关于 sqlite3.dll 动态加载使用问题

写 pbidea 库时，有考虑把 sqlite3 作为一个功能封装进去，静态编译又觉得太大，所以决定把 sqlite3.dll 当作插件动态加载使用。

官网下载的 DLL，2 个文件，sqlite3.dll 和 sqlite3.def。通常是使用 lib 生成 sqlite3.lib，然后引入使用。这个方法不能动态加载作为插件使用，当 sqlite3.dll 不存在时，整个 pbidea 都会报错。所以，必须考虑 LoadLibrary 后作为插件使用。这就有个难题 ：它有好几百个函数，需要自己一个一个去声明，太费劲了。而且不同版本，它函数也有变化。

下载了 sqlite3 的源码，研究了一下，发现其实它有个接口

```
struct sqlite3_api_routines {

  void * (*aggregate_context)(sqlite3_context*,int nBytes);
  int  (*aggregate_count)(sqlite3_context*);
  int  (*bind_blob)(sqlite3_stmt*,int,const void*,int n,void(*)(void*));
  int  (*bind_double)(sqlite3_stmt*,int,double);
  int  (*bind_int)(sqlite3_stmt*,int,int);
  int  (*bind_int64)(sqlite3_stmt*,int,sqlite_int64);
          int  (*bind_null)(sqlite3_stmt*,int);

        ……..

static const sqlite3_api_routines sqlite3Apis = {
  sqlite3_aggregate_context,
#ifndef SQLITE_OMIT_DEPRECATED
          sqlite3_aggregate_count,

        ……..
```

sqlite3Apis 是可以直接导出所有函数指针的。那这个接口怎么取得呢？进一步研究，它这个接口是给它的扩展库使用的，具体的 sqlite3_load_extension 函数里面。sqlite3_load_extension 函数的第一个参数是 sqlite3 *db。那么问题来了：我还没加载它呢，哪来的 sqlite3 *db，这就成了先有鸡还是先有蛋的问题了。看来，走扩展库这条路也行不通。而我又不想几百个函数去折腾声明一遍。

于是，编译源码，在源码里添加了一个导出函数：

```
SQLITE_API const sqlite3_api_routines* get_sqlite3_interface()

{

    return &sqlite3Apis;
}
```

然后然后这个函数，果然可以取得一个接口，需要时 LoadLibrary(“sqlite3.dll”)，用完后 FreeLibrary。挺爽的。

可是，又遇到一个新问题，不小心用官网下载的 DLL 覆盖后我编译的 DLL 后，整个接口消失了。所以，自己编译的 DLL 不能再命名为 sqlite3.dll，需要改名。作为有点小洁癖的人来说，还是不够爽。

貌似到此处，已经没有办法了！

看到 static const sqlite3_api_routines sqlite3Apis = {}，忽然灵光一现，好象还能想点办法。能不能从 DLL 里查找到它呢？如果直接查找到它，就能直接取得这个接口，不必使用函数导出。

说干就干！（这个方法要求你对 PE 格式有一定了解）

```
//加载DLL
HMODULE m_dll_sqlite3 = LoadLibraryW(L"sqlite3.dll");
//取PE头信息
IMAGE_DOS_HEADER* pDosHeader = (IMAGE_DOS_HEADER*)m_dll_sqlite3;
IMAGE_NT_HEADERS* pNtHeader = (IMAGE_NT_HEADERS*)((unsigned long)pDosHeader + pDosHeader->e_lfanew);
IMAGE_OPTIONAL_HEADER* pOption = (IMAGE_OPTIONAL_HEADER*)&pNtHeader->OptionalHeader;
DWORD dwImageBase = pOption->ImageBase;

PIMAGE_EXPORT_DIRECTORY pImageExportDirectory = (PIMAGE_EXPORT_DIRECTORY)((DWORD)m_dll_sqlite3 +
pNtHeader->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
PIMAGE_IMPORT_DESCRIPTOR pImportTable = (PIMAGE_IMPORT_DESCRIPTOR)((DWORD)m_dll_sqlite3 +
pNtHeader->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress);

DWORD* pAddressOfFunction = (DWORD*)(pImageExportDirectory->AddressOfFunctions + (DWORD)m_dll_sqlite3);
DWORD* pAddressOfNames = (DWORD*)(pImageExportDirectory->AddressOfNames + (DWORD)m_dll_sqlite3);
DWORD dwNumberOfNames = (DWORD)(pImageExportDirectory->NumberOfNames);
WORD* pAddressOfNameOrdinals = (WORD*)(pImageExportDirectory->AddressOfNameOrdinals + (DWORD)m_dll_sqlite3);

//获取所有导出函数
std::map<DWORD, char*> funcs;
for (int i = 0; i < (int)dwNumberOfNames; i++)
{
	char* strFunction = (char*)(pAddressOfNames[i] + (DWORD)m_dll_sqlite3);
	funcs[pAddressOfFunction[i] + dwImageBase] = strFunction;
}

unsigned long  dwSectionNumber = pNtHeader->FileHeader.NumberOfSections;
IMAGE_SECTION_HEADER* pSectionHeader = IMAGE_FIRST_SECTION(pNtHeader);
IMAGE_DATA_DIRECTORY* pDataDirectory = pOption->DataDirectory;
for (DWORD i = 0; i < dwSectionNumber; i++)
{
	DWORD dwBeginVA = pSectionHeader[i].VirtualAddress + dwImageBase;
	DWORD dwEndVA = pSectionHeader[i].VirtualAddress + pSectionHeader[i].SizeOfRawData + dwImageBase;
	//static const sqlite3_api_routines* sqlite3_api; 所在 section，想知道为什么它在这里？经验很重要
	if (memcmp(pSectionHeader[i].Name, ".rdata", 6) == 0)
	{
		DWORD* p = (DWORD*)dwBeginVA;
		while (p < (DWORD*)dwEndVA)
		{
			//根据sqlite3_api_routines结构，成员是指向函数的指针，判断是不是 sqlite3_api
```

```
                        bool find = true;
                        for (int i = 0; i < 32; i++)
                        {
                                if (HIWORD(p[i]) && funcs.find(p[i]) == funcs.end())
                                {
                                        find = false;
                                        break;
                                }
                        }
                        if (find)
                        {
                                sqlite3_api = (const sqlite3_api_routines*)p;
                                break;
                        }
                        p++;
                }
        }
        if (sqlite3_api)
                break;
}
```

代码写好，编译完运行，果然取得了接口指针，完全实现了 sqlite3.dll 的动态加载使用了。

爽！爽！爽！

大自在,QQ:781770213

2023/9/19