ONVIF项目操作学习文档

[一、 需要准备的软件和文档 2](#_Toc59105013)

[二、 gsoap  环境的搭建 3](#_Toc59105014)

[三、 onvif.h文件生成 4](#_Toc59105015)

[四、 框架代码生成 5](#_Toc59105016)

[五、 设备搜索 7](#_Toc59105017)

[六、 设备能力获取 11](#_Toc59105018)

[七、 Services获取多个媒体地址 13](#_Toc59105019)

[八、 profile获取媒体信息文件 14](#_Toc59105020)

[九、 获取RTSP地址 16](#_Toc59105021)

[十、 libonvif库封装 20](#_Toc59105022)

# 需要准备的软件和文档

**VMware® Workstation 15 Pro**

**ubuntu-18.04.4-desktop-amd64.iso**

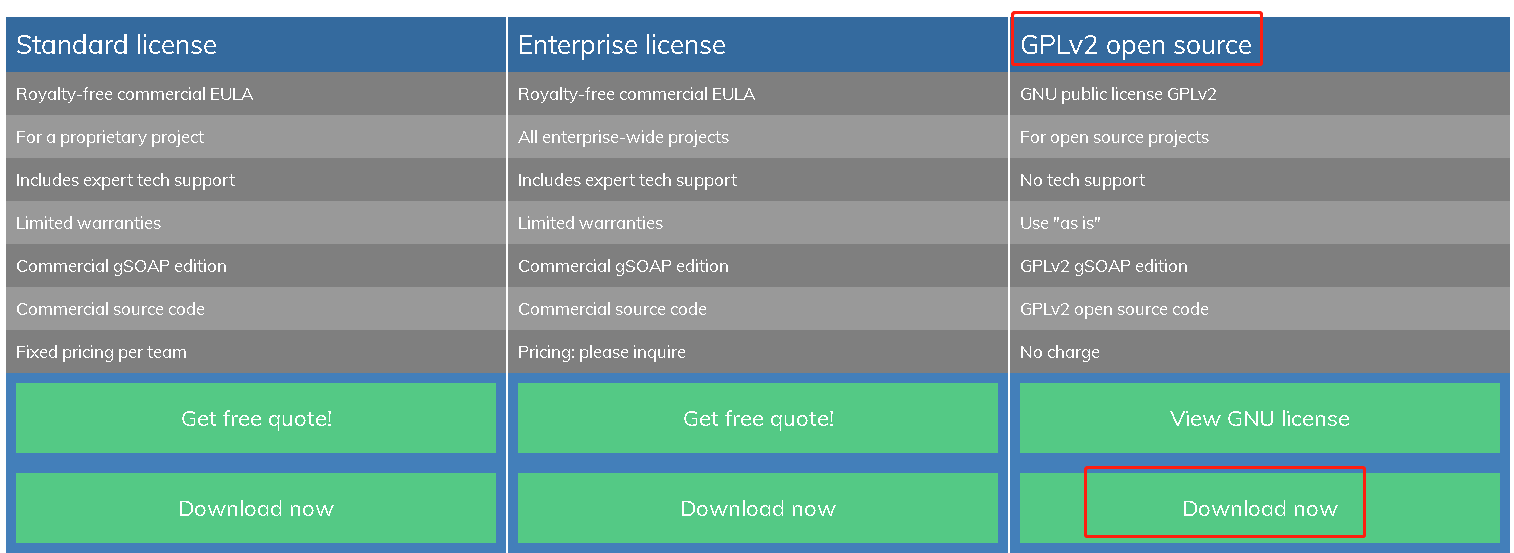
**openssl**

**gsoap\_2.8.109**

# [gsoap  环境的搭建](https://www.cnblogs.com/croxd/p/10684693.html)

* **下载gsoap**

网址：<http://www.genivia.com/products.html#notice>，选择Open Source gSOAP版本，目前最新版本为**gsoap\_2.8.109**



* **依赖工具安装**

阅读gSoap解压的INSTALL.txt文件中，关于《Installing gSOAP on Unix/Linux》的说明，防止安装出现问题，先检查下依赖是否安装，依赖的工具如下

- Automake tools installed (make and GNU m4)

- Bison installed from www.gnu.org/software/bison or Yacc

- Flex installed from flex.sourceforge.net

- OpenSSL from www.openssl.org or GNUTLS from www.gnu.org/software/gnutls

- Optionally Zlib to support compression from www.zlib.net.

安装依赖工具命令如下：

apt-get install bison

apt-get install flex

apt-get install openssl

apt-get install zlib1g-dev

* **编译**

解压进入目录，configure后面的选项可以根据自己的需求添加，--prefix后面填写的gsoa安装所在的路径，我安装的是当前的路径，**如果prefix出现安装错误，则需要把路径改成绝对路径。**

unzip gsoap\_2.8.109.zip

cd gsoap\_2.8.109/ gsoap-2.8/

./configure --enable-ipv6 --enable-debug --prefix=./

make

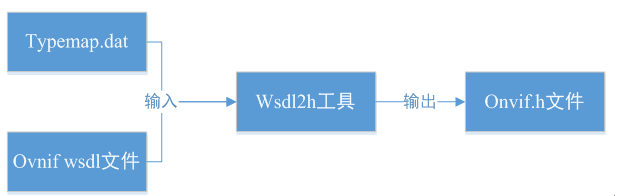
make install

安装完成后，通过以下命令检查安装是否成功：



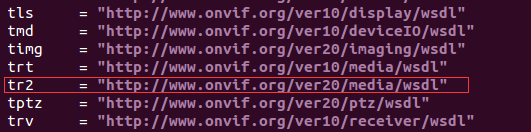
gSoap源码编译后，生成wsdl2h和soapcpp2工具，查看这两个工具的版本，版本应该与gSoap源码版本保持一致，均为2.8.109。

# [onvif.h文件生成](https://www.cnblogs.com/croxd/p/10684693.html)



* **修改**typemap.dat

在当前目录新建example文件夹，目前我的路径为gsoap\_2.8.109/gsoap-2.8/example，将gsoap-2.8/gsoap/typemap.dat拷贝到example目录下，并修改该文件的内容，在命名空间已经存在的trt下，**新增支持h265的代码行 tr2     =**[**http://www.onvif.org/ver20/media/wsdl**](http://www.onvif.org/ver20/media/wsdl)**，tr2可以自己随便命名，以后关于h265接口的都会以tr2名字出现。**

​

* 生成onvif.h：

../bin/wsdl2h -o onvif.h -c -s -t typemap.dat http://www.onvif.org/onvif/ver10/network/wsdl/remotediscovery.wsdl http://www.onvif.org/onvif/ver10/device/wsdl/devicemgmt.wsdl http://www.onvif.org/onvif/ver10/media/wsdl/media.wsdl https://www.onvif.org/ver20/media/wsdl/media.wsdl http://www.onvif.org/onvif/ver20/ptz/wsdl/ptz.wsdl

http://www.onvif.org/onvif/ver10/receiver.wsdl http://www.onvif.org/onvif/ver10/events/wsdl/event.wsdl http://www.onvif.org/onvif/ver10/display.wsdl http://www.onvif.org/onvif/ver10/deviceio.wsdl http://www.onvif.org/onvif/ver20/imaging/wsdl/imaging.wsdl http://www.onvif.org/onvif/ver10/search.wsdl http://www.onvif.org/onvif/ver10/replay.wsdl http://www.onvif.org/onvif/ver20/analytics/wsdl/analytics.wsdl http://www.onvif.org/onvif/ver10/analyticsdevice.wsdl http://www.onvif.org/onvif/ver10/schema/onvif.xsd

**其中 https://www.onvif.org/ver20/media/wsdl/media.wsdl：用于获取h265视频流地**

# [框架代码生成](https://www.cnblogs.com/croxd/p/10684693.html)



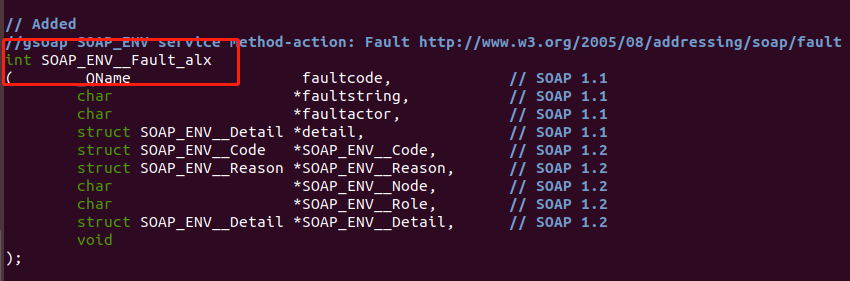
* 加入鉴权功能

在onvif.h的第100行加上 #import "wsse.h"



* 结构体修改

复制gsoap-2.8\gsoap\import文件夹至example下面，并修改wsa5.h文件。**将SOAP\_ENV\_\_Fault结构体注释掉或改为其他名字，如SOAP\_ENV\_\_Fault\_alex。**



**原因：onvif.h包含的wsdd10.h 和 wsa5.h两个头文件，而头文件wsdd10.h又包含wsa.h这个头文件，头文件wsa.h和wsa5.h中对SOAP\_ENV\_\_Fault重复定义了。**

* 生成框架代码

../bin/soapcpp2 -2 -C -x -I ./import ./custom onvif.h

* 拷贝代码

cd /home/cftc/onvif/gsoap\_2.8.109/gsoap-2.8 #切换到gsoap目录下

cp plugin/wsaapi.h plugin/wsaapi.c ../example/ #拷贝鉴权相关代码

cp plugin/wsseapi.c plugin/wsseapi.h ../example/

cp plugin/smdevp.c plugin/smdevp.h ../example/

cp plugin/mecevp.c plugin/mecevp.h ../example/

cp plugin/threads.c plugin/threads.h ../example/

cp custom/struct\_timeval.c ../example/

cp ./dom.c ../example/

cp stdsoap2.h stdsoap2.c ../example/ #拷贝soap 1.2 源码

* 修改stdsoap2.c文件

注释stdsoap2.c如下代码：不注释的话会在编译运行的时候产生log日志，最后会发现磁盘已满的现象。

/\*

#ifdef SOAP\_DEBUG

#ifdef TANDEM\_NONSTOP

soap\_set\_test\_logfile(soap, "TESTLOG");

soap\_set\_sent\_logfile(soap, "SENTLOG");

soap\_set\_recv\_logfile(soap, "RECVLOG");

#else

soap\_set\_test\_logfile(soap, "TEST.log");

soap\_set\_sent\_logfile(soap, "SENT.log");

soap\_set\_recv\_logfile(soap, "RECV.log");

#endif

#endif

\*/

修改stdsoap2.c文件如下

if (/\*s == r || \*r || \*/n < -128 || n > 127)

**（onvif对接海康设备出现soap->error=4的问题**）

# 设备搜索

Onvif协议要求实现Onvif协议服务的设备支持设备发现以及探测，也就是discovery和probe

discovery，客户端向网段内的组播地址为239.255.255.250，端口为3702发送广播消息，等待网内Onvif协议设备响应，响应消息的设备会返回自己的IP、UUID、EP Address、Type等，最重要的会返回设备的Device Service Address，格式如:http://192.168.1.2:8080/onvif/devices，此地址就是为设备提供Onvif服务的服务地址。

**具体代码如下：**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "soapH.h"

#include "stdsoap2.h"

#include "soapStub.h"

#include "wsdd.nsmap" //命名空间

static struct soap\* ONVIF\_Initsoap(struct SOAP\_ENV\_\_Header \*header, const char \*was\_To, const char \*was\_Action, int timeout)

{

struct soap \*soap = NULL; // soap环境变量

unsigned char macaddr[6];

char \_HwId[1024];

unsigned int Flagrand;

soap = soap\_new();

if(soap == NULL)

{

printf("[%d]soap = NULL\n", \_\_LINE\_\_);

return NULL;

}

soap\_set\_namespaces(soap, namespaces); // 设置soap的namespaces，即设置命名空间

// 设置超时（超过指定时间没有数据就退出）

if(timeout > 0)

{

soap->recv\_timeout = timeout;

soap->send\_timeout = timeout;

soap->connect\_timeout = timeout;

}

else

{

//Maximum waittime : 20s

soap->recv\_timeout = 20;

soap->send\_timeout = 20;

soap->connect\_timeout = 20;

}

soap\_default\_SOAP\_ENV\_\_Header(soap, header);

//Create SessionID randomly,生成uuid(windows下叫guid，linux下叫uuid)，格式为urn:uuid:8-4-4-4-12，由系统随机产生

srand((int)time(0));

Flagrand = rand()%9000 + 8888;

macaddr[0] = 0x1;

macaddr[1] = 0x2;

macaddr[2] = 0x3;

macaddr[3] = 0x4;

macaddr[4] = 0x5;

macaddr[5] = 0x6;

sprintf(\_HwId, "urn:uuid:%ud68a-1dd2-11b2-a105-%02X%02X%02X%02X%02X%02X", Flagrand, macaddr[0], macaddr[1], macaddr[2],macaddr[3],macaddr[4],macaddr[5]);

header->wsa\_\_MessageID = (char \*)malloc(100);

memset(header->wsa\_\_MessageID, 0, 100);

strncpy(header->wsa\_\_MessageID, \_HwId, strlen(\_HwId)); //wsa\_\_MessageID存放的是uuid

if(was\_Action != NULL)

{

header->wsa\_\_Action = (char\*)malloc(1024);

memset(header->wsa\_\_Action, '\0', 1024);

strncpy(header->wsa\_\_Action, was\_Action, 1024); //

}

if(was\_To != NULL)

{

header->wsa\_\_To = (char \*)malloc(1024);

memset(header->wsa\_\_To, '\0', 1024);

strncpy(header->wsa\_\_To, was\_To, 1024);//"urn:schemas-xmlsoap-org:ws:2005:04:discovery";

}

soap->header = header;

return soap;

}

//释放函数

void ONVIF\_soap\_delete(struct soap \*soap)

{

soap\_destroy(soap); // remove deserialized class instances (C++ only)

soap\_end(soap); // Clean up deserialized data (except class instances) and temporary data

soap\_free(soap); // Reset and deallocate the context created with soap\_new or soap\_copy

}

int ONVIF\_ClientDiscovery()

{

int FoundDevNo = 0;

int retval = SOAP\_OK;

wsdd\_\_ProbeType req; // 用于发送Probe消息

struct \_\_wsdd\_\_ProbeMatches resp; // 用于接收Probe应答

wsdd\_\_ScopesType sScope;

struct SOAP\_ENV\_\_Header header;

struct soap\* soap;

const char \*was\_To = "urn:schemas-xmlsoap-org:ws:2005:04:discovery";

const char \*was\_Action = "http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe";

//IP Adress and PortNo, broadCast

const char \*soap\_endpoint = "soap.udp://239.255.255.250:3702/"; //设备上服务器监听239.255.255.250的3702端口

//Create new soap object with info

soap = ONVIF\_Initsoap(&header, was\_To, was\_Action, 10);

soap\_default\_SOAP\_ENV\_\_Header(soap, &header);

soap->header = &header;

soap\_default\_wsdd\_\_ScopesType(soap, &sScope); // 设置寻找设备的范围

sScope.\_\_item = NULL;

soap\_default\_wsdd\_\_ProbeType(soap, &req); // 设置寻找设备的类型

req.Scopes = &sScope;

req.Types = NULL; //"dn:NetworkVideoTransmitter";

//sent the message broadcast and wait

retval = soap\_send\_\_\_wsdd\_\_Probe(soap, soap\_endpoint, NULL, &req); // 向组播地址广播Probe消息

while(retval == SOAP\_OK)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

retval = soap\_recv\_\_\_wsdd\_\_ProbeMatches(soap, &resp);

if(retval == SOAP\_OK)

{

if(soap->error)

{

printf("[%d]:recv soap error :%d, %s, %s\n", \_\_LINE\_\_, soap->error, \*soap\_faultcode(soap), \*soap\_faultstring(soap));

retval = soap->error;

}

else //we find a device

{

FoundDevNo++;

if(resp.wsdd\_\_ProbeMatches->ProbeMatch != NULL && resp.wsdd\_\_ProbeMatches->ProbeMatch->XAddrs != NULL)

{

printf("\*\*\*\*\* No %d Devices Information \*\*\*\*\*\n", FoundDevNo);

printf("Device Service Address : %s\r\n", resp.wsdd\_\_ProbeMatches->ProbeMatch->XAddrs);

printf("Device EP Address : %s\r\n", resp.wsdd\_\_ProbeMatches->ProbeMatch->wsa\_\_EndpointReference.Address);

printf("Device Type : %s\r\n", resp.wsdd\_\_ProbeMatches->ProbeMatch->Types);

printf("Device Metadata Version: %d\r\n", resp.wsdd\_\_ProbeMatches->ProbeMatch->MetadataVersion);

printf("[%d]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", \_\_LINE\_\_);

}

}

}

else if(soap->error)

{

if(FoundDevNo == 0)

{

printf("No Device found!\n");

retval = soap->error;

}

else

{

printf("Search end! Find %d Device! \n", FoundDevNo);

retval = 0;

}

break;

}

}

//释放函数

ONVIF\_soap\_delete(soap);

return retval;

}

int main(int argc, char \*argv[])

{

if(ONVIF\_ClientDiscovery() != 0)

{

printf("discover failed! \n");

return -1;

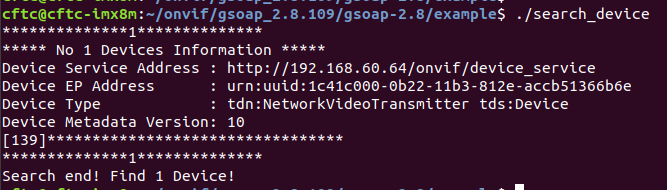
}

return 0;

}

编译指令：

gcc -o search\_device search\_device.c stdsoap2.c soapC.c soapClient.c struct\_timeval.c -I import/



**需要注意：虚拟机进行设备搜索时，网卡IP地址需设置成单一IP！**

soap\_send\_\_\_wsdd\_\_Probe无法发现设备：  
测试时开发机配置了多个固定ip，使用组播IP：239.255.255.250无法发现设备。开发机改为固定唯一IP，成功发现目标。

# 设备能力获取

int main(int argc,char \*argv[])

{

int ret = 0;

char sercer\_addr[] = "http://192.168.60.64/onvif/device\_service"; //设备搜索得到的地址

struct SOAP\_ENV\_\_Header header;

struct soap\* soap = ONVIF\_Initsoap(&header, NULL, NULL, 5);

struct \_tds\_\_GetCapabilities \*req;

struct \_tds\_\_GetCapabilitiesResponse \*Response;

if(NULL == (req = (struct \_tds\_\_GetCapabilities \*)calloc(1,sizeof(struct \_tds\_\_GetCapabilities))))

{

printf("calloc is error \n");

ret = -1;

return ret;

}else{

req->\_\_sizeCategory = 1;

req->Category = (enum tt\_\_CapabilityCategory \*)soap\_malloc(soap, sizeof(int));

\*(req->Category) = (enum tt\_\_CapabilityCategory)5; //5表示：tt\_\_CapabilityCategory\_\_Media

ONVIF\_SetAuthInfo(soap,"admin","cx123456"); //鉴权，输入摄像头的用户名、密码

ret = soap\_call\_\_\_tds\_\_GetCapabilities(soap, sercer\_addr, NULL,req, Response);

if(soap->error){

ret = -1;

printf("soap error: %d, %s, %s\n", soap->error, \*soap\_faultcode(soap), \*soap\_faultstring(soap));

return ret;

}else{;

if(NULL != Response->Capabilities)

{

if (Response->Capabilities->Media != NULL){

if (Response->Capabilities->Media->XAddr != NULL){

printf(" media\_addr: %s \n", Response->Capabilities->Media->XAddr);

}

}

}

}

}

if(NULL != req)

{

free(req);

req = NULL;

}

ONVIF\_soap\_delete(soap);

return ret;

}

编译指令：

gcc -o capabilities get\_capabilities.c stdsoap2.c soapC.c dom.c mecevp.c smdevp.c threads.c wsaapi.c wsseapi.c soapClient.c struct\_timeval.c -I import/ -DWITH\_OPENSSL -lssl -lcrypto -ldl



# Services获取多个媒体地址

**H265必须通过Services接口得到地址**

int main(int argc,char \*argv[])

{

int i = 0;

int ret = 0;

char secvre\_addr[] = "http://192.168.60.64/onvif/device\_service"; //设备搜索获取得到的服务地址

struct SOAP\_ENV\_\_Header header;

struct \_tds\_\_GetServices tds\_\_GetServices;

struct \_tds\_\_GetServicesResponse tds\_\_GetServicesResponse;

struct soap\* soap = ONVIF\_Initsoap(&header, NULL, NULL, 5);

//tds\_\_GetServices->IncludeCapability = (enum xsd\_\_boolean \*)soap\_malloc(soap, sizeof(int));

//\*(tds\_\_GetServices->IncludeCapability) = (enum xsd\_\_boolean)0;

tds\_\_GetServices.IncludeCapability = 0;

memset(&tds\_\_GetServices, 0x0, sizeof(struct \_tds\_\_GetServices));

memset(&tds\_\_GetServicesResponse, 0x0, sizeof(struct \_tds\_\_GetServicesResponse));

ONVIF\_SetAuthInfo(soap,"admin","cx123456"); //鉴权

soap\_call\_\_\_tds\_\_GetServices(soap,secvre\_addr,NULL, &tds\_\_GetServices, &tds\_\_GetServicesResponse);

if(soap->error){

ret = -1;

printf("soap error: %d, %s, %s\n", soap->error, \*soap\_faultcode(soap), \*soap\_faultstring(soap));

return ret;

}else{

if (tds\_\_GetServicesResponse.Service[i].Namespace != NULL ){

for(i=0; i<tds\_\_GetServicesResponse.\_\_sizeService; i++)

{

if(strcmp(tds\_\_GetServicesResponse.Service[i].Namespace, "http://www.onvif.org/ver20/media/wsdl") == 0)

{

printf(" media\_addr[%d] %s\n", i, tds\_\_GetServicesResponse.Service[i].XAddr);

}

if(strcmp(tds\_\_GetServicesResponse.Service[i].Namespace, "http://www.onvif.org/ver10/media/wsdl") == 0)

{

printf(" media\_addr->XAddr[%d] %s\n", i, tds\_\_GetServicesResponse.Service[i].XAddr);

}

}

}

}

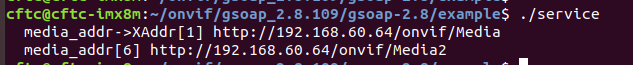
ONVIF\_soap\_delete(soap);

return ret;

}

编译指令：

gcc -o service get\_service.c stdsoap2.c soapC.c dom.c mecevp.c smdevp.c threads.c wsaapi.c wsseapi.c soapClient.c struct\_timeval.c -I import/ -DWITH\_OPENSSL -lssl -lcrypto -ldl



# profile获取媒体信息文件

int main(int argc,char \*argv[])

{

int i = 0;

int ret = 0;

char media\_addr[] = "http://192.168.60.64/onvif/media\_service"; //GetCapabilities得到的地址

char media\_addr2[] = "http://192.168.60.64/onvif/media2"; //GetServices得到的地址

struct SOAP\_ENV\_\_Header header;

struct soap\* soap = ONVIF\_Initsoap(&header, NULL, NULL, 5);

struct \_tr2\_\_GetProfiles tr2\_\_GetProfiles;

struct \_tr2\_\_GetProfilesResponse tr2\_\_GetProfilesResponse;

//tr2\_\_GetProfiles.\_\_sizeType = 1;

//tr2\_\_GetProfiles.Token = NULL;

//tr2\_\_GetProfiles.Type = NULL;

memset(&tr2\_\_GetProfilesResponse, 0x0, sizeof(struct \_tr2\_\_GetProfilesResponse));

memset((void \*)&tr2\_\_GetProfiles, 0x0, sizeof(struct \_tr2\_\_GetProfiles));

tr2\_\_GetProfiles.\_\_sizeType = 1;

tr2\_\_GetProfiles.Token = NULL;

char\* type[1] = {"VideoEncoder"};

tr2\_\_GetProfiles.Type = type;

ONVIF\_SetAuthInfo(soap,"admin","cx123456"); //鉴权

soap\_call\_\_\_tr2\_\_GetProfiles(soap, media\_addr2, NULL, &tr2\_\_GetProfiles, &tr2\_\_GetProfilesResponse);

if(soap->error){

ret = -1;

printf("soap error: %d, %s, %s\n", soap->error, \*soap\_faultcode(soap), \*soap\_faultstring(soap));

return ret;

}else{

for(i=0; i<tr2\_\_GetProfilesResponse.\_\_sizeProfiles; i++)

{

printf( "Profiles Name:%s \n",tr2\_\_GetProfilesResponse.Profiles[i].Name);

printf( "Profiles Taken:%s\n",tr2\_\_GetProfilesResponse.Profiles[i].token);

}

}

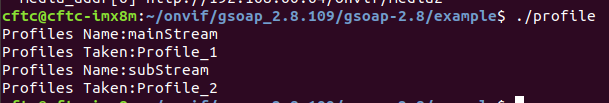
ONVIF\_soap\_delete(soap);

return ret;

}

编译指令：

gcc -o profile get\_profile.c stdsoap2.c soapC.c dom.c mecevp.c smdevp.c threads.c wsaapi.c wsseapi.c soapClient.c struct\_timeval.c -I import/ -DWITH\_OPENSSL -lssl -lcrypto -ldl



# 获取RTSP地址

int main(int argc,char \*argv[])

{

int i = 0;

int ret = 0;

char media\_addr[] = "http://192.168.60.64/onvif/media"; //GetCapabilities得到的地址

char media\_addr2[] = "http://192.168.60.64/onvif/media2"; //GetServices得到的地址

char taken[] = "Profile\_1"; //get\_profiles获取

struct SOAP\_ENV\_\_Header header;

struct soap\* soap = ONVIF\_Initsoap(&header, NULL, NULL, 5);

//...............................................h264通道....................................................

struct \_trt\_\_GetStreamUri trt\_\_GetStreamUri;

struct \_trt\_\_GetStreamUriResponse response;

trt\_\_GetStreamUri.StreamSetup = (struct tt\_\_StreamSetup\*)soap\_malloc(soap, sizeof(struct tt\_\_StreamSetup));

if (NULL == trt\_\_GetStreamUri.StreamSetup){

printf("soap\_malloc is error\n");

ret = -1;

}

trt\_\_GetStreamUri.StreamSetup->Stream = tt\_\_StreamType\_\_RTP\_Unicast;//stream type

trt\_\_GetStreamUri.StreamSetup->Transport = (struct tt\_\_Transport \*)soap\_malloc(soap, sizeof(struct tt\_\_Transport));

if (NULL == trt\_\_GetStreamUri.StreamSetup->Transport){

printf("soap\_malloc is error\n");

ret = -1;

}

trt\_\_GetStreamUri.StreamSetup->Transport->Protocol = 1;

trt\_\_GetStreamUri.StreamSetup->Transport->Tunnel = 0;

trt\_\_GetStreamUri.StreamSetup->\_\_size = 1;

trt\_\_GetStreamUri.StreamSetup->\_\_any = NULL;

trt\_\_GetStreamUri.StreamSetup->\_\_anyAttribute = NULL;

trt\_\_GetStreamUri.ProfileToken = (char \*)soap\_malloc(soap, 128\*sizeof(char ));//

if (NULL == trt\_\_GetStreamUri.ProfileToken){

printf("soap\_malloc is error\n");

ret = -1;

}

strcpy(trt\_\_GetStreamUri.ProfileToken, taken);

ONVIF\_SetAuthInfo(soap,"admin","cx123456"); //鉴权

soap\_call\_\_\_trt\_\_GetStreamUri(soap, media\_addr, NULL, &trt\_\_GetStreamUri, &response);

if(soap->error){

ret = -1;

printf("soap error: %d, %s, %s\n", soap->error, \*soap\_faultcode(soap), \*soap\_faultstring(soap));

return ret;

}else{

printf("rtsp\_addr: %s\n", response.MediaUri->Uri);

}

//...............................................h264通道....................................................

//...............................................h265通道....................................................

/\* struct \_tr2\_\_GetStreamUri tr2\_\_GetStreamUri;

struct \_tr2\_\_GetStreamUriResponse tr2\_\_GetStreamUriResponse;

tr2\_\_GetStreamUri.Protocol = (char \*)soap\_malloc(soap, 128\*sizeof(char));

if (NULL == tr2\_\_GetStreamUri.Protocol){

printf("soap\_malloc is error\n");

ret = -1;

}

tr2\_\_GetStreamUri.ProfileToken = (char \*)soap\_malloc(soap, 128\*sizeof(char ));

if (NULL == tr2\_\_GetStreamUri.ProfileToken){

printf("soap\_malloc is error\n");

ret = -1;

}

strcpy(tr2\_\_GetStreamUri.Protocol, "tcp");

strcpy(tr2\_\_GetStreamUri.ProfileToken, taken);

ONVIF\_SetAuthInfo(soap,"admin","cx123456"); //鉴权

soap\_call\_\_\_tr2\_\_GetStreamUri(soap, media\_addr2, NULL, &tr2\_\_GetStreamUri, &tr2\_\_GetStreamUriResponse);

if(soap->error){

ret = -1;

printf("soap error: %d, %s, %s\n", soap->error, \*soap\_faultcode(soap), \*soap\_faultstring(soap));

return ret;

}else{

printf("rtsp\_addr: %s\n", tr2\_\_GetStreamUriResponse.Uri);

}\*/

//...............................................h265通道....................................................

ONVIF\_soap\_delete(soap);

return ret;

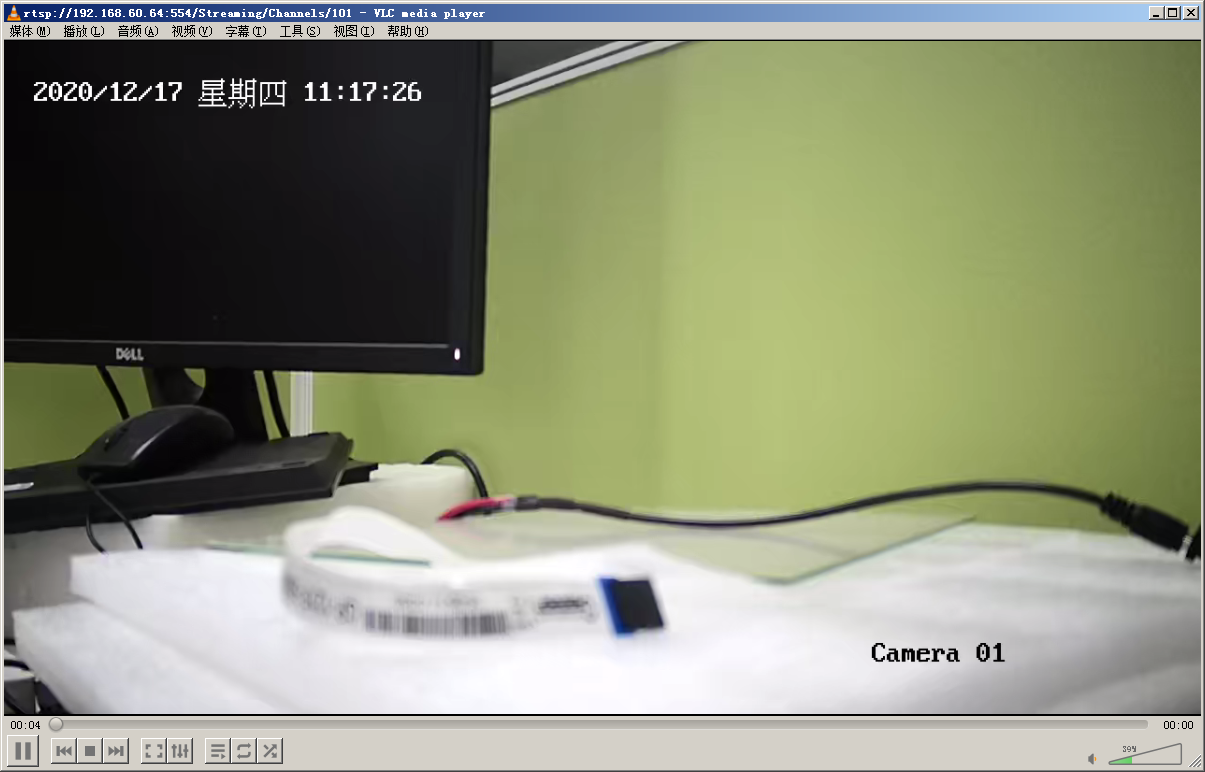
}

编译指令：

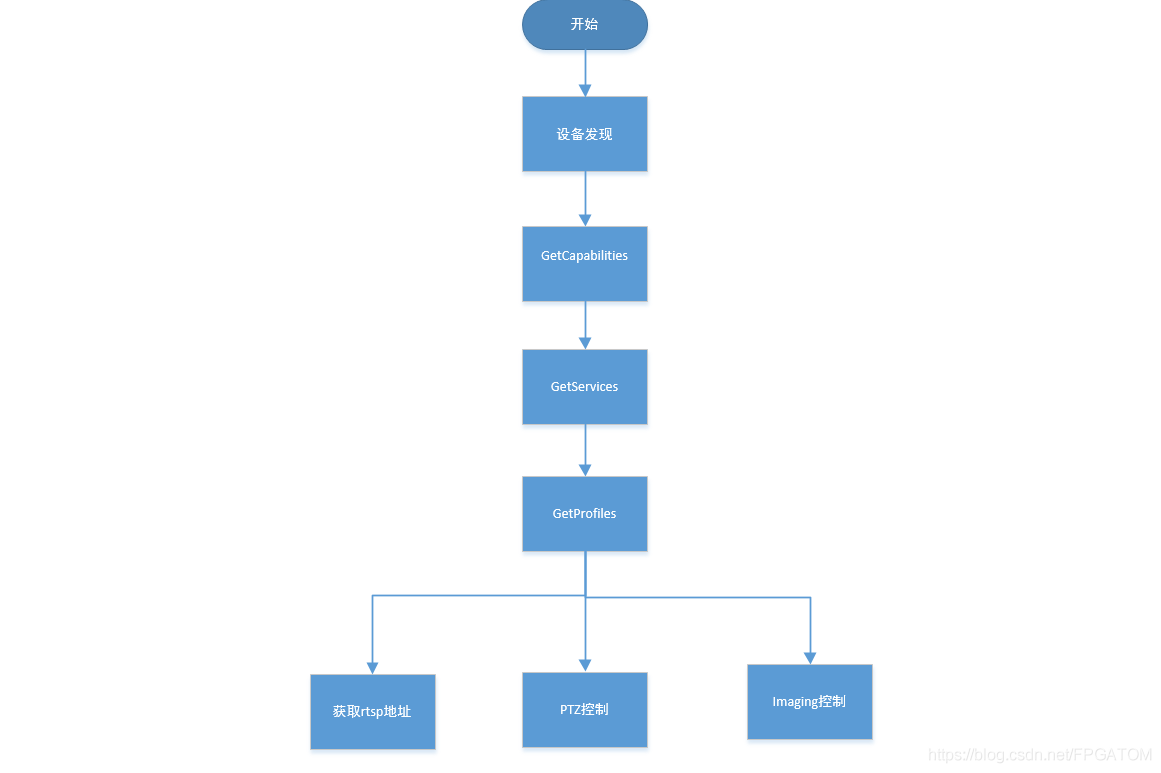
gcc -o rtsp get\_rtsp.c stdsoap2.c soapC.c dom.c mecevp.c smdevp.c threads.c wsaapi.c wsseapi.c soapClient.c struct\_timeval.c -I import/ -DWITH\_OPENSSL -lssl -lcrypto -ldl

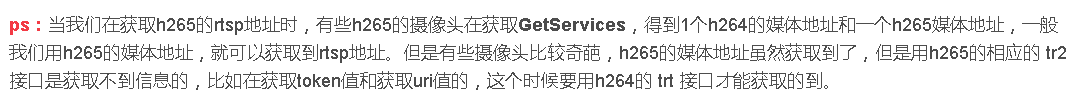


将获取到的rtsp地址通过VLC media player进行播放



至此，整个onvif获取rtsp流程已完成。总结整个步骤如下所示：





# libonvif库封装

参照原有库代码，增加对H265相机的支持。具体修改如下：

* 增加GetProfiles2接口，用于获取H265的媒体信息文件

static int GetProfiles2(struct soap \* pstSoap, const char \* acHttpAddr, struct \_tr2\_\_GetProfilesResponse \* ptr2\_\_GetProfilesResponse)

{

//struct SOAP\_ENV\_\_Header header;

struct \_tr2\_\_GetProfiles tr2\_\_GetProfiles;

pstSoap->recv\_timeout = 3; //超过5秒钟没有数据就退出

//soap\_default\_SOAP\_ENV\_\_Header(pstSoap, &header);

memset((void \*)&tr2\_\_GetProfiles, 0x0, sizeof(struct \_tr2\_\_GetProfiles));

tr2\_\_GetProfiles.\_\_sizeType = 1;

tr2\_\_GetProfiles.Token = NULL;

char\* type[1] = {"All"};

tr2\_\_GetProfiles.Type = type;

soap\_call\_\_\_tr2\_\_GetProfiles(pstSoap, acHttpAddr, NULL, &tr2\_\_GetProfiles, ptr2\_\_GetProfilesResponse);

if ( pstSoap->error )

{

ONVIF\_ERROR\_PRINT(pstSoap);

return pstSoap->error;

}

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s: Success! [Addr: %s]\n", \_\_FUNCTION\_\_, acHttpAddr);

return 0;

}

* 增加FinishCamInfo2接口，用于获取H265相机的具体参数信息

static int FinishCamInfo2(struct soap \* pstSoap, int channel, ST\_ONVIF\_CAM\_SERVICE\_ADDRS \* pstServerAddrs, ST\_ONVIF\_CAMINFO\_SET \* pstCamInfo)

{

int iRet = 0;

struct \_tr2\_\_GetProfilesResponse tr2\_\_GetProfilesResponse;

//char acNetAddr[SER\_NET\_ADDR\_LEN] = {0x0};

char acUril[SER\_NET\_ADDR\_LEN] = {0x0};

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

if ( (channel >= MAX\_ONVIF\_CAM\_NUM) || (channel < 0) || ( NULL == pstCamInfo) )

{

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s: %s\n", \_\_FUNCTION\_\_, ONVIF\_ERR2STR(E\_OAR\_INPUT\_ERR));

return E\_OAR\_INPUT\_ERR;

}

if ( 0 == strlen(pstServerAddrs->acMedia2ServiceAddr) )

{

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s: %s\n", \_\_FUNCTION\_\_, ONVIF\_ERR2STR(E\_OAR\_ADDR\_ERR));

return E\_OAR\_ADDR\_ERR;

}

memset(&tr2\_\_GetProfilesResponse, 0x0, sizeof(struct \_tr2\_\_GetProfilesResponse));

CamAuthorized(pstSoap, channel);

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

iRet = GetProfiles2(pstSoap, pstServerAddrs->acMedia2ServiceAddr, &tr2\_\_GetProfilesResponse);

if ( 0 != iRet )

{

return E\_OAR\_SOAP\_ERR;

}

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

/\* 主码流 \*/

pstCamInfo->stCamMainVideo.eFixed = (ONVIF\_BOOL)tr2\_\_GetProfilesResponse.Profiles[0].fixed;

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

if(tr2\_\_GetProfilesResponse.Profiles[0].Configurations)

{

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

if(tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder)

{

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

if(tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->RateControl)

{

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

pstCamInfo->stCamMainVideo.iEncBitRate = tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->RateControl->BitrateLimit;

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n iEncBitRate:%d",\_\_FUNCTION\_\_,\_\_LINE\_\_,pstCamInfo->stCamMainVideo.iEncBitRate);

pstCamInfo->stCamMainVideo.iFps = tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->RateControl->FrameRateLimit;

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n iFps:%d",\_\_FUNCTION\_\_,\_\_LINE\_\_,pstCamInfo->stCamMainVideo.iFps);

if(strcmp(tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->Encoding , "JPEG") == 0)

{

pstCamInfo->stCamMainVideo.iEncType = 0;

}

else if(strcmp(tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->Encoding , "MPEG4") == 0)

{

pstCamInfo->stCamMainVideo.iEncType = 1;

}

else if(strcmp(tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->Encoding , "H264") == 0)

{

pstCamInfo->stCamMainVideo.iEncType = 2;

}

else if(strcmp(tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->Encoding , "H265") == 0)

{

pstCamInfo->stCamMainVideo.iEncType = 3;

}

else

{

pstCamInfo->stCamMainVideo.iEncType = -1;

}

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n iEncType:%d",\_\_FUNCTION\_\_,\_\_LINE\_\_,pstCamInfo->stCamMainVideo.iEncType);

}

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

if(tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->Resolution)

{

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

pstCamInfo->stCamMainVideo.iWidth = tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->Resolution->Width;

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d iWidth:%d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_,pstCamInfo->stCamMainVideo.iWidth);

pstCamInfo->stCamMainVideo.iHeight = tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoEncoder->Resolution->Height;

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d iHeight:%d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_,pstCamInfo->stCamMainVideo.iHeight);

}

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

}

}

/\* 用strncpy更加安全，且最大拷贝字串长度预留1个结束符 \*/

strncpy(pstCamInfo->stCamMainVideo.acProfile, tr2\_\_GetProfilesResponse.Profiles[0].token, ONVIF\_STRING\_LEN - 1);

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

strncpy(pstCamInfo->stCamMainVideo.acSrcToken, tr2\_\_GetProfilesResponse.Profiles[0].Configurations->VideoSource->SourceToken, ONVIF\_STRING\_LEN - 1);

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

memset(acUril, 0x0, SER\_NET\_ADDR\_LEN);

CamAuthorized(pstSoap, channel);

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

iRet = GetRtspUril(pstSoap, tr2\_\_GetProfilesResponse.Profiles[0].token, pstServerAddrs->acMediaServiceAddr, acUril);

strcpy(pstCamInfo->stCamMainVideo.acRtspUri, acUril);

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

memset(acUril, 0x0, SER\_NET\_ADDR\_LEN);

CamAuthorized(pstSoap, channel);

iRet = GetSnapShortUril(pstSoap, tr2\_\_GetProfilesResponse.Profiles[0].token, pstServerAddrs->acMediaServiceAddr, acUril);

strcpy(pstCamInfo->stCamMainVideo.acSnapShortUri, acUril);

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

/\* 辅码流 \*/

if ( tr2\_\_GetProfilesResponse.\_\_sizeProfiles < 2 )

{

DebugPrint(ONVIF\_DEBUG\_LEVEL, "No SubStream Exsit\n");

//return E\_OAR\_OTHER\_ERR;

}

else

{

;

pstCamInfo->stCamSubVideo.eFixed = (ONVIF\_BOOL)tr2\_\_GetProfilesResponse.Profiles[1].fixed;

pstCamInfo->stCamSubVideo.iEncBitRate = tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoEncoder->RateControl->BitrateLimit;

pstCamInfo->stCamSubVideo.iFps = tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoEncoder->RateControl->FrameRateLimit;

pstCamInfo->stCamSubVideo.iWidth = tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoEncoder->Resolution->Width;

pstCamInfo->stCamSubVideo.iHeight = tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoEncoder->Resolution->Height;

if(strcmp(tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoEncoder->Encoding , "JPEG") == 0)

{

pstCamInfo->stCamMainVideo.iEncType = 0;

}

else if(strcmp(tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoEncoder->Encoding , "MPEG4") == 0)

{

pstCamInfo->stCamMainVideo.iEncType = 1;

}

else if(strcmp(tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoEncoder->Encoding , "H264") == 0)

{

pstCamInfo->stCamMainVideo.iEncType = 2;

}

else if(strcmp(tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoEncoder->Encoding , "H265") == 0)

{

pstCamInfo->stCamMainVideo.iEncType = 3;

}

else

{

pstCamInfo->stCamMainVideo.iEncType = -1;

}

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n iEncType:%d",\_\_FUNCTION\_\_,\_\_LINE\_\_,pstCamInfo->stCamMainVideo.iEncType);

/\* 用strncpy更加安全，且最大拷贝字串长度预留1个结束符 \*/

strncpy(pstCamInfo->stCamSubVideo.acProfile, tr2\_\_GetProfilesResponse.Profiles[1].token, ONVIF\_STRING\_LEN - 1);

strncpy(pstCamInfo->stCamSubVideo.acSrcToken, tr2\_\_GetProfilesResponse.Profiles[1].Configurations->VideoSource->SourceToken, ONVIF\_STRING\_LEN - 1);

memset(acUril, 0x0, SER\_NET\_ADDR\_LEN);

CamAuthorized(pstSoap, channel);

iRet = GetRtspUril(pstSoap, tr2\_\_GetProfilesResponse.Profiles[1].token, pstServerAddrs->acMediaServiceAddr, acUril);

strcpy(pstCamInfo->stCamSubVideo.acRtspUri, acUril);

memset(acUril, 0x0, SER\_NET\_ADDR\_LEN);

CamAuthorized(pstSoap, channel);

iRet = GetSnapShortUril(pstSoap, tr2\_\_GetProfilesResponse.Profiles[1].token, pstServerAddrs->acMediaServiceAddr, acUril);

strcpy(pstCamInfo->stCamSubVideo.acSnapShortUri, acUril);

}

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

/\* 拷贝服务地址 \*/

memcpy(&(pstCamInfo->stServiceAddrs), pstServerAddrs, sizeof(ST\_ONVIF\_CAM\_SERVICE\_ADDRS));

pstCamInfo->eValid = ONVIF\_TRUE;

/\* 获取预置位信息 \*/

/\*

if ( 0 != strlen(pstServerAddrs->acPTZServiceAddr) )

{

CamAuthorized(m\_pstSoap, channel);

GetAllPresets(m\_pstSoap, pstServerAddrs->acPTZServiceAddr,

trt\_\_GetProfilesResponse.Profiles[0].token, &g\_CamPresetInfo[channel]);

}

\*/

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

if ( 0 != strlen(pstServerAddrs->acPTZServiceAddr) )

{

CamAuthorized(pstSoap, channel);

iRet = GetAllPresets(pstSoap, pstServerAddrs->acPTZServiceAddr,tr2\_\_GetProfilesResponse.Profiles[0].token);

if ( 0 != iRet )

{

CamAuthorized(pstSoap, channel);

GetAllPresets(pstSoap, pstServerAddrs->acCamHttpAddr,tr2\_\_GetProfilesResponse.Profiles[0].token);

}

}

DebugPrint(ONVIF\_DEBUG\_LEVEL, "%s %d\n",\_\_FUNCTION\_\_,\_\_LINE\_\_);

return E\_OAR\_OK;

}

* 修改逻辑，先搜索H265的相机信息，如果失败则继续搜索H264相机信息

/\*20201217-pw: 先获取H265相机相关信息，如果不成功再获取H264相机相关信息\*/

if(E\_OAR\_SOAP\_ERR == FinishCamInfo2(pSoap, i, &stServiceAddrs, &stCamInfo))

{

//H265解码

FinishCamInfo(pSoap, i, &stServiceAddrs, &stCamInfo);

}

* 修改结构体，增加编码类型参数

/\* onvif摄像头视频参数 \*/

typedef struct tagST\_ONVIF\_CAM\_VIDEO\_PARAM

{

int iWidth;

int iHeight;

int iFps;

int iEncBitRate;

int iEncType; /\* 20201217-pw:当前编码类型：0：JPEG 1：MPEG4 2：H264 3：H265 -1：无效 \*/

ONVIF\_BOOL eFixed; /\* 启用标志 \*/

char acRtspUri[SER\_NET\_ADDR\_LEN]; /\* rtsp流地址 \*/

char acSnapShortUri[SER\_NET\_ADDR\_LEN]; /\* 抓拍地址 \*/

char acProfile[ONVIF\_STRING\_LEN]; /\* profile \*/

char acSrcToken[ONVIF\_STRING\_LEN]; /\* SourceToken \*/

} ST\_ONVIF\_CAM\_VIDEO\_PARAM;

* 修改结构体，增加H265设备地址

typedef struct tagST\_ONVIF\_CAM\_SERVICE\_ADDRS

{

char acDeviceServiceAddr[SER\_NET\_ADDR\_LEN];

char acMediaServiceAddr[SER\_NET\_ADDR\_LEN]; /\* H264设备地址 \*/

char acMedia2ServiceAddr[SER\_NET\_ADDR\_LEN]; /\* 20201217-pw: H265设备地址 \*/

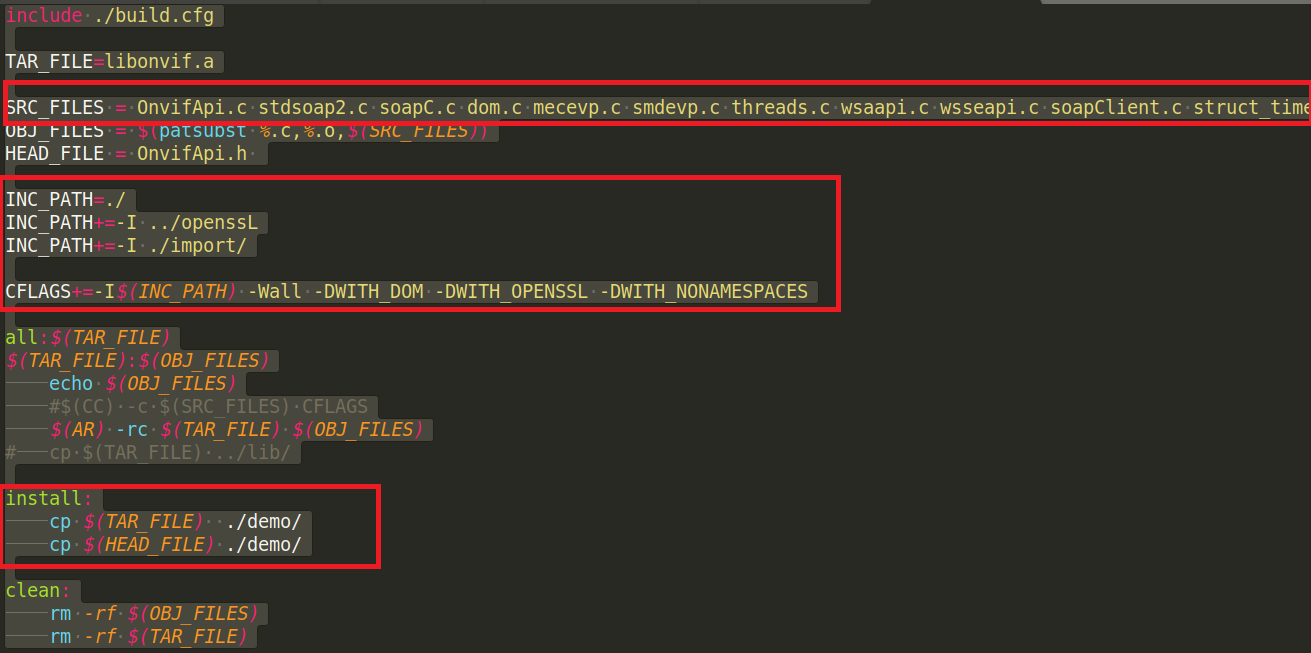
char acPTZServiceAddr[SER\_NET\_ADDR\_LEN];

char acIMGServiceAddr[SER\_NET\_ADDR\_LEN];

char acCamHttpAddr[SER\_NET\_ADDR\_LEN]; /\* 摄像头http地址 \*/

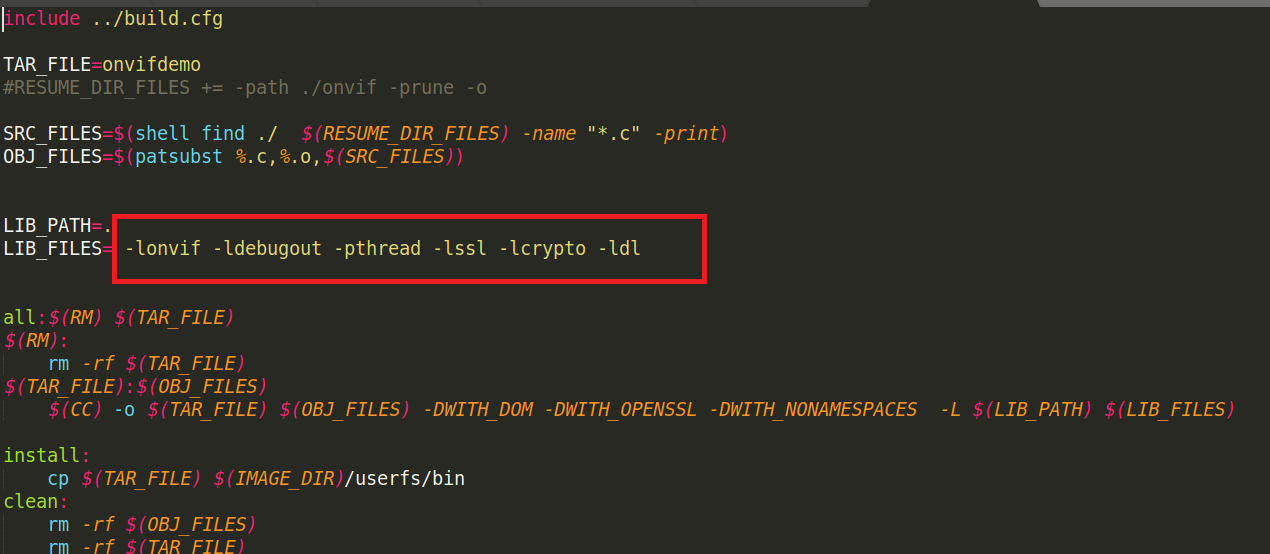
} ST\_ONVIF\_CAM\_SERVICE\_ADDRS;

* 修改Makefile，手动指定需要编译的c源文件



* 修改测试用例的Makefile，.a文件的链接顺序，不能错！！！！否则会导致编译错误。

-lpthread修改为-pthread



参考链接：

<https://blog.csdn.net/xuebing1995/article/details/102534494>

<https://blog.csdn.net/kangroger/article/details/39999573>

https://blog.csdn.net/qq\_22122811/article/details/52738134