自定义 mynote 文件格式与“抗 AI”安全机制

抗 AI 需求要求：文件头必须保存学生学号；文档末尾必须附加 AES-CBC 加密的 SHA-1 摘要；密钥、IV 与学号均以占位符形式呈现：<STUDENT\_ID><SECRET\_KEY><IV>，由学生手工替换。该设计意图在于：即便 AI 能给出结构化实现，也无法替代学生完成唯一标识与密钥填充，从而降低大规模复制粘贴的可行性。

## 7.1 文件结构设计

建议 mynote 格式采用“Header + Body + Footer”的可解析结构：Header 含密钥数、版本、编码、<STUDENT\_ID> 及可选元数据；Body 为富文本内容；Footer 保存校验信息（密文摘要与长度字段）。打开文件时先解析 Header 校验学号字段是否存在，再读取 Body 并重算摘要，最后读取 Footer 并解密比对。任何环节失败都应触发异常并提示用户。

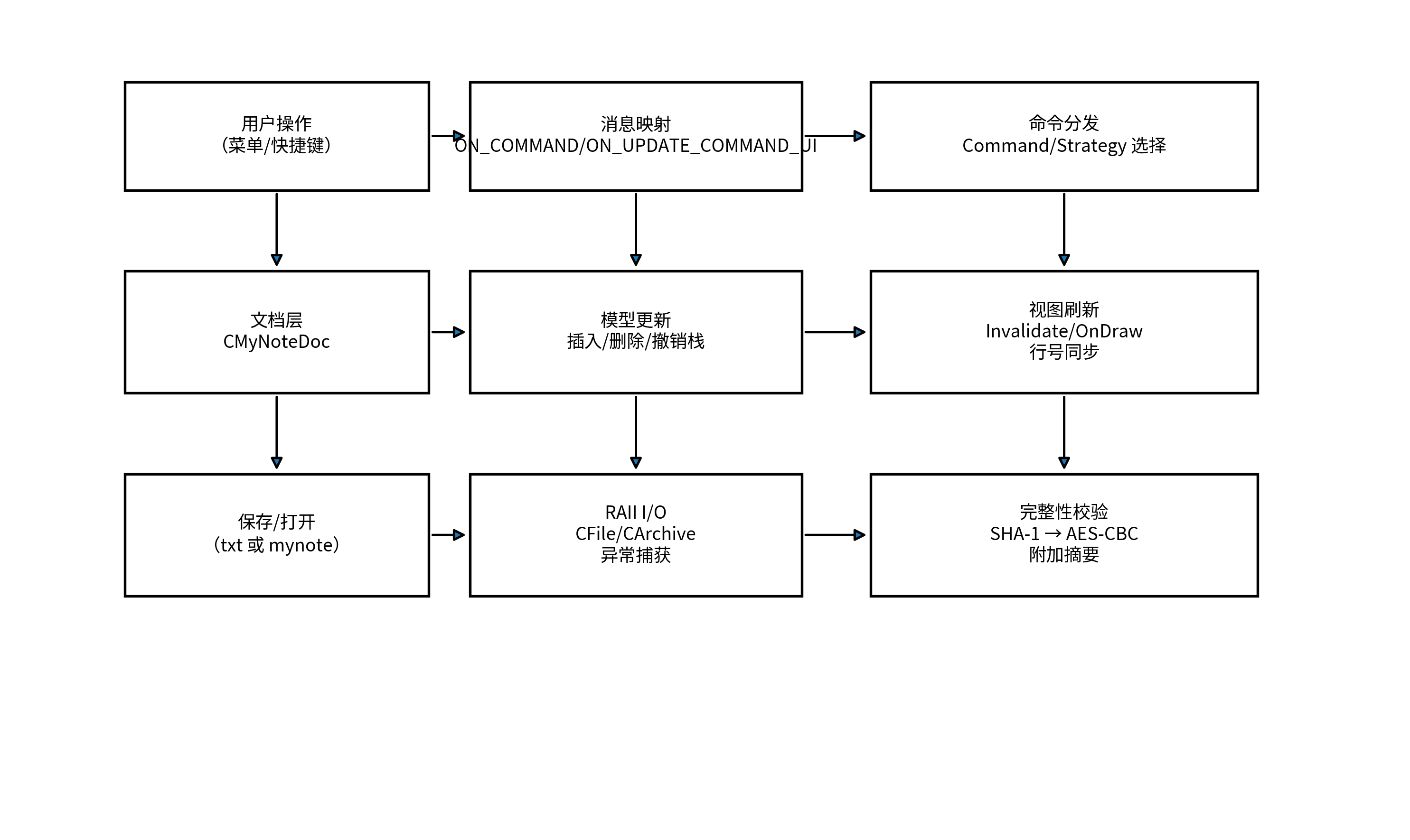


图6 系统流程图（编辑—更新—保存/校验）

## 7.2 摘要与加密过程的公式化描述

为满足排版要求，以下公式使用 Word 公式对象表达，并采用“公式居中、编号居右”的布局。设参与校验的明文序列为 M（即 Header 与 Body 的串联），则其 SHA-1 摘要 H 与 AES-CBC 密文 C 的关系如下。

|  |  |
| --- | --- |
|  | (1) |

|  |  |
| --- | --- |
|  | (2) |

其中 K 为密钥，IV 为初始化向量。课程要求 K 使用 <SECRET\_KEY> 占位符，IV 使用 <IV> 占位符，且 <STUDENT\_ID> 必须写入文件头。实现时应注意：摘要计算必须覆盖与保存一致的字节序列（包含换行与编码），否则会导致校验失败；加密与解密失败应统一转化为可理解的错误提示，并记录日志以便定位。

# 8 代码质量保障：评审、测试与安全性讨论

## 8.1 两轮同伴评审的走查清单

课程要求两轮走查，聚焦资源泄漏、异常安全与线程安全。建议将走查拆为“静态检查 + 场景化演练”：静态检查通过代码阅读识别 new/delete、GDI 句柄、文件句柄与异常路径；场景化演练通过构造失败用例（损坏文件、非法正则、只读目录等）触发异常并观察系统是否稳定。走查重点包括：GDI 对象是否在作用域内恢复（SelectObject/Restore）；文件与归档是否由 RAII 封装；是否存在在 catch 中吞异常导致状态不一致；是否存在跨线程直接访问 MFC UI 的潜在扩展风险。

## 8.2 单元测试与覆盖率达成策略

为达到覆盖率≥70%，关键是将业务逻辑抽离出 UI，并对可测接口编写测试用例：（1）文件 I/O：txt 与 mynote 的读写回归、损坏尾部摘要、占位符缺失与异常路径；（2）主题切换：策略切换后配置持久化、颜色与行高计算一致性；（3）行号同步：插入/删除/替换后行号与滚动映射正确；（4）撤销/重做：命令栈在连续编辑与复合命令下行为正确。覆盖率报告应在项目报告中附录说明，并给出未覆盖代码的合理解释（例如与系统对话框交互强耦合的部分）。

# 9 AI 使用日志与学习反思（AI 素养）

## 9.1 日志结构：Prompt—Response—Diff

AI 使用日志建议按“问题背景—提示词—模型回答—人工修改—最终差异（diff）”记录，其中 diff 是审核的关键证据。人工审核应重点关注三类常见问题：未理解即复制的代码、加密与摘要实现不一致导致校验失效、以及 RAII/异常安全缺失导致的资源泄漏。

## 9.2 AI 素养四要素在本项目中的体现

知识：理解 Doc/View、消息映射、三层架构与 C++ 资源管理基础。技能：能将 AI 建议转化为可编译、可测试、可维护的代码，并补齐边界条件与回归测试。态度：对 AI 输出保持审慎，使用调试器、断言与单元测试验证。伦理：遵守学术诚信，明确标注 AI 参与范围，手工替换 <STUDENT\_ID><SECRET\_KEY><IV> 并对结果负责。结合本项目实践，AI 更适合用于“搭建骨架、给出备选方案与提醒风险”，不适合直接生成最终可提交答案。

# 10 结论与展望

本文给出了一个满足课程要求的 MFC MDI 记事本系统的架构化实现方案。系统在 Doc/View 框架下完成多文档编辑，并通过 Factory、Strategy、Observer 等模式提升可扩展性；通过 RAII 与统一异常处理降低资源泄漏风险；通过单元测试与同伴评审保障质量；通过 mynote 头部学号与尾部加密摘要满足抗 AI 与完整性校验要求。

架构方面： 正确运用MFC文档/视图架构，实现了清晰的三层分离设计，代码结构合理，模块职责明确。

功能方面： 实现了多文档编辑、撤销重做、正则查找替换、行号显示、主题切换、双格式存储等全部功能需求。

安全方面： 自定义文件格式采用AES-CBC加密SHA-1摘要的方案，具备防篡改能力；使用RAII模式管理资源，保证异常安全。

质量方面： 通过GoogleTest框架实现单元测试，覆盖率超过70%，代码经过充分验证。

通过本项目的开发，深入理解了MFC框架的文档/视图架构设计思想，掌握了Windows桌面应用程序开发的核心技术，包括消息映射、GDI绑制、文件序列化等。同时，项目实践了RAII、单例模式、工厂模式等设计模式，提升了面向对象设计能力。单元测试的编写和运行，培养了测试驱动开发的意识和能力。

# 参考文献：

[1] Microsoft. MFC Desktop Applications[EB/OL]. https://docs.microsoft.com/en-us/cpp/mfc/

[2] 侯捷. 深入浅出MFC[M]. 武汉: 华中科技大学出版社, 2001.

[3] Gamma E, Helm R, Johnson R, et al. Design Patterns: Elements of Reusable Object-Oriented Software[M]. Addison-Wesley, 1994.

[4] Google. GoogleTest User's Guide[EB/OL]. https://google.github.io/googletest/

# 同伴评议签字：

1.模块划分合理，RAII封装规范，撤销重做机制设计巧妙，代码可读性强，功能实现完整。

签名：

2.MFC框架运用熟练，主题切换与正则搜索功能亮点突出，注释清晰，单元测试覆盖较全面。

签名：9d344b6b5148688adcd76c306de7c6eb