ALIBABA CLOUD

阿里云

视频直播

播放器SDK

文档版本：20200904

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.RtsSDK编程手册(C接口)

本文介绍了RtsSDK的C API，以及如何编程订阅RTS音视频流

V1.3.0

2020/09/02

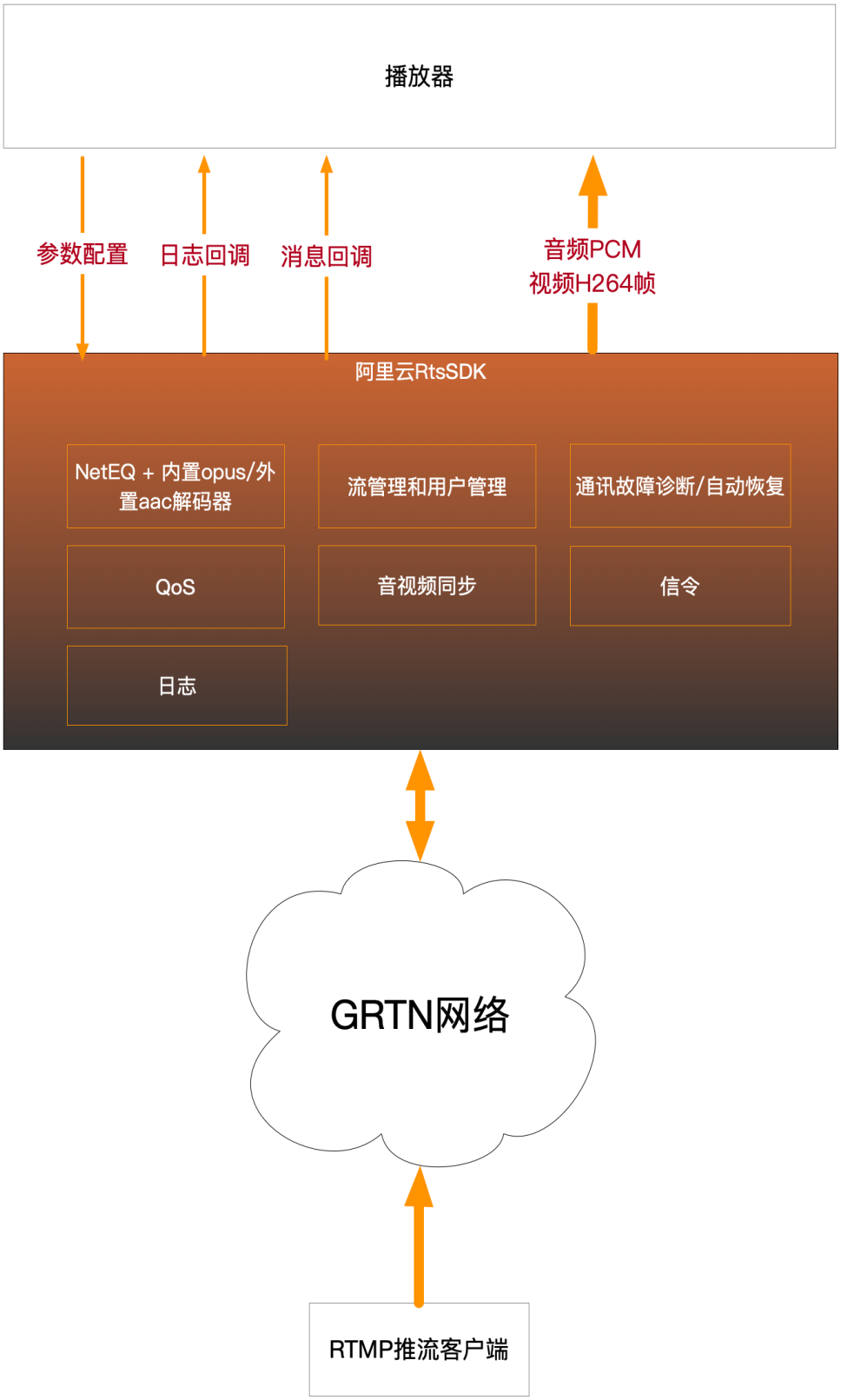
目录

1 介绍

阿里云RtsSDK是介于播放器和阿里云GRTN网络之间的客户端SDK，用来从GRTN网络实时拉取音视频流，处理后将音频pcm和视频h264帧提供给播放器进行解码/渲染。

RtsSDK集成进播放器，有2种方式，一种是使用随包提供的rtsdec.c，该文件将RtsSDK封装成了ffmpeg demuxer插件（直接阅读rtsdec.c源代码，本文档不对该文件做介绍）。另一种方式是直接使用本文描述的API。

2 架构

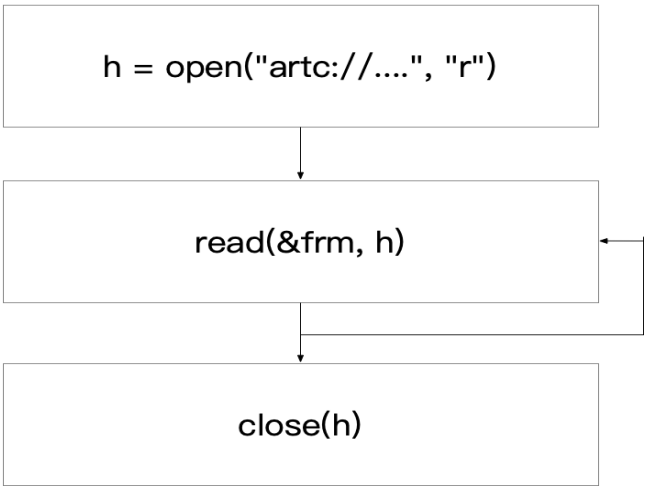


阿里云RtsSDK从阿里云GRTN网络拉音视频，经过demuxer，jitterbuffer，音频解码和视频组帧后送到一个frame queue中，供播放器来读取（pull方式）。播放器得到音频帧，直接送renderer；得到视频帧，送解码器，然后将解码输出的帧送到renderer。视频renderer通过pts和音频renderer同步。

RtsSDK提供参数配置接口，播放器可以对RtsSDK的参数进行调整。其中有2个重要的参数：日志回调和消息回调函数指针（如何设置日志/消息回调函数，参见参数配置）。

3 编程模型

编程是非常直接的：open一个artc的url，然后不断read，最后调用close关闭：



Example (以Mac平台为例):

```
// test_rtssdk.c

#include "rts_api.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 rts_glue_funcs *fs = get_rts_funcs(2); // 2: current version
if(fs == NULL) {
printf("Error: failed to get rts functions. check version.\n");
return -1;
}

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

const char *url = argv[1];

char buf[64];
// 设置support id prefix
fs->preconfig("HelpSupportIDPrefix", "yourInstanceID");
// 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;
}
```

编译：

```
gcc -I<path to header files> -o test_rtssdk test_rtssdk.c -L<path to library file> -lRtsSDK
```

运行：

```
DYLD_LIBRARY_PATH=<path to library file> ./test_rtssdk artc://<domain/app/name>
```

结果：

```
> 2 Start @1599029585691, Net sdk version 0.0.1_networkfd7ced1_LIBd84a462_sophoncb55d0435c_sophontb55d0435c_builde5fbe24_date2020.09.02.14:26-tag_rtssdk_1.3.0_startpoint-17-gfd7c
> 2 OS: Mac
> 2 Command queue thread running
> 2 help support id: <yourInstanceID-pull%2Ertsdemo%2Egrtn%2Ealiyunlive%2Ecom-mac-20200902065305-wxnArRGbLt5gGgJN>
try again
msg: 100 code=100,when=1599029585691,where=,who=0,desc="yourInstanceID-pull%2Ertsdemo%2Egrtn%2Ealiyunlive%2Ecom-mac-20200902065305-wxnArRGbLt5gGgJN"
<此处省略大量的打印>
got one frame
got one frame
got one frame
try again
try again
got one frame
try again
got one frame
got one frame
try again
try again
统计数据消息的内容：
msg: 105 code=105,when=1599029589707,where=,who=0,desc="haveVideo:1,videoCodecType:1,width:720,height:1280,receivedKeyFramePerSec:0.50,receivedVideoKbps:735,videoPktLostPerSec:0,receivedVideoPktPerSec:90,receivedVideoFramePerSec:12,videoNackPerSec:0,haveAudio:1,channels:2,sampleRate:48000,codecType:1,receivedAudioKbps:70,audioPktLostPerSec:0,receivedAudioPktPerSec:31,receivedAudioFramePerSec:31,audioNackPerSec:45,cacheDuration:0"
```

4 API

4.1 数据结构

4.1.1 rts_worker_demux_info

查询媒体信息返回的数据类型。

定义

```
struct rts_worker_demux_info
{
int audio_flag;
int audio_channels;
int audio_sample_rate;

int video_flag;
int video_codec;
int video_width;
int video_height;
int video_profile;
int video_level;

unsigned char spspps[10 * 1024];
int spspps_len;
};
```

成员

成员	解释
audio_flag	指示跟随其后的audio信息是否valid的flag。1表示valid, 0表示invalid
audio_channels	音频channel数。当audio_flag == 1的时候才有意义
audio_sample_rate	音频采样率。当audio_flag == 1的时候才有意义
video_flag	指示跟随其后的video信息是否valid的flag。1表示valid, 0表示invalid
video_codec	视频帧类型。1表示h264。当video_flag == 1的时候才有意义
video_width	视频分辨率宽度。当video_flag == 1的时候才有意义
video_height	视频分辨率高度。当video_flag == 1的时候才有意义
video_profile	视频编码使用的profile。当videoflag == 1，且videocodec == 1的时候才有意义
video_level	视频编码使用的level。当videoflag == 1，且videocodec == 1的时候才有意义
spspps	存放sps和pps数据，可以用于提前初始化解码器。有可能为空。当videoflag == 1，且videocodec == 1的时候才有意义
spspps_len	spspps的长度（字节为单位）。当videoflag == 1，且videocodec == 1的时候才有意义

remarks

无

4.1.2 rts_frame

封装音频和视频帧的数据类型

定义

```
struct rts_frame
{
void buf;
int size;
int is_audio;
unsigned long long pts;
unsigned long long dts;
int flag;
int duration;
void (free_ptr)(struct rts_frame *);
unsigned int uid;
};
```

成员

成员	解释
buf	frame数据buffer
size	buf的字节数
is_audio	1: 音频帧；0: 视频帧
pts	presentation time stamp, in ms
dts	decoding time stamp, in ms
flag	当视频帧的时候(is_audio == 0), bit 0: key frame flag; bit 1: corruption flag
duration	frame duration, in ms
free_ptr	函数指针，用来释放当前的rts_frame对象
uid	reserved

remarks

free_ptr: 参数是需要释放的rts_frame对象，比如：

frm->free_ptr(frm);

4.1.3 rts_glue_funcs

这个数据结构以函数指针的形式保存了api函数。调用者首先通过const struct rts_glue_funcs *get_rts_funcs(int version)来获取rts_glue_funcs变量，然后才能通过rts_glue_funcs变量访问其他api函数

定义

```
struct rts_glue_funcs
{
int api_version;
int (* preconfig)(const char *key, const char *val);
void (* open)(const char *url, const char mode);
void (* close)(void handle);
long long (* ioctl)(void *handle, const char *cmd, void arg);
int (* read)(struct rtsframe **frame, void handle);
int (* write)(struct rtsframe **frame, void *handle);
};
```

成员

成员	解释
api_version	api版本号。必须是2
preconfig	全局参数配置函数。调用顺序：在open前调用，不要在open后调用。详细描述参见function preconfig
open	打开一个流。详细描述参见function open
close	关闭一个流。详细描述参见function close
ioctl	参数配置/查询。详细描述参见function ioctl
read	读取一帧数据。详细描述参见function read
write	发送一帧数据。详细描述参见function write

remarks

无

4.2 函数

4.2.1 get_rts_funcs

这是唯一一个对外可见的api。通过这个api获得其他api的指针。

原型

```
const struct rts_glue_funcs *get_rts_funcs(
int version
);
```

参数

参数	解释
version	api兼容版本。必须是2

Return value

如果成功，返回rts_glue_funcs指针。如果失败（一般是version参数不匹配），返回NULL

Remarks

获取rts_glue_funcs变量。通过该变量得到其他api

4.2.2 function preconfig

设置参数，在open之前，影响全局

原型

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

参数

参数	解释
key	参数名称，大小写敏感。字符串
val	参数值。字符串

Return value

0 for OK, otherwise FAIL

Remarks

支持的参数列表

参数

Key	解释
AutoReconnect	是否允许RtsSDK发现网络断开后自动重联。缺省值：允许
BufferingDuration	Jitter buffer缓存时间。缺省值：500 ms
LogCallback	设置外部的日志的回调函数。需要同时设置LogCbParam才能生效
LogCbParam	设置外部的日志的回调函数的参数。需要同时设置LogCallback才能生效
LogToConsole	设置内置的日志输出到控制台。缺省值：不输出
LogToFile	设置内置的日志输出到文件。缺省值：不输出

Key	解释
LogToServer	设置内置的日志输出到服务器。缺省值：不输出
LogLevel	设置内置的日志输出level。缺省值：LOG_INFO=2
MessageCallback	设置消息回调函数。需要同时设置MessageCbParam才能生效
MessageCbParam	设置消息回调函数的参数。需要同时设置MessageCallback才能生效
AacdCreateCallback	设置外部aac解码器创建回调
AacdDecodeCallback	设置外部aac解码器解码回调
AacdCloseCallback	设置外部aac解码器关闭回调
HelpSupportIDPrefix	设置help support id的前缀

4.2.3 function open

打开一个流

原型

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

参数

参数	解释
url	流的url。现在支持artc://格式的url
mode	如果是播放一个流，设置为"r"；推送一个流，设置为"w"

Return value

如果执行成功，返回一个句柄，后续的ioctl, read, write, close针对这个句柄操作。如果执行失败，返回NULL

Remarks

无

4.2.4 function close

关闭一个流

原型

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

参数

参数	解释
handle	open返回的句柄

Return value

void

Remarks

无

4.2.5 function ioctl

访问rtssdk参数

原型

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

参数

参数	解释
handle	open返回的句柄
cmd	发送的命令名称
arg	发送的命令参数。有些命令不带参数，填NULL

Return value

0 for OK and otherwise FAIL

Remarks

实现的2个参数访问：

cmd	arg	解释
"get_stream_info"	rts_worker_demux_info变量地址，不能为NULL	从RtsSDK获得stream信息，存储到arg指向的rts_worker_demux_info变量中去

cmd	arg	解释
"reload"	NULL	提示RtsSDK重新连接

4.2.6 function read

从RtsSDK读取一帧数据

原型

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

参数

参数	解释
frame	where to store returned audio/video frame
handle	open返回的句柄

Return value

1 for one frame read into '*frame'; 0 for try later; -1 for EOF; or other negative value for fatal error

Remarks

返回的frame需要调用者释放。释放的示例代码：

```
struct rts_frame *f = NULL;
int r = __rts_funcs->read(&f, handle);
...
if(f != NULL)
f->free_ptr(f);
```

4.2.7 function write

这个api是推流用的。播放器可以跳过这个api

原型

```
int (* write)(
struct rts_frame **frame,
void *handle
);
```

参数

参数	解释
frame	audio/video frame to send
handle	open返回的句柄

Return value

1 for ok; 0 for try later; -1 for EOF; or other negative value for fatal error

Remarks

如果返回成功，那么frame的ownership转移至RtsSDK，由RtsSDK销毁。否则ownership仍然在调用者这里

5. 参数设置

RtsSDK提供了2种参数设置途径：

- 全局参数设置
- 实例参数设置

二者不同之处是，全局参数设置preconfig需要在open之前调用。而实例参数设置ioctl操作的对象是open返回的句柄

注：ioctl除了设置参数外，还可以获取参数，执行一个命令

全局参数设置示例

设置消息回调函数和日志回调函数的例子(参考rtsdec.c)：

```
__rts_funcs->preconfig("LogCallback", addr_to_string(output_log, buf));
__rts_funcs->preconfig("LogCbParam", addr_to_string(s, buf));
__rts_funcs->preconfig("MessageCallback", addr_to_string(format_control_message, buf));
__rts_funcs->preconfig("MessageCbParam", addr_to_string(s, buf));
```

实例参数设置

等待补充

6. 消息回调

原型

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

参数

参数	解释
opaque	这个参数是通过preconfig("MessageCbParam")设置的值，rtssdk原封不动回传给event_callbac
type	消息ID
data	消息内容，格式是：when=<time_in_ms>,where=<place>,who=<id>,desc=<extra text>
data_size	data长度，in bytes

Return value

未定义

Remarks

消息ID

消息ID	说明
100	Help support ID消息。发生在open, 在connect之前
102	Help support ID消息。发生在connect操作
103	Help support ID消息。发生在推流操作
104	Help support ID消息。发生在拉流操作
104	104
105	统计数据消息。每4秒一次
20000 ~ 30000	错误码
20001	DNS出错，无法解析域名
20002	域名访问权限验证失败
20009	成功连接到流媒体服务器
20010	连接流媒体服务器失败
20012	订阅流超时
20013	订阅流不存在
20014	订阅流没有音频
20015	订阅流没有视频
20020	其他未知订阅错误
20050	丢包率高，拥塞严重

消息ID	说明
20051	丢包率降到正常水平，拥塞恢复
20052	流中断，没有任何音频或者视频包
20053	流中断恢复
20054	流结束
20055	连接丢失
20056	检测到流重新推送
20057	无法播放，需要降级到RTMP

当消息ID是105 (统计数据消息)时，desc的内容是如下格式：

\"key1:val1,key2:val2,...\"

data临时变量。出了回调函数，该变量会销毁

Example

消息回调函数通过[preconfig](#rts_glue_funcs.preconfig_detail)设置给RtsSDK。下面这个例子摘自rtsdec.c:

```
static int on_message(void *s,
int type,
void *data,
long long data_size)
{
(void) s;

// TODO: process message, do not take too long
switch(type) {
case 105:
printf("Profiling message %s.\n", (const char *)data);
break;
default:
break;
}
return 0;
}

__rts_funcs->preconfig("MessageCallback", addr_to_string(on_message, buf));
__rts_funcs->preconfig("MessageCbParam", addr_to_string(s, buf));
```

7. 日志回调

RtsSDK支持内置日志系统和外部日志系统。无论是内置还是外置，都通过preconfig来配置日志参数。外置日志系统需要调用者提供一个日志回调函数

原型

```
int (* output_log)(
void *s,
int level,
const char *fmt,
va_list args
);
```

参数

参数	解释
s	这个参数是通过preconfig("LogCbParam")设置的值，rtssdk原封不动回传给output_log
level	本条日志的等级
fmt	格式化字符串，和printf中的format参数一致
args	可变参数列表

Return value

未定义

Remarks

日志的等级定义：

Level	等级
0	Error
1	Warning
2	Info
3	Debug

Example

外置日志回调函数通过preconfig设置给RtsSDK。下面这个例子摘自rtsdec.c:

```
// 提供一个回调函数
static int output_log(struct AVFormatContext *s, int level, const char *fmt, va_list args)
{
// TODO: 处理log
return 0;
}

// 注册这个回调函数给rtssdk
__rts_funcs->preconfig("LogCallback", addr_to_string(output_log, buf));
__rts_funcs->preconfig("LogCbParam", addr_to_string(s, buf));
```

内置log使用方法

```
__rts_funcs->preconfig("LogToConsole", "true"); // output to console
__rts_funcs->preconfig("LogToFile", "true"); // output to file
__rts_funcs->preconfig("LogToServer", "true"); // output to server
__rts_funcs->preconfig("LogLevel", "2"); // LOG_INFO
```

2.播放器SDK

直播播放器已和点播播放器合并，您可以直接参考点播提供的播放器。

详情参考 播放器SDK产品介绍及使用说明。

> **说明** 已经使用阿里云直播播放器的用户可以使用新版提供的基础播放器进行升级（接口兼容）。