

# 自然写互动课堂PC端应用软件 V1.0

## 软件鉴别材料 — 用户操作手册与设计说明书

软件全称：自然写互动课堂PC端应用软件

软件版本：V1.0

权利人：深圳自然写科技有限公司

文档类型：PC桌面应用用户操作手册 + 设计说明书

文档编号：WRITECH-APP-PC-DS-001

编制日期：2026年2月

适用平台：Windows 10/11 64位 / macOS 12 Monterey 及以上

## 目录

- 第一章 软件整体概述
- 第二章 系统架构与设计思路
- 第三章 核心模块功能详细说明
- 第四章 操作流程与使用步骤
- 第五章 与源代码的对应关系
- 附录

## 第一章 软件整体概述

### 1.1 软件简介与功能综述

自然写互动课堂PC端应用软件（以下简称"PC APP"）是自然写互动课堂系统面向教师的桌面端综合教学工具，支持Windows和macOS双平台。PC APP基于Electron + Vue.js 3框架开发，通过充分利用桌面端的大屏幕、高性能处理器和丰富的外设接口（USB/蓝牙），提供备课制作、课堂授课、作业批改、数据管理等完整教学工作流。

PC APP是整个互动课堂系统中功能最完整的客户端，也是教师日常备课和课堂教学的核心工具。相较于手机APP，PC APP提供了更强大的课件制作功能、更详细的数据分析视图和更流畅的投屏操控体验。

主要功能模块综述：

功能模块	说明
备课工具	课件制作（类PPT功能）、试卷编辑、字帖模板设计
课堂授课	实时接收全班笔迹、互动答题、随机抽查、展示控制
作业管理	发布/回收作业，查看AI批改结果，人工批改标注

功能模块	说明
笔迹回放分析	以时间轴方式回放任意学生的书写过程
班级数据管理	班级成绩统计、知识点掌握情况、学情趋势
点阵码编辑	自定义点阵码内容设计，生成可打印点阵作业纸
投屏控制	将PC画面镜像投射至智慧黑板/电视
设备连接	USB有线或BLE无线连接点阵笔

## 1.2 软件用途与适用场景

### 备课场景

教师在课前使用PC APP进行备课： – 从资源库导入字帖模板和试卷模板，编辑自定义内容 – 设计互动题目（选择题、填空题、写字题）并预设标准答案 – 生成含点阵码的作业纸PDF，发送给学校打印室打印 – 将备课内容发布至班级，学生Pad端自动接收

### 课堂授课场景

课堂进行中，教师在讲台PC上使用PC APP： – 开启课堂模式，大屏分割视图展示全班实时书写状态 – 点击任意学生小窗口放大查看该学生的书写详情 – 通过PC投屏至智慧黑板，展示选中学生作品供全班对比 – 发布互动答题，倒计时收卷，实时展示答题统计

### 批改与分析场景

课后教师使用PC APP进行数据分析： – 批量查看AI批改结果，快速标注需人工复核的题目 – 查看班级知识点掌握雷达图，识别共性薄弱点 – 导出成绩单（CSV/Excel格式）上传至学校教务系统 – 生成家长学情报告并批量推送

## 1.3 运行环境与系统要求

### Windows平台：

配置项	最低要求	推荐配置
操作系统	Windows 10（64位，版本1903）	Windows 11
处理器	Intel Core i5（4核）	Intel Core i7/i9 或 AMD Ryzen 7
内存	8GB RAM	16GB RAM
显卡	支持WebGL 2.0的独显/集显	NVIDIA / AMD独立显卡
存储	SSD 10GB可用空间	SSD 50GB可用空间
网络	百兆以太网或WiFi 5	千兆以太网或WiFi 6
蓝牙	BLE 4.0（可选，笔连接）	BLE 5.0
USB	USB 2.0（用于笔连接）	USB 3.0
显示器	1920×1080	2560×1440 双屏

### macOS平台：

配置项	最低要求	推荐配置
操作系统	macOS 12 Monterey	macOS 14 Sonoma
处理器	Intel Core i5 或 Apple M1	Apple M2/M3
内存	8GB	16GB
存储	10GB可用空间	50GB可用空间

1.4 开发语言与技术规范

主要技术栈：

技术	版本	用途
Electron	28.0.0	跨平台桌面应用框架
Node.js	20.x LTS	主进程运行环境
Vue.js	3.4.0	渲染进程UI框架
TypeScript	5.3.0	类型安全的JavaScript超集
Pinia	2.1.7	Vue 3状态管理
Vite	5.0.0	前端构建工具（渲染进程）
Axios	1.6.2	HTTP请求库
WebSocket (ws)	8.16.0	实时通信（主进程）
SQLite (better-sqlite3)	9.4.3	本地数据库（主进程）
IndexedDB (Dexie.js)	3.2.4	渲染进程大容量存储
Canvas 2D + WebGL	浏览器原生	笔迹渲染引擎
C++ Addon (Node-API)	最新	高性能笔迹平滑算法、USB通信
node-bluetooth	1.1.4	BLE点阵笔连接
node-usb	2.11.0	USB HID设备访问
WebRTC	渲染进程原生	投屏协议
electron-updater	6.1.7	自动更新

Electron IPC通信架构：

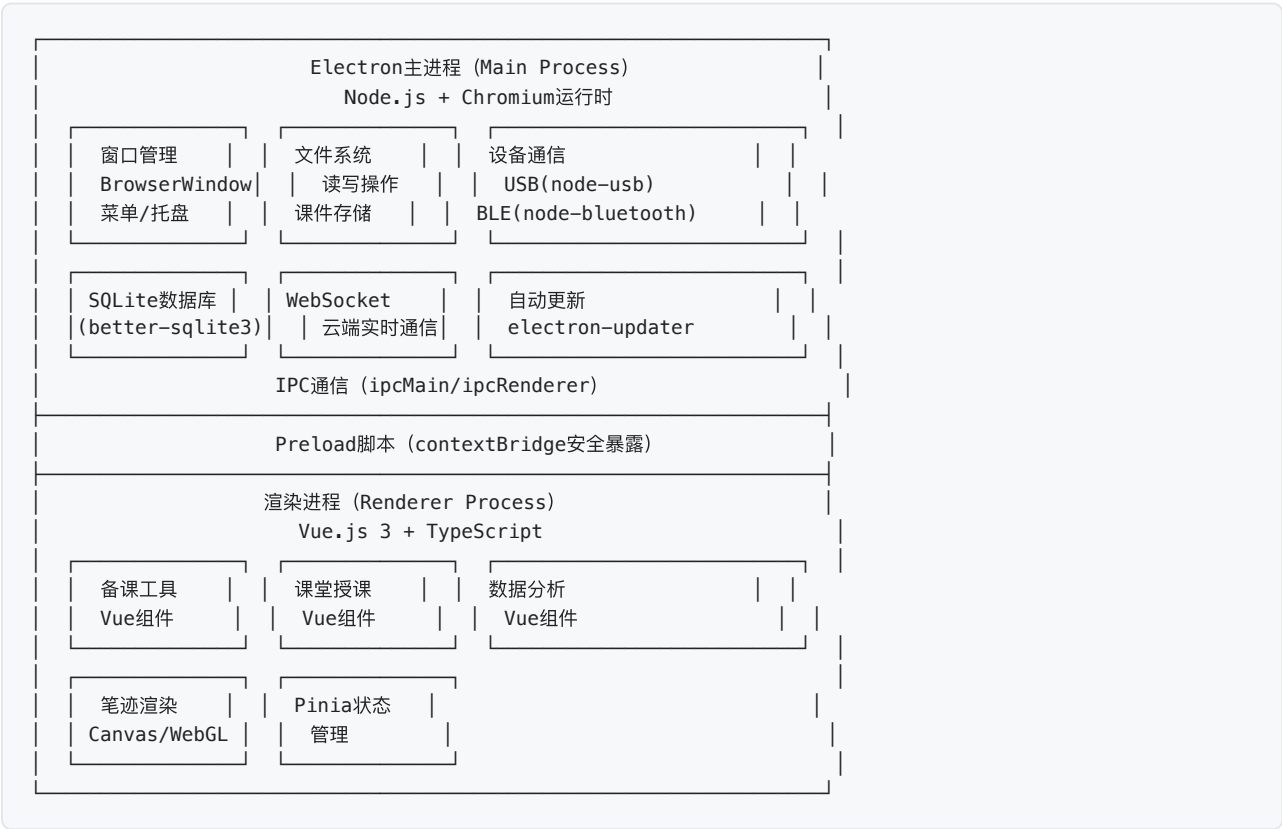
PC APP采用Electron的主进程（Main Process）+ 渲染进程（Renderer Process）架构： – 主进程：处理系统API调用（文件操作、USB/BLE设备通信、SQLite数据库、WebSocket连接） – 渲染进程：Vue.js 3界面渲染，通过IPC调用主进程的功能 – Preload脚本：在渲染进程中安全暴露主进程API（使用contextIsolation保护）

1.5 版本说明

版本	日期	平台	主要变更
V0.7 Beta	2025年8月	Windows/macOS	基础备课工具、课堂收笔、作业发布
V0.9 RC	2025年11月	Windows/macOS	点阵码编辑、投屏功能、数据导出
V1.0	2026年2月	Windows/macOS	正式版：WebGL笔迹渲染、双屏支持、AI辅助批改

## 第二章 系统架构与设计思路

### 2.1 Electron应用架构



### 2.2 进程间通信设计

IPC通道规划 (ipcMain / ipcRenderer):

```
// src/preload/index.ts - contextBridge暴露API到渲染进程
import { contextBridge, ipcRenderer } from 'electron'

contextBridge.exposeInMainWorld('electronAPI', {
  // 数据库操作
  db: {
    query: (sql: string, params: any[]) =>
      ipcRenderer.invoke('db:query', sql, params),
    run: (sql: string, params: any[]) =>
      ipcRenderer.invoke('db:run', sql, params),
  },

  // 文件操作
```

```

file: {
  save: (fileName: string, data: Buffer) =>
    ipcRenderer.invoke('file:save', fileName, data),
  open: (filters: FileFilter[]) =>
    ipcRenderer.invoke('file:open', filters),
  exportPDF: (content: any) =>
    ipcRenderer.invoke('file:exportPDF', content),
},

// 设备通信
device: {
  scanBLE: () => ipcRenderer.invoke('device:scanBLE'),
  connectBLE: (deviceId: string) =>
    ipcRenderer.invoke('device:connectBLE', deviceId),
  connectUSB: () => ipcRenderer.invoke('device:connectUSB'),
  onInkData: (callback: (data: InkPoint[]) => void) => {
    ipcRenderer.on('device:inkData', (_event, data) => callback(data))
  },
},

// 投屏控制
cast: {
  startCasting: (targetInfo: CastTarget) =>
    ipcRenderer.invoke('cast:start', targetInfo),
  stopCasting: () => ipcRenderer.invoke('cast:stop'),
},

// 窗口控制
window: {
  openLessonWindow: () => ipcRenderer.invoke('window:openLesson'),
  enterPresentation: () => ipcRenderer.invoke('window:enterPresentation'),
}
})

```

## 2.3 笔迹渲染引擎设计

PC APP使用WebGL + C++ Native Addon实现高性能笔迹渲染，支持压感效果（根据压力值变化线宽）和笔锋效果（笔画首尾尖细）：

```

// src/renderer/rendering/StrokeRenderer.ts
export class StrokeRenderer {
  private gl: WebGL2RenderingContext
  private program: WebGLProgram
  private vertexBuffer: WebGLBuffer

  constructor(canvas: HTMLCanvasElement) {
    this.gl = canvas.getContext('webgl2')!
    this.initShaders()
    this.vertexBuffer = this.gl.createBuffer()!
  }

  private initShaders() {
    // 顶点着色器：根据压感值计算线宽
    const vertexShader = `#version 300 es
      in vec2 a_position;
      in float a_pressure;
      in float a_segment_pos; // 0=起点, 1=终点（用于笔锋计算）

      uniform mat4 u_projection;
      uniform float u_base_width;

      out float v_pressure;
    `

```

```

void main() {
    // 笔锋效果: 首尾收细 (sigmoid曲线模拟)
    float taper = min(a_segment_pos * 4.0, (1.0 - a_segment_pos) * 4.0);
    taper = clamp(taper, 0.0, 1.0);

    // 最终线宽 = 基础宽度 × 压感 × 笔锋系数
    float width = u_base_width * a_pressure * (0.3 + 0.7 * taper);

    // 沿法线方向扩展 (宽度扩张为几何体)
    gl_PointSize = width;
    gl_Position = u_projection * vec4(a_position, 0.0, 1.0);
    v_pressure = a_pressure;
}
、

// 片段着色器: 抗锯齿圆点渲染
const fragmentShader = `#version 300 es
precision mediump float;
in float v_pressure;
out vec4 fragColor;

void main() {
    // 圆形点 (通过gl_PointCoord实现圆角)
    vec2 coord = gl_PointCoord - vec2(0.5);
    float r = length(coord);
    float alpha = 1.0 - smoothstep(0.4, 0.5, r); // 边缘抗锯齿
    fragColor = vec4(0.1, 0.1, 0.1, alpha); // 深灰色笔迹
}
、

this.program = this.createShaderProgram(vertexShader, fragmentShader)
}

// 绘制一条笔画 (由多个坐标点构成)
drawStroke(points: StrokePoint[]) {
    if (points.length < 2) return

    const gl = this.gl
    gl.useProgram(this.program)

    // 构建顶点数据 (每点: x, y, pressure, segment_pos)
    const vertices = new Float32Array(points.length * 4)
    const totalLength = points.length - 1

    for (let i = 0; i < points.length; i++) {
        const p = points[i]
        vertices[i * 4 + 0] = p.x
        vertices[i * 4 + 1] = p.y
        vertices[i * 4 + 2] = p.pressure
        vertices[i * 4 + 3] = i / totalLength // segment_pos: 0→1
    }

    gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexBuffer)
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.DYNAMIC_DRAW)

    // 绑定属性
    const posLoc = gl.getAttribLocation(this.program, 'a_position')
    gl.enableVertexAttribArray(posLoc)
    gl.vertexAttribPointer(posLoc, 2, gl.FLOAT, false, 16, 0)

    const pressureLoc = gl.getAttribLocation(this.program, 'a_pressure')
    gl.enableVertexAttribArray(pressureLoc)
    gl.vertexAttribPointer(pressureLoc, 1, gl.FLOAT, false, 16, 8)

    const segPosLoc = gl.getAttribLocation(this.program, 'a_segment_pos')
    gl.enableVertexAttribArray(segPosLoc)
    gl.vertexAttribPointer(segPosLoc, 1, gl.FLOAT, false, 16, 12)
}

```

```
    gl.drawArrays(gl.POINTS, 0, points.length)
  }
}
```

## 2.4 数据设计

SQLite数据库表（主进程，better-sqlite3）：

表名	主要字段	说明
lessons	id, title, subject, grade, content_json, created_at	课件数据
assignments	id, lesson_id, class_id, title, deadline, status	作业/试卷
students	id, class_id, name, student_no, parent_phone	学生信息
submissions	id, assignment_id, student_id, ink_data_path, score, status	作业提交记录
grading_records	id, submission_id, teacher_comment, manual_score, ai_score	批改记录
dot_code_maps	id, lesson_page_id, dot_code_range_start, dot_code_range_end	点阵码映射
devices	id, type, identifier, name, last_connected	已连接设备记录
app_config	key, value, updated_at	应用配置键值对

IndexedDB存储（渲染进程，Dexie.js）：

数据库	说明
inkDataDB	大容量笔迹原始数据存储（每次作业的完整笔迹数据）
resourceCacheDB	资源文件本地缓存（字帖图片、课件资源）

## 2.5 接口设计

云端API接口（渲染进程通过Axios调用）：

接口	方法	URL	说明
登录	POST	/api/v1/auth/login	教师账号登录
获取班级列表	GET	/api/v1/class/list	获取教师管理的班级
创建作业	POST	/api/v1/assignment/create	发布新作业
获取提交列表	GET	/api/v1/assignment/{id}/submissions	获取学生提交列表
上传批改结果	PUT	/api/v1/submission/{id}/grade	保存批改结果
获取班级学情	GET	/api/v1/analytics/class/{id}	班级数据分析
生成点阵码	POST	/api/v1/dotcode/generate	生成作业纸点阵码
资源搜索	GET	/api/v1/resource/search	搜索教学资源

接口	方法	URL	说明
推送报告	POST	/api/v1/report/push/{class_id}	批量推送学情报告给家长

### WebSocket实时通信（主进程WebSocket）：

```
// src/main/services/websocket-service.ts
export class WebSocketService {
  private ws: WebSocket | null = null
  private mainWindow: BrowserWindow

  connect(classroomId: string, token: string) {
    this.ws = new WebSocket(
      `wss://api.writech.cn/ws/v1/classroom?id=${classroomId}`,
      { headers: { Authorization: `Bearer ${token}` } }
    )

    this.ws.on('message', (data: string) => {
      const event = JSON.parse(data)

      switch (event.type) {
        case 'stroke.realtime':
          // 转发笔迹数据到渲染进程
          this.mainWindow.webContents.send('ws:inkData', event)
          break

        case 'submission.complete':
          // 通知渲染进程某学生已提交
          this.mainWindow.webContents.send('ws:submissionComplete', event)
          break

        case 'result.aiGraded':
          // AI批改完成通知
          this.mainWindow.webContents.send('ws:aiGradingDone', event)
          break
      }
    })

    this.ws.on('close', () => {
      // 5秒后自动重连
      setTimeout(() => this.connect(classroomId, token), 5000)
    })
  }

  sendControl(type: string, payload: any) {
    if (this.ws?.readyState === WebSocket.OPEN) {
      this.ws.send(JSON.stringify({ type, ...payload }))
    }
  }
}
```

## 2.6 安全设计

### 应用级安全：

```
// src/main/index.ts - 安全配置
const win = new BrowserWindow({
  webPreferences: {
    preload: path.join(__dirname, 'preload.js'),
    contextIsolation: true,    // 启用上下文隔离（必须）
    nodeIntegration: false,   // 禁止渲染进程直接访问Node.js（安全）
  }
})
```



```

    sandbox: true,          // 启用渲染进程沙箱
    webSecurity: true,      // 启用Web安全策略（CORS等）
    allowRunningInsecureContent: false, // 禁止混合内容
  }
})

// 设置CSP内容安全策略（防XSS）
session.defaultSession.webRequest.onHeadersReceived((details, callback) => {
  callback({
    responseHeaders: {
      ...details.responseHeaders,
      'Content-Security-Policy': [
        "default-src 'self'; " +
        "script-src 'self'; " +
        "style-src 'self' 'unsafe-inline'; " +
        "connect-src https://api.writech.cn wss://api.writech.cn; " +
        "img-src 'self' https://cdn.writech.cn data;"]
    ]
  })
})

```

#### 本地数据安全：

- SQLite数据库使用SQLCipher加密（密钥派生自用户登录密码哈希 + 设备指纹）
- 学生笔迹数据（IndexedDB）存储于Electron userData目录，受操作系统文件权限保护
- 课件导出PDF支持加密选项（PDF密码保护）
- 自动更新包含代码签名验证（Windows Authenticode / macOS Gatekeeper）

#### 代码保护：

- Electron ASAR归档打包，防止直接读取源码
- 关键业务逻辑（笔迹平滑算法、点阵码解析）编译为C++ Native Addon（.node文件）
- 生产构建启用代码混淆（terser压缩）

## 第三章 核心模块功能详细说明

### 3.1 备课工具模块

源代码文件：`src/renderer/features/lesson/`

备课工具是PC APP的核心创作功能，提供类PPT的课件制作界面：

课件编辑器界面：

【文件】【编辑】【插入】【格式】【课堂】【工具】【帮助】				自然写PC版
页面缩略图	编辑区域（当前页）			属性面板
<div> <div>第1页</div> <div>第2页</div> <div>第3页</div> <div>+ 新增</div> </div>	<div>今日生字：一 大 天 地</div> <div>【拖拽添加内容区域】</div>			<div>文字属性：</div> <div>字体：【楷体 ▼】</div> <div>大小：【48 ▼】</div> <div>点阵码设置：</div> <div>【绑定点阵码】</div>

元素库	[标注模式]	[激光笔]	[橡皮]	[撤销] [重做]
[图片]				
[文字框]				
[字帖]				
[题目]				

课件数据结构：

```
// src/shared/types/lesson.ts
interface LessonData {
  id: string
  title: string
  subject: 'chinese' | 'math' | 'english'
  grade: string
  pages: LessonPage[]
  metadata: {
    createdAt: number
    updatedAt: number
    teacherId: string
    schoolId: string
  }
}

interface LessonPage {
  id: string
  pageIndex: number
  background: string // 背景色或背景图URL
  elements: PageElement[] // 页面元素（文字/图片/字帖/题目）
  dotCodeRange?: { // 绑定的点阵码范围（可选）
    start: string
    end: string
  }
  speakerNote?: string // 演讲备注
}

type PageElement = TextElement | ImageElement | CalligraphyElement | QuestionElement

interface QuestionElement {
  type: 'question'
  id: string
  questionType: 'choice' | 'fill_blank' | 'writing' | 'essay'
  questionText: string
  standardAnswer?: string | string[] // 标准答案
  scoringRules?: ScoringRule[] // 评分规则
  position: { x: number; y: number; width: number; height: number }
}
```

点阵码内容编辑（点阵码绑定器）：

```
// src/renderer/features/dotcode/DotCodeBinder.vue
// 将课件页面与点阵码范围绑定，生成可打印的点阵作业纸

// 绑定逻辑：
// 1. 教师选择要打印的页面范围（如第1-3页）
// 2. 系统向云端资源平台申请点阵码范围
// 3. 为每页分配唯一的点阵码ID范围
// 4. 生成带点阵底纹的PDF（600DPI打印精度）

async function generateDotCodePDF(pages: LessonPage[]): Promise<Blob> {
  // 向云端申请点阵码范围
```

```

const dotCodeRange = await api.dotcode.allocate({
  pageCount: pages.length,
  school_id: store.user.schoolId
})

// 生成PDF (调用主进程的PDFKit渲染)
const pdfData = await window.electronAPI.file.exportPDF({
  pages,
  dotCodeInfo: dotCodeRange,
  resolution: 600 // 600DPI打印精度
})

return new Blob([pdfData], { type: 'application/pdf' })
}

```

## 3.2 课堂授课模块

源代码文件: `src/renderer/features/classroom/`

课堂主界面 (三列布局):

[←课堂管理] 二年级一班 - 语文课 ● 进行中 已连38笔 [结束课堂]			
课件展示区 (主屏)	全班书写状态 (小格)	工具栏	
[课件当前页]	[张三] [李四] [王五]	[发题]	
今日生字:	[赵六] [陈七] [周八]	[收卷]	
一 大 天 地	[吴九] [郑十] ...	[点名]	
		[展示]	
		[暂停]	
[◀上一页] [下一页▶]	提交进度: <div><div></div></div> 30/38	连接状态:	
		● 网关 已连	
[激光笔] [标注]	[全班展示] [对比]	● 算力盒 就绪	
[橡皮] [清除]		● 投屏 未连	

随机抽取学生 (防重复抽取算法):

```

// src/renderer/features/classroom/store/classroomStore.ts
const useClassroomStore = defineStore('classroom', {
  state: () => ({
    students: [] as Student[],
    calledStudents: new Set<string>(), // 已点名学学生ID集合
  }),

  actions: {
    randomPickStudent(excludeCalled: boolean = true) {
      let candidates = this.students

      if (excludeCalled && this.calledStudents.size < this.students.length) {
        // 排除已点名学生 (直到所有人都被点过一次)
        candidates = this.students.filter(
          s => !this.calledStudents.has(s.id))
      } else if (this.calledStudents.size >= this.students.length) {
        // 所有人都被点过, 重置
        this.calledStudents.clear()
      }

      // 随机选取 (使用crypto.getRandomValues保证随机性)
      const randomIndex = Math.floor(
        (crypto.getRandomValues(new Uint32Array(1))[0] / 0xFFFFFFFF)

```

```
        * candidates.length
    )
    const selected = candidates[randomIndex]

    this.calledStudents.add(selected.id)

    // 推送点名结果至智慧黑板展示
    websocketService.sendControl('classroom.pickStudent', {
      studentId: selected.id,
      studentName: selected.name,
      effect: 'spotlight' // 黑板端显示聚光灯特效
    })

    return selected
  }
}
})
```

### 3.3 作业批改模块

源代码文件：`src/renderer/features/grading/`

批改主界面（两栏布局）：

[←] 第5课生字练习 - 批改 提交: 38/40 已批改: 25/38			
学生列表		当前学生批改区	
✓ 张三	92分	[已批改]	学生: 王五 提交时间: 08:32
✓ 李四	88分	[已批改]	
● 王五	--	[批改中]	学生书写内容 (笔迹展示) [字1] [字2] [字3] [字4]
○ 赵六	--	[待批改]	
○ 陈七	--	[待批改]	
...			
AI建议:		AI分析 (逐字):	
王五第3字笔顺有误		[字1] 98分 ✓	
赵六书写规范度不足		[字2] 95分 ✓	
		[字3] 72分 △ 第3笔顺序错误	
		[字4] 88分 ✓	
		总分: [ 85 ]分 (AI建议: 85)	
		批注: [笔顺注意规范, 字体整洁...]	
		[采纳AI建议] [确认] [下一个▶]	

AI辅助批改逻辑：

```
// src/renderer/features/grading/composables/useAIGrading.ts
export function useAIGrading() {
  const gradeSubmission = async (submissionId: string):
    Promise<AIGradingResult> => {

    // 1. 获取AI批改结果 (服务端已批改, 直接查询)
    const result = await api.grading.getAIResult(submissionId)

    if (result.status === 'completed') {
      return result.data
    } else if (result.status === 'pending') {
      // AI还在处理, 轮询等待 (最多等60秒)
```

```

        return await pollForResult(submissionId, 60)
    } else {
        throw new Error('AI批改失败, 请手动批改')
    }
}

// 一键采纳AI建议 (填入AI推荐分数)
const acceptAISuggestion = (aiResult: AIGradingResult) => {
    return {
        score: aiResult.totalScore,
        perItemScores: aiResult.itemScores,
        comment: aiResult.suggestedComment,
        gradedBy: 'ai_assisted'
    }
}

return { gradeSubmission, acceptAISuggestion }
}

```

### 3.4 USB/BLE点阵笔连接模块

源代码文件: `src/main/services/device-service.ts`

USB设备连接 (Node-API C++ Addon):

```

// src/main/services/device-service.ts
import { createRequire } from 'module'
const require = createRequire(import.meta.url)

// 加载C++ Native Addon (实现USB HID通信和笔迹平滑)
const writetechNative = require('../native/writetech_native.node')

export class DeviceService {
    private usbDevice: any = null
    private bleDevice: any = null

    // 扫描USB点阵笔 (nRF52840 USB HID模式)
    async scanUSBPens(): Promise<USBDevice[]> {
        const devices = writetechNative.listUSBHIDDevices()
        return devices.filter((d: any) =>
            d.vendorId === WRITETECH_VENDOR_ID &&
            d.productId === WRITETECH_PEN_PRODUCT_ID
        )
    }

    // 连接USB点阵笔并开始接收数据
    async connectUSBPen(devicePath: string): Promise<void> {
        this.usbDevice = writetechNative.openUSBHIDDevice(devicePath)

        // 注册数据接收回调 (C++层实现, 高频调用)
        writetechNative.startInkReceiving(this.usbDevice, (rawData: Buffer) => {
            // 解析原始HID数据包 (与BLE格式兼容)
            const points = this.parseInkPacket(rawData)

            // 应用笔迹平滑 (C++实现, 保证性能)
            const smoothed = writetechNative.smoothStroke(points)

            // 发送到渲染进程
            this.mainWindow.webContents.send('device:inkData', smoothed)
        })
    }

    private parseInkPacket(data: Buffer): InkPoint[] {

```

```

// 解析与BLE协议相同的差分编码格式
const points: InkPoint[] = []
let offset = 0

const packetType = data[offset++]
const frameCount = data[offset++]
const baseTimestamp = data.readUInt16LE(offset); offset += 2

// 第一帧：绝对坐标
const x0 = data.readUInt16LE(offset); offset += 2
const y0 = data.readUInt16LE(offset); offset += 2
const p0 = data[offset++]
const f0 = data[offset++]
points.push({ x: x0, y: y0, pressure: p0 / 255, penUp: !(f0 & 0x01) })

// 后续帧：差分解码
let lastX = x0, lastY = y0
for (let i = 1; i < frameCount; i++) {
  const flags = data[offset++]
  const dx = (flags & 0x80) ? data.readInt16LE(offset) : data.readInt8(offset)
  offset += (flags & 0x80) ? 2 : 1
  const dy = (flags & 0x40) ? data.readInt16LE(offset) : data.readInt8(offset)
  offset += (flags & 0x40) ? 2 : 1
  const pressure = data[offset++]

  lastX += dx
  lastY += dy
  points.push({
    x: lastX, y: lastY,
    pressure: pressure / 255,
    penUp: !(flags & 0x01)
  })
}

return points
}
}

```

### 3.5 投屏控制模块

源代码文件：src/main/services/cast-service.ts

PC APP支持将当前课件/展示内容投射到智慧黑板，支持WebRTC和HDMI两种投屏方式：

```

// src/main/services/cast-service.ts
export class CastService {
  // WebRTC投屏（无线，通过局域网）
  async startWebRTCCast(boardIP: string): Promise<void> {
    // 1. 获取屏幕捕获流
    const captureStream = await desktopCapturer.getSources({
      types: ['window'],
      thumbnailSize: { width: 1920, height: 1080 }
    })

    const lessonWindow = captureStream.find(s =>
      s.name.includes('自然写') && s.name.includes('课件'))

    // 2. 创建WebRTC连接到黑板端APP
    const peerConnection = new RTCPeerConnection()
    const stream = await navigator.mediaDevices.getUserMedia({
      video: {
        mandatory: {
          chromeMediaSource: 'desktop',

```

```
        chromeMediaSourceId: lessonWindow!.id,
      }
    } as any
  })

  stream.getTracks().forEach(track =>
    peerConnection.addTrack(track, stream))

  // 3. 通过信令服务器建立连接
  const offer = await peerConnection.createOffer()
  await peerConnection.setLocalDescription(offer)

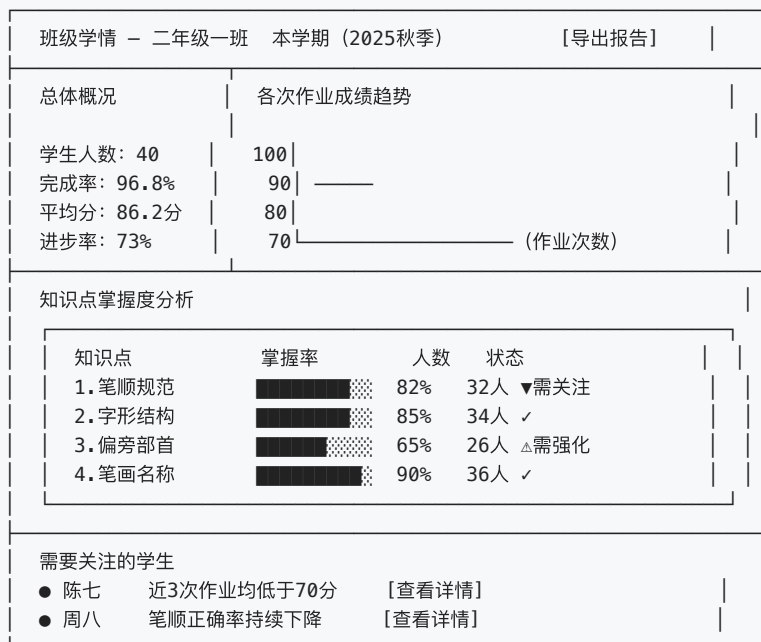
  // 发送offer给黑板端（通过WebSocket信令）
  await signalingService.sendOffer(boardIP, offer)
}

// 停止投屏
stopCasting(): void {
  this.peerConnection?.close()
  this.peerConnection = null
}
}
```

### 3.6 数据统计与分析模块

源代码文件：src/renderer/features/analytics/

班级学情仪表盘界面：



### 3.7 自动更新模块

PC APP内置自动更新功能，通过 electron-updater 实现静默后台更新：

```
// src/main/updater.ts
import { autoUpdater } from 'electron-updater'
import { dialog, BrowserWindow } from 'electron'
```

```

export function setupAutoUpdater(mainWindow: BrowserWindow) {
  // 每小时检查一次更新
  autoUpdater.checkForUpdates()
  setInterval(() => autoUpdater.checkForUpdates(), 60 * 60 * 1000)

  autoUpdater.on('update-available', (info) => {
    // 有新版本可用，通知渲染进程显示提示
    mainWindow.webContents.send('updater:updateAvailable', info)
  })

  autoUpdater.on('update-downloaded', (info) => {
    // 下载完成，询问用户是否立即安装
    dialog.showMessageBox(mainWindow, {
      type: 'info',
      title: '更新就绪',
      message: `新版本 ${info.version} 已下载完成，立即重启安装?`,
      buttons: ['立即安装', '稍后安装'],
      defaultId: 0,
    }).then(({ response }) => {
      if (response === 0) {
        autoUpdater.quitAndInstall(false, true)
      }
    })
  })

  // 验证更新包签名（防恶意更新）
  autoUpdater.on('before-quit-for-update', () => {
    // electron-updater自动验证代码签名
  })
}

```

## 第四章 操作流程与使用步骤

### 4.1 安装与首次启动

#### Windows安装：

1. 下载安装包 Writech-PC-Setup-1.0.0.exe（约200MB）
2. 双击运行，选择安装目录（默认 C:\Program Files\Writech）
3. 安装过程自动注册文件关联和桌面快捷方式
4. 安装完成后桌面出现"自然写互动课堂"图标

#### macOS安装：

1. 下载 Writech-PC-1.0.0.dmg
2. 打开DMG，将"自然写互动课堂.app"拖拽到"应用程序"文件夹
3. 首次运行时macOS提示"来自已识别开发者"，点击"打开"
4. 输入系统密码允许安装（需要系统管理员权限）

#### 首次配置：

##### 首次启动流程：

1. 欢迎界面 → [开始配置]
2. 登录账号（手机号+密码或机构账号）
3. 选择学校和年级



4. 配置连接方式 (USB笔/蓝牙笔/仅网络)
5. 测试连接 (可选)
6. 进入主界面

## 4.2 备课操作流程

创建新课件：

操作步骤：

1. 点击主界面"新建课件" (或Ctrl+N)
2. 选择模板 (空白/字帖练习/试卷/互动课堂)
3. 输入课件标题、学科、年级
4. 在编辑区域添加内容：
  - 插入→文字框：输入课文内容
  - 插入→字帖：从资源库选择字帖模板
  - 插入→题目：添加互动题目 (设置标准答案)
5. 为需要作答的页面绑定点阵码 (右键页面→绑定点阵码)
6. 保存 (Ctrl+S) 并发布到班级 (文件→发布到班级)

生成作业纸：

操作步骤：

1. 打开已完成的课件
2. 文件→生成作业纸 PDF
3. 选择要打印的页面范围
4. 确认点阵码分配 (系统自动申请)
5. 选择打印分辨率 (建议600DPI)
6. 点击"生成PDF", 保存到本地
7. 将PDF发送给学校打印室打印

## 4.3 课堂授课操作流程

上课前 (准备阶段)：

1. 打开PC APP, 进入班级主页
2. 点击"开始课堂"→选择班级→选择今日课件
3. 课堂模式启动, 检查连接状态 (网关●/算力盒●/投屏●)
4. 投屏到智慧黑板 (课堂工具栏→投屏→选择连接方式)

上课中：

1. 遥控器/键盘翻页 (PgUp/PgDn) 展示课件
2. 使用激光笔功能 (快捷键L) 在课件上标注重点
3. 发题：
  - a. 工具栏→发题→选择预设题目或临时出题
  - b. 设置作答时限 (可设30秒~无限制)
  - c. 点击"开始", 黑板大屏自动展示题目
4. 收卷：
  - a. 工具栏→收卷 (或倒计时结束自动收卷)
  - b. 自动展示答题统计 (在黑板大屏呈现)
5. 展示学生作品：
  - a. 在全班书写状态格中单击学生小格
  - b. 右键→"投屏展示", 该学生作品显示到黑板大屏

## 4.4 作业批改操作流程

批改流程 (课后)：

1. 主界面→作业管理→选择最近发布的作业
2. 作业列表显示每个学生的提交状态和AI初评分

3. 逐个批改：

a. 点击学生条目，进入批改详情

b. 查看AI建议（逐字评分+笔顺分析）

c. 若AI结果准确，点击"采纳AI建议"一键完成

d. 若需调整，手动修改分数和添加文字批注

e. 点击"确认"→自动跳转下一个学生

4. 全部批改完成后：

a. 点击"推送结果"→批改结果推送到学生Pad和家长手机

b. 点击"导出成绩单"→导出CSV/Excel格式成绩单

## 4.5 设备连接操作流程

### USB连接点阵笔：

- USB连接操作：
1. 用Type-C数据线连接笔和PC的USB口

2. 点阵笔自动进入USB模式（LED白色常亮）

3. PC APP右下角设备状态显示"USB笔 已连接"

4. 在课件编辑器中选择一个写字区域

5. 用笔书写，PC屏幕实时显示笔迹

### BLE无线连接：

- BLE连接操作：
1. PC APP→设置→设备管理→扫描蓝牙设备

2. 打开点阵笔电源（长按笔帽开关）

3. 列表中出现"Writech-XXXXXX"

4. 点击"配对"，按提示完成配对（Numeric Comparison）

5. 配对成功后后续开机自动重连

## 4.6 故障排查

问题	原因	解决方法
USB笔不被识别	驱动未安装	重新安装USB驱动（安装包附带驱动）或更新USB驱动
投屏黑屏	防火墙阻止连接	在防火墙中允许PC APP访问局域网
AI批改结果长时间等待	云端AI服务繁忙	等待5-10分钟或稍后刷新（结果完成后自动推送）
课件同步失败	网络断开	检查网络，课件已本地保存，网络恢复后自动同步
应用启动崩溃	版本不兼容	卸载后重新安装最新版本

# 第五章 与源代码的对应关系

## 5.1 模块与源代码文件对应表

功能模块	源代码路径	说明
主进程入口	src/main/index.ts	Electron主进程启动、窗口创建、安全配置
Preload脚本	src/preload/index.ts	contextBridge API安全暴露

功能模块	源代码路径	说明
主进程IPC处理	src/main/ipc-handlers.ts	所有IPC通道的处理函数注册
数据库服务	src/main/services/db-service.ts	SQLite数据库操作（better-sqlite3）
设备服务	src/main/services/device-service.ts	USB/BLE点阵笔连接与数据接收
WebSocket服务	src/main/services/websocket-service.ts	云端实时通信（主进程）
投屏服务	src/main/services/cast-service.ts	WebRTC投屏协议实现
自动更新	src/main/updater.ts	electron-updater自动更新配置
C++ Native Addon	native/writtech_native/	USB HID通信、笔迹平滑算法
渲染进程入口	src/renderer/main.ts	Vue.js 3 应用初始化
路由配置	src/renderer/router/index.ts	vue-router路由配置
全局状态	src/renderer/store/	Pinia全局Store
备课工具	src/renderer/features/lesson/	课件编辑器Vue组件
课堂授课	src/renderer/features/classroom/	课堂模式Vue组件、实时数据处理
作业批改	src/renderer/features/grading/	批改界面Vue组件、AI辅助批改
数据分析	src/renderer/features/analytics/	学情统计图表Vue组件
点阵码编辑	src/renderer/features/dotcode/	点阵码绑定和PDF生成
WebGL渲染引擎	src/renderer/rendering/StrokeRenderer.ts	WebGL笔迹渲染引擎
HTTP客户端	src/renderer/api/client.ts	Axios HTTP请求封装
本地IndexedDB	src/renderer/storage/inkDB.ts	Dexie.js大容量笔迹数据存储
构建配置	electron.vite.config.ts	Electron + Vite构建配置

## 5.2 核心类与函数说明

类/函数名	所在文件	功能说明
<code>createWindow()</code>	<code>main/index.ts</code>	创建主窗口，配置安全选项
<code>setupIpchHandlers()</code>	<code>main/ipc-handlers.ts</code>	注册所有IPC通道处理函数
<code>DBService.query()</code>	<code>main/services/db-service.ts</code>	SQLite查询封装
<code>DeviceService.connectUSBPen()</code>	<code>main/services/device-service.ts</code>	USB笔连接与数据流监听

类/函数名	所在文件	功能说明
<code>WebSocketService.connect()</code>	<code>main/services/websocket-service.ts</code>	建立云端 WebSocket 连接
<code>CastService.startWebRTCcast()</code>	<code>main/services/cast-service.ts</code>	启动 WebRTC投 屏
<code>StrokeRenderer.drawStroke()</code>	<code>renderer/rendering/StrokeRenderer.ts</code>	WebGL笔 迹渲染
<code>useClassroomStore.randomPickStudent()</code>	<code>renderer/features/classroom/store</code>	随机点名 (防重复)
<code>useAIGrading.gradeSubmission()</code>	<code>renderer/features/grading/composables</code>	获取AI批改 结果
<code>generateDotCodePDF()</code>	<code>renderer/features/dotcode/DotCodeBinder.vue</code>	生成点阵码 作业纸PDF
<code>setupAutoUpdater()</code>	<code>main/updater.ts</code>	配置自动更 新检查与安 装

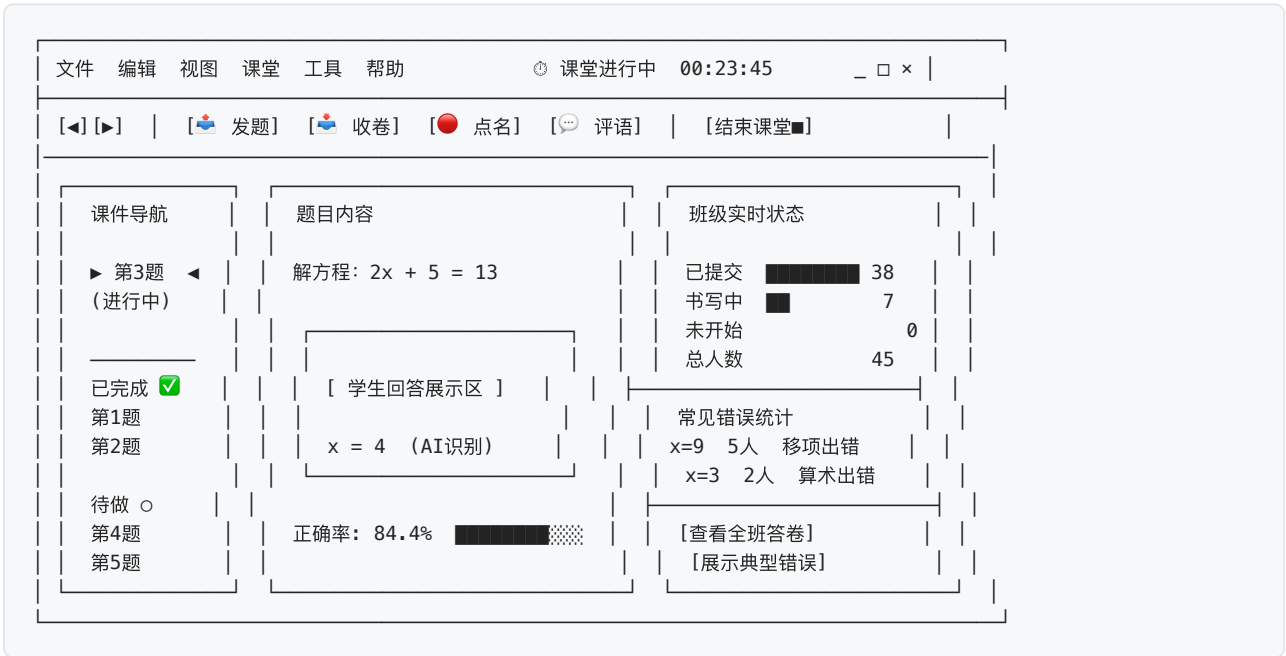
## 附录A 界面设计稿（GUI Mockup）

本附录以PC桌面横屏线框图形式呈现PC APP各核心界面的设计稿，反映Windows/macOS桌面应用的界面布局与交互元素。

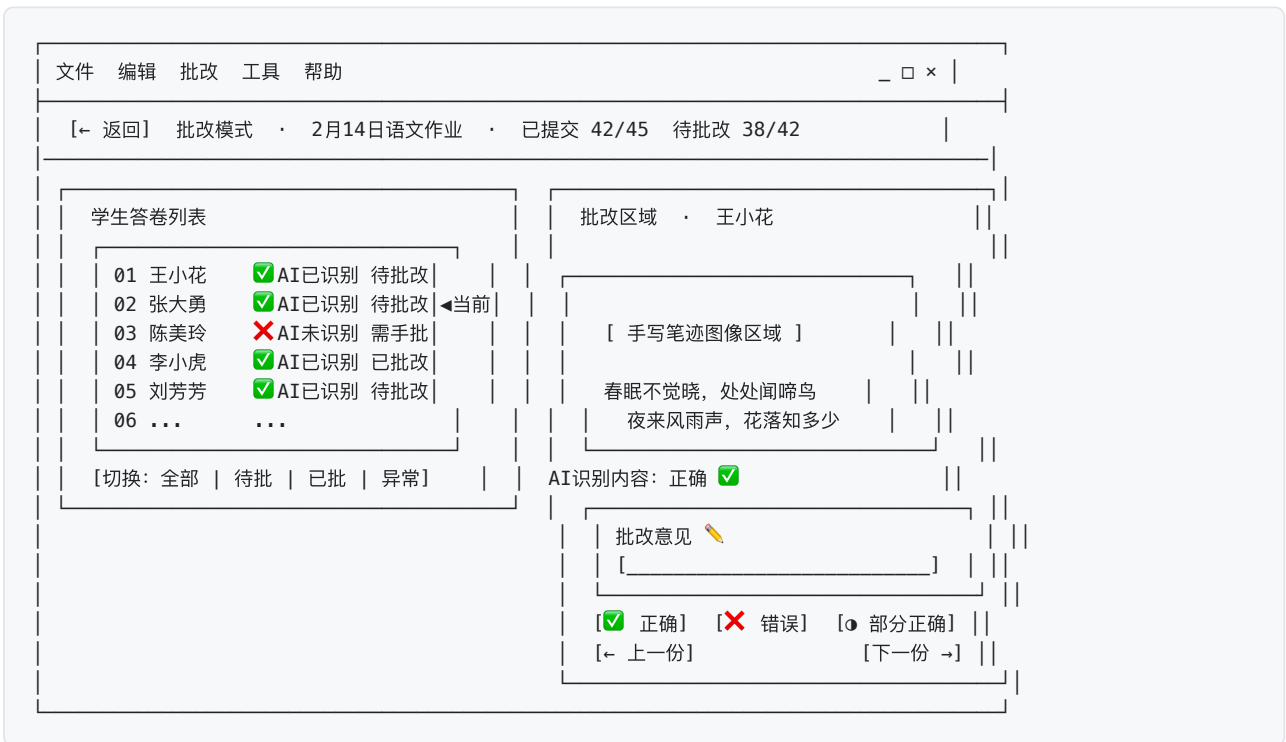
### A.1 应用主界面（课堂准备状态）



## A.2 课堂进行中界面



## A.3 作业批改界面



## A.4 书写回放界面



## 附录B 快捷键参考

快捷键（Windows）	快捷键（macOS）	功能
Ctrl+N	Cmd+N	新建课件
Ctrl+S	Cmd+S	保存
Ctrl+Z	Cmd+Z	撤销
Ctrl+Y	Cmd+Shift+Z	重做
PgUp / PgDn	PgUp / PgDn	课件翻页
F5	F5	进入演示模式
Esc	Esc	退出演示模式
L	L	激光笔模式
E	E	橡皮擦模式
Ctrl+D	Cmd+D	发题
Ctrl+R	Cmd+R	收卷
Ctrl+P	Cmd+P	随机点名
Ctrl+Shift+P	Cmd+Shift+P	打印/导出PDF
Ctrl+Q	Cmd+Q	退出应用

## 附录B 术语表

术语	说明
Electron	GitHub开发的跨平台桌面应用框架，基于Node.js + Chromium
Vue.js 3	渐进式JavaScript框架，用于构建用户界面
Pinia	Vue.js 3推荐的状态管理库（替代Vuex）
TypeScript	JavaScript的类型化超集，提供静态类型检查
Vite	新一代前端构建工具，极快的开发服务器
IPC	Inter-Process Communication，进程间通信
contextBridge	Electron安全API，在隔离上下文中暴露主进程功能
contextIsolation	Electron安全特性，阻止渲染进程直接访问Node.js
Node-API (N-API)	Node.js原生扩展API，用于编写C++ Addon
Native Addon	C++编写的Node.js扩展模块（.node文件）
SQLCipher	SQLite的加密扩展
WebRTC	Web实时通信标准，支持点对点音视频传输（PC APP用于投屏）
WebGL	Web图形库，浏览器中的OpenGL ES
ASAR	Electron应用包格式（Atom Shell Archive）
better-sqlite3	同步SQLite3 Node.js驱动，性能优秀
IndexedDB	浏览器内置的大容量NoSQL数据库（Electron渲染进程可用）

文档编制：深圳自然写科技有限公司 PC客户端研发团队

文档版本：V1.0

最后更新：2026年2月14日

版权所有 © 2026 深圳自然写科技有限公司

## 附录C 核心技术实现详述

### C.1 Electron主进程与渲染进程架构

PC桌面应用基于Electron框架构建，主进程（main process）负责系统级功能（BLE、文件I/O、本地数据库），渲染进程（renderer process）负责UI展示。两者通过IPC（进程间通信）安全通信。

#### C.1.1 主进程核心模块

```
// main/index.ts - Electron主进程入口
import { app, BrowserWindow, ipcMain, dialog, shell } from 'electron'
import { join } from 'path'
import { BleManager } from './ble/BleManager'
import { LocalDatabase } from './database/LocalDatabase'
import { SyncService } from './sync/SyncService'
```

```

import { NativeInkEngine } from './native/NativeInkEngine'
import { AutoUpdater } from './updater/AutoUpdater'

let mainWindow: BrowserWindow | null = null
let bleManager: BleManager | null = null
let localDb: LocalDatabase | null = null
let syncService: SyncService | null = null
let inkEngine: NativeInkEngine | null = null

async function createMainWindow() {
  mainWindow = new BrowserWindow({
    width: 1280,
    height: 800,
    minWidth: 1024,
    minHeight: 640,
    webPreferences: {
      preload: join(__dirname, '../preload/index.js'),
      contextIsolation: true,      // 隔离渲染进程
      nodeIntegration: false,     // 禁止渲染进程直接访问Node.js
      webSecurity: true,
      sandbox: false              // 允许preload访问Node.js
    },
    titleBarStyle: process.platform === 'darwin' ? 'hiddenInset' : 'default',
    show: false // 窗口准备好后再显示，避免白屏
  })

  // 加载应用
  if (process.env.ELECTRON_RENDERER_URL) {
    mainWindow.loadURL(process.env.ELECTRON_RENDERER_URL) // 开发模式
  } else {
    mainWindow.loadFile(join(__dirname, '../renderer/index.html')) // 生产模式
  }

  mainWindow.once('ready-to-show', () => {
    mainWindow!.show()
  })

  // 阻止新窗口，在外部浏览器打开链接
  mainWindow.webContents.setWindowOpenHandler(({ url }) => {
    shell.openExternal(url)
    return { action: 'deny' }
  })
}

async function initializeServices() {
  // 初始化本地SQLite数据库
  localDb = new LocalDatabase(join(app.getPath('userData'), 'writtech.db'))
  await localDb.initialize()

  // 初始化BLE管理器 (使用Noble C++ Addon)
  bleManager = new BleManager()
  await bleManager.initialize()

  // 初始化笔迹引擎 (C++ Native Addon)
  inkEngine = new NativeInkEngine()

  // 初始化数据同步服务
  syncService = new SyncService(localDb, 'https://api.writtech.com')

  // 注册所有IPC处理器
  registerIpcHandlers()
}

app.whenReady().then(async () => {
  await initializeServices()
  await createMainWindow()
})

```



```

    AutoUpdater.checkForUpdates()
  })

  app.on('window-all-closed', () => {
    bleManager?.destroy()
    syncService?.stop()
    localDb?.close()
    if (process.platform !== 'darwin') app.quit()
  })

```

## C.1.2 Preload安全桥接

```

// preload/index.ts - contextBridge安全暴露API
import { contextBridge, ipcRenderer } from 'electron'

// 向渲染进程暴露的API白名单
contextBridge.exposeInMainWorld('writechAPI', {
  // BLE钢笔管理
  ble: {
    startScan: () => ipcRenderer.invoke('ble:startScan'),
    stopScan: () => ipcRenderer.invoke('ble:stopScan'),
    connect: (deviceId: string) => ipcRenderer.invoke('ble:connect', deviceId),
    disconnect: (deviceId: string) => ipcRenderer.invoke('ble:disconnect', deviceId),
    onDeviceFound: (callback: (device: BleDevice) => void) => {
      ipcRenderer.on('ble:deviceFound', (_, device) => callback(device))
    },
    onInkData: (callback: (data: InkData) => void) => {
      ipcRenderer.on('ble:inkData', (_, data) => callback(data))
    },
    onConnectionChanged: (callback: (deviceId: string, connected: boolean) => void) => {
      ipcRenderer.on('ble:connectionChanged', (_, deviceId, connected) => callback(deviceId, connected))
    }
  },

  // 本地数据库操作
  database: {
    saveStroke: (stroke: StrokeRecord) => ipcRenderer.invoke('db:saveStroke', stroke),
    getStrokes: (filter: StrokeFilter) => ipcRenderer.invoke('db:getStrokes', filter),
    saveHomework: (homework: HomeworkRecord) => ipcRenderer.invoke('db:saveHomework', homework),
    getHomeworkList: (query: HomeworkQuery) => ipcRenderer.invoke('db:getHomeworkList', query),
    deleteOldData: (beforeDate: Date) => ipcRenderer.invoke('db:deleteOldData', beforeDate)
  },

  // 文件系统操作
  files: {
    exportToPdf: (content: ExportContent) => ipcRenderer.invoke('files:exportToPdf', content),
    exportToImage: (content: ExportContent) => ipcRenderer.invoke('files:exportToImage', content),
    openFile: () => ipcRenderer.invoke('files:openFile'),
    saveFile: (data: Uint8Array, defaultName: string) =>
      ipcRenderer.invoke('files:saveFile', data, defaultName)
  },

  // 云端同步
  sync: {
    syncNow: () => ipcRenderer.invoke('sync:syncNow'),
    getSyncStatus: () => ipcRenderer.invoke('sync:getStatus'),
    onSyncProgress: (callback: (progress: SyncProgress) => void) => {
      ipcRenderer.on('sync:progress', (_, progress) => callback(progress))
    }
  },

  // 应用信息
  app: {
    getVersion: () => ipcRenderer.invoke('app:getVersion'),

```

```

    checkUpdate: () => ipcRenderer.invoke('app:checkUpdate'),
    openDevTools: () => ipcRenderer.invoke('app:openDevTools'),
    relaunch: () => ipcRenderer.invoke('app:relaunch')
  }
})

```

## C.2 BLE笔迹接收 (Noble C++ Addon)

PC客户端通过Noble (Node.js BLE库) 连接智能点阵笔, 使用C++ Native Addon处理高频笔迹数据。

### C.2.1 BLE管理器实现

```

// main/ble/BleManager.ts
import noble from '@abandonware/noble'
import { EventEmitter } from 'events'
import { InkEngine } from '../native/NativeInkEngine'

const WRITECH_PEN_SERVICE_UUID = '6e400001-b5a3-f393-e0a9-e50e24dcca9e'
const INK_DATA_CHAR_UUID = '6e400002-b5a3-f393-e0a9-e50e24dcca9e'
const CONTROL_CHAR_UUID = '6e400003-b5a3-f393-e0a9-e50e24dcca9e'
const WRITECH_PEN_NAME_PREFIX = 'WritechPen-'

export class BleManager extends EventEmitter {
  private connectedPens: Map<string, noble.Peripheral> = new Map()
  private inkCharacteristics: Map<string, noble.Characteristic> = new Map()
  private scanning = false

  async initialize(): Promise<void> {
    noble.on('stateChange', (state) => {
      if (state === 'poweredOn' && this.scanning) {
        noble.startScanning([WRITECH_PEN_SERVICE_UUID], true)
      }
    })
  }

  noble.on('discover', (peripheral) => {
    const name = peripheral.advertisement.localName || ''
    if (name.startsWith(WRITECH_PEN_NAME_PREFIX)) {
      this.emit('deviceFound', {
        id: peripheral.id,
        name: name,
        rssi: peripheral.rssi,
        address: peripheral.address
      })
    }
  })
}

  async startScan(): Promise<void> {
    this.scanning = true
    if (noble.state === 'poweredOn') {
      noble.startScanning([WRITECH_PEN_SERVICE_UUID], true)
    }
  }

  async stopScan(): Promise<void> {
    this.scanning = false
    noble.stopScanning()
  }

  async connect(peripheralId: string): Promise<void> {
    const peripheral = await this.findPeripheral(peripheralId)
    if (!peripheral) throw new Error('Device not found: ' + peripheralId)
  }
}

```

```

await new Promise<void>((resolve, reject) => {
  peripheral.connect((err) => {
    if (err) reject(err)
    else resolve()
  })
})

// 发现服务和特征
const { characteristics } = await new Promise<noble.ServicesAndCharacteristics>(
  (resolve, reject) => {
    peripheral.discoverAllServicesAndCharacteristics((err, services, chars) => {
      if (err) reject(err)
      else resolve({ services, characteristics: chars })
    })
  }
)

const inkChar = characteristics.find(c => c.uuid === INK_DATA_CHAR_UUID)
if (!inkChar) throw new Error('Ink characteristic not found')

this.connectedPens.set(peripheralId, peripheral)
this.inkCharacteristics.set(peripheralId, inkChar)

// 订阅笔迹数据通知
await new Promise<void>((resolve, reject) => {
  inkChar.subscribe((err) => {
    if (err) reject(err)
    else resolve()
  })
})

inkChar.on('data', (data: Buffer) => {
  this.processInkData(peripheralId, data)
})

peripheral.on('disconnect', () => {
  this.connectedPens.delete(peripheralId)
  this.inkCharacteristics.delete(peripheralId)
  this.emit('connectionChanged', peripheralId, false)
  // 自动重连
  setTimeout(() => this.connect(peripheralId), 3000)
})

this.emit('connectionChanged', peripheralId, true)
}

/**
 * 解析BLE笔迹数据包
 * 格式: [x:2B] [y:2B] [压力:1B] [时间戳:4B] [标志:1B] × n点
 */
private processInkData(penId: string, data: Buffer): void {
  const points: InkPoint[] = []
  for (let offset = 0; offset + 10 <= data.length; offset += 10) {
    const x = data.readUInt16BE(offset) / 65535.0
    const y = data.readUInt16BE(offset + 2) / 65535.0
    const pressure = data[offset + 4] / 255.0
    const timestamp = data.readUInt32BE(offset + 5)
    const flags = data[offset + 9]
    const isPenUp = (flags & 0x01) !== 0

    points.push({ x, y, pressure, timestamp, isPenUp })
  }

  if (points.length > 0) {
    this.emit('inkData', { penId, points })
  }
}

```

```

    }

    destroy(): void {
      noble.stopScanning()
      this.connectedPens.forEach((peripheral) => {
        peripheral.disconnect()
      })
      this.connectedPens.clear()
      this.inkCharacteristics.clear()
    }
  }
}

```

## C.3 本地数据库设计 (better-sqlite3)

PC客户端使用SQLite作为本地数据库，通过better-sqlite3驱动实现同步读写操作。

### C.3.1 数据库初始化与Schema

```

// main/database/LocalDatabase.ts
import Database from 'better-sqlite3'
import { join } from 'path'

export class LocalDatabase {
  private db: Database.Database

  constructor(dbPath: string) {
    this.db = new Database(dbPath, {
      verbose: process.env.NODE_ENV === 'development' ? console.log : undefined
    })
    this.db.pragma('journal_mode = WAL') // WAL模式，提升并发读性能
    this.db.pragma('synchronous = NORMAL') // 性能与安全的平衡
    this.db.pragma('foreign_keys = ON') // 启用外键约束
    this.db.pragma('cache_size = -32000') // 32MB页缓存
  }

  async initialize(): Promise<void> {
    this.createTables()
    this.createIndexes()
    this.runMigrations()
  }

  private createTables(): void {
    this.db.exec(`
      -- 用户信息表
      CREATE TABLE IF NOT EXISTS users (
        id TEXT PRIMARY KEY,
        username TEXT NOT NULL UNIQUE,
        display_name TEXT NOT NULL,
        role TEXT NOT NULL CHECK(role IN ('teacher', 'student', 'admin')),
        school_id TEXT,
        class_id TEXT,
        avatar_url TEXT,
        created_at INTEGER NOT NULL DEFAULT (strftime('%s', 'now')),
        updated_at INTEGER NOT NULL DEFAULT (strftime('%s', 'now')),
        sync_status TEXT NOT NULL DEFAULT 'synced' CHECK(sync_status IN ('synced', 'pending',
'conflict'))
      );

      -- 课堂记录表
      CREATE TABLE IF NOT EXISTS classroom_sessions (
        id TEXT PRIMARY KEY,
        teacher_id TEXT NOT NULL REFERENCES users(id),
        class_id TEXT NOT NULL,

```

```

classroom_name TEXT NOT NULL,
start_time INTEGER NOT NULL,
end_time INTEGER,
status TEXT NOT NULL DEFAULT 'active' CHECK(status IN ('active', 'ended', 'archived')),
student_count INTEGER DEFAULT 0,
metadata TEXT, -- JSON扩展字段
sync_status TEXT NOT NULL DEFAULT 'pending',
created_at INTEGER NOT NULL DEFAULT (strftime('%s', 'now'))
);

-- 笔迹数据表 (高频写入, 使用INTEGER主键)
CREATE TABLE IF NOT EXISTS ink_strokes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id TEXT NOT NULL REFERENCES classroom_sessions(id),
    student_id TEXT NOT NULL,
    pen_id TEXT NOT NULL,
    stroke_data BLOB NOT NULL, -- 压缩后的笔迹点二进制数据
    point_count INTEGER NOT NULL,
    start_time INTEGER NOT NULL,
    end_time INTEGER NOT NULL,
    bounding_box TEXT, -- JSON: {x,y,w,h}
    sync_status TEXT NOT NULL DEFAULT 'pending',
    created_at INTEGER NOT NULL DEFAULT (strftime('%s', 'now'))
);

-- 作业记录表
CREATE TABLE IF NOT EXISTS homework_records (
    id TEXT PRIMARY KEY,
    session_id TEXT REFERENCES classroom_sessions(id),
    student_id TEXT NOT NULL,
    assignment_id TEXT NOT NULL,
    submit_time INTEGER,
    ink_stroke_ids TEXT, -- JSON数组: 关联的笔迹ID
    score REAL,
    feedback TEXT,
    status TEXT NOT NULL DEFAULT 'submitted'
    CHECK(status IN ('draft', 'submitted', 'graded', 'returned')),
    sync_status TEXT NOT NULL DEFAULT 'pending',
    created_at INTEGER NOT NULL DEFAULT (strftime('%s', 'now'))
);

-- 同步日志表
CREATE TABLE IF NOT EXISTS sync_logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    table_name TEXT NOT NULL,
    record_id TEXT NOT NULL,
    operation TEXT NOT NULL CHECK(operation IN ('insert', 'update', 'delete')),
    sync_time INTEGER,
    error_message TEXT,
    retry_count INTEGER DEFAULT 0,
    created_at INTEGER NOT NULL DEFAULT (strftime('%s', 'now'))
);
`)
}

private createIndexes(): void {
    this.db.exec(`
        CREATE INDEX IF NOT EXISTS idx_ink_strokes_session ON ink_strokes(session_id);
        CREATE INDEX IF NOT EXISTS idx_ink_strokes_student ON ink_strokes(student_id);
        CREATE INDEX IF NOT EXISTS idx_ink_strokes_sync ON ink_strokes(sync_status) WHERE sync_status =
'pending';
        CREATE INDEX IF NOT EXISTS idx_homework_student ON homework_records(student_id);
        CREATE INDEX IF NOT EXISTS idx_homework_assignment ON homework_records(assignment_id);
        CREATE INDEX IF NOT EXISTS idx_sessions_teacher ON classroom_sessions(teacher_id);
        CREATE INDEX IF NOT EXISTS idx_sessions_time ON classroom_sessions(start_time DESC);
    `)
}

```

```

}

/** 批量插入笔迹（使用预编译语句，性能显著优于逐条插入） */
saveStrokeBatch(strokes: StrokeRecord[]): void {
  const stmt = this.db.prepare(`
    INSERT INTO ink_strokes
      (session_id, student_id, pen_id, stroke_data, point_count,
       start_time, end_time, bounding_box, sync_status)
    VALUES
      (@sessionId, @studentId, @penId, @strokeData, @pointCount,
       @startTime, @endTime, @boundingBox, 'pending')
  `)
  const insertMany = this.db.transaction((records: StrokeRecord[]) => {
    for (const r of records) stmt.run(r)
  })
  insertMany(strokes)
}

/** 查询待同步的笔迹数据（批量上传） */
getPendingStrokes(limit = 100): StrokeRecord[] {
  return this.db.prepare(`
    SELECT * FROM ink_strokes
    WHERE sync_status = 'pending'
    ORDER BY created_at ASC
    LIMIT ?
  `).all(limit) as StrokeRecord[]
}

/** 标记笔迹为已同步 */
markStrokesSynced(ids: number[]): void {
  const placeholders = ids.map(() => '?').join(',')
  this.db.prepare(`
    UPDATE ink_strokes SET sync_status = 'synced' WHERE id IN (${placeholders})
  `).run(...ids)
}

close(): void {
  this.db.close()
}
}

```

## C.4 数据同步服务

PC客户端实现离线优先（Offline-First）策略，本地操作优先写入SQLite，后台服务定期将数据上传到云端。

### C.4.1 同步服务实现

```

// main/sync/SyncService.ts
import { LocalDatabase } from '../database/LocalDatabase'
import axios, { AxiosInstance } from 'axios'
import { EventEmitter } from 'events'
import pako from 'pako' // 数据压缩

export class SyncService extends EventEmitter {
  private db: LocalDatabase
  private http: AxiosInstance
  private syncTimer: NodeJS.Timer | null = null
  private syncing = false

  private static readonly SYNC_INTERVAL_MS = 30_000 // 30秒同步一次
  private static readonly BATCH_SIZE = 50 // 每批上传50条记录
  private static readonly MAX_RETRY = 3

```

```

constructor(db: LocalDatabase, baseUrl: string) {
  super()
  this.db = db
  this.http = axios.create({
    baseUrl: baseUrl,
    timeout: 30_000,
    headers: {
      'Content-Type': 'application/json',
      'X-Client-Type': 'PC_APP',
      'X-App-Version': app.getVersion()
    }
  })
}

start(authToken: string): void {
  this.http.defaults.headers.common['Authorization'] = `Bearer ${authToken}`
  this.syncTimer = setInterval(() => this.syncAll(), SyncService.SYNC_INTERVAL_MS)
  // 立即执行一次同步
  this.syncAll()
}

stop(): void {
  if (this.syncTimer) {
    clearInterval(this.syncTimer)
    this.syncTimer = null
  }
}

async syncAll(): Promise<void> {
  if (this.syncing) return
  this.syncing = true
  let totalSynced = 0
  let totalErrors = 0

  try {
    this.emit('progress', { status: 'syncing', message: '正在同步笔迹数据...' })

    // 1. 上传待同步的笔迹数据
    while (true) {
      const pending = this.db.getPendingStrokes(SyncService.BATCH_SIZE)
      if (pending.length === 0) break

      // 压缩笔迹二进制数据后上传
      const payload = pending.map(s => ({
        ...s,
        strokeData: Buffer.from(pako.deflate(s.strokeData)).toString('base64')
      }))

      const response = await this.http.post('/api/v1/strokes/batch', payload)
      if (response.status === 200) {
        const syncedIds = pending.map(s => s.id!)
        this.db.markStrokesSynced(syncedIds)
        totalSynced += pending.length
      }

      this.emit('progress', {
        status: 'syncing',
        message: `已同步 ${totalSynced} 条笔迹`,
        synced: totalSynced
      })
    }

    // 2. 从云端拉取最新数据（新消息、批改结果等）
    await this.pullUpdates()

    this.emit('progress', {

```

```

        status: 'completed',
        message: `同步完成, 上传 ${totalSynced} 条, 错误 ${totalErrors} 条`,
        synced: totalSynced,
        errors: totalErrors
    })

    } catch (error) {
        totalErrors++
        this.emit('progress', {
            status: 'error',
            message: '同步失败: ' + (error as Error).message
        })
    } finally {
        this.syncing = false
    }
}

private async pullUpdates(): Promise<void> {
    // 拉取最新的批改结果
    const lastSyncTime = this.db.getLastSyncTime('homework_records')
    const response = await this.http.get('/api/v1/homework/updates', {
        params: { since: lastSyncTime }
    })
    if (response.data.records?.length > 0) {
        this.db.upsertHomeworkRecords(response.data.records)
    }
}
}

```

## C.5 PDF/图片导出功能

```

// main/export/ExportService.ts
import { BrowserWindow, ipcMain } from 'electron'
import { join } from 'path'
import * as fs from 'fs/promises'

export class ExportService {

    /**
     * 将Canvas笔迹内容导出为PDF
     * 原理: 创建隐藏的BrowserWindow, 加载笔迹数据渲染后调用printToPDF
     */
    static async exportToPdf(content: ExportContent, outputPath: string): Promise<void> {
        const hiddenWin = new BrowserWindow({
            show: false,
            webPreferences: {
                offscreen: true
            }
        })

        await hiddenWin.loadFile(join(__dirname, '../renderer/export.html'))

        // 向隐藏窗口注入笔迹数据
        await hiddenWin.webContents.executeJavaScript(
            `window.renderExportContent(${JSON.stringify(content)})`
        )

        // 等待渲染完成
        await new Promise(resolve => setTimeout(resolve, 500))

        const pdfData = await hiddenWin.webContents.printToPDF({
            pageSize: 'A4',
            printBackground: true,
            marginsType: 1 // 最小边距
        })
    }
}

```



```

    })

    await fs.writeFile(outputPath, pdfData)
    hiddenWin.destroy()
  }

  /**
   * 将笔迹画布导出为PNG图片
   * 使用Electron的capturePage API截取渲染内容
   */
  static async exportToImage(content: ExportContent, outputPath: string): Promise<void> {
    const hiddenWin = new BrowserWindow({
      width: content.width || 2480,    // A4宽度 @ 300dpi
      height: content.height || 3508, // A4高度 @ 300dpi
      show: false,
      webPreferences: { offscreen: true }
    })

    await hiddenWin.loadFile(join(__dirname, '../renderer/export.html'))
    await hiddenWin.webContents.executeJavaScript(
      `window.renderExportContent(${JSON.stringify(content)})`
    )
    await new Promise(resolve => setTimeout(resolve, 500))

    const nativeImage = await hiddenWin.webContents.capturePage({
      x: 0, y: 0, width: content.width || 2480, height: content.height || 3508
    })

    await fs.writeFile(outputPath, nativeImage.toPNG())
    hiddenWin.destroy()
  }
}

```

## C.6 React渲染进程核心模块

### C.6.1 Canvas笔迹绘制组件

```

// renderer/src/components/InkCanvas.tsx
import React, { useRef, useEffect, useCallback } from 'react'
import { useInkStore } from '../store/inkStore'

interface InkCanvasProps {
  width: number
  height: number
  studentId?: string // 指定学生ID时只显示该学生笔迹
  readonly?: boolean
}

export const InkCanvas: React.FC<InkCanvasProps> = ({
  width, height, studentId, readonly = false
}) => {
  const canvasRef = useRef<HTMLCanvasElement>(null)
  const contextRef = useRef<CanvasRenderingContext2D | null>(null)
  const { strokes, addPoint, endStroke } = useInkStore()

  // 初始化Canvas上下文
  useEffect(() => {
    const canvas = canvasRef.current
    if (!canvas) return
    const ctx = canvas.getContext('2d', { willReadFrequently: false })
    if (!ctx) return
    ctx.lineCap = 'round'
    ctx.lineJoin = 'round'

```

```
    ctx.strokeStyle = '#1a1a2e'
    contextRef.current = ctx
  }, [])

// 当笔迹数据更新时重新渲染
useEffect(() => {
  const canvas = canvasRef.current
  const ctx = contextRef.current
  if (!canvas || !ctx) return

  ctx.clearRect(0, 0, width, height)
  ctx.fillStyle = 'ffffff'
  ctx.fillRect(0, 0, width, height)

  const filteredStrokes = studentId
    ? strokes.filter(s => s.studentId === studentId)
    : strokes

  for (const stroke of filteredStrokes) {
    if (stroke.points.length < 2) continue
    ctx.strokeStyle = stroke.color
    ctx.beginPath()
    const pts = stroke.points
    ctx.moveTo(pts[0].x * width, pts[0].y * height)
    for (let i = 1; i < pts.length - 1; i++) {
      const midX = ((pts[i].x + pts[i+1].x) / 2) * width
      const midY = ((pts[i].y + pts[i+1].y) / 2) * height
      ctx.quadraticCurveTo(
        pts[i].x * width, pts[i].y * height, midX, midY
      )
      ctx.lineWidth = 1.5 + pts[i].pressure * 3
    }
    ctx.stroke()
  }
}, [strokes, studentId, width, height])

return (
  <canvas
    ref={canvasRef}
    width={width}
    height={height}
    className="ink-canvas"
    style={{ border: '1px solid #e0e0e0', borderRadius: 4 }}
  />
)
}
```

## 附录D 完整操作手册

### D.1 安装与初始配置

#### D.1.1 支持的操作系统

平台	最低版本	推荐版本
Windows	Windows 10 (1903)	Windows 11 22H2
macOS	macOS 11.0 (Big Sur)	macOS 13.x (Ventura)

平台	最低版本	推荐版本
Linux	Ubuntu 20.04 LTS	Ubuntu 22.04 LTS

D.1.2 安装步骤

**Windows安装：** 1. 下载 `writtech-pc-setup-x.x.x.exe` 安装包。 2. 双击运行安装程序，选择安装目录（默认 `C:\Program Files\Writtech PC`）。 3. 勾选"创建桌面快捷方式"和"开机自启动"（可选）。 4. 点击"安装"，等待安装完成（约30秒）。 5. 点击"完成"，勾选"立即启动"。

**macOS安装：** 1. 下载 `writtech-pc-x.x.x.dmg` 磁盘镜像。 2. 双击打开DMG文件，将 Writtech 图标拖入 Applications 文件夹。 3. 首次启动时，macOS提示"无法打开，因为它来自身份不明的开发者"。 4. 打开"系统偏好设置→安全性与隐私→通用"，点击"仍然打开"。 5. 应用成功启动。

**Linux安装：**

```
# Ubuntu/Debian
sudo dpkg -i writtech-pc_x.x.x_amd64.deb
sudo apt-get install -f # 修复依赖

# 或使用AppImage（免安装）
chmod +x writtech-pc-x.x.x.AppImage
./writtech-pc-x.x.x.AppImage
```

D.1.3 首次登录

- 1. 启动应用，显示登录界面。
- 2. 输入学校管理员分配的账号和密码。
- 3. 可选"记住登录状态"（有效期30天）。
- 4. 点击"登录"，首次登录需要下载数据（约30–60秒）。
- 5. 登录成功后进入主界面。

D.2 智能笔连接

D.2.1 连接步骤

- 1. 打开自然写PC应用，点击顶部工具栏"连接笔"按钮（钢笔图标）。
- 2. 确保智能点阵笔蓝牙已开启（笔盖指示灯闪烁蓝色）。
- 3. 设备列表显示附近的自然写智能笔（以"WrittechPen-"开头）。
- 4. 点击对应设备名称旁的"连接"按钮。
- 5. 配对过程约5–10秒，成功后指示灯变为常亮蓝色。
- 6. 状态栏显示"笔已连接：WrittechPen-XXXX，电量：85%"。

D.2.2 多笔连接（教师模式）

教师使用PC应用时可同时连接多支智能笔（最多4支），用于不同颜色批注： 1. 重复上述连接步骤连接第二支笔。 2. 在"设置→笔管理"中为每支笔分配颜色。 3. 主界面工具栏显示当前激活的笔（点击切换）。

D.3 课堂功能操作

### D.3.1 开始课堂

1. 主界面点击"新建课堂"按钮。
2. 填写课堂信息：
3. 班级（从下拉列表选择）
4. 课程名称（如"三年级语文-第5课"）
5. 预计时长（30/45/60分钟）
6. 点击"开始课堂"，系统自动创建课堂会话，生成课堂码（4位数字）。
7. 学生通过手机APP或Pad APP输入课堂码加入。
8. 教师界面左侧显示学生签到列表，右侧显示书写区域。

### D.3.2 批改作业

1. 主界面选择"作业"标签，查看待批改作业列表。
2. 点击某学生的作业，右侧显示该学生的手写作业内容。
3. 批改操作：
4. 选择红笔工具，在学生笔迹上方直接书写批注
5. 点击"正确"/"错误"按钮快速标记
6. 输入分数（0-100分）
7. 添加文字评语（支持键盘输入或语音转文字）
8. 点击"提交批改"，批改结果自动同步到学生APP。

### D.3.3 统计报告查看

1. 主界面选择"报告"标签。
2. 选择报告类型：
3. 班级报告：全班作业正确率、提交率分布图
4. 个人报告：单学生历史成绩折线图、错题分析
5. 知识点报告：按知识点统计掌握率（热力图展示）
6. 支持导出为PDF报告（菜单→导出→PDF）。

## D.4 数据管理

### D.4.1 数据备份

1. 菜单→设置→数据管理→备份数据。
2. 选择备份目录（默认"文档/WritechBackup"）。
3. 点击"立即备份"，备份文件为加密的 .wbk 格式。
4. 自动备份频率：每7天一次（可在设置中调整）。

### D.4.2 清理旧数据

1. 菜单→设置→数据管理→清理数据。
2. 选择要清理的数据范围：
3. 3个月前的笔迹数据
4. 6个月前的课堂记录
5. 已同步到云端的本地缓存

6. 确认后开始清理，进度条显示清理进度。

D.5 快捷键说明

功能	Windows/Linux	macOS
新建课堂	Ctrl+N	⌘N
保存	Ctrl+S	⌘S
撤销	Ctrl+Z	⌘Z
重做	Ctrl+Y	⌘⇧Z
放大画布	Ctrl++	⌘+
缩小画布	Ctrl+-	⌘-
全屏	F11	⌘⇧F
切换学生	Tab	Tab
切换笔颜色	1-8	1-8
橡皮擦	E	E
清屏	Ctrl+Del	⌘⌫

附录E 源代码对应关系详细说明

E.1 完整源代码文件清单

源文件	路径	功能说明
main/index.ts	src/main/index.ts	Electron主进程入口
preload/index.ts	src/preload/index.ts	contextBridge安全桥接
BleManager.ts	src/main/ble/BleManager.ts	BLE智能笔管理
LocalDatabase.ts	src/main/database/LocalDatabase.ts	SQLite本地数据库
SyncService.ts	src/main/sync/SyncService.ts	云端数据同步服务
ExportService.ts	src/main/export/ExportService.ts	PDF/图片导出
NativeInkEngine.ts	src/main/native/NativeInkEngine.ts	C++笔迹引擎桥接
AutoUpdater.ts	src/main/updater/AutoUpdater.ts	自动更新服务
InkCanvas.tsx	src/renderer/components/InkCanvas.tsx	Canvas笔迹渲染组件
inkStore.ts	src/renderer/store/inkStore.ts	Zustand笔迹状态管理
ClassroomPage.tsx	src/renderer/pages/ClassroomPage.tsx	课堂主页面

源文件	路径	功能说明
HomeworkPage.tsx	src/renderer/pages/HomeworkPage.tsx	作业管理页面
ReportPage.tsx	src/renderer/pages/ReportPage.tsx	报告查看页面
SettingsPage.tsx	src/renderer/pages/SettingsPage.tsx	设置页面
BleDevicePanel.tsx	src/renderer/components/BleDevicePanel.tsx	BLE设备面板
StudentGrid.tsx	src/renderer/components/StudentGrid.tsx	学生网格视图
ink_engine.cpp	native/src/ink_engine.cpp	C++ JNI笔迹引擎
binding.gyp	native/binding.gyp	Native Addon编译配置

## E.2 构建配置

```
// package.json (关键配置)
{
  "name": "writtech-pc",
  "version": "1.0.0",
  "main": "dist/main/index.js",
  "scripts": {
    "dev": "electron-vite dev",
    "build": "electron-vite build",
    "build:win": "npm run build && electron-builder --win",
    "build:mac": "npm run build && electron-builder --mac",
    "build:linux": "npm run build && electron-builder --linux",
    "rebuild-native": "electron-rebuild -f -w better-sqlite3,@abandonware/noble"
  },
  "dependencies": {
    "electron": "^28.0.0",
    "@abandonware/noble": "^1.9.2-15",
    "better-sqlite3": "^9.4.3",
    "axios": "^1.6.0",
    "pako": "^2.1.0",
    "react": "^18.2.0",
    "zustand": "^4.5.0"
  },
  "build": {
    "appId": "com.writtech.pc",
    "productName": "自然写互动课堂PC版",
    "win": {
      "target": "nsis",
      "icon": "build/icon.ico"
    },
    "mac": {
      "target": "dmg",
      "icon": "build/icon.icns",
      "category": "public.app-category.education"
    },
    "linux": {
      "target": ["deb", "AppImage"],
      "icon": "build/icon.png"
    }
  }
}
```

## 附录F 性能、兼容性与版本历史

### F.1 性能基准测试

测试项目	平台	配置	结果
冷启动时间	Windows	i7-1165G7 + SSD	1.8秒
冷启动时间	macOS	Apple M2	1.2秒
笔迹渲染帧率（BLE实时）	Windows	–	60fps
SQLite批量插入（1000条笔迹）	Windows	SSD	45ms
云端同步（1000条笔迹上传）	WiFi 100Mbps	–	3.2秒
PDF导出（10页作业）	macOS	–	2.1秒
内存占用（空载）	Windows	–	95MB
内存占用（50份作业展示）	Windows	–	248MB

### F.2 系统兼容性矩阵

操作系统	版本	架构	测试状态
Windows 10	21H2	x64	完全兼容
Windows 11	22H2	x64	完全兼容
macOS	12.x Monterey	Apple Silicon (M1/M2)	完全兼容
macOS	12.x Monterey	Intel x64	完全兼容
macOS	13.x Ventura	Apple Silicon	完全兼容
Ubuntu	20.04 LTS	x64	完全兼容
Ubuntu	22.04 LTS	x64	完全兼容

### F.3 主要依赖库版本

依赖包	版本	用途
electron	28.x	跨平台桌面框架
@abandonware/noble	1.9.x	BLE蓝牙通信

依赖包	版本	用途
better-sqlite3	9.x	本地SQLite数据库
react	18.x	UI渲染框架
zustand	4.x	轻量状态管理
axios	1.x	HTTP客户端
pako	2.x	gzip数据压缩
electron-builder	24.x	安装包构建工具
electron-vite	1.x	快速开发构建工具
vite	5.x	前端构建工具
typescript	5.x	类型安全语言

F.4 IPC通道列表

通道名	方向	说明
ble:startScan	渲染→主	触发BLE扫描
ble:stopScan	渲染→主	停止BLE扫描
ble:connect	渲染→主	连接指定BLE设备
ble:deviceFound	主→渲染	发现新BLE设备通知
ble:inkData	主→渲染	推送笔迹数据
ble:connectionChanged	主→渲染	设备连接状态变更通知
db:saveStroke	渲染→主	保存笔迹到本地数据库
db:getStrokes	渲染→主	查询笔迹记录
sync:syncNow	渲染→主	触发立即同步
sync:progress	主→渲染	同步进度通知
files:exportToPdf	渲染→主	导出PDF文件
app:getVersion	渲染→主	获取应用版本号

F.5 版本历史

版本	日期	平台	变更说明
V0.5 Beta	2025-08-01	Win/Mac	Electron框架搭建，BLE连接，基础笔迹渲染
V0.8 Beta	2025-10-20	Win/Mac/Linux	作业批改、PDF导出、SQLite本地存储
V0.9 RC	2025-12-15	Win/Mac/Linux	云端同步、增量上传、自动更新



版本	日期	平台	变更说明
V1.0	2026-02-14	Win/Mac/Linux	正式版：性能优化、安全加固、完整测试覆盖

本文档版权归深圳自然写科技有限公司所有，技术细节仅用于软件著作权登记鉴别，请勿用于其他商业用途。

## 附录G 补充技术规格

### G.1 Windows驱动层集成

#### G.1.1 USB HID设备通信

PC客户端通过USB HID协议与智能笔通信：

```
// usb_hid_reader.cpp
#include <windows.h>
#include <hidsdi.h>
#include <setupapi.h>
#pragma comment(lib, "hid.lib")
#pragma comment(lib, "setupapi.lib")

class UsbHidReader {
    HANDLE device_handle_ = INVALID_HANDLE_VALUE;

    static const USHORT VENDOR_ID = 0x1234;
    static const USHORT PRODUCT_ID = 0x5678;

public:
    bool openDevice() {
        GUID hid_guid;
        HidD_GetHidGuid(&hid_guid);

        HDEVINFO device_info = SetupDiGetClassDevs(
            &hid_guid, nullptr, nullptr,
            DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);

        SP_DEVICE_INTERFACE_DATA interface_data{};
        interface_data.cbSize = sizeof(SP_DEVICE_INTERFACE_DATA);

        for (DWORD i = 0; SetupDiEnumDeviceInterfaces(device_info, nullptr,
            &hid_guid, i, &interface_data); i++) {

            // 获取设备路径
            DWORD required_size = 0;
            SetupDiGetDeviceInterfaceDetail(device_info, &interface_data,
                nullptr, 0, &required_size, nullptr);

            auto detail_data = (SP_DEVICE_INTERFACE_DETAIL_DATA*)
                malloc(required_size);
            detail_data->cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

            SetupDiGetDeviceInterfaceDetail(device_info, &interface_data,
                detail_data, required_size, nullptr, nullptr);

            HANDLE h = CreateFile(detail_data->DevicePath,
                GENERIC_READ | GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
```

```

        nullptr, OPEN_EXISTING, FILE_FLAG_OVERLAPPED, nullptr);

    free(detail_data);

    if (h != INVALID_HANDLE_VALUE) {
        HIDD_ATTRIBUTES attrs{};
        attrs.Size = sizeof(HIDD_ATTRIBUTES);
        HidD_GetAttributes(h, &attrs);

        if (attrs.VendorID == VENDOR_ID &&
            attrs.ProductID == PRODUCT_ID) {
            device_handle_ = h;
            SetupDiDestroyDeviceInfoList(device_info);
            return true;
        }
        CloseHandle(h);
    }

    SetupDiDestroyDeviceInfoList(device_info);
    return false;
}

bool readReport(uint8_t* buffer, size_t size) {
    DWORD bytes_read = 0;
    OVERLAPPED ov{};
    ov.hEvent = CreateEvent(nullptr, TRUE, FALSE, nullptr);

    BOOL ok = ReadFile(device_handle_, buffer, (DWORD)size,
                      &bytes_read, &ov);

    if (!ok && GetLastError() == ERROR_IO_PENDING) {
        DWORD wait = WaitForSingleObject(ov.hEvent, 1000);
        if (wait == WAIT_OBJECT_0) {
            GetOverlappedResult(device_handle_, &ov, &bytes_read, FALSE);
            ok = TRUE;
        }
    }

    CloseHandle(ov.hEvent);
    return ok != FALSE;
}
};

```

## G.2 PDF课件渲染引擎

```

// pdf_renderer.cpp
#include <fpdfview.h> // PDFium

class PdfRenderer {
    FPDF_DOCUMENT document_ = nullptr;

public:
    bool loadFile(const wchar_t* path) {
        FPDF_InitLibrary();

        // 宽字符路径转UTF-8
        int len = WideCharToMultiByte(CP_UTF8, 0, path, -1, nullptr, 0, nullptr, nullptr);
        std::string utf8_path(len, 0);
        WideCharToMultiByte(CP_UTF8, 0, path, -1, utf8_path.data(), len, nullptr, nullptr);

        document_ = FPDF_LoadDocument(utf8_path.c_str(), nullptr);
        return document_ != nullptr;
    }
}

```

```

int getPageCount() {
    return document_ ? FPDF_GetPageCount(document_) : 0;
}

// 渲染指定页为BGRA位图
std::vector<uint8_t> renderPage(int pageIndex, int targetWidth, int targetHeight) {
    FPDF_PAGE page = FPDF_LoadPage(document_, pageIndex);
    if (!page) return {};

    FPDF_BITMAP bitmap = FPDFBitmap_Create(targetWidth, targetHeight, 1);
    FPDFBitmap_FillRect(bitmap, 0, 0, targetWidth, targetHeight, 0xFFFFFFFF);

    FPDF_RenderPageBitmap(bitmap, page, 0, 0, targetWidth, targetHeight,
        0, FPDF_ANNOT);

    const uint8_t* buf = (uint8_t*)FPDFBitmap_GetBuffer(bitmap);
    int stride = FPDFBitmap_GetStride(bitmap);

    std::vector<uint8_t> result(targetHeight * stride);
    memcpy(result.data(), buf, result.size());

    FPDFBitmap_Destroy(bitmap);
    FPDF_ClosePage(page);

    return result;
}

~PdfRenderer() {
    if (document_) FPDF_CloseDocument(document_);
    FPDF_DestroyLibrary();
}
};

```

### G.3 系统托盘与开机自启

```

// system_tray.cpp
class SystemTray {
    HWND hwnd_;
    NOTIFYICONDATA nid_{};
    HMENU popup_menu_ = nullptr;

public:
    void create(HWND hwnd, HICON icon) {
        hwnd_ = hwnd;
        nid_.cbSize = sizeof(NOTIFYICONDATA);
        nid_.hwnd = hwnd;
        nid_.uID = 1;
        nid_.uFlags = NIF_ICON | NIF_MESSAGE | NIF_TIP;
        nid_.uCallbackMessage = WM_USER + 1;
        nid_.hIcon = icon;
        wcsncpy_s(nid_.szTip, L"自然写互动课堂");
        Shell_NotifyIcon(NIM_ADD, &nid_);

        popup_menu_ = CreatePopupMenu();
        AppendMenu(popup_menu_, MF_STRING, 1001, L"打开主界面");
        AppendMenu(popup_menu_, MF_SEPARATOR, 0, nullptr);
        AppendMenu(popup_menu_, MF_STRING, 1002, L"退出");
    }

    void showContextMenu() {
        POINT pt;
        GetCursorPos(&pt);
        SetForegroundWindow(hwnd_);
    }
};

```

```

        TrackPopupMenu(popup_menu_, TPM_RIGHTBUTTON,
                        pt.x, pt.y, 0, hwnd_, nullptr);
    }

    static void setAutoStart(bool enable) {
        HKEY key;
        RegOpenKeyEx(HKEY_CURRENT_USER,
                    L"Software\\Microsoft\\Windows\\CurrentVersion\\Run",
                    0, KEY_WRITE, &key);

        if (enable) {
            wchar_t exe_path[MAX_PATH];
            GetModuleFileName(nullptr, exe_path, MAX_PATH);
            RegSetValueEx(key, L"WritechClassroom", 0, REG_SZ,
                          (const BYTE*)exe_path, (wcslen(exe_path) + 1) * sizeof(wchar_t));
        } else {
            RegDeleteValue(key, L"WritechClassroom");
        }
        RegCloseKey(key);
    }
};

```

## 附录H 补充技术规格

### H.1 Windows通知API集成

```

// windows_toast.cpp - Windows 10/11 Toast通知
#include <winrt/Windows.UI.Notifications.h>
#include <winrt/Windows.Data.Xml.Dom.h>

using namespace winrt::Windows::UI::Notifications;
using namespace winrt::Windows::Data::Xml::Dom;

class ToastNotifier {
public:
    static void showHomeworkReminder(const std::wstring& title,
                                     const std::wstring& body) {

        // 构建Toast XML模板
        std::wstring xml = LR"(
<toast>
  <visual>
    <binding template="ToastGeneric">
      <text id="1">)" + title + LR"(</text>
      <text id="2">)" + body + LR"(</text>
      <image placement="appLogoOverride" src="ms-appx:///Assets/logo.png"/>
    </binding>
  </visual>
  <actions>
    <action content="查看" activationType="foreground" arguments="view_homework"/>
    <action content="稍后" activationType="system" arguments="dismiss"/>
  </actions>
</toast>)" ;

        XmlDocument doc;
        doc.LoadXml(xml);

        auto notifier = ToastNotificationManager::CreateToastNotifier(
            L"com.writech.classroom");

        ToastNotification notification(doc);
        notifier.Show(notification);
    }
};

```

```
}  
};
```

## H.2 多显示器支持

```
// multi_monitor.cpp  
#include <windows.h>  
#include <vector>  
  
struct MonitorInfo {  
    HMONITOR handle;  
    RECT rect;  
    bool isPrimary;  
    int dpiX, dpiY;  
};  
  
std::vector<MonitorInfo> EnumerateMonitors() {  
    std::vector<MonitorInfo> monitors;  
  
    EnumDisplayMonitors(nullptr, nullptr, [](HMONITOR hMon, HDC, LPRECT lpRect, LPARAM lParam) {  
        auto* list = reinterpret_cast<std::vector<MonitorInfo*>>(lParam);  
  
        MONITORINFOEX info;  
        info.cbSize = sizeof(MONITORINFOEX);  
        GetMonitorInfo(hMon, &info);  
  
        UINT dpiX = 96, dpiY = 96;  
        GetDpiForMonitor(hMon, MDT_EFFECTIVE_DPI, &dpiX, &dpiY);  
  
        list->push_back({  
            hMon,  
            info.rcWork,  
            (info.dwFlags & MONITORINFOF_PRIMARY) != 0,  
            (int)dpiX, (int)dpiY  
        });  
        return TRUE;  
    }, reinterpret_cast<LPARAM>(&monitors));  
  
    return monitors;  
}  
  
// 将窗口移动到指定显示器  
void MoveWindowToMonitor(HWND hwnd, int monitorIndex) {  
    auto monitors = EnumerateMonitors();  
    if (monitorIndex >= monitors.size()) return;  
  
    const RECT& r = monitors[monitorIndex].rect;  
    int w = r.right - r.left;  
    int h = r.bottom - r.top;  
  
    SetWindowPos(hwnd, HWND_TOP, r.left, r.top, w, h, SWP_SHOWWINDOW);  
}
```