cursor续杯工具分析

前言：本文通过cursor分析cursor续杯工具的实现原理，并分析如何绕过设备限制，同时对API进行了一些简单的安全测试。

地址： `aHR0cHM6Ly93d3cueHhkbHpzLnRvcC8=`

解包

程序目录如下：

```
Administrator@CHINAMI-A6GAF0V ~/Desktop/344Fa0LnFrV8N05kLAzrq0Q1sRO
$ ll
total 216M
-rw-r--r-- 1 Administrator 197121 164K 10月 15 02:03 chrome_100_percent.pak
-rw-r--r-- 1 Administrator 197121 223K 10月 15 02:03 chrome_200_percent.pak
-rwxr-xr-x 1 Administrator 197121 169M 10月 15 02:03 CursorRenewal.exe
-rwxr-xr-x 1 Administrator 197121 4.7M 10月 15 02:03 d3dcompiler_47.dll
-rwxr-xr-x 1 Administrator 197121 2.8M 10月 15 02:03 ffmpeg.dll
-rw-r--r-- 1 Administrator 197121  11M 10月 15 02:03 icudtl.dat
-rwxr-xr-x 1 Administrator 197121 467K 10月 15 02:03 libEGL.dll
-rwxr-xr-x 1 Administrator 197121 7.5M 10月 15 02:03 libGLESv2.dll
-rw-r--r-- 1 Administrator 197121 1.1K 10月 15 02:03 LICENSE.electron.txt
-rw-r--r-- 1 Administrator 197121 8.8M 10月 15 02:03 LICENSES.chromium.html
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:22 locales
drwxr-xr-x 1 Administrator 197121    0 10月 28 06:39 resources
-rw-r--r-- 1 Administrator 197121 5.1M 10月 15 02:03 resources.pak
-rw-r--r-- 1 Administrator 197121 271K 10月 15 02:03 snapshot_blob.bin
-rw-r--r-- 1 Administrator 197121 628K 10月 15 02:03 v8_context_snapshot.bin
-rwxr-xr-x 1 Administrator 197121 5.0M 10月 15 02:03 vk_swiftshader.dll
-rw-r--r-- 1 Administrator 197121  106 10月 15 02:03 vk_swiftshader_icd.json
-rwxr-xr-x 1 Administrator 197121 925K 10月 15 02:03 vulkan-1.dll
Administrator@CHINAMI-A6GAF0V ~/Desktop/344Fa0LnFrV8N05kLAzrq0Q1sRO
$
```

resources文件夹内容如下：

```
Administrator@CHINAMI-A6GAF0V ~/Desktop/344Fa0LnFrV8N05kLAzrq0Q1sRO/resources
$ ll
total 106M
-rw-r--r-- 1 Administrator 197121 106M 10月 28 06:35 app.asar
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:22 app.asar.unpacked
-rw-r--r-- 1 Administrator 197121 1.1K 10月 15 02:03 app.manifest
-rwxr-xr-x 1 Administrator 197121 105K 10月 15 02:03 elevate.exe
-rw-r--r-- 1 Administrator 197121   36 10月 28 06:24 uuid.txt
Administrator@CHINAMI-A6GAF0V ~/Desktop/344Fa0LnFrV8N05kLAzrq0Q1sRO/resources
$
```

根据 `app.asar` 搜索发现是 `electron` 程序，通过npm安装对应的 `asar` 工具。

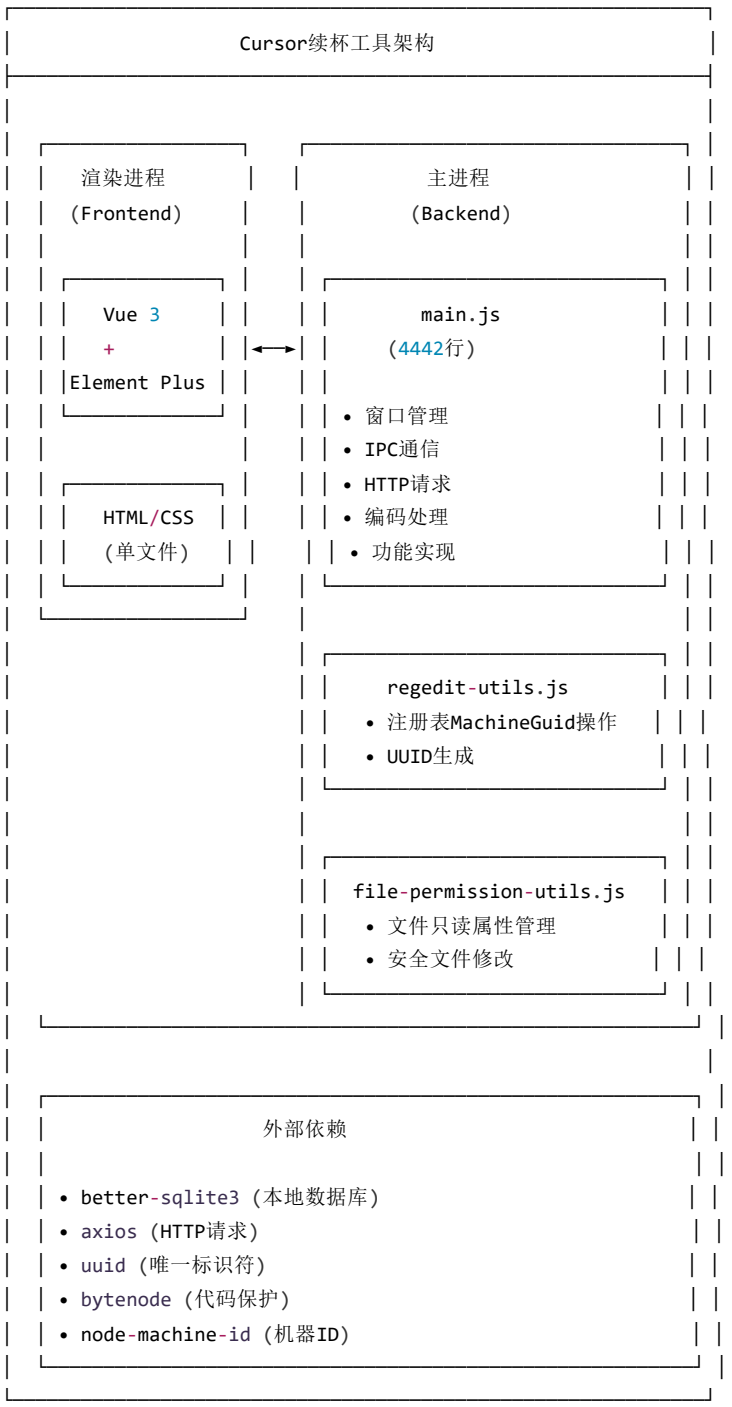
```
npm install -g asar
asar -V
```

在asar文件所在的根目录执行解压命令，此刻拿到项目源码：

```
asar extract app.asar ./
```

架构

通过cursor打开相应的项目文件夹，输入提示词 分析代码架构 。



开控制台

```
const isDev = process.argv.includes('--dev') || process.env.NODE_ENV === 'development';
```

- `--dev` 仅用于打开控制台，可以通过命令行启动或者创建快捷方式来传入这个参数。
- `development` 除了打开控制台还有API测试相关功能， `set NODE_ENV=development` 后命令行输入 `.\CursorRenewal.exe` 即可。

本次用 `development` 进行测试，程序要求输入激活码，随便输入即可。

配置文件夹

根据log，发现config文件夹路径：

```
C:\Users\Administrator\AppData\Roaming\cursor-renewal-client
Administrator@CHINAMI-A6GAF0V ~/AppData/Roaming/cursor-renewal-client
$ ll
total 16K
drwxr-xr-x 1 Administrator 197121    0 10月 28 18:38  blob_storage
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:20  Cache
-rw-r--r-- 1 Administrator 197121  284 10月 28 18:39  card_info.json
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:20 'Code Cache'
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:20  DawnCache
-rw-r--r-- 1 Administrator 197121   64 10月 28 04:55  device_id.txt
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:20  Dictionaries
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:20  GPUCache
-rw-r--r-- 1 Administrator 197121  434 10月 28 04:20 'Local State'
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:20 'Local Storage'
drwxr-xr-x 1 Administrator 197121    0 10月 28 18:38  Network
-rw-r--r-- 1 Administrator 197121 1.3K 10月 28 18:39  Preferences
drwxr-xr-x 1 Administrator 197121    0 10月 28 18:38 'Session Storage'
-rw-r--r-- 1 Administrator 197121  164 10月 28 04:58  settings.json
drwxr-xr-x 1 Administrator 197121    0 10月 28 04:20 'Shared Dictionary'
-rw-r--r-- 1 Administrator 197121    0 10月 28 04:36  SharedStorage
Administrator@CHINAMI-A6GAF0V ~/AppData/Roaming/cursor-renewal-client
$
```

机器码

根据cursor提供的架构图和配置文件夹，发现 `uuid` 和 `device_id.txt` 俩个配置文件，输入提示词 根据uuid和device_id寻找机器码相关函数 。

函数名	用途	生成的 ID 类型	返回值
get-machine-id (IPC)	获取系统真实机器 ID	系统 UUID（跨平台）	字符串
generateMachineId()	生成 / 获取设备 ID（带缓存）	基于真实机器 ID 或稳定标识符	字符串
generateNewIds()	生成新的 UUID 集合（用于SQL数据库）	machineId, anonymousId, machineIdHash	对象
generateMachineIds()	生成 storage.json 所需的机器码	machineId(SHA256), devDeviceId(UUID), sqmId	对象
resetMachineIds()	重置 Cursor 机器 ID 文件	deviceId, anonymousId, machineGuid	void

据此，我们主要通过操作 `get-machine-id (IPC)` 和 `generateMachineId` 来达到突破机器码限制的问题。提示词

修改 get-machine-id (IPC)和generateMachineId()函数体，返回指定ID：a4675d2ca478e02b55507892b77b0e5a5b0b249371aaa8977a665b1e14c616c9 。

```
// 获取机器ID
ipcMain.handle('get-machine-id', async () => {
  // 返回指定的机器ID
  const machineId = "a4675d2ca478e02b55507892b77b0e5a5b0b249371aaa8977a665b1e14c616c9";
  return machineId;
});

// 生成设备ID - 使用缓存确保稳定性
function generateMachineId() {
  // 返回指定的设备ID
  const deviceId = "a4675d2ca478e02b55507892b77b0e5a5b0b249371aaa8977a665b1e14c616c9";
  return deviceId;
}

// 两个函数均已简化为直接返回指定的ID：// a4675d2ca478e02b55507892b77b0e5a5b0b249371aaa8977a665b1e14c616c9。
// 移除了原有的系统检测逻辑
// 移除了缓存机制
// 移除了备用方案
// 直接返回固定ID
```

突破设备限制方案

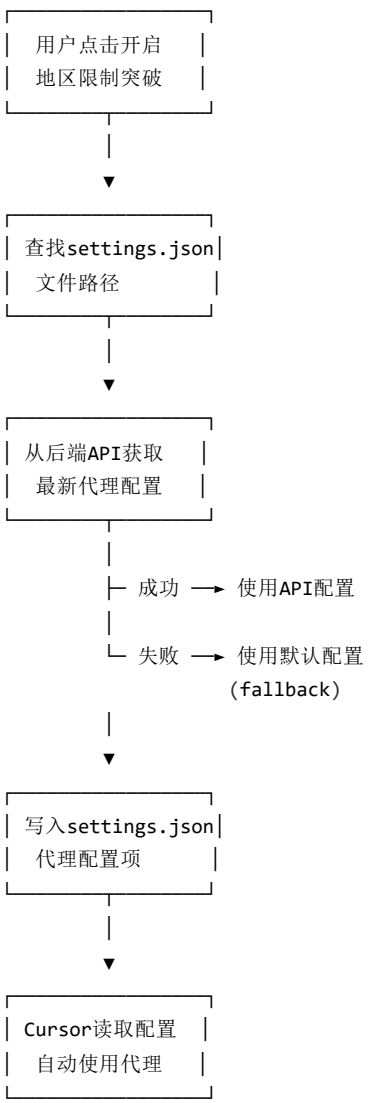
1. 固定机器码
2. 固定IP

突破地区限制

提示词 `main.js`是如何完成突破地区限制的？ 。

核心实现原理
通过修改 `Cursor` 的配置文件，强制 `Cursor` 使用代理服务器访问网络，从而突破地区限制。

流程图



1. 查找 settings.json 文件路径

```
// 搜索Cursor settings.json文件
function findCursorSettingsPath() {
  // 如果已经缓存了路径，直接返回
  if (cachedSettingsPath && fs.existsSync(cachedSettingsPath)) {
    console.log(`使用缓存的settings.json路径: ${cachedSettingsPath}`);
    return cachedSettingsPath;
  }

  console.log('开始全盘搜索Cursor settings.json文件...');

  // 获取所有驱动器（Windows）
  // ... 路径搜索逻辑 ...

  // Windows路径: C:\Users\[用户名]\AppData\Roaming\Cursor\User\settings.json
  // Linux/Mac路径: ~/.config/Cursor/User/settings.json
}
```

2. 从后端获取代理配置

```
// 从后端API获取当前代理配置
async function fetchCurrentProxyConfig() {
  try {
    const apiUrl = 'http://129.204.108.166:2486/csk/proxy-config/current';
    console.log('从后端获取当前代理配置...');

    const response = await axios.get(apiUrl, { timeout: 5000 });

    if (response.data.success && response.data.data) {
      console.log('成功获取代理配置');
      return response.data.data;
    } else {
      console.warn('获取代理配置失败:', response.data.message);
      return null;
    }
  } catch (error) {
    console.error('获取代理配置出错:', error.message);
    return null;
  }
}
```

3. 检查当前代理状态

```
// 检查Cursor Settings.json当前状态
ipcMain.handle('check-cursor-settings-status', async () => {
  // 1. 查找settings.json文件
  // 2. 读取并解析配置
  // 3. 检查是否包含代理配置字段
  // 4. 验证代理是否在数据库列表中
  // 5. 返回当前状态
})
```

检查的关键字段：

- http.proxy - 代理地址
- http.proxySupport - 代理支持模式（需为 "override"）
- cursor.general.disableHttp2 - 禁用 HTTP/2（需为 true）

4. 更新 settings.json（开启/关闭）

```
// 更新Cursor Settings.json（突破地区限制）
ipcMain.handle('update-cursor-settings', async (event, enabled) => {
  if (enabled) {
    // ===== 开启地区限制突破 =====
    // 1. 从后端获取代理配置（优先）
    // 2. 如果获取失败，使用默认配置
    // 3. 写入完整的代理配置
  } else {
    // ===== 关闭地区限制 =====
    // 写入基础配置（移除代理设置）
  }
})
```

开启时写入的配置项

```
settingsContent = {
  "database-client.autoSync": true,
  "update.enableWindowsBackgroundUpdates": false,
  "update.mode": "none",
  "http.proxyAuthorization": null,
  "json.schemas": [],
  "window.commandCenter": true,
  "http.proxy": "socks5://xc999:xc123@154.201.91.204:38999",  // 代理地址
  "http.systemCertificates": false,
  "http.proxySupport": "override",  // 强制使用代理
  "http.experimental.systemCertificatesV2": false,
  "http.experimental.systemCertificates": false,
  "cursor.general.disableHttp2": true  // 禁用HTTP/2
};
```

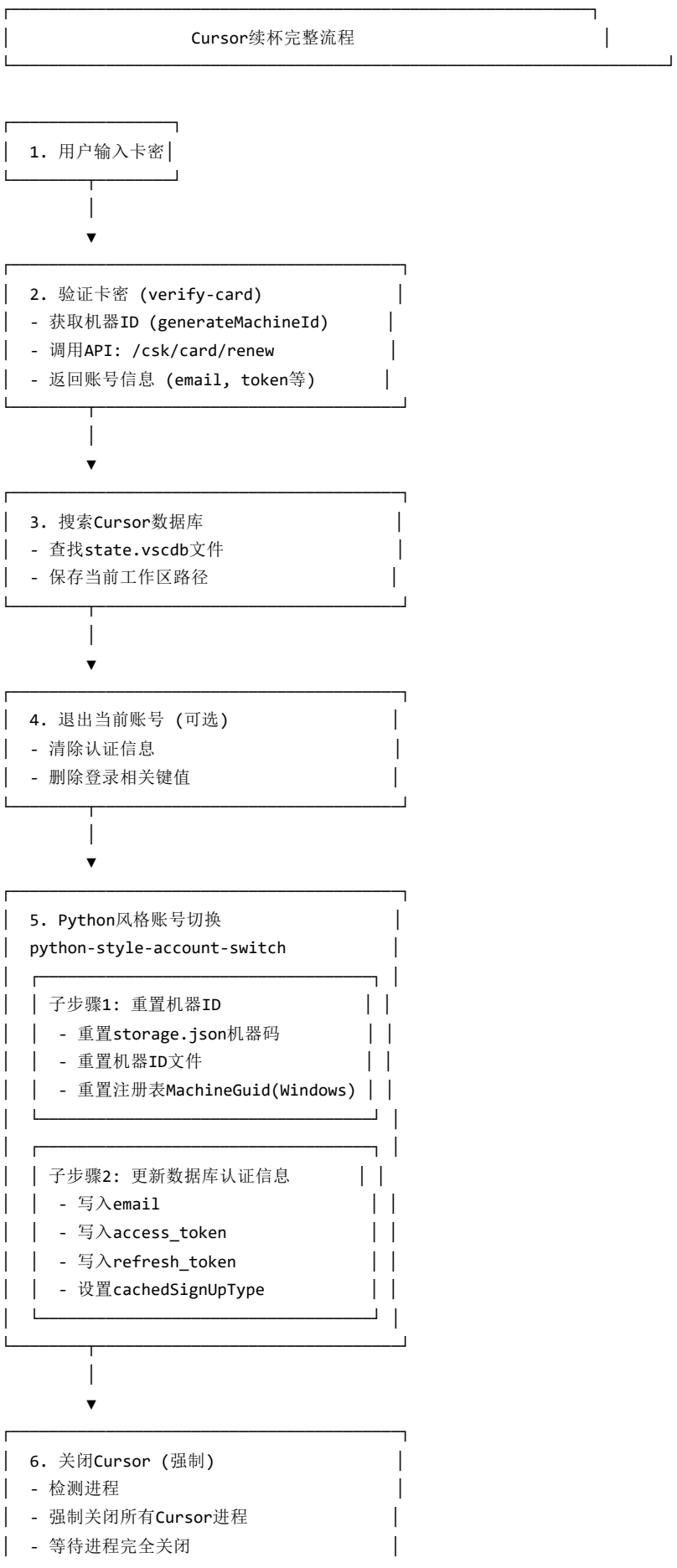
关键配置说明：

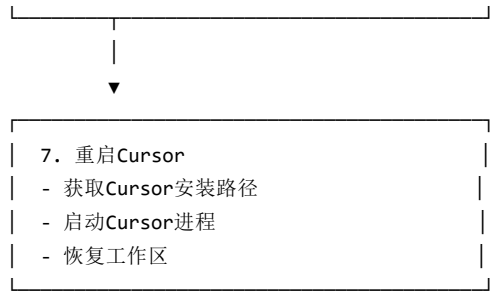
- http.proxy - SOCKS5 代理地址（含认证）
- http.proxySupport: "override" - 强制使用代理，覆盖系统设置
- cursor.general.disableHttp2: true - 禁用 HTTP/2，确保代理兼容性
- http.systemCertificates: false - 不使用系统证书，避免证书验证问题

续杯流程

提示词 分析`main.js`中`cursor`续杯的逻辑 。

流程图


Cursor续杯完整流程
1. 用户输入卡密
2. 验证卡密 (verify-card)
- 获取机器ID (generateMachineId)
- 调用API: /csk/card/renew
- 返回账号信息 (email, token等)
3. 搜索Cursor数据库
- 查找state.vscdb文件
- 保存当前工作区路径
4. 退出当前账号 (可选)
- 清除认证信息
- 删除登录相关键值
5. Python风格账号切换
python-style-account-switch
子步骤1: 重置机器ID
- 重置storage.json机器码
- 重置机器ID文件
- 重置注册表MachineGuid(Windows)
子步骤2: 更新数据库认证信息
- 写入email
- 写入access_token
- 写入refresh_token
- 设置cachedSignUpType
6. 关闭Cursor (强制)
- 检测进程
- 强制关闭所有Cursor进程
- 等待进程完全关闭



1. 卡密验证函数 - verify-card

```
// 添加续杯卡密的处理器（会更新数据库）
ipcMain.handle('verify-card', async (event, cardCode) => {
  // 1. 获取设备ID（用于绑定）
  const machineId = generateMachineId();

  // 2. 调用续杯API（会更新数据库）
  const apiUrl = 'http://129.204.108.166:2486/csk/card/renew';

  // 3. 返回账号信息 (email, token等)
  // ...
})
```

功能：

- 使用固定设备ID与后端验证
- 调用续杯接口（会更新后端数据库）
- 返回账号信息：email、access_token、refresh_token

返回数据结构：

```
{
  success: true,
  data: {
    card_info: {
      card: "卡密",
      address: "卡类型",
      start: "开始时间",
      end: "结束时间",
      usetime: "使用天数"
    },
    account_info: {
      userid: "用户ID",
      email: "用户邮箱",
      token: "访问令牌",
      current_time: "当前时间"
    },
    remaining_days: 剩余天数
  }
}
```

2. 退出当前账号函数 - logout-current-cursor-account

```
// 退出当前Cursor登录账号
ipcMain.handle('logout-current-cursor-account', async (event, dbPath) => {
  // 清除的认证相关键值
  const keysToDelete = [
    'cursorAuth/cachedEmail',
    'cursorAuth/accessToken',
    'cursorAuth/refreshToken',
    'cursorAuth/cachedSignUpType',
    'cursorai/featureStatusCache',
    'cursorai/featureConfigCache',
    'cursorAuth/stripeMembershipType',
    'cursorai/serverConfig',  // 重要：删除此配置以激活新账号
    'auth/user',
    'auth/session',
    'vscode.chat.access-token'
  ];
  // ...
})
```

功能：

- 清除旧账号的认证信息
- 确保切换新账号时无残留数据

3. 账号切换函数 - python-style-account-switch

3.1: 重置机器ID

```
// ========== 子步骤1：重置机器ID（重要：每次刷新Cursor时都要重置机器码）==========
// 1.1 重置storage.json中的机器码
await resetStorageMachineIds();

// 1.2 重置机器ID文件
await resetMachineIds(cursorDir);

// 1.3 重置Windows注册表中的MachineGuid（仅在Windows平台）
if (process.platform === 'win32') {
  await resetMachineGuid();
}
```

重置内容：

- storage.json 中的 telemetry.machineId、telemetry.devDeviceId、telemetry.sqmId
- 机器ID文件（machineid.json、machineid、anonymousid）
- Windows 注册表 MachineGuid

3.2: 更新数据库认证信息

```
// ========== 子步骤2: 更新数据库认证信息 ==========
const updates = [];
if (email) {
  updates.push(["cursorAuth/cachedEmail", email]);
}
if (access_token) {
  updates.push(["cursorAuth/accessToken", access_token]);
}
if (refresh_token) {
  updates.push(["cursorAuth/refreshToken", refresh_token]);
  updates.push(["cursorAuth/cachedSignUpType", "Auth_0"]);
}

// 使用事务更新数据库
const transaction = db.transaction(() => {
  for (const [key, value] of updates) {
    // 更新或插入键值对
  }
});
```

写入的数据库键：

- cursorAuth/cachedEmail - 用户邮箱
- cursorAuth/accessToken - 访问令牌
- cursorAuth/refreshToken - 刷新令牌
- cursorAuth/cachedSignUpType - 设置为 "Auth_0"

4. Cursor 重启流程 - restart-cursor-complete

```
// 完整的Cursor重启流程（关闭->等待->启动）
ipcMain.handle('restart-cursor-complete', async () => {
  // 1. 检查Cursor是否运行
  // 2. 如果运行中，强制关闭
  // 3. 等待进程完全关闭
  // 4. 启动Cursor
})
```

功能：

- 检测并强制关闭所有 Cursor 进程
- 等待进程完全关闭（约 2 秒）
- 启动新的 Cursor 进程
- 支持恢复工作区路径

API鉴权问题

所有API都没有鉴权，以下是proxy-config的示例。

```
Administrator@CHINAMI-A6GAF0V ~
$ curl http://129.204.108.166:2486/csk/proxy-config/current
{"data":{"update.enableWindowsBackgroundUpdates":false,"http.experimental.systemCertificates":false,"http.proxy":"socks5://hff:
$
```

总结

> 本文通过cursor分析cursor续杯工具的实现原理，并分析如何绕过设备限制，同时对API进行了一些简单的安全测试。cursor还是很强大的，但是输出内容导出不了markdown，这方面感觉挺难用的。我用豆包将输出的ascii表格将其转成md格式的表格，这个似乎又叫做aigc之类的。

> 邮箱：1458643480@QQ.COM
> QQ交流群：563882916

附录

1. 使用alert卡住electron界面,打开调试控制台