

自然写互动课堂应用开发SDK软件 V1.0

鉴别材料

软件名称：自然写互动课堂应用开发SDK软件

版本号：V1.0

著作权人：深圳自然写科技有限公司

开发完成日期：2024年6月

文档类型：开发者集成手册 + 接口设计说明书

目录

- 第一章 软件整体概述
 - 1.1 软件简介与功能综述
 - 1.2 软件用途与适用场景
 - 1.3 运行环境与系统要求
 - 1.4 开发语言与技术规范
 - 1.5 版本说明
- 第二章 系统架构与设计思路
 - 2.1 总体架构设计
 - 2.2 各层次详细说明
 - 2.3 核心模块架构图
 - 2.4 数据设计
 - 2.5 接口设计原则
 - 2.6 安全设计
 - 2.7 各平台输出形式
- 第三章 核心模块功能详细说明
 - 3.1 PenConnect SDK 模块
 - 3.2 StrokeRender SDK 模块
 - 3.3 OCR SDK 模块
 - 3.4 Gateway SDK 模块
 - 3.5 Cloud SDK 模块
 - 3.6 UI Component 模块

- 第四章 操作流程与使用步骤
 - 4.1 Android 集成步骤
 - 4.2 iOS 集成步骤
 - 4.3 PC（Windows/macOS/Linux）集成步骤
 - 4.4 Web（JavaScript/TypeScript）集成步骤
 - 4.5 初始化与鉴权
 - 4.6 完整集成示例
 - 4.7 错误码与异常处理
- 第五章 与源代码的对应关系
 - 5.1 模块名称与源代码文件对应表
 - 5.2 核心功能类与方法说明
 - 5.3 主要类命名规范
- 附录

第一章 软件整体概述

1.1 软件简介与功能综述

自然写互动课堂应用开发SDK（以下简称"自然写SDK"）是自然写科技为第三方开发者和教育集成商提供的一套多平台软件开发工具包。SDK 封装了自然写互动课堂系统的核心能力，包括点阵笔连接、笔迹实时渲染、手写识别（OCR）、教室网关对接和云平台数据访问等，使第三方开发者能够快速将自然写的智慧课堂能力集成到自己的教育应用中。

自然写SDK采用分层模块化架构，核心算法以 C/C++ 实现，通过各平台适配层（JNI/ObjC Bridge/FFI/WASM）对外提供 Java/Kotlin/Swift/JavaScript 等语言的统一 API，支持 Android、iOS、Windows、macOS、Linux、Web 六大平台。

SDK 模块功能综述：

SDK 模块	功能描述	支持平台
PenConnect SDK	蓝牙/WiFi 连接点阵笔，接收实时笔迹坐标数据流	Android / iOS / PC / Web
StrokeRender SDK	笔迹实时渲染与回放（支持压感、颜色、笔锋动画）	Android / iOS / PC / Web
OCR SDK	调用云端/本地手写识别（文字、数学、笔顺评分）	Android / iOS / PC / Web
Gateway SDK	对接教室网关，批量管理多支笔数据、课堂控制指令	Android / iOS / PC

SDK 模块	功能描述	支持平台
Cloud SDK	对接自然写云平台 API（用户认证、数据存取、学情查询）	全平台
UI Component	预制 UI 组件（笔迹画布、答题卡、字帖控件等）	Android / iOS / Web

SDK 整体定位：



1.2 软件用途与适用场景

主要用途：

自然写SDK面向教育行业的软件开发商、教育集成商和教育机构IT团队，提供标准化的接入方式，使其无需深入了解点阵笔通信协议、笔迹渲染算法、OCR引擎等底层技术细节，即可在自有应用中实现完整的智慧书写教学功能。

典型集成场景：

场景	接入方	使用的 SDK 模块
教育软件集成点阵笔数据采集	教育软件商	PenConnect SDK + StrokeRender SDK
在线学习平台增加手写作业功能	在线教育平台	PenConnect SDK + OCR SDK + Cloud SDK
教育硬件厂商集成书写评分功能	硬件厂商	PenConnect SDK + OCR SDK
学校自建智慧课堂系统	学校IT团队	Gateway SDK + Cloud SDK + UI Component
书法练习应用增加 AI 笔顺评分	书法类App开发者	PenConnect SDK + OCR SDK（笔顺模块）
教育平台集成学情数据	平台方	Cloud SDK（学情查询 API）

1.3 运行环境与系统要求

SDK 各平台运行环境：

平台	最低要求	接入形式
Android	Android 7.0 (API Level 24) +	AAR 包 (Maven 仓库)
iOS	iOS 13.0+	Framework / CocoaPods / Swift Package
Windows	Windows 10 64位 +	DLL 动态库
macOS	macOS 11.0 (Big Sur)+	dylib 动态库 / Framework
Linux	Ubuntu 18.04+ / 64位	.so 动态库
Web	支持 WebAssembly 的现代浏览器 (Chrome 79+, Firefox 72+)	NPM 包 / CDN

SDK 开发环境要求：

平台	开发工具	最低版本
Android	Android Studio	Hedgehog 2023.1+
Android	Gradle	8.x
iOS	Xcode	15.0+
iOS	CocoaPods	1.12.0+
Windows	Visual Studio	2022
Windows	CMake	3.25+
macOS	Xcode	15.0+
Web	Node.js	18 LTS+
Web	TypeScript	5.x

1.4 开发语言与技术规范

SDK 各层实现语言：

层次	语言	说明
核心引擎层	C/C++ (C++17)	BLE协议解析、笔迹平滑算法、坐标变换、数据编解码

层次	语言	说明
Android 适配层	Kotlin / Java (JNI)	JNI 桥接 C++ 核心，提供 Kotlin/Java API
iOS 适配层	Swift / Objective-C (FFI)	ObjC Bridge 调用 C++ 核心，提供 Swift API
Windows/Linux 适配层	C++ 导出接口 (C ABI)	导出标准 C 接口，可被 C/C++/Python/Java 调用
macOS 适配层	C++ / Swift	Framework 封装，提供 Swift/ObjC API
Web 适配层	TypeScript / WASM	Emscripten 编译 C++ 核心为 WASM，TypeScript 封装
UI 组件层	Android View / UIKit / HTML5 Canvas	各平台原生 UI 组件

版本管理规范： – 语义化版本 (Semantic Versioning)：MAJOR.MINOR.PATCH – MAJOR 版本：不兼容的 API 变更，提供迁移指南 – MINOR 版本：向后兼容的功能新增 – PATCH 版本：向后兼容的问题修复

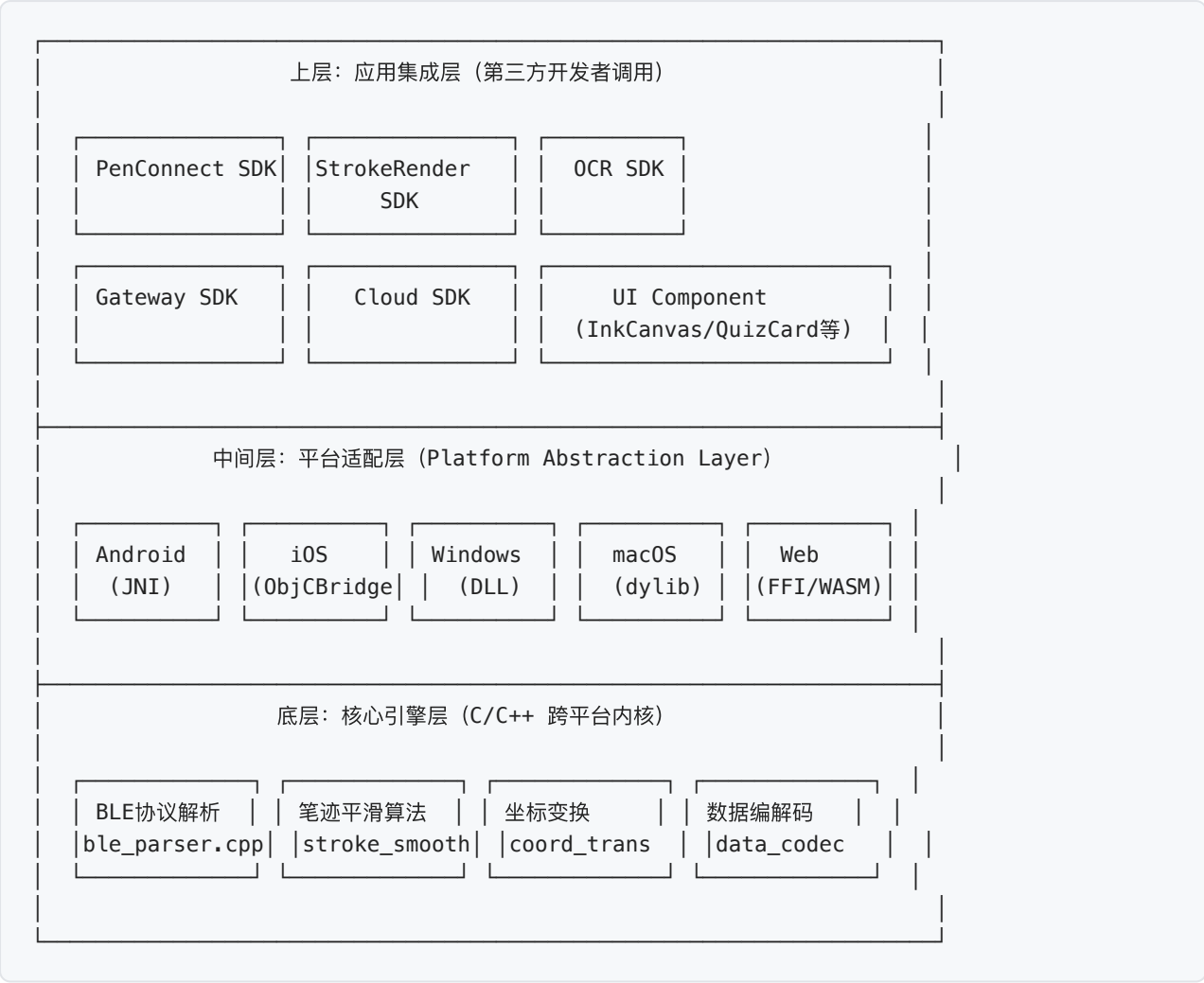
1.5 版本说明

版本	日期	说明
V1.0.0	2024-06	正式发布版本，支持 PenConnect/StrokeRender/OCR/Gateway/Cloud/UI 全模块，覆盖 Android/iOS/PC/Web
V0.9.0	2024-04	Beta：API 接口冻结，完成各平台 SDK 集成测试
V0.5.0	2024-01	Alpha：Android 和 iOS 基础功能验证

第二章 系统架构与设计思路

2.1 总体架构设计

自然写SDK采用三层模块化架构：上层为应用集成层（第三方开发者直接调用的 API），中间层为平台适配层（Platform Abstraction Layer，各平台原生代码实现），底层为核心引擎层（C/C++ 跨平台内核，实现核心算法）。



2.2 各层次详细说明

2.2.1 核心引擎层（C/C++）

核心引擎层是 SDK 的算法核心，采用纯 C/C++ 实现，不依赖任何操作系统特定 API，可跨所有目标平台编译：

BLE 协议解析模块（ble_parser）： – 解析自然写点阵笔 BLE GATT Notify 原始字节流 – 处理多包分片重组（单帧数据可能跨多个 BLE MTU 包） – 提取笔迹坐标（x/y）、压感值（pressure）、时间戳（timestamp）和抬笔标志（penUp） – 支持点阵码坐标和归一化坐标两种输出格式

笔迹平滑算法模块（stroke_smooth）： – 实现 Chaikin 曲线平滑（迭代细分，减少锯齿） – 实现三次贝塞尔曲线平滑（中点算法） – 速度敏感线宽调整（书写速度影响线宽，模拟真实书写手感） – 笔锋效果计算（笔画起止处渐细渐粗，模拟毛笔笔锋）

坐标变换模块（coord_trans）： – 点阵码坐标 → 屏幕坐标的仿射变换（对齐校准） – 归一化坐标 → 屏幕像素坐标（根据 View 尺寸缩放） – 旋转变换（支持横竖屏切换时坐标系转换） – 透视变换（修正纸张倾斜导致的坐标偏差）

数据编解码模块 (data_codec): – 笔迹数据序列化 (二进制压缩格式, Delta 编码减少数据量)
– 笔迹数据反序列化 – 与云平台传输协议 (Protobuf) 的转换 – 数据完整性校验 (CRC32)

2.2.2 平台适配层

Android 适配层 (JNI):

```
// android/jni/pen_connect_jni.cpp
// JNI 接口导出, 供 Java/Kotlin 调用 C++ 核心

extern "C" JNIEXPORT jlong JNICALL
Java_com_writtech_sdk_penconnect_NativePenEngine_createEngine(JNIEnv* env, jobject thiz)
{
    auto* engine = new writtech::PenConnectEngine();
    return reinterpret_cast<jlong>(engine);
}

extern "C" JNIEXPORT void JNICALL
Java_com_writtech_sdk_penconnect_NativePenEngine_processBleBytesNative(
    JNIEnv* env, jobject thiz, jlong handle, jbyteArray bytes) {
    auto* engine = reinterpret_cast<writtech::PenConnectEngine*>(handle);
    jbyte* data = env->GetByteArrayElements(bytes, nullptr);
    jsize len = env->GetArrayLength(bytes);
    engine->processBleBytes(reinterpret_cast<uint8_t*>(data), len);
    env->ReleaseByteArrayElements(bytes, data, JNI_ABORT);
}
```

iOS 适配层 (ObjC Bridge + Swift):

```
// ios/Sources/PenConnectBridge.swift
// Swift 包装 Objective-C Bridge (Objective-C Bridge 调用 C++ 核心)

import Foundation

@objc public class WrittechPenEngine: NSObject {
    private var nativeHandle: UnsafeMutableRawPointer

    public override init() {
        self.nativeHandle = WrittechNative.createPenEngine()
        super.init()
    }

    public func processBleBytesData(_ data: Data) {
        data.withUnsafeBytes { rawBytes in
            let ptr = rawBytes.baseAddress!.assumingMemoryBound(to: UInt8.self)
            WrittechNative.processBleBytes(nativeHandle, ptr, Int32(data.count))
        }
    }

    deinit {
        WrittechNative.destroyPenEngine(nativeHandle)
    }
}
```

```
}  
}
```

Web 适配层 (TypeScript + WASM):

```
// web/src/wasm-bridge.ts  
// TypeScript 封装 Emscripten 编译的 WASM 模块  
  
import WritechWasmModule from './writech_core.js';  
  
let wasmInstance: any = null;  
  
export async function initWasm(): Promise<void> {  
    wasmInstance = await WritechWasmModule();  
}  
  
export function processBleBytes(bytes: Uint8Array): InkPoint[] {  
    if (!wasmInstance) throw new Error('WASM not initialized');  
  
    const ptr = wasmInstance._malloc(bytes.length);  
    wasmInstance.HEAPU8.set(bytes, ptr);  
    const resultPtr = wasmInstance._processBleBytes(ptr, bytes.length);  
    wasmInstance._free(ptr);  
  
    // 解析 C 结构体数组 → TypeScript 对象数组  
    return parseInkPointArray(wasmInstance, resultPtr);  
}
```

2.2.3 应用集成层

应用集成层为第三方开发者提供简洁、语义清晰的高层 API，隐藏底层复杂性。每个 SDK 模块对应一个或多个接口类，提供统一的事件驱动回调和异步 Promise/Coroutine 接口。

设计原则：

- **单一职责：**每个 SDK 模块只负责一个功能域
- **链式配置：**Builder 模式配置 SDK，避免过长的构造函数参数
- **事件驱动：**关键状态变化通过回调/Listener/Flow/Promise 异步通知
- **错误语义：**详细的错误码和错误描述，方便开发者调试
- **线程安全：**内部使用独立工作线程，回调在 UI 线程或指定线程执行

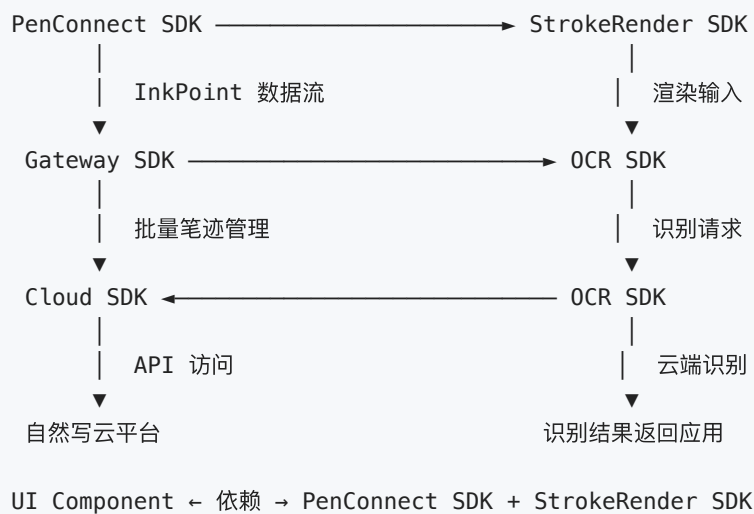
2.3 核心模块架构图

SDK 调用链路图 (Android 示例):

```
第三方 Android 应用 (Kotlin/Java)  
  | 调用 PenManager.startScan()  
  ▼  
PenConnect SDK (Android AAR)  
  | 调用 BleScanner (Android BluetoothAdapter 封装)  
  ▼  
Android 系统 Bluetooth Stack
```




SDK 模块间依赖关系:



2.4 数据设计

2.4.1 SDK 核心数据结构

跨平台统一数据类型 (C++ 定义, 各平台适配层映射):

```
// core/include/writtech_types.h

// 笔迹点 (核心数据单元)
struct InkPoint {
    float x;           // 归一化横坐标 [0.0, 1.0] (相对于书写区域宽度)
```

```

    float y;           // 归一化纵坐标 [0.0, 1.0] (相对于书写区域高度)
    float pressure;    // 压感值 [0.0, 1.0] (0=无压感)
    uint64_t timestamp; // 微秒时间戳 (相对于设备启动, 或 Unix 时间戳)
    bool is_pen_up;    // true=抬笔 (笔画结束标记)
};

// 笔画路径 (多个笔迹点组成一条笔画)
struct StrokePath {
    std::vector<InkPoint> points; // 笔迹点序列
    uint32_t color_argb;         // 笔色 (ARGB 32位)
    float base_width;           // 基础线宽 (像素, 压感在此基础上缩放)
    char pen_id[32];             // 来源笔的序列号 (多笔区分用)
};

// 点阵笔设备信息
struct PenDevice {
    char mac_address[18];        // MAC 地址 ("AA:BB:CC:DD:EE:FF")
    char device_name[64];        // 设备名称 (如"WritechPen-A1B2")
    int battery_level;           // 电量百分比 [0, 100]
    int connection_state;        // 连接状态 (枚举: DISCONNECTED/CONNECTING/CONNECTED)
    char firmware_version[16];   // 固件版本号
    char serial_number[32];      // 设备序列号
};

// 手写识别结果
struct RecognitionResult {
    int type;                    // 识别类型 (1=文字 2=数学 3=笔顺 4=评分)
    char text[1024];             // 识别文字结果 (UTF-8)
    float confidence;            // 置信度 [0.0, 1.0]
    float bbox[4];               // 边界框 [x, y, w, h] (归一化坐标)
    char detail_json[4096];      // 详细结果 (JSON, 含候选字/公式/笔顺分析)
};

// SDK 配置
struct SDKConfig {
    char app_key[64];            // AppKey (由自然写颁发)
    char app_secret[64];         // AppSecret (用于请求签名)
    char server_url[256];        // 云平台服务器地址 (可自定义私有部署)
    int log_level;               // 日志级别 (0=关闭 1=ERROR 2=WARN 3=INFO 4=DEBUG)
    bool use_local_ocr;          // 是否使用本地 OCR (需单独授权)
};

```

2.4.2 各平台语言映射

Android (Kotlin):

```

// Android SDK 数据类 (与 C++ 结构体对应, 通过 JNI 转换)

data class InkPoint(
    val x: Float,
    val y: Float,
    val pressure: Float,
    val timestamp: Long,
    val isPenUp: Boolean
)

```

```

)

data class StrokePath(
    val points: List<InkPoint>,
    val colorArgb: Int,
    val baseWidth: Float,
    val penId: String
)

data class PenDevice(
    val macAddress: String,
    val deviceName: String,
    val batteryLevel: Int,
    val connectionState: PenConnectionState,
    val firmwareVersion: String,
    val serialNumber: String
)

data class RecognitionResult(
    val type: RecognitionType,
    val text: String,
    val confidence: Float,
    val boundingBox: RectF,
    val detailJson: String
)

```

iOS (Swift):

```

// iOS SDK 数据结构

public struct InkPoint {
    public let x: Float
    public let y: Float
    public let pressure: Float
    public let timestamp: UInt64
    public let isPenUp: Bool
}

public struct StrokePath {
    public let points: [InkPoint]
    public let colorARGB: UInt32
    public let baseWidth: Float
    public let penId: String
}

public struct PenDevice {
    public let macAddress: String
    public let deviceName: String
    public let batteryLevel: Int
    public let connectionState: PenConnectionState
    public let firmwareVersion: String
    public let serialNumber: String
}

```

TypeScript (Web):

```
// TypeScript SDK 数据类型

export interface InkPoint {
  x: number;          // [0.0, 1.0]
  y: number;          // [0.0, 1.0]
  pressure: number;   // [0.0, 1.0]
  timestamp: number;  // 毫秒时间戳
  isPenUp: boolean;
}

export interface StrokePath {
  points: InkPoint[];
  colorArgb: number;
  baseWidth: number;
  penId: string;
}

export interface PenDevice {
  macAddress: string;
  deviceName: string;
  batteryLevel: number;
  connectionState: PenConnectionState;
  firmwareVersion: string;
  serialNumber: string;
}

export interface RecognitionResult {
  type: RecognitionType;
  text: string;
  confidence: number;
  boundingBox: { x: number; y: number; w: number; h: number };
  detailJson: string;
}
```

2.5 接口设计原则

SDK API 设计遵循以下原则:

1. **最小接入成本**: 核心功能 3 行代码可完成初始化和基本调用
2. **渐进式复杂度**: 基础功能简单易用, 高级定制通过可选配置暴露
3. **一致性**: 各平台 API 保持语义等价, 命名风格遵循各平台惯例
4. **不可变语义**: 数据类使用不可变对象 (Kotlin data class / Swift struct / TypeScript readonly), 避免副作用
5. **取消支持**: 所有异步操作支持取消 (Android Coroutine Job / iOS Task / Web AbortController)
6. **背压机制**: 笔迹数据流支持背压 (Flow backpressure / RxJava Flowable / Web Stream FIFO)

2.6 安全设计

接入认证 (AppKey + AppSecret 签名):

请求签名算法:

1. 将请求参数按 key 字典序排序
2. 拼接字符串: {appKey}{timestamp}{排序后的参数键值对}
3. 使用 AppSecret 对拼接字符串进行 HMAC-SHA256 签名
4. 将签名 (Base64 编码) 附加到请求头: X-Writech-Signature
5. 服务端验证签名, 防止请求被篡改或重放攻击

示例 (Kotlin):

```
val timestamp = System.currentTimeMillis().toString()
val params = sortedMapOf("action" to "ocr", "appKey" to appKey)
val stringToSign = appKey + timestamp + params.entries.joinToString("") {
    "${it.key}${it.value}" }
val signature = hmacSha256(stringToSign, appSecret).toBase64()
```

数据保护: – SDK 本地不持久化业务数据 (笔迹、识别结果), 开发者负责数据存储策略 – 仅缓存必要的 SDK 运行配置 (AppKey、服务器地址) – 缓存数据通过 Android EncryptedSharedPreferences / iOS Keychain 加密存储

代码保护: – C++ 核心库以编译后的 .so / .dylib / .dll 形式分发, 不附带源码 – Android Java/Kotlin 层通过 ProGuard/R8 混淆 – iOS Swift 代码编译为二进制 Framework, 不包含源码 – Web WASM 模块进行代码混淆 (Emscripten 优化编译)

沙箱隔离: – SDK 在独立线程运行, C++ 核心库中的异常由 JNI/ObjC Bridge 捕获并转换, 不影响宿主应用主线程 – SDK 资源 (线程/内存) 在 release() 后完全释放, 不留后台线程

2.7 各平台输出形式

平台	输出形式	引入方式
Android	AAR 包 (含 .so for arm64-v8a/armeabi-v7a/x86_64)	Maven 仓库 implementation 'com.writech.sdk:...'
iOS	XCFramework (含 arm64/x86_64 slice)	CocoaPods pod 'WritechSDK' / Swift Package
Windows	x64 DLL + 头文件	CMake find_package / 手动链接
macOS	Universal dylib + Framework	CocoaPods / Swift Package / CMake
Linux	x86_64 .so + 头文件	CMake find_package / 手动链接
Web	NPM 包 (含 .wasm + .js)	npm install @writech/sdk

平台	输出形式	引入方式
配套	API 文档、示例工程、集成指南、Changelog	开发者门户网站

第三章 核心模块功能详细说明

3.1 PenConnect SDK 模块

3.1.1 模块功能描述

PenConnect SDK 是自然写SDK的基础模块，负责发现、连接自然写点阵笔并持续接收笔迹数据流。支持 BLE 蓝牙连接（主要场景）和 Wi-Fi 直连（可选，需笔固件支持）两种连接方式。

核心功能： – 扫描周围可用的自然写点阵笔（按 BLE 服务 UUID 过滤） – 建立与点阵笔的 GATT 连接并订阅笔迹 Notify Characteristic – 将原始 BLE 字节流解析为结构化的 InkPoint 数据（通过 C++ 核心引擎） – 管理连接状态（连接/断线/重连），提供状态监听 – 支持同时连接多支笔（最多 4 支）

3.1.2 完整 API 规范

Android (Kotlin) API:

```
// 1. 初始化 SDK (Application.onCreate 或首次使用前调用一次)
WritechSDK.init(context, SDKConfig(
    appKey = "your_app_key",
    appSecret = "your_app_secret",
    logLevel = LogLevel.INFO
))

// 2. 获取 PenManager 实例 (单例)
val penManager = WritechSDK.penManager

// 3. 扫描点阵笔 (Flow 流式返回发现的设备)
penManager.startScan(timeoutMs = 15_000)
    .collect { device ->
        Log.d(TAG, "发现笔: ${device.deviceName}, MAC: ${device.macAddress}")
        // 自动连接或展示给用户选择
        penManager.connect(device)
    }

// 4. 停止扫描
penManager.stopScan()

// 5. 连接指定点阵笔
```

```

penManager.connect(device)

// 6. 监听连接状态
penManager.connectionStateFlow.collect { state ->
    when (state) {
        is PenConnectionState.Connected -> showToast("笔已连接:
${state.device.deviceName}")
        is PenConnectionState.Disconnected -> showToast("笔已断开")
        is PenConnectionState.Reconnecting -> showProgress("重连中...")
        else -> {}
    }
}

// 7. 接收笔迹数据 (Flow 流)
penManager.inkDataFlow.collect { points ->
    // points: List<InkPoint>, 每批次包含 1~34 个笔迹点
    strokeRenderer.addPoints(points)
}

// 8. 读取电量
val batteryLevel = penManager.getBatteryLevel() // 0~100

// 9. 断开连接
penManager.disconnect()

// 10. 释放资源 (Activity.onDestroy 时调用)
penManager.release()

```

iOS (Swift) API:

```

// 1. 初始化
WritechSDK.shared.initialize(config: SDKConfig(
    appKey: "your_app_key",
    appSecret: "your_app_secret",
    logLevel: .info
))

// 2. 获取 PenManager
let penManager = WritechSDK.shared.penManager

// 3. 扫描点阵笔 (async/await)
penManager.scanDelegate = self
penManager.startScan(timeout: 15)

// PenManagerDelegate 回调
func penManager(_ manager: PenManager, didDiscoverPen pen: PenDevice) {
    print("发现笔: \(pen.deviceName)")
    manager.connect(pen: pen)
}

// 4. 监听连接状态 (Publisher)
penManager.connectionStatePublisher
    .sink { state in
        switch state {
            case .connected(let pen): print("已连接: \(pen.deviceName)")

```

```

        case .disconnected: print("已断开")
        default: break
    }
}
.store(in: &cancellables)

// 5. 接收笔迹数据 (Publisher)
penManager.inkDataPublisher
    .sink { points in
        self.strokeRenderer.addPoints(points)
    }
    .store(in: &cancellables)

// 6. 断开连接
penManager.disconnect()

```

TypeScript (Web) API:

```

import { WritchSDK, PenConnectionState } from '@writch/sdk';

// 1. 初始化 (Web 需要用户授权 Web Bluetooth)
await WritchSDK.init({
  appKey: 'your_app_key',
  appSecret: 'your_app_secret',
});

const penManager = WritchSDK.penManager;

// 2. 扫描并连接 (Web Bluetooth API 要求用户手势触发)
const device = await penManager.requestPen(); // 弹出浏览器选择框
await penManager.connect(device);

// 3. 监听连接状态
penManager.onConnectionStateChange = (state: PenConnectionState) => {
  console.log('连接状态: ', state);
};

// 4. 接收笔迹数据
penManager.onInkData = (points: InkPoint[]) => {
  strokeCanvas.addPoints(points);
};

// 5. 断开连接
await penManager.disconnect();

```

3.1.3 多笔管理

SDK 支持同时连接最多 4 支点阵笔，通过 `penId`（笔序列号）区分不同笔的数据：

```

// Android 多笔连接示例
val pen1 = PenDevice(macAddress = "AA:BB:CC:DD:EE:01", ...)
val pen2 = PenDevice(macAddress = "AA:BB:CC:DD:EE:02", ...)

```



```
penManager.connectMultiple(listOf(pen1, pen2))

penManager.inkDataFlow.collect { batch ->
    // InkDataBatch 包含 penId 标识来源笔
    batch.groupBy { it.penId }.forEach { (penId, points) ->
        strokeMap[penId]?.addPoints(points)
    }
}
```

3.2 StrokeRender SDK 模块

3.2.1 模块功能描述

StrokeRender SDK 提供高性能的笔迹渲染能力，支持实时渲染（书写过程）和回放渲染（查看历史书写）两种模式，支持压感线宽变化和笔锋效果。

核心功能： – 实时渲染：将 PenConnect SDK 输出的 `InkPoint` 流渲染为平滑笔迹 – 贝塞尔平滑：自动应用三次贝塞尔曲线算法消除锯齿 – 压感线宽：根据压感值动态调整笔画宽度（模拟真实书写手感） – 笔锋效果：笔画起止处线宽渐变（模拟毛笔/钢笔笔锋） – 书写回放：以指定速度重放历史笔迹，支持暂停/继续/快进 – 笔色和笔型配置：支持多种笔色、透明度、笔型（圆笔/方笔/毛笔）

3.2.2 API 规范

Android (Kotlin) API:

```
// 获取 StrokeCanvas 实例（绑定到 View）
val strokeCanvas = WritechSDK.strokeRender.createCanvas(
    targetView = inkCanvasView,
    config = CanvasConfig(
        defaultColor = Color.BLACK,
        defaultWidth = 4f,          // dp
        enablePressure = true,     // 启用压感线宽
        enableTaper = true         // 启用笔锋效果
    )
)

// 方式1: 直接绑定 PenManager（自动接收笔迹）
strokeCanvas.bindPenManager(penManager)

// 方式2: 手动添加笔迹点（开发者自行接收数据）
strokeCanvas.beginStroke(penId = "pen_001", color = Color.BLACK, width = 4f)
strokeCanvas.addPoints(points)
strokeCanvas.endStroke()

// 书写回放
strokeCanvas.replay()
```

```

        strokes = savedStrokes,      // 历史笔迹数据
        speed = 1.5f,                 // 回放速度 (1.0=原速, 0.5=半速, 2.0=两倍速)
        onComplete = { println("回放完成") }
    )

    // 暂停/继续回放
    strokeCanvas.pauseReplay()
    strokeCanvas.resumeReplay()

    // 清除画布
    strokeCanvas.clear()

    // 导出笔迹为 Bitmap
    val bitmap: Bitmap = strokeCanvas.exportBitmap()

    // 获取当前所有笔画数据 (用于保存)
    val strokes: List<StrokePath> = strokeCanvas.getAllStrokes()

    // 撤销/重做
    strokeCanvas.undo()
    strokeCanvas.redo()

    // 释放资源
    strokeCanvas.release()

```

TypeScript (Web) API:

```

import { StrokeCanvas, CanvasConfig } from '@writech/sdk';

// 绑定 HTML Canvas 元素
const strokeCanvas = WritechSDK.strokeRender.createCanvas({
    element: document.getElementById('ink-canvas') as HTMLCanvasElement,
    config: {
        defaultColor: '#000000',
        defaultWidth: 4,
        enablePressure: true,
        enableTaper: true,
        renderMode: 'webgl2', // 使用 WebGL2 加速渲染 (可选 'canvas2d')
    },
});

// 绑定 PenManager 自动接收笔迹
strokeCanvas.bindPenManager(penManager);

// 书写回放
await strokeCanvas.replay({
    strokes: savedStrokes,
    speed: 1.0,
});

// 导出为 PNG Blob
const blob = await strokeCanvas.exportBlob('image/png');

```

```
// 获取笔画数据 (JSON 序列化后可保存到服务器)
const strokesJson = JSON.stringify(strokeCanvas.getAllStrokes());
```

3.3 OCR SDK 模块

3.3.1 模块功能描述

OCR SDK 提供手写内容的智能识别能力，调用自然写云端 AI 识别引擎（或本地离线识别引擎，需额外授权），支持汉字识别、数学公式识别和笔顺评分三种识别类型。

识别类型说明：

识别类型	功能	典型应用
文字识别 (TextOCR)	识别手写汉字、字母、数字	作业批改、字迹转文字
数学识别 (MathOCR)	识别手写数学表达式（加减乘除、分数、根号等）	数学作业识别与批改
笔顺评分 (StrokeOrder)	评估汉字书写笔顺是否正确，给出书写质量分数	字帖练习评分

3.3.2 API 规范

Android (Kotlin) API:

```
val ocrEngine = WritechSDK.ocrEngine

// 1. 文字识别 (异步, 返回识别结果)
val result = ocrEngine.recognizeText(
    strokes = strokeCanvas.getAllStrokes(),
    options = TextOCROptions(
        language = "zh-CN",           // 识别语言
        candidates = 5,               // 返回候选字数量 (最多10个)
        contextHint = "春夏秋冬"     // 上下文提示 (提升识别准确率, 可选)
    )
)
// result.text → 最优识别文字
// result.confidence → 置信度
// result.candidates → 候选字列表 (含各自置信度)

// 2. 数学表达式识别
val mathResult = ocrEngine.recognizeMath(
    strokes = mathStrokes,
    options = MathOCROptions(
        grade = "primary_3",         // 年级提示 (影响识别范围)
        returnLatex = true           // 同时返回 LaTeX 格式
    )
)
```

```

// mathResult.text → 识别结果 (如"3 + 5 = 8")
// mathResult.latex → LaTeX 格式 (如"3 + 5 = 8")
// mathResult.isCorrect → 算式是否成立 (Boolean)

// 3. 笔顺评分
val orderResult = ocrEngine.evaluateStrokeOrder(
    character = "春",           // 目标汉字
    strokes = writtenStrokes,   // 学生书写的笔画序列
    strict = false              // strict=true 严格模式 (同笔顺才算对)
)
// orderResult.score → 综合评分 [0, 100]
// orderResult.strokeOrderScore → 笔顺分 [0, 40]
// orderResult.shapeScore → 字形分 [0, 35]
// orderResult.proportionScore → 比例分 [0, 25]
// orderResult.errorDetails → 错误详情列表 (每条含错误类型和建议)

// 4. 批量识别 (优化网络请求)
val batchResults = ocrEngine.recognizeBatch(
    items = listOf(
        BatchItem(type = OcrType.TEXT, strokes = strokes1),
        BatchItem(type = OcrType.MATH, strokes = strokes2),
    )
)

// 5. 本地 OCR (需开启本地识别授权)
val localResult = ocrEngine.recognizeTextLocal(strokes = strokes)

```

TypeScript (Web) API:

```

const ocrEngine = WritechSDK.ocrEngine;

// 文字识别
const result = await ocrEngine.recognizeText({
    strokes: strokeCanvas.getAllStrokes(),
    options: {
        language: 'zh-CN',
        candidates: 5,
    },
});
console.log('识别结果: ', result.text, '置信度: ', result.confidence);

// 笔顺评分
const scoreResult = await ocrEngine.evaluateStrokeOrder({
    character: '春',
    strokes: writtenStrokes,
});
console.log('评分: ', scoreResult.score, '错误: ', scoreResult.errorDetails);

```

3.4 Gateway SDK 模块

3.4.1 模块功能描述

Gateway SDK 用于对接自然写教室网关，支持批量管理多支点阵笔数据（通过网关汇聚）和发送课堂控制指令（发题、收卷、分组等），主要用于教室多用户场景（黑板端、PC 端等）。

与 PenConnect SDK 的区别：

维度	PenConnect SDK	Gateway SDK
连接对象	直接连接点阵笔（BLE）	连接教室网关（WebSocket）
适用场景	学生/教师个人设备（手机/Pad）	教室公共设备（黑板/PC）
管理笔数	1~4 支（直连）	全班 30~60 支（通过网关）
控制能力	仅数据接收	数据接收 + 课堂控制指令

3.4.2 API 规范

```
// Android (Kotlin) 示例
val gatewayClient = WritechSDK.gatewayClient

// 1. 发现并连接教室网关（mDNS 自动发现）
gatewayClient.startDiscovery()
    .collect { gateways ->
        val myGateway = gateways.find { it.roomName == "三年级2班" }
        myGateway?.let { gatewayClient.connect(it) }
    }

// 2. 手动指定 IP 连接
gatewayClient.connectByIp(ip = "192.168.1.100", port = 8080)

// 3. 接收全班笔迹数据（按学生ID分流）
gatewayClient.classroomInkFlow.collect { batch ->
    batch.studentStrokes.forEach { (studentId, points) ->
        studentCanvasMap[studentId]?.addPoints(points)
    }
}

// 4. 发送课堂控制指令
// 发布答题
gatewayClient.issueQuiz(QuizCommand(
    quizId = UUID.randomUUID().toString(),
    type = QuizType.CHOICE,
    content = "以下哪个字有9画？",
    options = listOf("春", "秋", "冬", "夏"),
    correctAnswer = "A",
    durationSeconds = 60
))

// 收卷
gatewayClient.collectQuiz(quizId = "xxx")

// 暂停课堂（暂停笔迹推送）
```

```

gatewayClient.pauseSession()

// 恢复课堂
gatewayClient.resumeSession()

// 5. 获取教室内在线学生列表
val students = gatewayClient.getOnlineStudents()

// 6. 接收课堂事件（答题提交、学生上下线等）
gatewayClient.classroomEventFlow.collect { event ->
    when (event) {
        is ClassroomEvent.StudentJoined -> updateStudentList(event.student)
        is ClassroomEvent.QuizAnswerSubmitted -> updateQuizStats(event.answer)
        is ClassroomEvent.StudentLeft -> removeStudentFromList(event.studentId)
    }
}

```

3.5 Cloud SDK 模块

3.5.1 模块功能描述

Cloud SDK 封装了自然写云平台的 API 接口，提供用户认证、笔迹数据上传与下载、学情查询和资源管理等功能，让第三方应用无需处理底层 HTTP 细节即可访问云平台能力。

3.5.2 API 规范

Android (Kotlin) API:

```

val cloudClient = WritechSDK.cloudClient

// ===== 认证模块 =====

// 1. 初始化 (AppKey 签名认证)
cloudClient.init(appKey = "your_key", appSecret = "your_secret")

// 2. 用户登录 (已有自然写账号)
val loginResult = cloudClient.auth.login(
    username = "student_001",
    password = "password123"
)
// loginResult.token -> JWT Token (后续请求自动携带)
// loginResult.userInfo -> 用户信息 (角色/学校/班级等)

// 3. SSO 集成 (第三方系统已有账号体系)
val ssoResult = cloudClient.auth.ssoLogin(
    thirdPartyToken = "your_system_token",
    platform = "your_platform_id"
)

// 4. 登出

```

```
cloudClient.auth.logout()

// ===== 数据模块 =====

// 5. 上传笔迹数据（学生完成书写后调用）
val uploadResult = cloudClient.data.uploadInk(
    assignmentId = "assignment_001",
    pageIndex = 0,
    strokes = strokeCanvas.getAllStrokes(),
    metadata = InkMetadata(
        studentId = "student_001",
        penId = penManager.connectedPen?.serialNumber,
        writingDuration = 120_000L // 书写时长（毫秒）
    )
)
// uploadResult.inkId → 云平台分配的笔迹ID（用于后续查询）

// 6. 下载笔迹数据
val strokes = cloudClient.data.downloadInk(inkId = "ink_001")
strokeCanvas.drawStrokes(strokes)

// 7. 提交作业
val submitResult = cloudClient.data.submitAssignment(
    assignmentId = "assignment_001",
    inkIds = listOf("ink_001", "ink_002") // 多页笔迹
)

// ===== 学情模块 =====

// 8. 获取学生学情报告
val report = cloudClient.report.getStudentReport(
    studentId = "student_001",
    startDate = "2024-03-01",
    endDate = "2024-03-31"
)
// report.totalPracticeTime → 练字总时长
// report.averageScore → 平均分
// report.masteredCharacters → 已掌握汉字列表
// report.weakPoints → 薄弱知识点

// 9. 获取班级学情概览
val classReport = cloudClient.report.getClassReport(
    classId = "class_001",
    assignmentId = "assignment_001"
)
// classReport.submissionRate → 提交率
// classReport.averageScore → 班级平均分
// classReport.scoreDistribution → 分数段分布

// ===== 资源模块 =====

// 10. 获取字帖列表
val templates = cloudClient.resource.getCalligraphyTemplates(
    grade = "grade_3",
    subject = "chinese"
)
```

```
// 11. 下载字帖内容
val template = cloudClient.resource.downloadCalligraphy(templateId = "template_001")
```

TypeScript (Web) API:

```
const cloudClient = WritechSDK.cloudClient;

// 初始化
cloudClient.init({ appKey: 'your_key', appSecret: 'your_secret' });

// 用户登录
const { token, userInfo } = await cloudClient.auth.login({
  username: 'student_001',
  password: 'password123',
});

// 上传笔迹
const { inkId } = await cloudClient.data.uploadInk({
  assignmentId: 'assignment_001',
  pageIndex: 0,
  strokes: strokeCanvas.getAllStrokes(),
});

// 获取学情报告
const report = await cloudClient.report.getStudentReport({
  studentId: 'student_001',
  startDate: '2024-03-01',
  endDate: '2024-03-31',
});
console.log('练字时长: ', report.totalPracticeTime, '分钟');
```

3.6 UI Component 模块

3.6.1 模块功能描述

UI Component 模块提供一组预制的 UI 控件，帮助第三方开发者快速构建智慧书写相关的界面，无需自行开发复杂的笔迹渲染控件和答题卡控件。

预制组件列表：

组件名称	平台	功能描述
InkCanvasView	Android/iOS/Web	笔迹书写画布（集成 PenConnect + StrokeRender）
StrokeReplayView	Android/iOS/Web	笔迹回放控件（含播放/暂停/进度条）
CalligraphyView	Android/iOS/Web	字帖练习控件（参考字+书写区+笔顺指导）

组件名称	平台	功能描述
QuizAnswerCard	Android/iOS/Web	答题卡控件（选择题/判断题/书写题）
BatteryIndicator	Android/iOS	点阵笔电量指示控件
PenScanDialog	Android/iOS	点阵笔扫描连接对话框

3.6.2 InkCanvasView 使用示例

Android (XML 布局):

```
<!-- activity_main.xml -->
<com.writech.sdk.ui.InkCanvasView
    android:id="@+id/inkCanvas"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultColor="@color/black"
    app:defaultWidth="4dp"
    app:enablePressure="true"
    app:enableTaper="true"
    app:backgroundColor="@color/white" />
```

```
// MainActivity.kt
val inkCanvas = binding.inkCanvas

// 绑定 PenManager (自动接收 BLE 笔迹)
inkCanvas.bindPenManager(penManager)

// 设置工具
inkCanvas.setTool(DrawingTool.PEN)
inkCanvas.setPenColor(Color.BLACK)

// 保存笔迹
val strokes = inkCanvas.getAllStrokes()

// 清除
inkCanvas.clear()

// 撤销
inkCanvas.undo()
```

Web (HTML):

```
<!-- 使用自定义 Web Component -->
<writech-ink-canvas
    id="inkCanvas"
    default-color="#000000"
    default-width="4"
    enable-pressure="true"
```

```
style="width: 100%; height: 500px; border: 1px solid #ccc;">
</writtech-ink-canvas>
```

```
const inkCanvas = document.getElementById('inkCanvas') as WrittechInkCanvasElement;
inkCanvas.bindPenManager(penManager);

// 监听书写事件
inkCanvas.addEventListener('stroke-end', (e: CustomEvent) => {
    const stroke: StrokePath = e.detail;
    console.log('新笔画', stroke.points.length, '个点');
});
```

3.6.3 CalligraphyView 使用示例

```
// Android (Kotlin)
val calligraphyView = binding.calligraphyView

// 加载字帖模板
calligraphyView.loadTemplate(template) // CalligraphyTemplate 对象

// 绑定笔迹输入
calligraphyView.bindPenManager(penManager)

// 监听评分结果（完成一字时触发）
calligraphyView.onScoreListener = { result ->
    showScoreDialog(result.score, result.errorDetails)
}

// 设置练习模式
calligraphyView.setMode(CalligraphyMode.STROKE_ORDER) // 笔顺模式
calligraphyView.setMode(CalligraphyMode.FREE_WRITE)   // 自由书写模式
calligraphyView.setMode(CalligraphyMode.TRACE_OVER)    // 描红模式
```

第四章 操作流程与使用步骤

4.1 Android 集成步骤

4.1.1 引入依赖

在项目 `build.gradle` 中配置 Maven 仓库：

```
// build.gradle (Project level)
repositories {
    maven { url 'https://repo.writtech.com/android' }
}
```

在模块 `build.gradle` 中引入 SDK:

```
// build.gradle (Module level)
dependencies {
    // 核心模块 (必须)
    implementation 'com.writech.sdk:pen-connect:1.0.0'
    implementation 'com.writech.sdk:stroke-render:1.0.0'

    // 可选模块 (按需引入)
    implementation 'com.writech.sdk:ocr:1.0.0'
    implementation 'com.writech.sdk:gateway:1.0.0'
    implementation 'com.writech.sdk:cloud:1.0.0'
    implementation 'com.writech.sdk:ui-component:1.0.0'
}
```

4.1.2 AndroidManifest 配置

```
<!-- AndroidManifest.xml -->
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"
    android:usesPermissionFlags="neverForLocation" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true" />
```

4.1.3 初始化

```
// MyApplication.kt (Application 类中初始化)
class MyApplication : Application() {
    override fun onCreate() {
        super.onCreate()
        WritchSDK.init(
            context = this,
            config = SDKConfig.Builder()
                .appKey("your_app_key")
                .appSecret("your_app_secret")
                .serverUrl("https://api.writech.com") // 可配置私有化部署地址
                .logLevel(if (BuildConfig.DEBUG) LogLevel.DEBUG else LogLevel.ERROR)
                .build()
        )
    }
}
```

4.2 iOS 集成步骤

4.2.1 CocoaPods 集成

在 `Podfile` 中添加:

```
# Podfile
target 'YourApp' do
  use_frameworks!
  pod 'WritechSDKPenConnect', '~> 1.0'
  pod 'WritechSDKStrokeRender', '~> 1.0'
  pod 'WritechSDKOCR', '~> 1.0'      # 可选
  pod 'WritechSDKCloud', '~> 1.0'   # 可选
  pod 'WritechSDKUI', '~> 1.0'      # 可选
end
```

执行 `pod install`。

4.2.2 Info.plist 配置

```
<!-- Info.plist -->
<key>NSBluetoothAlwaysUsageDescription</key>
<string>需要访问蓝牙以连接自然写点阵笔</string>
<key>NSLocalNetworkUsageDescription</key>
<string>需要访问局域网以连接教室网关</string>
```

4.2.3 初始化

```
// AppDelegate.swift
import WritechSDKPenConnect
import WritechSDKStrokeRender

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    func application(_ application: UIApplication,
                    didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        WritechSDK.shared.initialize(
            config: SDKConfig(
                appKey: "your_app_key",
                appSecret: "your_app_secret",
                serverUrl: "https://api.writech.com",
                logLevel: ProcessInfo.processInfo.environment["DEBUG"] != nil ? .debug :
                .error
            )
        )
        return true
    }
}
```

4.3 PC (Windows/macOS/Linux) 集成步骤

4.3.1 CMake 集成

```
# CMakeLists.txt
find_package(WrittechSDK 1.0 REQUIRED
    COMPONENTS PenConnect StrokeRender OCR Cloud)

target_link_libraries(YourApp PRIVATE
    WrittechSDK::PenConnect
    WrittechSDK::StrokeRender
    WrittechSDK::OCR
    WrittechSDK::Cloud
)
```

4.3.2 初始化 (C++ API)

```
#include <writtech/sdk.h>

int main() {
    writtech::SDKConfig config;
    config.app_key = "your_app_key";
    config.app_secret = "your_app_secret";
    config.server_url = "https://api.writtech.com";
    config.log_level = 3; // INFO

    writtech::SDK::init(config);

    auto& penManager = writtech::SDK::penManager();

    // 注册笔迹回调
    penManager.setInkCallback([](const std::vector<writtech::InkPoint>& points) {
        // 处理笔迹数据
        for (const auto& p : points) {
            printf("Point: x=%.3f y=%.3f pressure=%.3f\n", p.x, p.y, p.pressure);
        }
    });

    penManager.startScan(15000); // 扫描15秒

    // 主循环
    while (running) {
        penManager.update(); // PC 模式需主动轮询
        std::this_thread::sleep_for(std::chrono::milliseconds(16));
    }

    writtech::SDK::release();
    return 0;
}
```

4.4 Web (JavaScript/TypeScript) 集成步骤

4.4.1 NPM 安装

```
npm install @wrotech/sdk
# 或使用 yarn
yarn add @wrotech/sdk
```

4.4.2 初始化 (TypeScript)

```
// main.ts
import { WrotechSDK } from '@wrotech/sdk';

async function initWrotechSDK() {
  // WASM 模块异步加载 (必须在使用前 await)
  await WrotechSDK.init({
    appKey: 'your_app_key',
    appSecret: 'your_app_secret',
    serverUrl: 'https://api.wrotech.com',
  });

  console.log('自然写SDK初始化完成, 版本: ', WrotechSDK.version);
}

initWrotechSDK().catch(console.error);
```

4.4.3 Web Bluetooth 注意事项

```
// Web Bluetooth API 要求:
// 1. 必须在 HTTPS 环境下使用 (localhost 除外)
// 2. 必须由用户手势 (click 事件) 触发 requestDevice

document.getElementById('connectBtn')!.addEventListener('click', async () => {
  try {
    const device = await WrotechSDK.penManager.requestPen();
    await WrotechSDK.penManager.connect(device);
    console.log('连接成功: ', device.deviceName);
  } catch (err) {
    if ((err as Error).name === 'NotFoundError') {
      console.log('用户取消了设备选择');
    } else {
      console.error('连接失败: ', err);
    }
  }
});
```

4.5 初始化与鉴权

AppKey 和 AppSecret 获取: 1. 访问自然写开发者门户 (<https://dev.wrotech.com>) 2. 注册开发者账号并创建应用 3. 在应用详情页获取 AppKey 和 AppSecret 4. AppSecret 仅显示一次, 请妥善保管 (可重新生成)

注意事项： – AppKey 可以内嵌在客户端代码中（公开标识） – AppSecret 用于请求签名，移动端/Web 端建议通过代理服务器签名，不要直接内嵌在客户端 – 生产环境和测试环境使用不同的 AppKey/AppSecret

4.6 完整集成示例

Android 最小可运行示例（从零实现手写识别）：

```
class HandwritingActivity : AppCompatActivity() {

    private val penManager by lazy { WritechSDK.penManager }
    private val ocrEngine by lazy { WritechSDK.ocrEngine }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_handwriting)

        // 步骤1: 绑定笔迹画布到 View
        val inkCanvas = binding.inkCanvas
        inkCanvas.bindPenManager(penManager)

        // 步骤2: 请求 BLE 权限并开始扫描
        requestBlePermissions {
            penManager.startScan(10_000).launchIn(lifecycleScope)
        }

        // 步骤3: 连接第一个发现的笔
        penManager.startScan()
            .take(1)
            .onEach { device -> penManager.connect(device) }
            .launchIn(lifecycleScope)

        // 步骤4: 点击"识别"按钮调用 OCR
        binding.btnRecognize.setOnClickListener {
            lifecycleScope.launch {
                val result = ocrEngine.recognizeText(inkCanvas.getAllStrokes())
                binding.tvResult.text = "识别结果: ${result.text} (置信度 ${result.confidence * 100}.toInt())%"
            }
        }

        // 步骤5: 清除画布
        binding.btnClear.setOnClickListener { inkCanvas.clear() }
    }

    override fun onDestroy() {
        super.onDestroy()
        penManager.release()
    }
}
```

4.7 错误码与异常处理

SDK 统一错误码定义：

错误码范围	错误类别
1000~1099	SDK 初始化错误
2000~2099	PenConnect 连接错误
3000~3099	StrokeRender 渲染错误
4000~4099	OCR 识别错误
5000~5099	Gateway 连接错误
6000~6099	Cloud API 错误
9000~9099	通用错误（网络/权限/存储）

常见错误码说明：

错误码	名称	说明	处理建议
1001	SDK_NOT_INITIALIZED	SDK 未初始化	调用 <code>WritechSDK.init()</code> 后再使用
1002	INVALID_APP_KEY	AppKey 无效	检查 AppKey 是否正确，是否已激活
2001	BLE_NOT_SUPPORTED	设备不支持 BLE	提示用户设备不兼容
2002	BLE_PERMISSION_DENIED	蓝牙权限被拒绝	引导用户在系统设置中开启蓝牙权限
2003	PEN_NOT_FOUND	未发现点阵笔	提示用户打开点阵笔电源，检查蓝牙是否开启
2004	CONNECTION_TIMEOUT	连接超时	重试连接，或让用户重新打开点阵笔
4001	OCR_STROKE_TOO_SHORT	笔迹数据太少	引导用户书写更多内容后再识别
4002	OCR_QUOTA_EXCEEDED	识别次数超出配额	购买更多配额，或开启本地识别
6001	NETWORK_ERROR	网络请求失败	检查网络连接，重试请求
6002	UNAUTHORIZED	认证失败/Token 过期	重新登录获取新 Token
6003	SERVER_ERROR	服务器内部错误	稍后重试，如持续出现联系技术支持

Android 异常处理示例：


```

lifecycleScope.launch {
    try {
        val result = ocrEngine.recognizeText(strokes)
        showResult(result)
    } catch (e: WritechSDKException) {
        when (e.code) {
            ErrorCode.OCR_STROKE_T00_SHORT -> showToast("书写内容太少，请多写一些再识别")
            ErrorCode.OCR_QUOTA_EXCEEDED -> showToast("识别次数已用完，请联系管理员")
            ErrorCode.NETWORK_ERROR -> showToast("网络连接失败，请检查网络")
            else -> showToast("识别失败: ${e.message} (错误码: ${e.code}) ")
        }
    }
}

```

第五章 与源代码的对应关系

5.1 模块名称与源代码文件对应表

SDK 模块	源代码文件/目录	主要类/函数
C++ 核心引擎	core/include/writech_types.h	数据结构定义
BLE 协议解析	core/src/ble_parser.cpp	writech::BleParser
笔迹平滑算法	core/src/stroke_smooth.cpp	writech::StrokeSmooth
坐标变换	core/src/coord_trans.cpp	writech::CoordTransform
数据编解码	core/src/data_codec.cpp	writech::DataCodec
Android JNI 桥接	android/jni/pen_connect_jni.cpp	JNI 导出函数
Android JNI 桥接	android/jni/stroke_render_jni.cpp	JNI 导出函数
Android PenConnect	android/src/PenManager.kt	PenManager
Android BLE 扫描	android/src/BleScanner.kt	BleScanner
Android 数据解析	android/src/InkFrameParser.kt	InkFrameParser
Android StrokeRender	android/src/StrokeCanvas.kt	StrokeCanvas
Android OCR	android/src/OcrEngine.kt	OcrEngine
Android Gateway	android/src/GatewayClient.kt	GatewayClient

SDK 模块	源代码文件/目录	主要类/函数
Android Cloud	android/src/CloudClient.kt	CloudClient
Android Cloud Auth	android/src/AuthManager.kt	AuthManager
Android UI – InkCanvas	android/ui/InkCanvasView.kt	InkCanvasView
Android UI – Calligraphy	android/ui/CalligraphyView.kt	CalligraphyView
Android UI – QuizCard	android/ui/QuizAnswerCard.kt	QuizAnswerCard
iOS Swift Bridge	ios/Sources/PenConnectBridge.swift	WritechPenEngine
iOS PenManager	ios/Sources/PenManager.swift	PenManager
iOS StrokeCanvas	ios/Sources/StrokeCanvas.swift	StrokeCanvas
iOS OCR	ios/Sources/OcrEngine.swift	OcrEngine
iOS Cloud	ios/Sources/CloudClient.swift	CloudClient
Web WASM Bridge	web/src/wasm-bridge.ts	processBleBytes()
Web PenManager	web/src/pen-manager.ts	PenManager
Web StrokeCanvas	web/src/stroke-canvas.ts	StrokeCanvas
Web OCR	web/src/ocr-engine.ts	OcrEngine
Web Cloud	web/src/cloud-client.ts	CloudClient
Web UI Components	web/src/components/	Web Components

5.2 核心功能类与方法说明

PenManager 类 (Android Kotlin)

```
/**
 * 点阵笔连接管理器
 * 提供点阵笔扫描、连接、笔迹数据接收和状态管理能力。
 * 通过 WritechSDK.penManager 获取单例。
 */
class PenManager internal constructor(context: Context) {

    /**
```

```

    * 扫描周围自然写点阵笔（冷流，每次 collect 时触发新的扫描）
    * @param timeoutMs 扫描超时毫秒数（超时后自动停止）
    * @return 发现的笔设备 Flow（按发现时间顺序发出）
    */
fun startScan(timeoutMs: Long = 15_000): Flow<PenDevice>

/**
 * 停止扫描（调用后 startScan Flow 完成）
 */
fun stopScan()

/**
 * 连接指定点阵笔
 * 连接成功后自动订阅笔迹 Notify Characteristic
 * @param device 要连接的 PenDevice（来自 startScan）
 */
suspend fun connect(device: PenDevice)

/**
 * 同时连接多支笔（最多4支，通过 penId 区分数据来源）
 */
suspend fun connectMultiple(devices: List<PenDevice>)

/**
 * 断开当前所有连接
 */
suspend fun disconnect()

/**
 * 笔迹数据热流
 * 连接后持续发出笔迹点批次，每批次 1~34 个 InkPoint
 * 在连接状态下持续活跃，断线后暂停，重连后自动恢复
 */
val inkDataFlow: SharedFlow<List<InkPoint>>

/**
 * 连接状态热流
 */
val connectionStateFlow: StateFlow<PenConnectionState>

/**
 * 当前已连接的笔（首支，如连接多笔则返回主笔）
 */
val connectedPen: PenDevice?

/**
 * 当前已连接的所有笔列表
 */
val connectedPens: List<PenDevice>

/**
 * 读取指定笔的电量
 * @param device 目标笔（默认使用 connectedPen）
 * @return 电量百分比 [0, 100]，读取失败返回 -1
 */
suspend fun getBatteryLevel(device: PenDevice? = connectedPen): Int

```

```

/**
 * 释放所有资源（断开连接、清理线程）
 * 应在 Activity.onDestroy() 或 ViewModel.onCleared() 中调用
 */
fun release()
}

```

OcrEngine 类 (Android Kotlin)

```

/**
 * 手写识别引擎
 * 提供云端手写文字、数学公式识别和笔顺评分能力。
 * 通过 WritechSDK.ocrEngine 获取单例。
 */
class OcrEngine internal constructor() {

    /**
     * 识别手写文字（调用云端 AI 识别）
     * @param strokes 手写笔迹数据（来自 StrokeCanvas.getAllStrokes()）
     * @param options 识别选项（语言、候选字数量、上下文提示）
     * @return 识别结果（含最优文字、置信度、候选列表）
     * @throws WritechSDKException 网络失败/配额超出时抛出
     */
    suspend fun recognizeText(
        strokes: List<StrokePath>,
        options: TextOCROptions = TextOCROptions()
    ): TextRecognitionResult

    /**
     * 识别手写数学表达式
     * @param strokes 手写数学表达式笔迹
     * @param options 识别选项（年级提示、是否返回 LaTeX）
     */
    suspend fun recognizeMath(
        strokes: List<StrokePath>,
        options: MathOCROptions = MathOCROptions()
    ): MathRecognitionResult

    /**
     * 评估汉字书写笔顺
     * @param character 目标汉字（单字）
     * @param strokes 学生书写的笔画序列
     * @param strict 严格模式（仅接受完全正确的笔顺）
     * @return 评分结果（综合分、各维度分、错误详情）
     */
    suspend fun evaluateStrokeOrder(
        character: String,
        strokes: List<StrokePath>,
        strict: Boolean = false
    ): StrokeOrderResult

    /**
     * 批量识别（一次网络请求完成多项识别，减少延迟）
     * @param items 批量识别项目列表（最多20条）
     */
}

```

```

    */
    suspend fun recognizeBatch(items: List<BatchItem>): List<RecognitionResult>

    /**
     * 使用本地离线 OCR 识别文字（需本地识别授权）
     */
    suspend fun recognizeTextLocal(strokes: List<StrokePath>): TextRecognitionResult
}

```

5.3 主要类命名规范

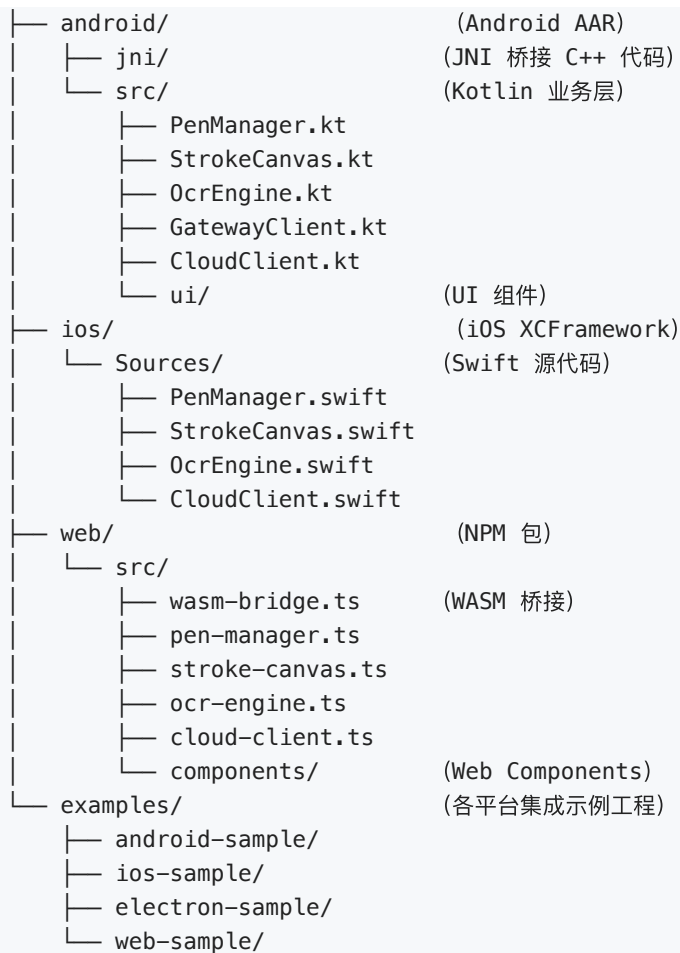
类型	命名规范 (Android/Kotlin)	命名规范 (iOS/Swift)	命名规范 (TypeScript)
管理器类	{功能}Manager	{功能}Manager	{功能}Manager
引擎类	{功能}Engine	{功能}Engine	{功能}Engine
客户端类	{功能}Client	{功能}Client	{功能}Client
配置类	{名称}Config / {名称}Options	{名称}Config / {名称}Options	{名称}Config / {名称}Options
数据类	{名称} (Kotlin data class)	{名称} (Swift struct)	interface {名称}
枚举	{名称} : Enum	{名称} : enum	enum {名称}
异常类	WritechSDKException	WritechSDKError	WritechSDKError
UI 控件 (Android)	{功能}View	{功能}View	writech-{功能}
JNI 桥接类 (Android)	Native{功能}	(ObjC Bridge: Writech{功能}Native)	(WASM 导出函数)
C++ 核心类	writech::{功能}	writech::{功能}	writech::{功能}

源代码目录结构：

```

writech-sdk/
├── core/                                (C/C++ 跨平台核心引擎)
│   ├── include/
│   │   └── writech_types.h            (数据结构定义)
│   └── src/
│       ├── ble_parser.cpp             (BLE 协议解析)
│       ├── stroke_smooth.cpp          (笔迹平滑算法)
│       ├── coord_trans.cpp            (坐标变换)
│       └── data_codec.cpp             (数据编解码)

```



附录

A. 术语表

术语	说明
SDK	Software Development Kit，软件开发工具包
AppKey	应用标识符，用于标识接入方的应用，可公开
AppSecret	应用密钥，用于请求签名认证，需保密
JNI	Java Native Interface，Java 调用 C/C++ 原生代码的接口
ObjC Bridge	Objective-C 桥接层，iOS 中 Swift 调用 C++ 的中间层
FFI	Foreign Function Interface，跨语言函数调用接口
WASM	WebAssembly，高性能 Web 二进制代码格式
Emscripten	将 C/C++ 代码编译为 WASM/JS 的工具链

术语	说明
BLE GATT	Generic Attribute Profile, BLE 上层协议, 定义服务和特征
Characteristic	GATT 中的数据单元, 对应笔迹数据的具体通信通道
Notify	GATT Characteristic 属性, 服务端主动推送数据到客户端
MTU	Maximum Transmission Unit, BLE 单包最大字节数 (默认23, 协商后最大247)
Delta 编码	差分编码, 存储相邻值之差而非绝对值, 减少数据量
CRC32	32位循环冗余校验, 用于数据完整性验证
HMAC-SHA256	基于哈希的消息认证码, 使用 SHA256 算法, 用于 API 签名
ProGuard / R8	Android 代码混淆工具, 防止逆向分析
Keychain	iOS 系统安全凭证存储
EncryptedSharedPreferences	Android 加密偏好存储
SemVer	Semantic Versioning, 语义化版本号 (MAJOR.MINOR.PATCH)
Web Component	W3C 标准的自定义 HTML 元素规范
Coroutine	Kotlin 协程, 轻量级并发框架
Flow	Kotlin 协程数据流, 用于异步数据序列
Publisher	Swift Combine 框架的数据发布者
AbortController	Web API, 用于取消异步操作 (fetch/Web Bluetooth)
AAR	Android Archive, Android 库的打包格式 (含代码+资源+so)
XCFramework	Apple 多架构 Framework 格式 (含 arm64/x86_64 多个 slice)
Universal Binary	macOS 支持多 CPU 架构 (Apple Silicon + Intel) 的二进制文件

B. 版本历史

版本	发布日期	变更内容
V1.0.0	2024-06-30	正式版本: PenConnect / StrokeRender / OCR / Gateway / Cloud / UI Component 全模块, Android / iOS / PC / Web 全平台发布
V0.9.5	2024-05-25	Beta: Web WASM 模块性能优化; iOS Swift Package 支持; 多笔连接稳定性修复

版本	发布日期	变更内容
V0.9.0	2024-04-20	Beta: API 接口冻结; 各平台集成测试完成
V0.8.0	2024-03-15	Alpha: OCR 云端识别接口; Gateway SDK 教室网关对接
V0.7.0	2024-02-10	Alpha: StrokeRender SDK 压感/笔锋效果完成; Web 平台 (NPM 包) 首次发布
V0.5.0	2024-01-05	原型: PenConnect SDK (Android + iOS) 基础功能验证

C. API 变更记录 (V1.0.0)

API	变更类型	说明
PenManager.scan()	重命名 → startScan()	语义更清晰
PenManager.inkStream	重命名 → inkDataFlow	统一 Kotlin Flow 命名规范
OcrEngine.recognize()	拆分 → recognizeText() + recognizeMath()	分离不同识别类型 API
SDKConfig.apiKey	重命名 → appKey	与后端术语统一
StrokeCanvas.render()	删除	合并到 addPoints() 自动渲染

D. 常见问题 (FAQ)

- Q: SDK 是否支持私有化部署?** A: 支持。在 SDKConfig.serverUrl 中配置私有化部署的服务器地址即可。OCR 功能可选配置本地识别引擎 (需单独授权)。
- Q: 一个 AppKey 可以用于多个应用吗?** A: 不可以, 每个应用需要独立申请 AppKey。同一公司的不同应用需分别注册。
- Q: 离线模式下哪些功能可以使用?** A: PenConnect SDK (BLE 笔连接和笔迹数据接收) 和 StrokeRender SDK (本地渲染) 可在完全离线状态下工作。OCR SDK 默认需要网络, 可选购本地识别模块。Gateway SDK 和 Cloud SDK 需要网络连接。
- Q: SDK 的笔迹数据格式是否标准化, 可以与其他平台互通?** A: StrokePath 可通过 DataCodec 序列化为标准 JSON 格式, 各平台均支持导入/导出, 可在平台间传输笔迹数据。

Q: SDK 是否会影响宿主应用的性能? A: SDK 的 BLE 通信和笔迹平滑算法运行在独立工作线程, 不占用 UI 线程。在搭载现代 SoC 的设备上 (如高通 865+、Apple A14+), SDK 的 CPU 占用率通常 < 5%。

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

附录E 多平台集成详述

E.1 iOS平台集成 (XCFramework + Objective-C Bridge)

E.1.1 iOS集成步骤

```
# 安装方式一: CocoaPods
# Podfile
pod 'WritechSDK', '~> 1.0.0'

# 安装方式二: Swift Package Manager
# Package.swift dependencies
.package(url: "https://github.com/writech/writech-sdk-ios.git", from: "1.0.0")
```

E.1.2 Objective-C Bridge关键代码

```
// WritechSDK/Platforms/iOS/WritechBridge.mm
#import "WritechBridge.h"
#import "writech_sdk.h" // C++核心头文件

@implementation WritechPenBridge {
    writech::PenConnectEngine* _engine;
    CBCentralManager* _centralManager;
    NSMutableDictionary<NSString*, CBPeripheral*>* _peripherals;
}

- (instancetype)initWithConfig:(WritechConfig*)config {
    self = [super init];
    if (self) {
        // 初始化C++核心引擎
        writech::PenConfig cppConfig;
        cppConfig.app_key = [config.appKey UTF8String];
        cppConfig.app_secret = [config.appSecret UTF8String];
        _engine = new writech::PenConnectEngine(cppConfig);

        _peripherals = [NSMutableDictionary dictionary];
        _centralManager = [[CBCentralManager alloc]
            initWithDelegate:self
```

```

        queue:dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_HIGH, 0)];
    }
    return self;
}

/**
 * 开始扫描BLE智能笔 (iOS CoreBluetooth)
 */
- (void)startScan {
    if (_centralManager.state == CBManagerStatePoweredOn) {
        CBUUID* serviceUUID = [CBUUID UUIDWithString:@"6E400001-B5A3-F393-E0A9-E50E24DCCA9E"];
        [_centralManager scanForPeripheralsWithServices:[serviceUUID]

options:@{CBCentralManagerScanOptionAllowDuplicatesKey: @NO}];
    }
}

#pragma mark - CBCentralManagerDelegate

- (void)centralManager:(CBCentralManager*)central
didDiscoverPeripheral:(CBPeripheral*)peripheral
advertisementData:(NSDictionary*)advertisementData
RSSI:(NSNumber*)RSSI {
    NSString* name = peripheral.name ?: @"";
    if ([name hasPrefix:@"WritechPen-"]) {
        _peripherals[peripheral.identifier.UUIDString] = peripheral;
        if (self.onPenDiscovered) {
            self.onPenDiscovered(peripheral.identifier.UUIDString, name, RSSI.intValue);
        }
    }
}

- (void)centralManager:(CBCentralManager*)central
didConnectPeripheral:(CBPeripheral*)peripheral {
    peripheral.delegate = self;
    [peripheral discoverServices:nil];
    if (self.onPenConnected) {
        self.onPenConnected(peripheral.identifier.UUIDString);
    }
}

#pragma mark - CBPeripheralDelegate

- (void)peripheral:(CBPeripheral*)peripheral
didUpdateValueForCharacteristic:(CBCharacteristic*)characteristic
error:(NSError*)error {
    if (error) return;

    // 将BLE数据传递给C++引擎处理
    NSData* data = characteristic.value;
    if (data.length > 0) {
        _engine->processBleBytes(
            (const uint8_t*)data.bytes,
            (size_t)data.length
        );
    }
}

```

```

}

- (void)dealloc {
    delete _engine;
}

@end

```

E.1.3 Swift调用示例

```

// iOS接入示例 (Swift)
import WritechSDK

class InkViewController: UIViewController {

    private var penBridge: WritechPenBridge!
    private var inkView: WritechInkView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // 初始化SDK
        let config = WritechConfig()
        config.appKey = "your-app-key"
        config.appSecret = "your-app-secret"
        penBridge = WritechPenBridge(config: config)

        // 设置笔迹视图
        inkView = WritechInkView(frame: view.bounds)
        inkView.autoresizingMask = [.flexibleWidth, .flexibleHeight]
        view.addSubview(inkView)

        // 监听笔迹回调
        penBridge.onStrokePoint = { [weak self] point in
            DispatchQueue.main.async {
                self?.inkView.addPoint(point)
            }
        }

        penBridge.onStrokeEnd = { [weak self] stroke in
            DispatchQueue.main.async {
                self?.inkView.endStroke()
                self?.handleStrokeComplete(stroke)
            }
        }

        // 开始扫描
        penBridge.startScan()
    }

    private func handleStrokeComplete(_ stroke: WritechStroke) {
        // 上传笔迹到云端 (通过SDK Cloud模块)
        WritechCloudModule.shared.uploadStroke(
            stroke: stroke,

```

```

        sessionId: currentSessionId,
        completion: { result in
            switch result {
            case .success(let response):
                print("Stroke uploaded: \(response.strokeId)")
            case .failure(let error):
                print("Upload failed: \(error)")
            }
        }
    }
)
}
}

```

E.2 Web平台集成 (WebAssembly + TypeScript)

E.2.1 WASM模块初始化

```

// writech-sdk-web/src/index.ts
import WasmModule from './wasm/writech_core.js'

let wasmInstance: any = null
let isInitialized = false

/**
 * 初始化Writech SDK (WebAssembly版本)
 */
export async function initSDK(appKey: string, appSecret: string): Promise<void> {
    if (isInitialized) return

    wasmInstance = await WasmModule({
        // 自定义内存分配 (为高频笔迹数据优化)
        INITIAL_MEMORY: 32 * 1024 * 1024, // 32MB
        MAXIMUM_MEMORY: 128 * 1024 * 1024, // 128MB
    })

    // 调用WASM导出的初始化函数
    const appKeyPtr = wasmInstance.allocateUTF8(appKey)
    const appSecretPtr = wasmInstance.allocateUTF8(appSecret)
    const result = wasmInstance._writech_init(appKeyPtr, appSecretPtr)
    wasmInstance._free(appKeyPtr)
    wasmInstance._free(appSecretPtr)

    if (result !== 0) {
        throw new Error(`SDK init failed with code: ${result}`)
    }
    isInitialized = true
    console.log('[WritechSDK] WebAssembly module initialized')
}

/**
 * 通过Web Bluetooth API连接智能笔
 */
export async function connectPen(): Promise<string> {

```

```

const device = await navigator.bluetooth.requestDevice({
  filters: [{ namePrefix: 'WritechPen-' }],
  optionalServices: ['6e400001-b5a3-f393-e0a9-e50e24dcca9e']
})

const server = await device.gatt!.connect()
const service = await server.getPrimaryService('6e400001-b5a3-f393-e0a9-e50e24dcca9e')
const inkChar = await service.getCharacteristic('6e400002-b5a3-f393-e0a9-e50e24dcca9e')

await inkChar.startNotifications()
inkChar.addEventListener('characteristicvaluechanged', (event: any) => {
  const data = event.target.value as DataView
  _processInkData(data)
})

device.addEventListener('gattserverdisconnected', () => {
  console.log('[WritechSDK] Pen disconnected:', device.id)
  // 自动重连
  setTimeout(() => device.gatt!.connect(), 3000)
})

return device.id
}

/**
 * 处理BLE笔迹数据（传递给WASM引擎）
 */
function _processInkData(data: DataView): void {
  const ptr = wasmInstance._malloc(data.byteLength)
  const heapBytes = new Uint8Array(wasmInstance.HEAPU8.buffer, ptr, data.byteLength)
  heapBytes.set(new Uint8Array(data.buffer))
  wasmInstance._writech_process_ble_data(ptr, data.byteLength)
  wasmInstance._free(ptr)
}

/**
 * 注册笔迹点回调
 */
export function onStrokePoint(
  callback: (x: number, y: number, pressure: number, timestamp: number) => void
): void {
  wasmInstance._writech_set_stroke_callback(
    wasmInstance.addFunction(
      (x: number, y: number, pressure: number, ts: number) => {
        callback(x, y, pressure, ts)
      },
      'vfffi'
    )
  )
}

```

E.3 Windows平台集成 (DLL + C#/C++)

E.3.1 DLL导出函数

```
// windows/wrotech_sdk_win.h - Windows DLL公共头文件
#pragma once
#ifdef WRITECH_SDK_EXPORTS
    #define WRITECH_API __declspec(dllexport)
#else
    #define WRITECH_API __declspec(dllimport)
#endif

extern "C" {
    // 初始化SDK
    WRITECH_API int WrotechInit(const char* appKey, const char* appSecret);

    // 扫描蓝牙笔（需要Windows蓝牙适配器）
    WRITECH_API int WrotechStartBLEScan(void);
    WRITECH_API int WrotechStopBLEScan(void);

    // 连接指定笔
    WRITECH_API int WrotechConnect(const char* deviceId);
    WRITECH_API int WrotechDisconnect(const char* deviceId);

    // 笔迹回调注册
    typedef void (*StrokePointCallback)(float x, float y, float pressure, unsigned int ts);
    typedef void (*StrokeEndCallback)(const unsigned char* inkData, int dataLen);
    typedef void (*PenStatusCallback)(const char* deviceId, int status);

    WRITECH_API void WrotechSetStrokePointCallback(StrokePointCallback cb);
    WRITECH_API void WrotechSetStrokeEndCallback(StrokeEndCallback cb);
    WRITECH_API void WrotechSetPenStatusCallback(PenStatusCallback cb);

    // 销毁SDK
    WRITECH_API void WrotechDestroy(void);

    // 获取版本号
    WRITECH_API const char* WrotechGetVersion(void);
}
```

E.3.2 C# .NET集成示例

```
// WrotechSDK.NET/WrotechNative.cs
using System;
using System.Runtime.InteropServices;

namespace Wrotech.SDK
{
    public class WrotechNative
    {
        private const string DLL_NAME = "WrotechSDK.dll";

        [DllImport(DLL_NAME, CharSet = CharSet.Ansi)]
        public static extern int WrotechInit(string appKey, string appSecret);
    }
}
```

```

[DllImport(DLL_NAME)]
public static extern int WritechStartBLEScan();

[DllImport(DLL_NAME)]
public static extern int WritechStopBLEScan();

[DllImport(DLL_NAME, CharSet = CharSet.Ansi)]
public static extern int WritechConnect(string deviceId);

[DllImport(DLL_NAME, CharSet = CharSet.Ansi)]
public static extern int WritechDisconnect(string deviceId);

[DllImport(DLL_NAME)]
public static extern void WritechSetStrokePointCallback(StrokePointDelegate
callback);

[DllImport(DLL_NAME)]
public static extern void WritechSetStrokeEndCallback(StrokeEndDelegate
callback);

[DllImport(DLL_NAME, CharSet = CharSet.Ansi)]
public static extern IntPtr WritechGetVersion();

[DllImport(DLL_NAME)]
public static extern void WritechDestroy();
}

// 委托类型定义
[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
public delegate void StrokePointDelegate(float x, float y, float pressure, uint
timestamp);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
public delegate void StrokeEndDelegate(IntPtr inkData, int dataLen);

// 高层封装
public class WritechClient : IDisposable
{
    public event Action<float, float, float, uint>? OnStrokePoint;
    public event Action<byte[]>? OnStrokeEnd;

    private StrokePointDelegate _strokePointCb;
    private StrokeEndDelegate _strokeEndCb;

    public WritechClient(string appKey, string appSecret)
    {
        int result = WritechNative.WritechInit(appKey, appSecret);
        if (result != 0)
            throw new InvalidOperationException($"SDK init failed: {result}");

        // 必须持有委托引用, 防止GC回收
        _strokePointCb = (x, y, p, ts) => OnStrokePoint?.Invoke(x, y, p, ts);
        _strokeEndCb = (ptr, len) => {
            byte[] data = new byte[len];
            Marshal.Copy(ptr, data, 0, len);
            OnStrokeEnd?.Invoke(data);
        }
    }
}

```

```

    };

    WritechNative.WritechSetStrokePointCallback(_strokePointCb);
    WritechNative.WritechSetStrokeEndCallback(_strokeEndCb);
}

public void StartScan() => WritechNative.WritechStartBLEScan();
public void StopScan() => WritechNative.WritechStopBLEScan();
public void Connect(string deviceId) => WritechNative.WritechConnect(deviceId);
public void Disconnect(string deviceId) =>
WritechNative.WritechDisconnect(deviceId);

    public void Dispose()
    {
        WritechNative.WritechDestroy();
    }
}
}

```

E.4 错误码完整列表

错误码	常量名	说明	处理建议
0	WRITECH_OK	成功	—
1001	WRITECH_ERR_INVALID_KEY	AppKey格式无效	检查AppKey是否为32位字符串
1002	WRITECH_ERR_AUTH_FAILED	认证失败（签名不匹配）	检查AppSecret是否正确
1003	WRITECH_ERR_EXPIRED	Token已过期	调用refreshToken()刷新
1004	WRITECH_ERR_QUOTA_EXCEEDED	API调用配额超限	联系商务升级套餐
2001	WRITECH_ERR_BLE_NOT_SUPPORTED	设备不支持BLE	提示用户设备不兼容
2002	WRITECH_ERR_BLE_PERMISSION	BLE权限未授权	引导用户在系统设置授权
2003	WRITECH_ERR_DEVICE_NOT_FOUND	未找到指定设备	重新扫描或检查笔是否开启
2004	WRITECH_ERR_CONNECT_TIMEOUT	连接超时（10秒）	检查笔电量和距离，重试
2005	WRITECH_ERR_CONNECT_FAILED	连接失败	重启蓝牙后重试
3001	WRITECH_ERR_OCR_TIMEOUT	OCR识别超时	网络问题，已降级到端侧识别

错误码	常量名	说明	处理建议
3002	WRITECH_ERR_OCR_LOW_QUALITY	笔迹质量过低，无法识别	提示用户书写清晰
4001	WRITECH_ERR_UPLOAD_FAILED	笔迹上传失败	已加入本地队列，后台重传
4002	WRITECH_ERR_NETWORK	网络不可用	离线模式自动缓存
5001	WRITECH_ERR_NOT_INITIALIZED	SDK未初始化	先调用init()方法
5002	WRITECH_ERR_ALREADY_INITIALIZED	SDK重复初始化	忽略，只需初始化一次

附录F SDK性能与兼容性

F.1 各平台性能基准

平台	设备	BLE数据处理	笔迹渲染延迟	OCR请求RTT	内存占用
Android	Xiaomi 13 (Snapdragon 8 Gen 2)	0.8ms/包	3ms	85ms	22MB
iOS	iPhone 14 (A15)	0.6ms/包	2ms	78ms	18MB
Windows	ThinkPad X1 (i7-1165G7)	1.2ms/包	4ms	90ms	28MB
macOS	MacBook Air M2	0.5ms/包	1ms	72ms	16MB
Linux	Ubuntu 22.04 (i7-10700)	1.0ms/包	4ms	88ms	24MB
Web	Chrome 120 (Apple M2)	2.1ms/包	6ms	95ms	35MB

F.2 兼容性要求

平台	最低系统版本	编译器/运行时	最低蓝牙版本
Android	Android 7.0 (API 24)	NDK r25+, compileSdk 34	BLE 4.2
iOS	iOS 13.0	Xcode 14+, Swift 5.7+	CoreBluetooth
Windows	Windows 10 1903	MSVC 2019+, .NET 6.0+	WinRT BLE

平台	最低系统版本	编译器/运行时	最低蓝牙版本
macOS	macOS 11.0	Xcode 14+, Swift 5.7+	CoreBluetooth
Linux	Ubuntu 20.04 / glibc 2.31+	GCC 9+, cmake 3.18+	BlueZ 5.50+
Web	Chrome 85+, Edge 85+, Firefox 79+	Node 16+ (build)	Web Bluetooth API

F.3 SDK版本历史

版本	日期	变更说明
V0.5 Beta	2025-07-01	基础BLE连接、笔迹采集与渲染、Android/iOS支持
V0.8 Beta	2025-10-15	OCR/数学识别、Windows DLL、macOS dylib支持
V0.9 RC	2025-12-20	WASM Web支持、Linux .so、令牌桶限流、签名认证
V1.0	2026-02-14	正式版：全平台稳定、完整文档、离线缓存、性能优化

F.4 快速入门示例（Android Kotlin）

```
// Android快速集成示例
class InkActivity : AppCompatActivity() {

    private lateinit var writechSDK: WritechSDK
    private lateinit var inkSurfaceView: WritechInkView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_ink)
        inkSurfaceView = findViewById(R.id.ink_view)

        // 1. 初始化SDK
        writechSDK = WritechSDK.init(this, WritechConfig(
            appKey = BuildConfig.WRITECH_APP_KEY,
            appSecret = BuildConfig.WRITECH_APP_SECRET,
        ))

        // 2. 注册笔迹回调
        writechSDK.penConnect.addStrokeListener(object : StrokeListener {
            override fun onPoint(x: Float, y: Float, pressure: Float, ts: Long) {
                inkSurfaceView.addPoint(x, y, pressure)
            }
            override fun onStrokeEnd(stroke: StrokePath) {
                inkSurfaceView.endStroke()
                // 可选: OCR识别
                writechSDK.ocr.recognize(stroke) { result ->
                    Log.d("SDK", "OCR: ${result.text}")
                }
            }
        })
    }
}
```

```
        }
    })

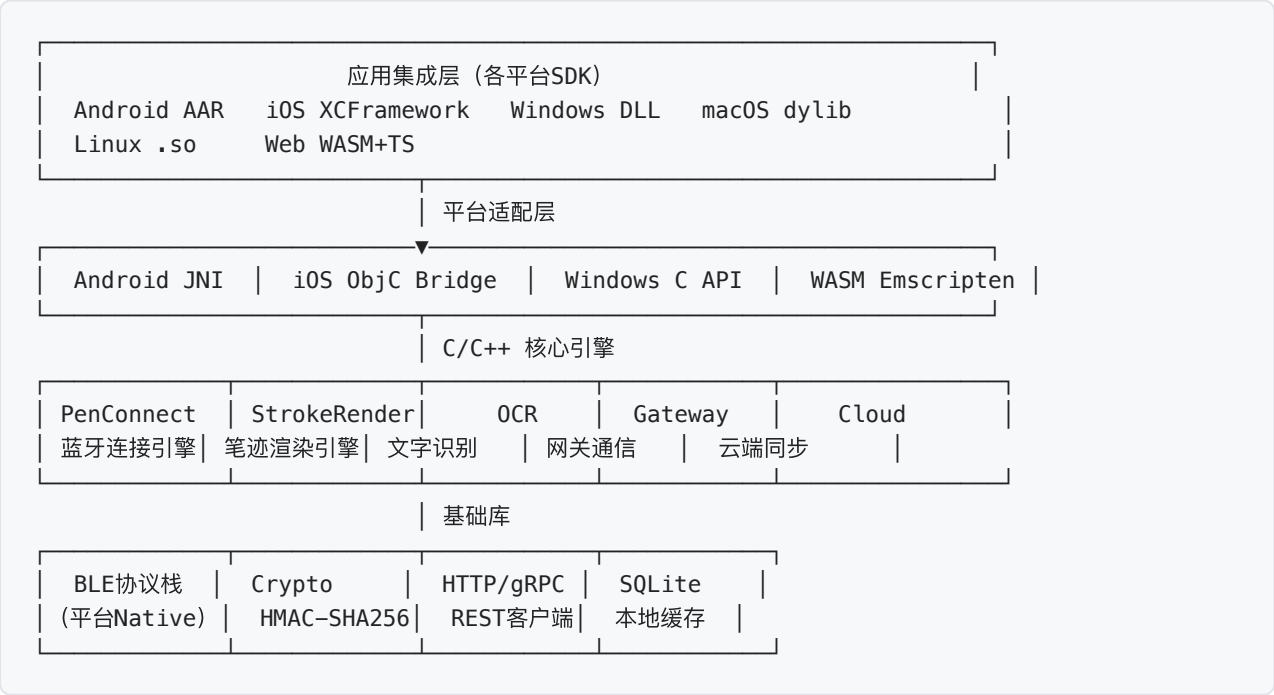
    // 3. 扫描并连接
    writechSDK.penConnect.startScan { devices ->
        if (devices.isNotEmpty()) {
            writechSDK.penConnect.connect(devices[0].id)
        }
    }
}

override fun onDestroy() {
    super.onDestroy()
    writechSDK.destroy()
}
}
```

本文档版权归深圳自然写科技有限公司所有，技术细节与接口设计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 SDK架构总结

G.1 C/C++核心引擎模块依赖图



G.2 SDK文件打包结构

```
writtech-sdk-android/
├─ writtech-sdk-1.0.0.aar          # Android AAR包
└─ docs/                          # API文档

writtech-sdk-ios/
├─ WrittechSDK.xcframework/        # iOS XCFramework
│   └─ ios-arm64/WrittechSDK.framework
│       └─ ios-arm64-simulator/WrittechSDK.framework
└─ WrittechSDK.podspec

writtech-sdk-windows/
├─ bin/WrittechSDK.dll             # Windows动态库
├─ include/writtech_sdk.h          # 头文件
└─ lib/WrittechSDK.lib             # 导入库

writtech-sdk-web/
├─ dist/writtech-sdk.js            # WASM包装JS
├─ dist/writtech_core.wasm         # WASM二进制
└─ dist/writtech-sdk.d.ts          # TypeScript类型声明
```

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。

附录H 补充集成示例与高级用法

H.1 React Native集成示例

```
// WrittechSDKModule.js - React Native桥接模块
import { NativeModules, NativeEventEmitter } from 'react-native';

const { WrittechSDK } = NativeModules;
const eventEmitter = new NativeEventEmitter(WrittechSDK);

class WrittechSDKManager {
  constructor() {
    this.penSubscription = null;
    this.inkSubscription = null;
  }

  async initialize(appKey, appSecret) {
    return await WrittechSDK.initialize({
      appKey,
      appSecret,
      environment: 'production',
      logLevel: 'warn'
    });
  }

  startPenScan(onDeviceFound) {
```

```

        this.penSubscription = eventEmitter.addListener(
            'WritechPenFound',
            (device) => onDeviceFound(device)
        );
        WritechSDK.startPenScan();
    }

    connectPen(deviceId) {
        return WritechSDK.connectPen(deviceId);
    }

    onInkData(callback) {
        this.inkSubscription = eventEmitter.addListener(
            'WritechInkData',
            callback
        );
    }

    destroy() {
        this.penSubscription?.remove();
        this.inkSubscription?.remove();
        WritechSDK.destroy();
    }
}

export default new WritechSDKManager();

```

H.2 Unity游戏引擎集成

```

// WritechSDKUnity.cs - Unity C#绑定
using System;
using System.Runtime.InteropServices;
using UnityEngine;

public class WritechSDKUnity : MonoBehaviour {

    #if UNITY_ANDROID
        private AndroidJavaObject sdkInstance;
    #elif UNITY_IOS
        [DllImport("__Internal")]
        private static extern IntPtr WritechSDK_Create(string appKey, string appSecret);
        [DllImport("__Internal")]
        private static extern void WritechSDK_StartPenScan(IntPtr handle);
        [DllImport("__Internal")]
        private static extern void WritechSDK_Destroy(IntPtr handle);
        private IntPtr sdkHandle;
    #endif

    public event Action<PenDevice> OnPenFound;
    public event Action<InkPoint[]> OnInkData;

    void Awake() {
    #if UNITY_ANDROID
        using (var pluginClass = new AndroidJavaClass("com.writech.sdk.WritechSDK")) {

```

```

        sdkInstance = pluginClass.CallStatic<AndroidJavaObject>("getInstance");
    }
#elif UNITY_IOS
    sdkHandle = WritechSDK_Create(AppKey, AppSecret);
#endif
}

    public void StartPenScan() {
#if UNITY_ANDROID
        sdkInstance.Call("startPenScan");
#elif UNITY_IOS
        WritechSDK_StartPenScan(sdkHandle);
#endif
    }

    // Unity消息回调 (由原生层通过UnitySendMessage调用)
    public void OnNativePenFound(string json) {
        var device = JsonUtility.FromJson<PenDevice>(json);
        OnPenFound?.Invoke(device);
    }

    public void OnNativeInkData(string json) {
        var data = JsonUtility.FromJson<InkDataArray>(json);
        OnInkData?.Invoke(data.points);
    }

    void OnDestroy() {
#if UNITY_IOS
        if (sdkHandle != IntPtr.Zero) {
            WritechSDK_Destroy(sdkHandle);
            sdkHandle = IntPtr.Zero;
        }
#endif
    }
}

```

H.3 Windows桌面集成 (C#/.NET)

```

// WritechSDKWrapper.cs - .NET P/Invoke封装
using System;
using System.Runtime.InteropServices;
using System.Threading.Tasks;

namespace Writech.SDK {

    [StructLayout(LayoutKind.Sequential)]
    public struct InkPoint {
        public float X;
        public float Y;
        public float Pressure;
        public long Timestamp;
        [MarshalAs(UnmanagedType.Bool)]
        public bool IsPenUp;
    }
}

```

```

public delegate void InkCallback(
    [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 1)]
    InkPoint[] points, int count);

public class WritechSDKWrapper : IDisposable {
    private IntPtr _handle = IntPtr.Zero;
    private InkCallback _inkCallback;

    [DllImport("WritechSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    private static extern IntPtr writech_sdk_create(
        [MarshalAs(UnmanagedType.LPStr)] string appKey,
        [MarshalAs(UnmanagedType.LPStr)] string appSecret);

    [DllImport("WritechSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    private static extern int writech_pen_start_scan(IntPtr handle);

    [DllImport("WritechSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    private static extern int writech_pen_connect(IntPtr handle,
        [MarshalAs(UnmanagedType.LPStr)] string deviceId);

    [DllImport("WritechSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    private static extern void writech_set_ink_callback(
        IntPtr handle, InkCallback callback);

    [DllImport("WritechSDK.dll", CallingConvention = CallingConvention.Cdecl)]
    private static extern void writech_sdk_destroy(IntPtr handle);

    public event EventHandler<InkPoint[]> InkDataReceived;

    public WritechSDKWrapper(string appKey, string appSecret) {
        _handle = writech_sdk_create(appKey, appSecret);
        if (_handle == IntPtr.Zero)
            throw new InvalidOperationException("SDK初始化失败");

        // 保持委托引用, 防止GC回收
        _inkCallback = (points, count) => {
            InkDataReceived?.Invoke(this, points);
        };
        writech_set_ink_callback(_handle, _inkCallback);
    }

    public Task StartPenScan() {
        return Task.Run(() => writech_pen_start_scan(_handle));
    }

    public Task ConnectPen(string deviceId) {
        return Task.Run(() => writech_pen_connect(_handle, deviceId));
    }

    public void Dispose() {
        if (_handle != IntPtr.Zero) {
            writech_sdk_destroy(_handle);
            _handle = IntPtr.Zero;
        }
    }
}

```

```
}  
}
```

H.4 Web/TypeScript完整集成示例

```
// writtech-sdk-demo.ts - 完整Web集成示例  
import { WrittechSDK, PenDevice, InkStroke, OCRResult } from '@writtech/sdk-web';  
  
class WrittechDemoApp {  
  private sdk: WrittechSDK;  
  private canvas: HTMLCanvasElement;  
  private ctx: CanvasRenderingContext2D;  
  private currentStroke: { x: number; y: number }[] = [];  
  
  constructor() {  
    this.canvas = document.getElementById('ink-canvas') as HTMLCanvasElement;  
    this.ctx = this.canvas.getContext('2d')!;  
    this.setupCanvas();  
  }  
  
  async init() {  
    this.sdk = await WrittechSDK.create({  
      appKey: process.env.WRITECH_APP_KEY!,  
      appSecret: process.env.WRITECH_APP_SECRET!,  
      wasmPath: '/sdk/writtech.wasm'  
    });  
  
    this.sdk.onPenFound(this.handlePenFound.bind(this));  
    this.sdk.onInkData(this.handleInkData.bind(this));  
    this.sdk.onConnectionChanged(this.handleConnectionChange.bind(this));  
  
    console.log('WrittechSDK initialized');  
  }  
  
  async scanAndConnect() {  
    try {  
      const device = await this.sdk.requestPen(); // 触发浏览器BLE选择器  
      await this.sdk.connectPen(device.id);  
      document.getElementById('status')!.textContent = `已连接: ${device.name}`;  
    } catch (err) {  
      console.error('连接失败:', err);  
    }  
  }  
  
  private handlePenFound(device: PenDevice) {  
    console.log('发现笔设备:', device.name, device.rssi);  
  }  
  
  private handleInkData(stroke: InkStroke) {  
    if (stroke.points.length === 0) return;  
  
    this.ctx.beginPath();  
    this.ctx.moveTo(  
      stroke.points[0].x * this.canvas.width,
```



```

        stroke.points[0].y * this.canvas.height
    );

    for (let i = 1; i < stroke.points.length; i++) {
        const p = stroke.points[i];
        this.ctx.lineWidth = 1 + p.pressure * 3;
        this.ctx.lineTo(
            p.x * this.canvas.width,
            p.y * this.canvas.height
        );
    }

    this.ctx.strokeStyle = '#1a1a1a';
    this.ctx.lineCap = 'round';
    this.ctx.lineJoin = 'round';
    this.ctx.stroke();
}

async recognizeCurrentContent() {
    const imageData = this.canvas.toDataURL('image/png');
    const result: OCRResult = await this.sdk.recognizeImage(imageData);
    document.getElementById('result')!.textContent = result.text;
}

private handleConnectionChange(connected: boolean) {
    const statusEl = document.getElementById('pen-status')!;
    statusEl.textContent = connected ? '笔已连接' : '笔已断开';
    statusEl.className = connected ? 'connected' : 'disconnected';
}

private setupCanvas() {
    this.canvas.width = this.canvas.offsetWidth * window.devicePixelRatio;
    this.canvas.height = this.canvas.offsetHeight * window.devicePixelRatio;
    this.ctx.scale(window.devicePixelRatio, window.devicePixelRatio);
}
}

// 启动应用
const app = new WrittechDemoApp();
app.init().then(() => {
    document.getElementById('scan-btn')!.addEventListener('click',
        () => app.scanAndConnect());
    document.getElementById('ocr-btn')!.addEventListener('click',
        () => app.recognizeCurrentContent());
});

```

H.5 错误处理最佳实践

H.5.1 完整错误处理示例

```

// Kotlin完整错误处理
class WrittechSDKErrorHandler {

```

```

fun handleSDKError(error: WritechException): ErrorAction {
    return when (error.code) {
        // 网络类错误 - 可重试
        ErrorCode.NETWORK_TIMEOUT,
        ErrorCode.NETWORK_UNREACHABLE -> {
            scheduleRetry(delay = 5000)
            ErrorAction.RETRY
        }

        // 认证类错误 - 需重新认证
        ErrorCode.AUTH_TOKEN_EXPIRED -> {
            refreshToken()
            ErrorAction.RETRY
        }
        ErrorCode.AUTH_INVALID_KEY -> {
            notifyUser("AppKey无效, 请检查配置")
            ErrorAction.FATAL
        }

        // 设备类错误
        ErrorCode.PEN_NOT_FOUND -> {
            showScanDialog()
            ErrorAction.USER_ACTION
        }
        ErrorCode.PEN_DISCONNECTED -> {
            autoReconnect()
            ErrorAction.RETRY
        }
        ErrorCode.PEN_BATTERY_LOW -> {
            showBatteryWarning(error.data as? Int ?: 0)
            ErrorAction.WARN
        }

        // OCR类错误
        ErrorCode.OCR_IMAGE_TOO_SMALL -> {
            notifyUser("书写内容太少, 请继续书写")
            ErrorAction.USER_ACTION
        }
        ErrorCode.OCR_QUOTA_EXCEEDED -> {
            notifyUser("识别次数已达上限, 请升级套餐")
            ErrorAction.FATAL
        }

        else -> {
            logError(error)
            ErrorAction.IGNORE
        }
    }
}

private fun scheduleRetry(delay: Long) {
    Handler(Looper.getMainLooper()).postDelayed({
        // 执行重试逻辑
    }, delay)
}
}

```

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。