KEDACOM 苏州科达科技有限公司
Suzhou Keda Technology Co.,Ltd.

SWIG封装C++库

(仅供内部使用)

版 本 号:	**V0.1**
保 密 等 级:	■秘密 □机密 □绝密
编 制:	李永军
审 核:	李永军

修订记录

日期	版本号	描述	作者
2024-06-18	0.1	初稿完成	李永军

目录

KEDACOM 苏州科达科技有限公司
Suzhou Keda Technology Co.,Ltd.

前言

关键词

`swig`、`golang`、`c++`、`cgo`

摘要

> 应项目需求，显控需要扩展开发一些会控功能，需要对接不同版本的会议平台。5.0+MCU 对接方式为 Http 协议。对接 4.7MCU，会议平台提供的是 C++ 风格的 SDK。
>
> 显控业务为 golang 语言开发。经调研评估，决定使用 SWIG 工具将 C++ 4.7MCUSDK 转换为 golang 接口直接调用。

案例描述

应项目需求，显控需要扩展开发一些会控功能，需要对接不同版本的会议平台。5.0+MCU 对接方式为 Http 协议。对接 4.7MCU，会议平台提供的是 C++ 风格的 SDK。显控业务已设计 MCU 适配器模块，封装各版本 MCU 差异，对显控客户端提供统一的会控接口。

案例分析

显控业务为 golang 语言开发。对接 4.7MCUSDK 有两种思路：

- 将 4.7MCUSDK 封装成 Http 协议接口或 RPC 接口，供适配器业务使用
- 基于 cgo 将 4.7MCUSDK 封装为 golang 接口，供适配器业务使用

第一种方案，显控需要新增一个业务进程对 4.7MCUSDK 进行封装，业务适配器模块调用封装后的接口，但是 4.7MCUSDK 接口和 5.0+Http协议并不相同，适配器模块也要做一些兼容逻辑。多一层封装，联调测试相对复杂些，新增进程还要考虑时序问题。

第二种方案，基于 cgo 将 4.7MCUSDK 封装为 golang 接口，在业务MCU适配器模块直接对接 4.7MCUSDK。cgo 封装难度也比较大。后询问 ChatGPT 了解到 SWIG 这个开源工具，只需要编写一个 mcusdk.i 接口文件就可以将 C++ 接口转换为 golang 接口。

经调研评估。决定使用 SWIG 工具将 C++ 4.7MCUSDK 转换为 golang 接口直接调用。

解决过程

一、SWIG 安装

1、源码编译

Getting Started · swig/swig Wiki · GitHub

2、apt install swig

我选择的是联网安装，方便快捷。SWIG 版本为V4.0，本文涉及的 SWIG 测试程序全部基于此版本。

```
root@Tiger:/mnt/lyjwin/project/cgo_demo/swig_kdvtype# swig -version

SWIG Version 4.0.1

Compiled with g++ [x86_64-pc-linux-gnu]

Configured options: +pcre

Please see http://www.swig.org for reporting bugs and further information
```

二、SWIG 学习

基本使用参阅 Go通过swig封装、调用c++共享库的技术总结

官方文档参阅 SWIG and Go

官方示例参阅 SWIG/Examples/go

cgo 不支持 C++，SWIG 是先将 C++ 接口封装为 C 接口，然后基于 cgo 封装为 go 包。

三、SWIG demo

自己编写的测试 Demo，用于学习测试 SWIG 转换规则。源码见附件 `swig_kdvtype.zip`。

```
root@ubuntu-Vostro-3268:/mnt/lyjwin/project/cgo_demo/swig_kdvtype# tree
.
├── swig                        #SWIG 转换测试源文件
│   ├── kdvtest                 #C++ 源代码
│   │   ├── build_cpp.sh
│   │   ├── kdvtest.cpp
│   │   ├── kdvtest.h
│   │   ├── kdvtype.h
│   │   └── libkdvtest.a
│   ├── kdvtest.go              #SWIG 生成的 Go 包
│   ├── kdvtest_wrap.cxx        #SWIG 生成的封装文件
│   ├── kdvtest_wrap.h          #SWIG 生成的封装文件
│   ├── kdvtest.i               #SWIG 转换接口文件(在这里定义转换规则)
│   ├── gocallback.go           #Go   中实现接收处理 C++ 回调
│   └── swig_build.sh
├── test.go                     #调用 SWIG 生成的 Go 包测试代码
└── go.mod

2 directories, 15 files
```

四、SWIG 4.7MCUSDK

SWIG 封装 4.7MCUSDK 源文件

源码见附件 `mcusdk.zip`

```
root@ubuntu-Vostro-3268:/mnt/lyjwin/project/MSP/40-servers/mpuaps/library/mcusdk# tree
.
├── include                             #4.7MCUSDK 头文件
│   ├── dcconst.h
│   ├── evmcslibadp.h
│   ├── kdvdef.h
│   ├── kdvtype.h
│   ├── mcsdllapi.h
│   ├── mcsdllconst.h
│   ├── mcsdllerrorid.h
│   └── mcsdllstruct.h
├── lib                                  #MCUSDK 依赖库
│   ├── euler_arm64
│   └── ubuntu_amd64
│       ├── debug
│       │   ├── libkdvencrypt.so
│       │   ├── libkdvlog.so
│       │   ├── libkdvmedianet.so
│       │   ├── libkprop.so
│       │   ├── libmcsdll_64.so
│       │   └── libosp.so
│       └── release
│           ├── libkdvencrypt.so
│           ├── libkdvlog.so
│           ├── libkdvmedianet.so
│           ├── libkprop.so
│           ├── libmcsdll_64.so
│           └── libosp.so
├── mcusdk.go                           #SWIG 生成的 Go 包
├── mcusdk_wrap.cxx                     #SWIG 生成的封装文件
├── mcusdk_wrap.h                       #SWIG 生成的封装文件
├── mcusdk.i                            #SWIG 转换接口文件(在这里定义转换规则)
├── gocallback.go                       #Go   中实现接收处理 C++ 回调
└── swig_convert.sh

6 directories, 27 files
```

SWIG 接口文件

```
%module(directors="1") mcusdk

//声明SWIG包装生成的cxx文件中包含的头文件
%{
#include "include/kdvdef.h"
#include "include/mcsdllapi.h"
%}

#define MCSDLL_API extern "C"

//C++回调类
%feature("director") CMcsCBHandler;

//包含SWIG标准接口文件
%include "typemaps.i"
%include "stdint.i"
%include "cstring.i"
%include "std_string.i"
%include "std_vector.i"

//64位编译器中， s32(int32_t) 映射到 golang 中应该是 int32 类型(4字节)
//但是swig默认将 s32(int32_t) 映射为 golang 中的 int 类型(golang中int是8字节)
//当s32值为负数时会有问题, 导致 c 侧和 golang 侧值不一样[占用字节数不同,值的表示范围不一样,会引发一些问题]
//例如:
//C语言           s32   -2            32bit  FFFF FFFE
//转为golang值为   int   4294967294    64bit  0000 0000 FFFF FFFE

//此处声明映射关系
//TODO 类型映射规则不生效, 待测试
// %typemap(in) int32_t {
//     $1 = (int32_t)$input;
// }
// %typemap(out) int32_t {
//     $result = (int32_t)$input;
// }
// %apply int32_t { int32 };

//C++[数组||指针]类型转换
%include "carrays.i"
%include "cpointer.i"
%array_functions(s8, byteArray)
%array_functions(s32, s32Array)
%array_functions(DCTCallMtInfo, DCTCallMtInfoArray)
%array_functions(DCTMtBitrate, DCTMtBitrateArray)
%array_functions(DCTVmpMember, DCTVmpMemberArray)
%array_functions(DCTConfVmpStatus, DCTConfVmpStatusArray)
%array_functions(DCTMtInfo, DCTMtInfoArray)
%array_functions(DCTMtParambyMcs, DCTMtParambyMcsArray)
%array_functions(DCTMtStatus, DCTMtStatusArray)
%array_functions(DCTMTStatusnfy, DCTMTStatusnfyArray)
%array_functions(DCTConfE164Info, DCTConfE164InfoArray)
%array_functions(DCTConfE164, DCTConfE164Array)
%array_functions(DCTHduChnlCfgInfo, DCTHduChnlCfgInfoArray)
%array_functions(DCTHduStyleCfgInfo, DCTHduStyleCfgInfoArray)
%array_functions(DCTHduChnStatus, DCTHduChnStatusArray)
%array_functions(DCTHduStatusList, DCTHduStatusListArray)
%array_functions(DCTMtPollParam, DCTMtPollParamArray)
%array_functions(DCTRecStatus, DCTRecStatusArray)
%array_functions(DCTRecChnnlStatus, DCTRecChnnlStatusArray)


//声明要导出的类和结构体定义
%include "include/kdvtype.h"
%include "include/kdvdef.h"
%include "include/dcconst.h"
%include "include/evmcslibadp.h"
%include "include/mcsdllconst.h"
%include "include/mcsdllerrorid.h"
%include "include/mcsdllstruct.h"
%include "include/mcsdllapi.h"

//声明需要导出的C++接口
```

问题记录

1、C++字节对齐宏错误

```
// SWIG ERR
//  mcusdk define err packed

//=============================================================================================
//dcconst.h::557
 #define PACKED __attribute__((__packed__)) // 取消编译器字节对齐优化

//mcsdllstruct.h
struct DCTMcsInfo {
    DCTMcsInfo(s8* szLogPath, u8 byFileNum=8, u32 dwFileSize=1<<20)
    {
        DCTMcsInfo() ;
        memcpy(m_LogPath, szLogPath, strlen(szLogPath) + 1 ) ;
        m_byLogFileNum = byFileNum;
        m_dwFileSize = dwFileSize;
    }

    DCTMcsInfo()
    {
        memset(this, 0, sizeof(DCTMcsInfo));
    }

    s8 m_LogPath[260];   //add by jianghuan 2014-4-3 增加日志文件路径和名称
    u8 m_byLogFileNum;  //日志文件个数
    u32 m_dwFileSize;   //单个日志文件大小
}
PACKED;
//=============================================================================================


//在数据及结构末尾添加 __attribute__((__packed__)) 取消编译器字节对齐优化，SWIG 不支持此语法。
//改用 #pragma pack(1) 设置1字节对齐
#define PACKED
#pragma pack(1)
```

2、C++编译字节对齐问题

```
// 类似这种数据结构定义，m_dwFileSize 参数传到MCU侧值为0 ???
struct DCTMcsInfo {
    DCTMcsInfo(s8* szLogPath, u8 byFileNum=8, u32 dwFileSize=1<<20)
    {
        DCTMcsInfo() ;
        memcpy(m_LogPath, szLogPath, strlen(szLogPath) + 1 ) ;
        m_byLogFileNum = byFileNum;
        m_dwFileSize = dwFileSize;
    }

    DCTMcsInfo()
    {
        memset(this, 0, sizeof(DCTMcsInfo));
    }

    s8 m_LogPath[260];  //add by jianghuan 2014-4-3 增加日志文件路径和名称
    u8 m_byLogFileNum;  //日志文件个数
    u32 m_dwFileSize;   //单个日志文件大小
}
PACKED;


//在使用 MCUSDK 头文件编译时，为正确设置字节对齐方式，编译的二进制程序与 MCUSDK 动态库字节对齐方式不一致。导致传参错误
//参照第一个问题，正确设置字节对齐方式即可。
```

3、C++内联警告

```
// SWIG 警告
// include/mcsdllstruct.h:3587: Warning 312: Nested union not currently supported (ignored).


//=============================================================================================
struct DCTFrameInfo
{
public:
    DCTFrameInfo()
```

```
    {
        memset(this, 0, sizeof(DCTFrameInfo));
    }
public:
    u8     m_byMediaType; //媒体类型
    u8    *m_pData;       //数据缓冲
    u32    m_dwPreBufSize;//m_pData缓冲前预留了多少空间，用于加
                          // RTP option的时候偏移指针一般为12+4+12
                          // (FIXED HEADER + Extence option + Extence bit)
    u32    m_dwDataSize;  //m_pData指向的实际缓冲大小缓冲大小
    u8     m_byFrameRate; //发送帧率,用于接收端
    u32    m_dwFrameID;   //帧标识，用于接收端
    u32    m_dwTimeStamp; //时间戳，用于接收端
    u32    m_dwSSRC;      //同步源，用于接收端

    union
    {
        struct{
            BOOL32    m_bKeyFrame;    //频帧类型（I or P）
            u16       m_wVideoWidth;  //视频帧宽
            u16       m_wVideoHeight; //视频帧宽
            BOOL32    m_bHighProfile; // H264 HP标识 - 2012/03/01
        }m_tVideoParam;
        u8    m_byAudioMode;//音频模式
    };

}
PACKED;
//==============================================================================================

//数据结构未使用，忽略未处理
```

4、C++ Const常量转换

```
// SWIG 生成封装文件后
// 编译报错：报未定义的常量类型
mcusdk_wrap.cxx: In function ‘uint16_t* _wrap_PORT_NMS_get_mcusdk_2ce382c49db0fb1b()’:
mcusdk_wrap.cxx:294:17: error: ‘PORT_NMS’ was not declared in this scope
   result = (u16)PORT_NMS;

//其实已在 kdvdef.h 定义



//SWIG 转换接口文件
```

5、C++ 回调类转换

官方示例：[SWIG/Examples/go/callback]

参考文章： [https://erpeng.github.io/2019/10/11/swig/]

```
//mcsdllstruct.h 修改如下
//=============================================================================================================================
class CMcsCBHandler
{
public:
    //virtual ~CMcsCBHandler() {}

    /*=============================================================================
    函 数 名： CBHandle
    功    能:
    算法实现:
    全局变量:
    参    数： u32 dwEventId      消息号;
               u32 dwMsgBody      消息体;
               u32 dwMsgLength    消息体大小;
               u32 nMcuIndex      MCU索引号;
    ==============================================================================*/
    virtual u16 CBHandle(u32 dwEventId, uintptr_t dwMsgBody, u32 dwMsgLength, u32 nMcuIndex) = 0;
};


//TODO C++头文件添加回调类定义
//     Swig Used
class CMcsCBCaller {
private:
    CMcsCBHandler *_callback;
```

```
public:
    CMcsCBCaller(): _callback(0) {}
    ~CMcsCBCaller() { delCallback(); }
    void delCallback() { delete _callback; _callback = 0; }
    void setCallback(CMcsCBHandler *cb) { delCallback(); _callback = cb; }
    u16 call(u32 dwEventId, uintptr_t dwMsgBody, u32 dwMsgLength, u32 nMcuIndex) {
        if (_callback) {
            return _callback->CBHandle(dwEventId, dwMsgBody, dwMsgLength, nMcuIndex);
        } else {
            return 0;
        }
    }
};
//=============================================================================================================================
```

6、cgo 混编 C 源码

```
# C源文件为[CPP|CXX]文件编译的，cgo 调用会有问题
# c++ 导出的函数和 .h 文件中定义的函数名称不相同

# 要编译 C++ 源码，不能使用 go build qsort.go 这样直接编译.
# 需要编译整个文件夹: go build -o qsort | go run .
```

7、MCUSDK 错误

```
// 编译报错 -- 会议 MCUSDK 中 DCTConfBrdstStatus::SetConfE164 未实现
/tmp/go-build1488731067/b001/_x003.o: In function `_wrap_DCTConfBrdstStatus_SetConfE164_mcusdk_835918b53861b283':
./mcusdk_wrap.cxx:10489: undefined reference to `DCTConfBrdstStatus::SetConfE164(char const*)'
collect2: error: ld returned 1 exit status

// mcsdllstruct.h 头文件中实现该接口
//=============================================================================
inline void DCTConfBrdstStatus::SetConfE164(const s8* pchConfE164)
{
    memset(m_achConfE164, 0, sizeof( m_achConfE164 ));
    if(pchConfE164 != NULL)
    {
        strncpy(m_achConfE164, pchConfE164, sizeof(m_achConfE164));
        m_achConfE164[sizeof( m_achConfE164 ) - 1] = '\0';
    }
}
//=============================================================================
```

8、运行报错(Go pointer)

```
# 运行报错
runtime error: cgo argument has Go pointer to Go pointer

# 可以通过设置 cgo 内存监测环境变量忽略此报错
export GODEBUG=cgocheck=0
https://blog.51cto.com/u_15127705/4565198
```

上述报错表明代码中存在一个Go语言指针的指针（即一个指向指针的指针）传递给C语言函数。CGO不允许这样做，因为Go的垃圾收集器无法跟踪C代码中的指针，特别是当它们指向Go分配的内存时。

这个问题通常发生在以下几种情况：

你尝试传递一个指向Go切片或数组的指针给C函数。
你尝试传递一个指向结构体的指针给C函数，而该结构体内部包含了指向Go内存的指针字段。
解决方法：

- 方法1：避免传递指针的指针
  最简单的解决方案是避免传递指针的指针。如果可能，重新设计你的C函数接口，使其接受一个指向值的指针而不是指针的指针。
- 方法2：使用C兼容的数据结构
  如果必须传递复杂的数据结构给C，考虑使用C兼容的数据结构，或者将Go的数据结构转换为C的数据结构。例如，你可以将Go的切片转换为C的数组。
- 方法3：传递副本
  如果数据结构不大，你可以传递数据的副本给C函数，而不是指针。这样，C函数可以修改副本而不影响原始的Go数据。
- 方法4：使用unsafe包
  作为最后的手段，你可以使用Go的unsafe包来操作指针。但是，这需要非常小心，因为unsafe操作可能会绕过Go的内存安全保证。

示例：
假设你有一个Go的切片，你想将其传递给C函数。错误的做法是：

```
// 错误的示例：不要这样做
var slice []int
cFunction(&slice) // 这会导致运行时错误
```

正确的做法是传递切片的头指针给C函数：

```
// 正确的示例
var slice []int
cSlice := (*C.int)(C.malloc(C.size_t(len(slice)) * int(unsafe.Sizeof(slice[0]))))
C.memcpy(cSlice, unsafe.Pointer(&slice[0]), C.size_t(len(slice))*int(unsafe.Sizeof(slice[0])))

// 调用C函数
cFunction(cSlice, C.int(len(slice)))

// 释放C分配的内存
C.free(unsafe.Pointer(cSlice))
//请注意，使用unsafe包和CGO时，你需要确保手动管理内存，包括分配和释放。这增加了代码的复杂性和出错的风险，因此应该谨慎使用。
```

9、MCUSDK 问题

```
#SDK初始化更改了程序工作路径
goframe 日志组件，MCUSDK加载后，不输出日志了？？？
```

10、cgo 引发内存越界问题排查方案

```
cgo 异常如何调试堆栈??
内存越界问题如何排查??
https://blog.csdn.net/flynetcn/article/details/125090377
```

总结

现在 AI 盛行，遇到问题询问 GPT 能够帮我们拓宽思路，并能了解到新的技术方案。

golang 业务引入了 cgo，内存变的不再安全，特别注意 cgo 相关代码内存的申请释放以及指针操作。

SWIG 不仅可以将 C++ 组件转换为 golang 接口，也可以转换为 jave、python 等语言的接口，是一个值得学习的强大工具。

引用

golang

cgo doc

cgo demo

swig doc

swig github rep

swig demo1

swig demo2

swig zh_CN doc

GitHub c-for-go