视频边缘智能服务

设备端开发指南

设备端开发指南

概述

概述

LinkVisual设备端SDK提供了网络摄像机（IPC）接入阿里云视频边缘智能服务的基础API，主要包括实时视频直播相关的接口、设备本地文件点播相关的接口、报警事件上报和图片上传的接口以及其他IPC控制相关的接口等，为物联网设备提供的视频直播、视频点播、图片上传的功能。

资源占用情况如下：

- 内存占用：1MB的码流的情况下，预估占用500K左右的内存
- ROM占用：2MB

平台支持：目前支持Linux平台，且需要C++11的支持

依赖：LinkVisual SDK依赖了物联网套件Link Kit C版本SDK。Link Kit C SDK提供了物联网通道能力，LinkVisual响应Link Kit C SDK的控制消息来进行流媒体业务处理。开发者需要同时获取LinkVisual SDK和物联网套件Link Kit C SDK来完成设备的接入。

- Link Kit C SDK主要是提供物联网控制通道能力，包括长连接、消息通知、事件上报等。
- LinkVisual SDK主要是提供音视频流通道能力；

Link Kit SDK

Link Kit SDK对接

设备接入视频边缘智能服务LinkVisual的一个前提是设备首先需要接入Link Kit SDK，成为一个物联网设备。Link Kit提供了物联网设备接入阿里云IoT云的基础能力，包括长连接消息通道、配网、OTA等能力。

Link Kit C SDK对接文档，**点击这里。**

开发者在设备端上集成Link Kit SDK，完成所需要的适配工作，更多参考给出链接的文档说明，这里不作赘述。简单说明下对接Link Kit所需工作如下：

必需：

- SDK下载、编译
- HAL层的适配
- 完成Link Kit C SDK中example/linkkit/linkkit_example_solo.c的示例

可选：

- WIFI配网开发
- OTA开发
- 其他

在Link Kit对接完成后，开发者需要把Link Kit接收到的消息通过调用Link Visual SDK的接口给到Link Visual SDK内部处理。下面给一个示例代码片段，

- Link Kit注册服务调用回调

```
IOT_RegisterCallback(ITE_SERVICE_REQUST, user_service_request_event_handler);
```

在Link Kit服务调用回调里，把消息通知给Link Visual

```
static int user_service_request_event_handler(const int devid, const char *id, const int id_len,
const char *serviceid, const int serviceid_len,
const char *request, const int request_len,
char **response, int *response_len)
{
EXAMPLE_TRACE("Service Request Received, Devid: %d, ID %.*s, Service ID: %.*s, Payload: %s",
devid, id_len, id, serviceid_len, serviceid, request);
//将服务类消息交由LinkVisual来处理，不需要处理response
lv_linkkit_adapter_tsl_service_s in = {0};
in.dev_id = devid;
in.id = id;
in.id_len = id_len;
in.service_id = serviceid;
in.service_id_len = serviceid_len;
in.request = request;
in.request_len = request_len;
int ret = linkkit_adapter_tsl_service(&in);
if (ret < 0) {
return -1;
}

return 0;
}
```

LinkVisual SDK

SDK获取

LinkVisual SDK获取

LinkVisual设备端SDK以静态库的形式提供，我们可以编译SDK支持不同芯片平台的接入，请发送邮件到ray.wl@antfin.com进行SDK申请，并确保在邮件内容中，请包含如下信息：

- 1.公司名称；
- 2.联系人及联系方式；
- 3.公司地址；
- 4.应用场景描述。

我们的商务人员在收到邮件后，会主动与您联系获取芯片平台的交叉编译工具链。我们将编译好对应平台的SDK提供给您。

使用指南

LinkVisual SDK使用说明

初始化及销毁

- 调用lv_get_version打印版本信息
- 完成lv_init所需要的参数，并且运行成功
- lv_init成功后，在需要退出的时候调用lv_destroy进行销毁

下行消息通知

Link Kit SDK初始化后，将Link Kit回调的物模型服务类消息通过调用LinkVisual的linkkit_adapter_tsl_service给LinkVisual进行处理。

视频直播功能

在LinkVisual SDK初始化时，注册以下三个回调，

- lv_start_push_streaming_cb；
- lv_stop_push_streaming_cb；
- lv_on_push_streaming_cmd_cb。

当LinkVisual SDK收到推流通知时，

- LinkVisual SDK会调用lv_start_push_streaming_cb回调；
- 设备调用lv_stream_send_config发送音视频参数；
- 设备调用lv_stream_send_video发送视频数据；
- 设备调用lv_stream_send_audio发送音频数据。

当LinkVisual SDK收到停止推流通知时，

- LinkVisual SDK会调用lv_stop_push_streaming_cb回调；
- 设备停止调用lv_stream_send_video；
- 设备停止调用lv_stream_send_audio；
- 设备自行处理音视频采集的资源释放。

直播时，

- LinkVisual SDK会通过lv_on_push_streaming_cmd_cb回调通知设备强制I帧；
- 设备调用lv_stream_send_config发送音视频参数；
- 设备调用lv_stream_send_video发送一个I帧。

视频点播功能

在LinkVisual SDK初始化时，注册以下四个回调

- lv_start_push_streaming_cb
- lv_stop_push_streaming_cb
- lv_on_push_streaming_cmd_cb
- lv_query_storage_record_cb

当LinkVisual SDK收到查询设备本地录像列表通知时，

- LinkVisual SDK会调用lv_query_storage_record_cb回调；
- 设备应完成录像列表查询并回报。

当LinkVisual SDK收到点播指定的文件的推流通知时，

- LinkVisual SDK会调用lv_start_push_streaming_cb回调；
- 设备调用lv_stream_send_config发送音视频参数；
- 设备调用lv_stream_send_video发送视频数据；

- 设备调用lv_stream_send_audio发送音频数据。

当LinkVisual SDK收到结束点播推流通知时，

- LinkVisual SDK会调用lv_stop_push_streaming_cb回调；
- 设备停止调用lv_stream_send_video；
- 设备停止调用lv_stream_send_audio；
- 设备自行处理音视频采集的资源释放。

点播过程中LinkVisual SDK收到暂停、恢复播放、seek等通知时，

- LinkVisual SDK会调用lv_on_push_streaming_cmd_cb通知设备。

图片服务

在LinkVisual SDK初始化时，注册trigger_pic_capture_cb这个回调。

当LinkVisual SDK收到抓图通知时，

- LinkVisual SDK会调用trigger_pic_capture_cb回调；
- 设备调用lv_post_alarm_image上报抓图。

设备主动上报移动报警、声音报警等事件时，

- 调用lv_post_alarm_image上报对应图片。

附录：LinkVisual SDK API定义

- LinkVisual SDK初始化

```
/**
@brief SDK初始化
@param [IN] config: SDK配置参数集合
@return lv_error_e
*/
int lv_init(const lv_config_s *config);
```

其中，结构体lv_config_s用于定义包括三元组等基础信息。

```
/* 配置参数结构体 */
typedef struct lv_config_s {
/* 三元组信息 */
char product_key[PRODUCT_KEY_LEN + 1];
char device_name[DEVICE_NAME_LEN + 1];
char device_secret[DEVICE_SECRET_LEN + 1];
```

```
/* SDK的日志配置 */
lv_log_level_e log_level;
lv_log_destination log_dest;
char log_dest_file[LOG_DEST_FILE_LEN + 1];

/* 音视频推流服务 */
lv_start_push_streaming_cb start_push_streaming_cb;
lv_stop_push_streaming_cb stop_push_streaming_cb;
lv_on_push_streaming_cmd_cb on_push_streaming_cmd_cb;

/* 语音对讲服务 */
lv_start_voice_intercom_cb start_voice_intercom_cb;
lv_stop_voice_intercom_cb stop_voice_intercom_cb;
lv_voice_intercom_receive_metadata_cb voice_intercom_receive_metadata_cb;
lv_voice_intercom_receive_data_cb voice_intercom_receive_data_cb;

/* 存储录像查询命令 */
lv_query_storage_record_cb query_storage_record_cb;

/* 触发设备抓图 */
lv_trigger_pic_capture_cb trigger_pic_capture_cb;

/* 触发设备录像 */
lv_trigger_record_cb trigger_record_cb;

} lv_config_s;
```

- LinkVisual SDK销毁

```
/**
* @brief SDK销毁
*
* @param [IN] void
*
* @return lv_error_e
*/
int lv_destroy(void);
```

- LinkVisual SDK版本信息

```
/**
* @brief 获取SDK版本信息
*
*/
void lv_get_version(void);
```

- LinkKit下行消息处理

```
/**
* @brief linkkit适配器，将物模型中的服务类消息注入该函数中，由LinkVisual代为处理
```

```
*
* @param [IN] lv_linkkit_adapter_tsl_service_s: 服务类消息结构体入参
*
* @return lv_error_e
*/
int linkkit_adapter_tsl_service(const lv_linkkit_adapter_tsl_service_s *in);
```

- 通知开始推流

```
/**
* @brief 通知设备开始推流
*
* @param [IN] service_id: 推流服务ID
* @param [IN] type: 推流服务的类型
* @param [IN] param: 可变参数，根据type会有不同
* type == LV_STREAM_CMD_LIVE,param->pre_time为预录时间，单位：s
* type == LV_STREAM_CMD_STORAGE_RECORD_BY_FILE,param->file_name为请求播放的录像文件名
*
* @retval < 0 : Fail.LV_STREAM_CMD_STORAGE_RECORD_BY_FILE
* @retval 0 : Success.
*
* @notice: type == LV_STREAM_CMD_LIVE时，需要调用lv_stream_start_service()来开启服务，
* 并调用lv_stream_send_video()/lv_stream_send_audio()直接开始推流；
* type == LV_STREAM_CMD_STOARGE_RECORD_BY_FILE或type ==
LV_STREAM_CMD_STORAGE_RECORD_BY_TIME时，
* 需要调用lv_stream_start_service()来开启服务，
* 并等待lv_on_push_streaming_cmd_cb()中的消息通知，当lv_on_push_streaming_cmd_cb()中通知
LV_STORAGE_RECORD_START时，
* 调用lv_stream_send_video()/lv_stream_send_audio()开始推流
* @see lv_stream_start_service() lv_on_push_streaming_cmd_cb()
*/
typedef int (*lv_start_push_streaming_cb)(int service_id, lv_stream_type_e type, const lv_stream_param_s *param);
```

- 通知停止推流

```
/**
* @brief 通知设备停止推流
*
* @param [IN] service_id: 推流服务ID
*
* @retval < 0 : Fail.
* @retval 0 : Success.
*/
typedef int (*lv_stop_push_streaming_cb)(int service_id);
```

- 推流过程中控制消息

```
/**
* @brief 推送直播/存储录像流的过程中，需要支持的命令
*
* @param [IN] service_id: 推流服务ID
```

```
* @param [IN] cmd: 命令类型
* @param [IN] timestamp: 时间戳参数，单位为秒
* cmd == LV_STORAGE_RECORD_SEEK时可用，其他命令无意义
*
* @retval < 0 : Fail.
* @retval 0 : Success.
*/
typedef int (*lv_on_push_streaming_cmd_cb)(int service_id, lv_on_push_streaming_cmd_type_e cmd, double
timestamp);
```

- 推送音视频参数信息

```
/**
* @brief 在发送实际视音频数据前发送视音频相关配置,用于直播推流和存储录像播放
*
* @param [IN] service_id: 服务ID，来自回调 lv_start_push_streaming_cb
* @param [IN] bitrate_kbps: 目标码率，单位为kbps，0~8000
* @param [IN] duration: 回放的文件长度，直播置为0
* @param [IN] video_param: 视频的相关参数配置
* @param [IN] audio_param: 音频的相关参数配置
*
* @return lv_error_e
* @see lv_start_push_streaming_cb()
*/
int lv_stream_send_config(int service_id,
unsigned int bitrate_kbps,
double duration,
const lv_video_param_s *video_param,
const lv_audio_param_s *audio_param);
```

- 推送视频数据

```
/**
* @brief 发送视频数据
*
* @param [IN] service_id: 服务ID
* @param [IN] buffer: 视频数据指针
* @param [IN] buffer_len: 视频数据长度
* @param [IN] key_frame: 视频帧是否为关键帧
* @param [IN] timestamp_ms: 视频帧时间戳，单位：ms
*
* @return lv_error_e
*/
int lv_stream_send_video(int service_id,
unsigned char* buffer,
unsigned int buffer_len,
bool key_frame,
unsigned int timestamp_ms);
```

- 推送音频数据

```
/**
* @brief 发送音频数据
*
* @param [IN] service_id: 服务ID
* @param [IN] buffer: 音频数据指针
* @param [IN] buffer_len: 音频数据长度
* @param [IN] timestamp_ms: 音频帧时间戳，单位：ms
*
* @return lv_error_e
*/
int lv_stream_send_audio(int service_id,
unsigned char* buffer,
unsigned int buffer_len,
unsigned int timestamp_ms);
```

- 录像列表查询

```
/**
* @brief 查询存储录像列表
*
* @param [IN] start_time : 查询的开始时间，UTC时间，秒数
* @param [IN] stop_time : 查询的结束时间，UTC时间，秒数
* @param [IN] num : 录像查询的数量
* @param [IN] id : 该次查询的会话标识符，需要回传到on_complete
* @param [IN] on_complete : 返回录像列表的函数，目前仅支持同步调用，否则会超时；
* num返回实际所查询到的数量，id回传，lv_query_storage_record_response_s里放置录像列表信息。
*
* @return void
*/
typedef void (*lv_query_storage_record_cb)(unsigned int start_time,
unsigned int stop_time,
int num,
const char *id,
int (*on_complete)(int num, const char *id,
const lv_query_storage_record_response_s *response));
```

- 主动上传报警图片

```
/**
* @brief 报警事件图片上传
*
* @param [IN] buffer: 事件图片数据指针
* @param [IN] buffer_len: 事件图片数据长度
* @param [IN] type: 事件类型，lv_event_type_e
* @param [IN] id: 当type=LV_TRIGGER_PIC时，需要回传lv_trigger_pic_capture_cb中的id，其他为空
*
* @return lv_error_e
* @see lv_trigger_pic_capture_cb()
*/
int lv_post_alarm_image(const char* buffer,
int buffer_len,
lv_event_type_e type,
const char *id);
```

- 触发设备抓图

```
/**
* @brief 触发设备抓图
*
* @param [IN] id: 触发ID，需回传
*
* @retval < 0 : Fail.
* @retval 0 : Success.
* @see lv_post_alarm_image()
*/
typedef int (*lv_trigger_pic_capture_cb)(const char *id);
```

SDK Demo

准备开发环境

我们提供Ubuntu16.04系统上编译的x86 64位的LinkVisual SDK和Demo供快速体验, 在其它Linux版本上尚未验证过, 推荐安装与阿里一致的发行版以避免碰到兼容性方面的问题。使用下面命令安装一些必备的软件：

```
$ sudo apt-get install -y build-essential make git gcc g++ cmake
```

下载

- 提供Ubuntu16.04下编译的设备端SDK和Demo供参考使用。单击下载LinkVisual SDK Demo。下载本LinkVisual SDK Demo将默认您同意本软件许可协议。

编译运行

- 内容确认

请将下载的后文件解压得到link_visual_ipc_v1.0.0文件夹，确认文件夹下的文件目录结构如下：

```
---- CMakeLists.txt
|
---- samples
|
---- sdk
```

- 编译

切换到link_visual_ipc_v1.0.0文件夹下，执行如下命令。

```
1. mkdir build
2. cd build
3. cmake ..
4. make
5. make install
```

- 创建产品和设备

请根据物联网平台**接入指引**来创建产品，并获取设备的三元组。

- 运行

在link_visual_ipc_v1.0.0文件夹下的build目录下，生成了对应的可执行程序。进入可执行程序目录下，运行如下的命令，其中product_name、device_name、device_secret需要替换为自己的设备三元组。

```
./link_visual_demo  -p product_name -n device_name -s device_secret，
```