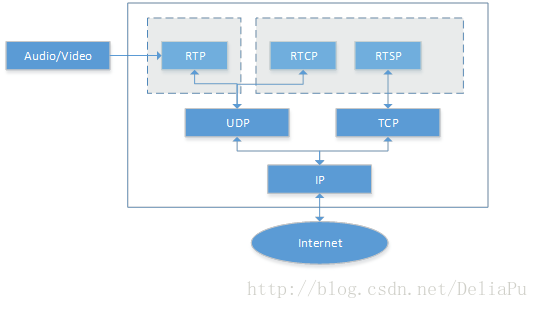
1. **live555简介**

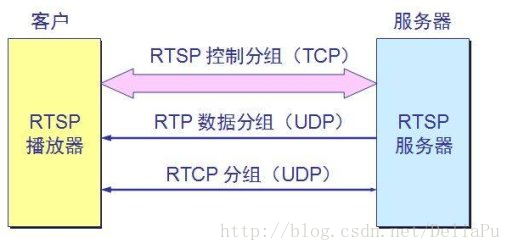
live555 是一个 C++ 开发的流媒体项目，它主要由几个用于多媒体流的库组成，其官方网站地址为 [http://www.live555.com/。live555](http://www.live555.com/。live555" \t "https://www.wolfcstech.com/2017/08/28/live555_src_analysis_introduction/_blank) 使用开放的标准协议 (RTP/RTCP，RTSP，SIP)，方便与其它标准的流媒体组件互操作。这些库可以为 Unix-like（包括 Linux 和 Mac OS X），Windows，和 QNX （及其它 POSIX 兼容系统）等系统进行编译，它们可以被用于构建流媒体应用。除了库之外，live555 还包含了两个流媒体应用程序 “[LIVE555 Media Server](http://www.live555.com/mediaServer/" \t "https://www.wolfcstech.com/2017/08/28/live555_src_analysis_introduction/_blank)“ 和 “[LIVE555 Proxy Server](http://www.live555.com/proxyServer/" \t "https://www.wolfcstech.com/2017/08/28/live555_src_analysis_introduction/_blank)“，它们都是 RTSP 服务器应用程序。  
live555 的库可以被用于处理 MPEG，H.265，H.264，H.263+，DV 或 JPEG 视频，及多种音频格式。它们还可以非常简单地进行扩展，以支持其它的音频或视频编解码格式，并可以被用于构建基本的 [RTSP](http://www.live555.com/openRTSP/" \t "https://www.wolfcstech.com/2017/08/28/live555_src_analysis_introduction/_blank) 或 [SIP](http://www.live555.com/playSIP/" \t "https://www.wolfcstech.com/2017/08/28/live555_src_analysis_introduction/_blank) 客户端和服务器。

1. **RTSP简介**

本次机器人采用rtsp流媒体传输协议实时传输摄像头数据，RTSP(Real-Time Stream Protocol)协议是一个基于文本的多媒体播放控制协议，属于应用层。RTSP以客户端方式工作，对流媒体提供播放、暂停、后退、前进等操作。该标准由IETF指定，对应的协议是RFC2326。RTSP作为一个应用层协议，提供了一个可供扩展的框架，使得流媒体的受控和点播变得可能，它主要用来控制具有实时特性的数据的发送，但其本身并不用于传送流媒体数据，而必须依赖下层传输协议(如RTP/RTCP)所提供的服务来完成流媒体数据的传送。RTSP负责定义具体的控制信息、操作方法、状态码，以及描述与RTP之间的交互操作。RTSP媒体服务协议框架如下：



RTSP包含Normal RTSP(数据通过RTP传输，应用厂商有苹果和[微软](https://www.baidu.com/s?wd=%E5%BE%AE%E8%BD%AF&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/deliapu/article/details/_blank)等)，以及Real-RTSP(数据通过RDT传输)。RTSP传输的一般是TS、MP4格式的流，其传输一般需要2~3个通道，命令和数据通道分离。使用RTSP协议传输流媒体数据需要有专门的媒体播放器和[媒体服务器](https://www.baidu.com/s?wd=%E5%AA%92%E4%BD%93%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/deliapu/article/details/_blank)，也就是需要支持RTSP协议的客户端和服务器。



客户端要播放RTSP媒体流，就需要知道媒体源的URL，RTSP的URL格式一般如下：

rtsp://host[:port]/[abs\_path]/content\_name

host: 有效的域名或IP地址；

port: 端口号，缺省为554，若为缺省可不填写，否则必须写明。

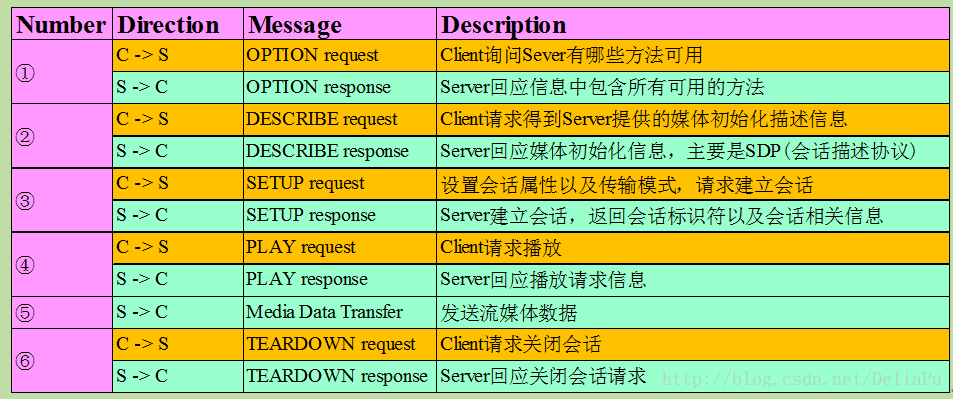
例如，一个完整的RTSP URL可写为：

rtsp://192.168.1.67:554/test

RTSP请求报文的常用方法与作用：



一次基本的RTSP交互过程如下，C表示客户端，S表示服务端



**三、 v4l2简介**

为了屏蔽具体的摄像头硬件，采用V4l2底层驱动接口，v4l2为linux下视频设备程序提供了一套接口规范，只能在linux下使用。它使程序有发现设备和操作设备的能力。它主要是用一系列的回调函数来实现这些功能。像设置摄像头的频率、帧频、视频压缩格式和图像参数等等。一般来说，采用V4L2驱动的摄像头设备文是/dev/v4l/video0。为了通用，可以建立一个到/dev/video0的链接。V4L2支持两种方式来采集图像：内存映射方式(mmap)和直接读取方式(read)。V4L2在include/linux/videodev.h文件中定义了一些重要的数据结构，在采集图像的过程中，就是通过对这些数据的操作来获得最终的图像数据。Linux系统V4L2的能力可在Linux内核编译阶段配置，默认情况下都有此开发接口。V4L2从Linux 2.5.x版本的内核中开始出现。

V4L2规范中不仅定义了通用API元素(Common API Elements)，图像的格式(Image Formats)，输入/输出方法(Input/Output)，还定义了Linux内核驱动处理视频信息的一系列接口(Interfaces)，这些接口主要有：

　　视频采集接口——Video Capture Interface;

　　视频输出接口—— Video Output Interface;

　　视频覆盖/预览接口——Video Overlay Interface;

　　视频输出覆盖接口——Video Output Overlay Interface;

编解码接口——Codec Interface。

**四 、主要功能代码**

开源的live555自带的livemedia sever用于播放本地视频，所以需要修改其代码用于获取摄像头数据以及在局域网内利用rtsp单播h264视频流；

其主要代码如下：

*/\*\*\*\*\*\*\*\*\*\**

*This library is free software; you can redistribute it and/or modify it under*

*the terms of the GNU Lesser General Public License as published by the*

*Free Software Foundation; either version 3 of the License, or (at your*

*option) any later version. (See <http://www.gnu.org/copyleft/lesser.html>.)*

*This library is distributed in the hope that it will be useful, but WITHOUT*

*ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS*

*FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for*

*more details.*

*You should have received a copy of the GNU Lesser General Public License*

*along with this library; if not, write to the Free Software Foundation, Inc.,*

*51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA*

*\*\*\*\*\*\*\*\*\*\*/// Copyright (c) 1996-2017, Live Networks, Inc. All rights reserved// A test program that demonstrates how to stream - via unicast RTP// - various kinds of file on demand, using a built-in RTSP server.// main program*

**#include "liveMedia.hh"#include "BasicUsageEnvironment.hh"#include <H264FramedLiveSource.hh>#include <sys/types.h>#include <sys/stat.h>**

UsageEnvironment\* env;**char** **const**\* inputFileName = "/tmp/fifo";**char** \*ptr;*//H264VideoStreamFramer\* videoSource;//RTPSink\* videoSink;***class** **Device** **Camera**;

*//void play(); // forward*

EventTriggerId DeviceSource::eventTriggerId = 0;

*// To make the second and subsequent client for each stream reuse the same// input stream as the first client (rather than playing the file from the// start for each client), change the following "False" to "True":*

Boolean reuseFirstSource = False;

*// To stream \*only\* MPEG-1 or 2 video "I" frames// (e.g., to reduce network bandwidth),// change the following "False" to "True":*

Boolean iFramesOnly = False;

**static** **void** **announceStream**(RTSPServer\* rtspServer, ServerMediaSession\* sms,

**char** **const**\* streamName, **char** **const**\* inputFileName); *// fwd*

**static** **char** newDemuxWatchVariable;

**static** MatroskaFileServerDemux\* matroskaDemux;**static** **void** **onMatroskaDemuxCreation**(MatroskaFileServerDemux\* newDemux, **void**\* */\*clientData\*/*) {

matroskaDemux = newDemux;

newDemuxWatchVariable = 1;

}

**static** OggFileServerDemux\* oggDemux;**static** **void** **onOggDemuxCreation**(OggFileServerDemux\* newDemux, **void**\* */\*clientData\*/*) {

oggDemux = newDemux;

newDemuxWatchVariable = 1;

}

**int** **main**(**int** argc, **char**\*\* argv) {

*// Begin by setting up our usage environment:*

TaskScheduler\* scheduler = BasicTaskScheduler::createNew();

env = BasicUsageEnvironment::createNew(\*scheduler);

UserAuthenticationDatabase\* authDB = NULL;**#ifdef ACCESS\_CONTROL**

*// To implement client access control to the RTSP server, do the following:*

authDB = **new** UserAuthenticationDatabase;

authDB->addUserRecord("username1", "password1"); *// replace these with real strings*

*// Repeat the above with each <username>, <password> that you wish to allow*

*// access to the server.***#endif**

*// Create the RTSP server:*

RTSPServer\* rtspServer = RTSPServer::createNew(\*env, 8554, authDB);

**if** (rtspServer == NULL) {

\*env << "Failed to create RTSP server: " << env->getResultMsg() << "\n";

exit(1);

}

**char** **const**\* descriptionString

= "Session streamed by \"testOnDemandRTSPServer\"";

Camera.Init();

mkfifo(inputFileName, 0777);

**if**(0 == fork())

{

Camera.pipe\_fd = fopen(inputFileName, "w");

**if**(NULL == Camera.pipe\_fd)

{

printf("===============child process open pipe err =======\n ");

}

**while**(1)

{

usleep(15000);

Camera.getnextframe();

}

}

*// Set up each of the possible streams that can be served by the*

*// RTSP server. Each such stream is implemented using a*

*// "ServerMediaSession" object, plus one or more*

*// "ServerMediaSubsession" objects for each audio/video substream.*

*// A MPEG-4 video elementary stream:*

*// A H.264 video elementary stream:*

{

**char** **const**\* streamName = "testStream";

*//char const\* inputFileName = "test.264";*

ServerMediaSession\* sms

= ServerMediaSession::createNew(\*env, "testStream", inputFileName,

descriptionString);

sms->addSubsession(H264VideoFileServerMediaSubsession

::createNew(\*env, inputFileName, reuseFirstSource));

rtspServer->addServerMediaSession(sms);

announceStream(rtspServer, sms, streamName, inputFileName);

}

*// A H.265 video elementary stream:*

*// A MPEG-1 or 2 audio+video program stream:*

*// A MPEG-1 or 2 video elementary stream:*

*// A MP3 audio stream (actually, any MPEG-1 or 2 audio file will work):*

*// To stream using 'ADUs' rather than raw MP3 frames, uncomment the following://#define STREAM\_USING\_ADUS 1*

*// To also reorder ADUs before streaming, uncomment the following://#define INTERLEAVE\_ADUS 1*

*// (For more information about ADUs and interleaving,*

*// see <http://www.live555.com/rtp-mp3/>)*

*// A WAV audio stream:*

*// An AMR audio stream:*

*// A 'VOB' file (e.g., from an unencrypted DVD):*

*// A MPEG-2 Transport Stream:*

*// An AAC audio stream (ADTS-format file):*

*// A DV video stream:*

*// A AC3 video elementary stream:*

*// A Matroska ('.mkv') file, with video+audio+subtitle streams:*

*// A WebM ('.webm') file, with video(VP8)+audio(Vorbis) streams:*

*// (Note: ".webm' files are special types of Matroska files, so we use the same code as the Matroska ('.mkv') file code above.)*

*// An Ogg ('.ogg') file, with video and/or audio streams:*

*// An Opus ('.opus') audio file:*

*// (Note: ".opus' files are special types of Ogg files, so we use the same code as the Ogg ('.ogg') file code above.)*

*// A MPEG-2 Transport Stream, coming from a live UDP (raw-UDP or RTP/UDP) source:*

*// Also, attempt to create a HTTP server for RTSP-over-HTTP tunneling.*

*// Try first with the default HTTP port (80), and then with the alternative HTTP*

*// port numbers (8000 and 8080).*

env->taskScheduler().doEventLoop(); *// does not return*

**return** 0; *// only to prevent compiler warning*

}

**static** **void** **announceStream**(RTSPServer\* rtspServer, ServerMediaSession\* sms,

**char** **const**\* streamName, **char** **const**\* inputFileName) {

**char**\* url = rtspServer->rtspURL(sms);

UsageEnvironment& env = rtspServer->envir();

env << "\n\"" << streamName << "\" stream, from the file \""

<< inputFileName << "\"\n";

env << "Play this stream using the URL \"" << url << "\"\n";

**delete**[] url;

}