CApi.h说明文档

1. 所有关于字符串的编码都是UTF-8
2. 网络会话的读写是多线程安全的, 其安全性依赖于事件核心. 注: **只是读写是多线程安全**

事件核心构建和功能说明

```
///< 构建是构建一个事件核心
HEventCore * CreateEventCore();

///< 这是释放掉事件核心, 内部会自动取消事件循环的阻塞状态.
void FreeEventCore(const HEventCore *core);

///< 阻塞当前进程, 将当前进程转换成事件进程.
HBool Exec(const HEventCore *core);

///< 不阻塞, 内部会自行开一个线程当事件线程.
HBool Run(const HEventCore *core);

///< 通知退出事件循环.
void Quit(const HEventCore *core);

///< 如有自己的事件框架, 又不使用Exec和Run时, 需自行推动事件任务则该调用下列3个接口, 需要和创建时同一进程
///< 需要自行同步事件需绑定通知是否有同步任务通知回调(请注意: 通知可同步的调用是调度线程, 而不是事件线程, 请勿在调度线程中执行SyncEvent, 此操作是无效的)
typedef void (*syncEvnetNofifyCallBack)(const HEventCore *core, void *userData);

void BindSyncEventNotifyCallBack(const HEventCore *core, syncEvnetNofifyCallBack
callBack, void *userData);
void FreeSyncEventNotifyCallBack(const HEventCore *core);
///< 推动事件任务, 需要和创建事件核心是同一进程
void SyncEvent(const HEventCore *core);
```

网络会话构建和功能说明

```
///< 创建一个网络会话, core是事件核心, protocol是协议枚举, 详情看头文件枚举, 当协议枚举是不存在的数值时, 默认构建SDK2.0
HSession * CreateNetSession(const HEventCore *core, eNetProtocol protocol);

///< 释放协议会话, 内部会释放该协议的内存
void FreeNetSession(HSession *session);

///< 设置协议会话功能
HBool SetNetSession(HSession *session, int type, void *data);
```

网络会话功能设置说明

```
///< 通用回调, 所有网络会话的协议都会触发这个回调

///< 网络状态回调, 网络连接或者断开时会触发该回调
```

```
///< currSession是当前的网络会话
///< status 是当前网络会话的状态，详情看头文件枚举
///< userData 是用户自己传递的数据
typedef void (*NetStatusCallBack)(HSession *currSession, eNetStatus status, void
*userData);

///< Tcp回调，只有类型是TCP的才会触发此回调

///< 数据读取回调，协议处理完数据后会触发该回调
///< data 是读取的数据
///< len 是读取数据的长度
typedef void (*ReadyReadCallBack)(HSession *currSession, const char *data,
huint32 len, void *userData);

///< 执行上传文件回调，定时触发该回调(已弃用，使用下面UploadFileFlowProgressCallBack)
///< fileName 是上传文件的文件名
///< sendSize 是当前发送数据的大小
///< fileSize 是文件的大小
///< status 是当前上传文件状态，详情看头文件枚举
typedef void (*UploadFileProgressCallBack)(HSession *currSession, const char
*fileName, hint64 sendSize, hint64 fileSize, eUploadFileStatus status, void
*userData);

///< 发送上传文件进度回调，定时会触发该回调
///< sendSize 是当前发送数据的大小
///< fileSize 是文件的大小
///< status 是当前上传文件状态，详情看头文件枚举
typedef void (*UploadFileFlowProgressCallBack)(HSession *currSession, hint64
sendSize, hint64 fileSize, eUploadFileStatus status, void *userData);

///< 错误信息回调，接收到错误码时会触发该回调
///< status 是文档中说明的错误码值
typedef void (*ErrorCodeCallBack)(HSession *currSession, int status, void
*userData);


///< Tcp服务端回调
///< currSession 当前服务的会话
///< newSession 连接过来的客户端会话，不需要使用时一样需要调用释放接口
typedef void (*NewConnect)(HSession *currSession, HSession *newSession, void
*userData);

///< Udp回调
///< 设置探测设备后会触发该回调.会有id，ip和原始的读取数据
typedef void (*DeviceInfoCallBack)(HSession *currSession, const char *id,
huint32 idLen, const char *ip, huint32 ipLen, const char *readData, huint32
dataLen, void *userData);
```

网络操作接口

```
///< 判断当前会话是否在连接状态
HBool Isconnect(const HSession *session);

///< 连接会话
HBool Connect(HSession *session, const char *ip, int port);

///< 断开连接
```

```
void Disconnect(HSession *session);

///< 发送数据接口, SDK直接发送xml数据即可. 内部会进行数据处理
HBool SendSDK(HSession *session, const char *data, huint32 len);

enum eFileType {
    kImageFile          = 0,    ///< 图片
    kVideoFile          = 1,    ///< 视频
    kFont               = 2,    ///< 字体
    kFireware           = 3,    ///< 固件
    kFPGAConfig         = 4,    ///< 常规情况下不需要使用
    kSettingCofnig      = 5,    ///< 常规情况下不需要使用
    KProjectResources   = 6,    ///< 资源文件, 常规情况下不需要使用
    kData               = 7,    ///< 常规情况下不需要使用
    kTemp               = 8,    ///< 常规情况下不需要使用
    kTempImageFile      = 128,  ///< 临时图片文件
    kTempVideoFile      = 129,  ///< 临时视频文件
};
///< 发送文件接口, 传入文件路径和文件类型
HBool SendFile(HSession *session, const char *filePath, int type);
```