iOS 网宿低延迟直播

接入手册

2021-07-20

目录

1.SDK 结构

- WsRTC.framework　　SDK 对应 Framework 动态库
  - wsrtc_sdk_wrapper.h　　SDK 头文件
- source (demuxer 插件源文件)
  - wsrtcdec.c　　demuxer 插件源文件

2.SDK 嵌入说明

使用网宿低延迟直播 SDK 可以集成到依赖 FFmpeg 的播放器引擎，以及不依赖 FFmpeg 的播放器引擎；对于非依赖 ffmpeg 的自研播放器引擎可以参照 wsrtcdec.c 文件完成 demuxer 的开发，本示例展示的为以 ffmpeg 为播放器引擎的 ijkplayer 集成说明；

2.1. Ijkplayer 集成说明

本文默认您已知悉 ijkplayer 的编译方式，具体方法参考相关开源项目：（https://github.com/bilibili/ijkplayer.git）。

Ijkplayer 集成方式主要有两种：

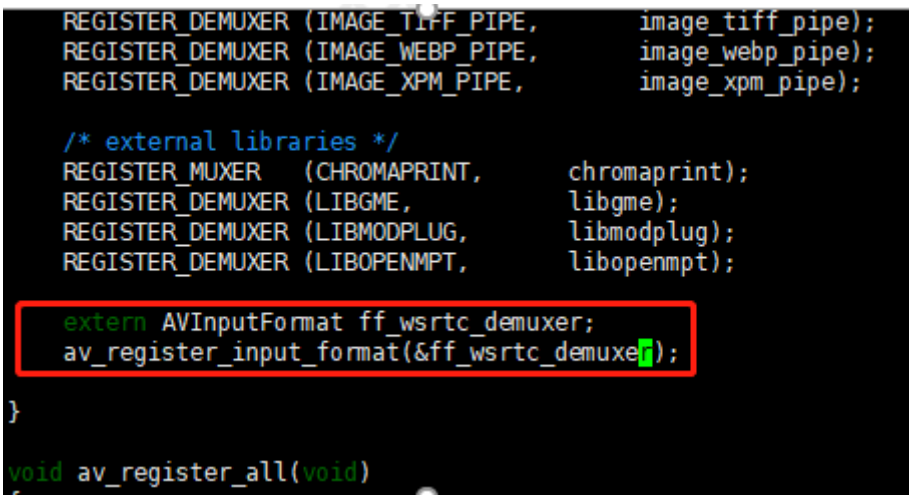
集成方法	优点	缺点
在 FFmpeg 中注入 demuxer 插件	使用简单，不需要修改播放器代码	需要重新编译 FFmpeg
Ijkplayer 中直接添加 AVInputFormat	不需要编译 ffmpeg	ff_player.c 中需要添加部分逻辑代码

2.1.1. 在 FFmpeg 中注入 demuxer 插件

a．在 ijkplayer 目录下，执行 init-ios.h

b. 进入 ijkplayer/ios 目录，将 wsrtcdec.c 复制到复制到 ffmpeg-$arch/libavformat 中，修改 libavformat/Makefile 和 allformats.c

修改 allformats.c 中的 register_all 函数，默认支持 WSRTC 协议：




```
extern AVInputFormat ff_wsrtc_demuxer;
av_register_input_format(&ff_wsrtc_demuxer);
```

c. 编译

在 ijkplayer/ios 目录下，执行./compile-ffmpeg.sh all

d. 查看 ijkplayer/ios/build/universal 目录下是否有输出的 FFmpeg 库

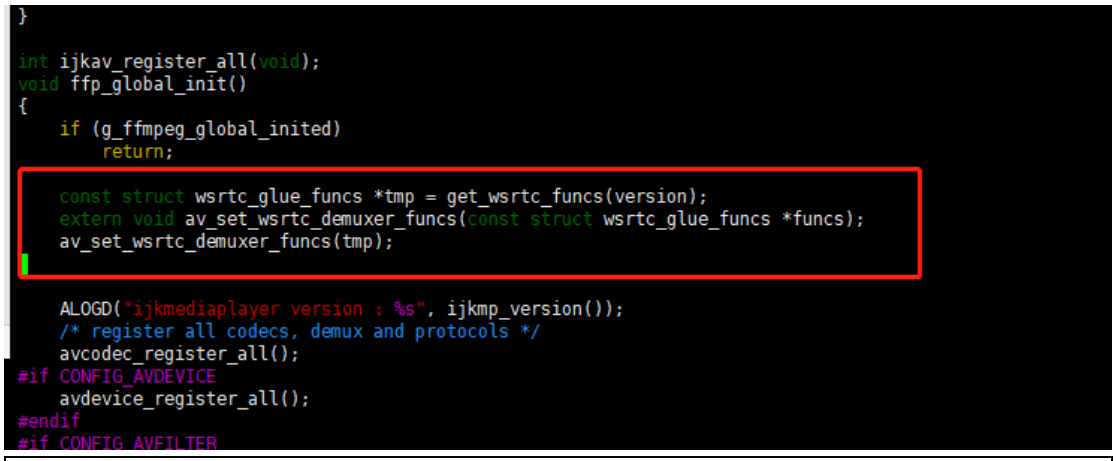
e. 将 WsRTC.framework 放入到 ijkplayer/ios/IJKMediaDemo/IJKMediaDemo 目录下

f. 使用 xcode 打开 ios/IJKMediaDemo/IJKMediaDemo.xcodeproj

g. 为 IJKMediaPlayer.xcodeproj 添加 WsRTC.framework 的依赖

h. ff_player.c 加入 wsrtc 逻辑

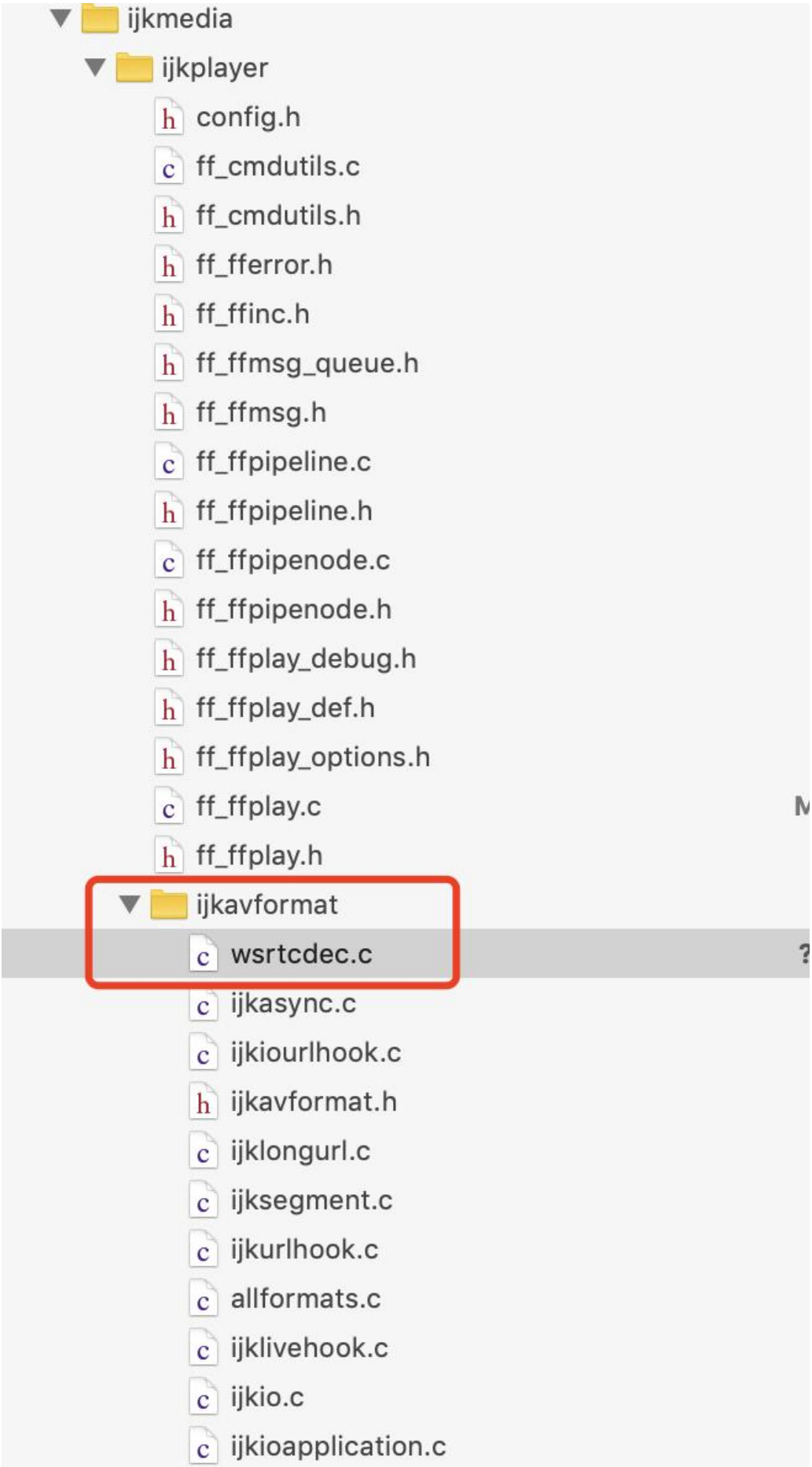
修改 ijkplayer/ijkmedia/ijkplayer/ff_ffplay.c，中的 ffp_global_init 函数：



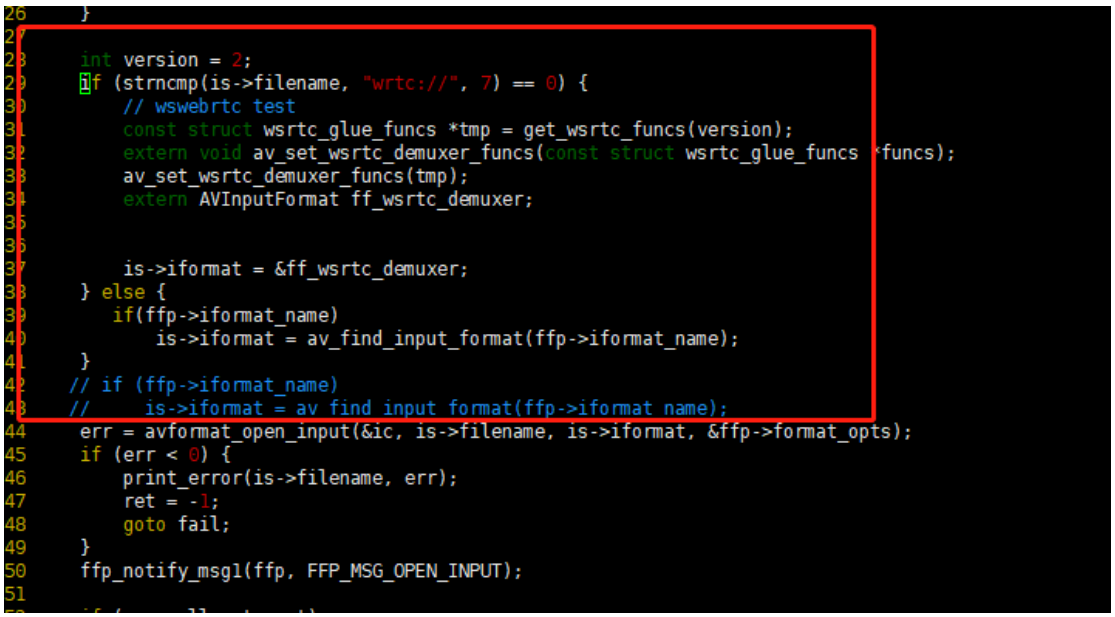

```
const struct wsrtc_glue_funcs *tmp = get_wsrtc_funcs(version);
extern void av_set_wsrtc_demuxer_funcs(const struct wsrtc_glue_funcs *funcs);
av_set_wsrtc_demuxer_funcs(tmp);
```

2.1.2. Ijkplayer 中直接添加 AVInputFormat

a．编译 ffmpeg，不需要依赖 wsrtc sdk 拓展 ffmpeg；
b． 将 wsrtcdec.c 拖入到工程中


ijkmedia
ijkplayer
config.h
ff_cmdutils.c
ff_cmdutils.h
ff_fferror.h
ff_ffinc.h
ff_ffmsg_queue.h
ff_ffmsg.h
ff_ffpipeline.c
ff_ffpipeline.h
ff_ffpipenode.c
ff_ffpipenode.h
ff_ffplay_debug.h
ff_ffplay_def.h
ff_ffplay_options.h
ff_ffplay.c
ff_ffplay.h
ijkavformat
wsrtcdec.c
ijkasync.c
ijkiourlhook.c
ijkavformat.h
ijklongurl.c
ijksegment.c
ijkurlhook.c
allformats.c
ijklivehook.c
ijkio.c
ijkioapplication.c

c. 修改 ff_player.c 代码。



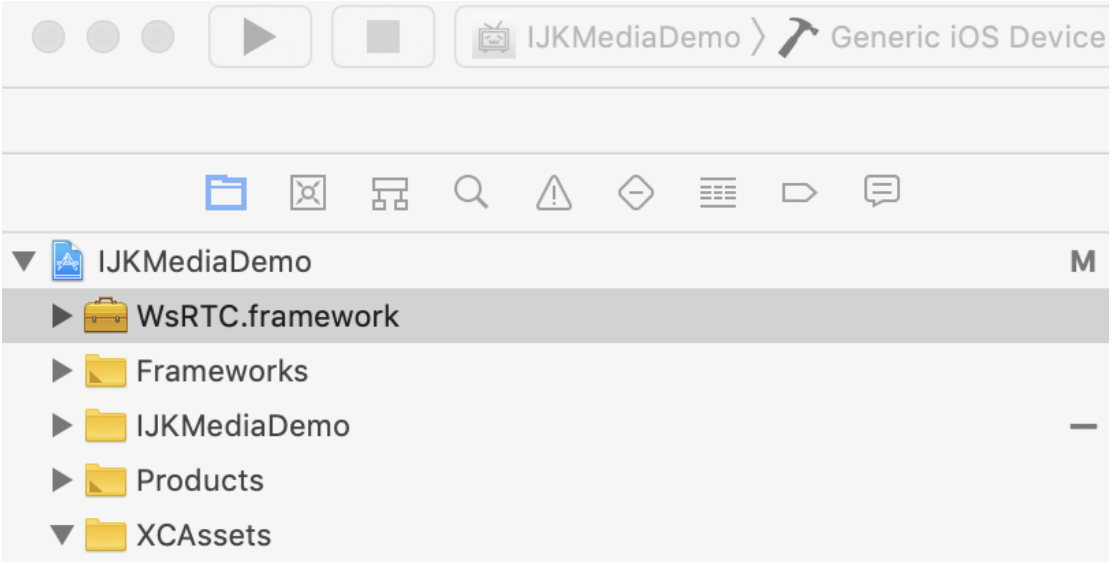
```
    int version = 2;
    if (strncmp(is->filename, "wrtc://", 7) == 0) {
        // wswebrtc test
        const struct wsrtc_glue_funcs *tmp = get_wsrtc_funcs(version);
        extern void av_set_wsrtc_demuxer_funcs(const struct wsrtc_glue_funcs *funcs);
        av_set_wsrtc_demuxer_funcs(tmp);
        extern AVInputFormat ff_wsrtc_demuxer;


        is->iformat = &ff_wsrtc_demuxer;
    } else {
       if(ffp->iformat_name)
           is->iformat = av_find_input_format(ffp->iformat_name);
    }
   // if (ffp->iformat_name)
   //     is->iformat = av_find_input_format(ffp->iformat_name);
    err = avformat_open_input(&ic, is->filename, is->iformat, &ffp->format_opts);
    if (err < 0) {
        print_error(is->filename, err);
        ret = -1;
        goto fail;
    }
    ffp_notify_msg1(ffp, FFP_MSG_OPEN_INPUT);
```

```
int version = 2;
    if (strncmp(is->filename, "wrtc://", 7) == 0) {
        // wswebrtc test
        const struct wsrtc_glue_funcs *tmp = get_wsrtc_funcs(version);
        extern void av_set_wsrtc_demuxer_funcs(const struct wsrtc_glue_funcs *funcs);
        av_set_wsrtc_demuxer_funcs(tmp);
        extern AVInputFormat ff_wsrtc_demuxer;


        is->iformat = &ff_wsrtc_demuxer;
    } else {
       if(ffp->iformat_name)
           is->iformat = av_find_input_format(ffp->iformat_name);
    }
   // if (ffp->iformat_name)
//      is->iformat = av_find_input_format(ffp->iformat_name);
```

d. 将 wsrtc sdk 的 framework 拖入到工程中



3.功能概览

网宿低延迟直播 SDK 是基于 webRTC 技术的一种音视频传输技术，可以从网宿 RTC 服务器拉取低延迟实时直播流，实现低延迟直播，可支持将直播流 demuxer 为 H264 以及 PCM 提供给播放器进行解码。

网宿低延迟直播 SDK 当前支持多实例，每条流均可以通过 open 创建，close 关闭，通过 read 读取数据。如下为示例代码:

```
/


#include "wsrtc_sdk_wrapper.h"
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <sys/time.h>
#include <unistd.h>


static char *addr_to_string(const void *v, char buf[64])
{
    sprintf(buf, "%llu", (unsigned long long)v);
    return buf;
```

```
}

//日志回调函数
static int output_log(void *s, int level, const char *fmt, va_list
args)
{
    (void) s;
    char str[1024];
    vsnprintf(str, 1000, fmt, args);
    printf("> %d %s\n", level, str);
    return 0;
}

//消息回调函数
static int on_event(void *s,
    int type,
    void *data, // temp data, do not cache it for later use!
    long long data_size)
{
    (void) s;
    printf("msg: %d %s\n", type, (data == NULL ? "null" : (const char
*)data));
    return 0;
}

int main(int argc, const char **argv)
{
    const struct wsrtc_glue_funcs *fs = get_wsrtc_funcs(1); /
    if(fs == NULL) {
        printf("Error: failed to get wsrtc functions. check
version.\n");
        return -1;
    }

    if(argc != 2) {
        printf("Usage: %s <wsrtc url>\n", argv[0]);
        return -1;
    }

    const char *url = argv[1];

    char buf[64];

    //设置外置 log function
```

```
    fs->preconfig("LogCallback",          addr_to_string(output_log,
buf));
    fs->preconfig("LogCbParam",         addr_to_string(NULL, buf));
    //设置消息回调
    fs->preconfig("MessageCallback",        addr_to_string(on_event,
buf));
    fs->preconfig("MessageCbParam",     addr_to_string(NULL, buf));

    //打开连接
    void *h = fs->open(url, "r");

    if(h == NULL) {
        printf("Error: failed to open %s!\n", url);
        return -1;
    }

    while(1) {
        struct rts_frame *frm = NULL;

        int c = fs->read(&frm, h);
        if(c == 1) {
            printf("got one frame\n");
            assert(frm != NULL);
            //TODO: 将该帧送给解码器
            frm->free_ptr(frm);
            frm = NULL;
        }
        else if(c == 0) {
            assert(frm == NULL);
            printf("try again\n");
            //sleep a while, 防止过于频繁的 read 占用太多 cpu
            usleep(5 * 1000);
        }
        else {
            assert(frm == NULL);
            printf("Error: unknown\n");
            break;
        }
    }

    //关闭连接
    fs->close(h);

    return 0;
```

```
}
```

4. 接口说明

4.1. 数据结构

```
struct wsrtc_frame {
    void *buf;
    int size;
    int is_audio;
    uint64_t pts; // pts & dts: in ms
    uint64_t dts;
    int flag;// 5 is for key frame;0 is no-key frame
    int duration;
    // use this function to free
    // struct rts_frame object
    void (*free_ptr)(struct wsrtc_frame *);
    unsigned int uid; // belongs to which user
};
```

成员	备注
buf	数据
size	buf 的大小
is_audio	是否为音频帧： 1 为音频帧 0 为视频帧
pts	渲染时间戳，单位 ms
dts	解码时间戳，单位 ms
flag	当 is_audio 为 0 时有效： 0 为非关键帧； 5 为关键帧；
duration	帧持续时间，单位 ms
free_ptr	用来释放 wsrtc_frame 对象
uid	暂未使用

```
struct wsrtc_glue_funcs {
    int api_version;

    /* configure globaly, before open is called
     * do not change values during an instance is running
```

```
     */
    void (* preconfig)(const char *key, const char *val);

    /* open a stream specified by url
     * url:   stream url. wrtc:// stream supported for now
     * mode:  "r" for subscribe, "w" for publish
     * return value: handle to the stream. NULL if open failed
     */
    void *(* open)(const char *url, const char *mode);

    /* close the stream
     * handle: returned by open
     */
    void (* close)(void *handle);

    /* runtime control (e.g. get/set parameters)
     * negative return value for error
     */
    long long (* ioctl)(void *handle, const char *cmd, void *arg);

    /* read one frame
     * caller need free the returned frame
     * return value: 1 for one frame read into '*frame'; 0 for try
     *               later; -1 for EOF; or other negative value for
     *               fatal error
     */
    int (* read)(struct wsrtc_frame **frame, void *handle);

     /* write one frame. callee free the frame
     * return value: 1 for ok; 0 for try
     *               later; -1 for EOF; or other negative value for
     *               fatal error
      */
    int (* write)(struct wsrtc_frame **frame, void *handle);
};
```

成员	备注
api_verison	api 版本号
preconfig	全局预配置，在 open 之前调用
open	打开流
close	关闭流
ioctl	配置参数，获取状态等
read	读取一帧数据
write	发送一帧数据，暂不支持

```
struct wsrts_worker_demux_info
{
    // audio part
    int audio_flag;        // 1 - following audio info is valid; 0 - in
valid
    int audio_channels;    // 1
    int audio_sample_rate; // 48000

    // video part
    int video_flag; // 1 - following video info is valid; 0 - invalid
    int codec;      // 1 - h264  2 - hevc
    int video_width;
    int video_height;
    int video_profile;
    int video_level;

    unsigned char spspps[10 * 1024]; // large enough
    int spspps_len;                  // actual bytes used in spspps
};
```

成员	备注
audio_flag	后续 audio 信息是否有效： 1 为有效； 0 为无效；
audio_channels	音频 channel 数，当 audio_flag 为 1 时有效；
audio_sample_rate	音频采样率，当 audio_flag 为 1 时有效；
video_flag	后续 video 信息是否有效： 1 为有效； 0 为无效；
video_codec	Video 编码类型： 1 为 H264 当 video_flag 为 1 时有效；
video_width	视频分辨率宽度，当 video_flag 为 1 时有效；
video_height	视频分辨率高度，当 video_flag 为 1 时有效；
video_profile	视频编码使用的 profile，当 video_flag 为 1 时有效；
video_level	视频编码使用的 video，当 video_flag 为 1 时有效；
spspps	存放 sps 以及 pps 信息，可以为空，video_flag 为 1 时有效；
spspps_len	spspps 的长度，单位字节，video_flag 为 1 时有效；

4.2. get_wsrtc_funcs

```
struct wsrtc_glue_funcs* JNIEXPORT get_wsrtc_funcs(int version);
```

函数说明：

对外统一调用 api，全局单例

参数说明：

参数	备注
version	api version

返回值:

调用成功返回 wsrtc_glue_funcs 指针；

调用失败返回 NULL；

示例代码:

```
WsrtcDemuxer::WsrtcDemuxer(int dummy) : avFormatDemuxer(dummy)
    {
        static std::once_flag oc;
        std::call_once(oc, [&] {
            int version = 2;
           const struct wsrtc_glue_funcs* rts_funcs = get_wsrtc_funcs(ver
sion);
            // set to ffmpeg plugin
            av_set_wsrtc_demuxer_funcs(rts_funcs);

            wsrtc_set_rts_param((char*)"AutoReconnect", (char*)"false");
        });


    }
```

4.3. preconfig

```
void (* preconfig)(const char *key, const char *val);
```

函数说明：

全局参数配置，只能在 open 函数调用前配置

参数说明：

参数	备注
key	配置名称

val	配置值

配置说明：

配置	备注
AutoReconnect	是否允许 SDK 自动重连，默认为 true
LogCallback	设置外部日志回调函数
LogCbParam	设置 LogCallback 回调的 arg 参数返回值
LogToConsole	日志是否打印到控制台，默认为 true
LogLevel	日志打印等级，默认为 0
MessageCallback	设置事件回调函数
MessageCbParam	设置事件回调的 arg 参数返回值

示例代码:

```
static int format_control_message(struct AVFormatContext *s,
                                  int type,
                                  void *data, // temp data, do not cache
it for later use!
                                  long long data_size)
{
}

static void output_log(void* s, const char* msg, int level) {

}



static char *addr_to_string(const void *v, char buf[64])
{
    sprintf(buf, "%llu", (unsigned long long)v);
    return buf;
}

   __rts_funcs->preconfig("LogCallback", addr_to_string(output_log, buf))
;
    __rts_funcs->preconfig("LogCbParam", addr_to_string(s, buf));
    int log_level = 0;
    __rts_funcs->preconfig("LogLevel", addr_to_string(&log_level, buf));
    __rts_funcs->preconfig("LogToConsole", addr_to_string("true", buf));
    __rts_funcs->preconfig("MessageCallback", addr_to_string(format_contr
ol_message, buf));
    __rts_funcs->preconfig("MessageCbParam", addr_to_string(s, buf));
```

4.4. open

```
void *(* open)(const char *url, const char *mode);
```

函数说明：

打开一条流

参数说明：

参数	备注
url	流名必须以 wrtc://开头
mode	r：播放一个流

返回值:

调用成功返回一个 fd 句柄，可以用于 read，ioctl，close;

调用失败返回 NULL;

示例代码:

```
const struct wsrtc_glue_funcs* rts_funcs = get_wsrtc_funcs(version);
void *rts_worker = rts_funcs->open(s->url, "r");
```

4.5. close

```
void (* close)(void *handle);
```

函数说明：

关闭一条流

参数说明：

参数	备注
handle	open 返回的 fd 句柄

示例代码:

```
const struct wsrtc_glue_funcs* rts_funcs = get_wsrtc_funcs(version);
void *rts_worker = rts_funcs->open(s->url, "r");
rts_funcs->close(rts_worker);
```

4.6. ioctl

```
long long (* ioctl)(void *handle, const char *cmd, void *arg);
```

函数说明：

配置对应流的参数

参数说明：

参数	备注
handle	open 返回的 fd 句柄
cmd	配置名
arg	配置值，无则填 NULL

配置说明：

配置名	配置参数	备注
get_stream_info	wsrts_worker_demux_info 指针	将流信息存储到 wsrts_worker_demux_info 中，返回 0 成功，非 0 失败；
reload	NULL	重新链接，返回 0 成功，非 0 失败；
get_state	int 指针	arg 值为 1 返回值为视频缓存 duration，arg 值为 0 返回值为音频缓存 duration

示例代码:

```
const struct wsrtc_glue_funcs* rts_funcs = get_wsrtc_funcs(version);
void *rts_worker = rts_funcs->open(s->url, "r");

struct wsrts_worker_demux_info stream_info;
int r = rts_funcs->ioctl(rts_worker, "get_stream_info", &stream_info);
rts_funcs->ioctl(rts_worker, "reload", NULL);
int key = 1;
int duration=rts_funcs->ioctl(rts_worker, "get_state", &key);
```

4.7. read

```
int (* read)(struct wsrtc_frame **frame, void *handle);
```

函数说明：

读取流中的一帧数据，保存在 wsrtc_frame 中

参数说明：

参数	备注
handle	open 返回的 fd 句柄
frame	帧数据存储，为 Null 则无数据

返回值：

0 为正常;

-1 为异常;

示例代码:

```
const struct wsrtc_glue_funcs* rts_funcs = get_wsrtc_funcs(version);
void *rts_worker = rts_funcs->open(s->url, "r");

struct wsrts_frame *f = NULL;
int r = rts_funcs->read(&f, rts_worker);
```

4.8. 消息回调

```
typedef void (* wsrtc_event_callback)(
   void *opaque,
   int type,
   void *data,
    long long data_size
);
```

函数说明：

设置消息回调通告的函数

参数说明：

参数	备注
opaue	通过 preconfig("MessageCbParam")设置的值，将原值回传给 wsrtc_event_callback。
type	消息 ID
data	消息描述信息
data_size	消息描述信息大小，单位字节

消息 ID 说明：

消息 ID	备注

10001	本地 SDP 设置成功
20001	SDP 获取失败
20002	连接断开
20003	数据获取超时
40001	流中断恢复

4.9. 日志回调

```
typedef void (*wsrtc_log_cb) (void* s, const char *msg, int log_level);
```

函数说明：

设置日志回调参数

参数说明：

参数	备注
s	通过 preconfig("LogCbParam")设置的值，将原值回传给 wsrtc_log_cb。
msg	日志内容
log_level	日志等级

日志等级说明：

消息 ID	备注
0	VERBOSE
1	INFO
2	WARNING
3	ERROR

4.备注

若有任何意见或建议，请及时联系网宿负责对接的产品工程师。