# 基于vlc的live555

live555下载网址： 参考vlc/contrib/src/live555/rules.mak文件

LIVE555\_FILE := live.2014.01.21.tar.gz

LIVEDOTCOM\_URL := http://download.videolan.org/pub/contrib/live555/$(LIVE555\_FILE)

curl -f -L "http://download.videolan.org/pub/contrib/live555/live.2014.01.21.tar.gz" > "live.2014.01.21.tar.gz"

=========================================================

live555编译: 生成动态lib库文件

修改文件：config.linux-with-shared-libraries

增加-m64 -g选项，支持linux-64bit, 支持debug模式： COMPILE\_OPTS += -m64 -g

#cd ./live

#./genMakefile linux-with-shared-libraries

#make

#sudo make install //将live555生成的文件加载到系统默认路径: /usr/local

#sudo ldconfig //更新动态lib库的加载信息

# 下载

源码的下载：[http://www.live555.com](http://www.live555.com/)

# 编译：

./genMakefile linux //linux 32为操作系统

./genMakefile linux-64bit //linux 64位操作系统， //-m64

./genMakefile linux-gdb //linux 32位操作系统，支持gdb //-g

./genMakefile linux-with-shared-libraries //linux 32为操作系统，支持动态库

#file /sbin/init //可以查看操作系统位数

在genMakefile文件所在目录中有很多config.打头的文件，这些文件的后缀对应不同平台的操作系统

Linux 64位操作系统编译动态库：

可以修改config.linux-with-shared-libraries文件，在COMPILE\_OPTS中增加 “–m64 –g”， 使其支持linux 64位操作系统，并支持dbg

修改安装目录：

可以修改config.linux-with-shared-libraries文件，在文件开始的地方添加”PREFIX=/home/user/live/out”和”LIBDIR=$(PREFIX)/lib”, 删除子文件夹中Makefile.tail文件中的” PREFIX=/usr/local”。

# 基本库说明：

**UsageEnvironment：**

UsageEnvironment包括抽象类UsageEnvironment和抽象类TaskScheduler，这两个类用于事件调度，其中包括实现了对事件的异步读取、对事件句柄的设置及对错误信息的输出等；该库中还有一个HashTable，这是一个通用的HashTable，在整个项目中都可以使用它，当然该HashTable也是一个抽象类。

**BasicUsageEnvironment：**

BasicUsageEnvironment中的类主要是对UsageEnvironment中对应类的实现。

**Groupsock：**

groupsock，顾名思义，用于数据包的接收和发送，其同时支持多播和单播。groupsock库中包括了GroupEId、Groupsock、GroupsockHelper、NetAddress、NetInterface等类，其中Groupsock类有两个构造函数，一个是“for a source-independent multicast group”，另一个是“for a source-specific multicast group”；而GroupsockHelper类主要用于读写Socket。

**liveMedia：**

liveMedia是很重要的一个库，其不仅包含了实现RTSP Server的类，还包含了针对不同流媒体类型（如TS流、PS流等）编码的类。在该库中，基类是Medium，层次关系非常清晰。在该库中，有几个很重要的类，如RTSPServer、ServerMediaSession、RTPSink、RTPInterface、FramedSource等。

# liveMedia库中的主要类结构

liveMedia库中的基类为Medium，其下又有几个非常重要的部分：

一个是×××Subsession，除Medium父类外，所有的×××Subsession类都继承于ServerMediaSubsession；

一个是×××Source，MediaSource的frameSource下主要包含FramedFileSource、RTPSource、FramedFilter等几个主要的部分；

一个是MediaSink，以继承于RTPSink的类居多。

此外，还包含了用于处理packet的BufferedPacketFactory和BufferedPacket及其相关子类，用于处理流分析的StreamParser及其子类。

  Medium

  RTSPServer

  RTSPOverHTTPServer

  MediaSession

  ServerMediaSession

  ServerMediaSubsession

  OnDemandServerMediaSubsession

  FileServerMediaSubsession

  ADTSAudioFileServerMediaSubsession

  AMRAudioFileServerMediaSubsession

  H263plusVideoFileServerMediaSubsession

  MP3AudioFileServerMediaSubsession

  MPEG1or2VideoFileServerMediaSubsession

  MPEG2TransportFileServerMediaSubsession

  MPEG4VideoFileServerMediaSubsession

  WAVAudioFileServerMediaSubsession

  MPEG1or2DemuxedServerMediaSubsession

  PassiveServerMediaSubsession

  MediaSource

  FramedSource

  FramedFileSource

  ByteStreamFileSource

  ADTSAudioFileSource

  MP3FileSource

  MP3HTTPSource

  BasicUDPSource

  RTPSource

  MultiFramedRTPSource

  RawQCELPRTPSource

  AC3AudioRTPSource

  MPEG4GenericRTPSource

  RawAMRRTPSource

  H261VideoRTPSource

  H263plusVideoRTPSource

  H264VideoRTPSource

  JPEGVideoRTPSource

  MP3ADURTPSource

  MPEG1or2AudioRTPSource

  MPEG1or2VideoRTPSource

  MPEG4ESVideoRTPSource

  MPEG4GenericRTPSource

  MPEG4LATMAudioRTPSource

  DVVideoRTPSource

  QuickTimeGenericRTPSource

  SimpleRTPSource

  AMRAudioSource

  AMRDeinterleaver

  AMRAudioFileSource

  ByteStreamMultiFileSource

  DeviceSource

  JPEGVideoSource

  MPEG1or2DemuxedElementaryStream

  MPEG2TransportStreamMultiplexor

  MPEG2TransportStreamFromESSource

  MPEG2TransportStreamFromPESSource

  AudioInputDevice

  WAVAudioFileSource

  FramedFilter

  H264FUAFragmenter

  QCELPDeinterleaver

  AC3AudioStreamFramer

  ADUFromMP3Source

  uLawFromPCMAudioSource

  H264VideoStreamFramer

  MP3FromADUSource

  MP3Transcoder

  PCMFromuLawAudioSource

  MPEG2IFrameIndexFromTransportStream

  NetworkFromHostOrder16

  HostFromNetworkOrder16

  MP3ADUinterleaverBase

  MP3ADUinterleaver

  MP3ADUdeinterleaver

  MPEG2TransportStreamFramer

  EndianSwap16

  H263plusVideoStreamFramer

  MPEGVideoStreamFramer

  MPEG1or2VideoStreamFramer

  MPEG1or2VideoStreamDiscreteFramer

  MPEG4VideoStreamFramer

  MPEG4VideoStreamDiscreteFramer

  MPEG1or2AudioStreamFramer

  DVVideoStreamFramer

  MP3ADUTranscoder

  MPEG2TransportStreamTrickModeFilter

  MediaSink

  DummySink

  BasicUDPSink

  RTPSink

  MultiFramedRTPSink

  MPEG4GenericRTPSink

  VideoRTPSink

  H264VideoRTPSink

  MPEG1or2VideoRTPSink

  H263plusVideoRTPSink

  JPEGVideoRTPSink

  DVVideoRTPSink

  MPEG4ESVideoRTPSink

  AudioRTPSink

  AC3AudioRTPSink

  MPEG4LATMAudioRTPSink

  GSMAudioRTPSink

  MPEG1or2AudioRTPSink

  AMRAudioRTPSink

  MP3ADURTPSink

  SimpleRTPSink

  HTTPSink

  MPEG1or2VideoHTTPSink

  FileSink

  AMRAudioFileSink

  H264VideoFileSink

  RTCPInstance

  RTSPClient

  SIPClient

  DarwinInjector

  QuickTimeFileSink

  MPEG1or2Demux

  MPEG2TransportStreamIndexFile

  MPEG1or2FileServerDemux

  AVIFileSink

  BufferedPacketFactory

  QCELPBufferedPacketFactory

  AMRBufferedPacketFactory

  MPEG4GenericBufferedPacketFactory

  ADUBufferedPacketFactory

  QTGenericBufferedPacketFactory

  LATMBufferedPacketFactory

  H264BufferedPacketFactory

  JPEGBufferedPacketFactory

  BufferedPacket

  QCELPBufferedPacket

  AMRBufferedPacket

  MPEG4GenericBufferedPacket

  ADUBufferedPacket

  QTGenericBufferedPacket

  LATMBufferedPacket

  H264BufferedPacket

  JPEGBufferedPacket

  StreamParser

  AC3AudioStreamParser

  MPEGVideoStreamParser

  MPEG1or2VideoStreamParser

  MPEG4VideoStreamParser

  MPEG1or2AudioStreamParser

  H263plusVideoStreamParser

  MPEGProgramStreamParser

# 处理连接请求的基本流程

  **Step 1：**与客户端建立RTSP连接（调用incomingConnectionHandler方法），创建ClientSession并关联fClientSocket与incomingRequestHandler（调用incomingConnectionHandler1）。

  **Step 2：**接收客户端请求（调用incomingRequestHandler方法）。

  **Step 3：**从客户端Socket读取数据，并对请求数据（即the request string）进行转换（调用parseRTSPRequestString方法，该方法在RTSPCommon类中）。

  **Step 4：**根据分离出来的指令进行分别处理：

  **OPTIONS→handleCmd\_OPTIONS**

  **DESCRIBE→handleCmd\_DESCRIBE**

handleCmd\_DESCRIBE这一个方法比较重要，首先根据urlSuffix查找ServerMediaSession是否存在（调用lookupServerMediaSession方法，该方法中通过HashTable来查找）。

在testOnDemandRTSPServer项目工程中，仅仅是通过streamName来确认session是否为NULL。而在完整的live555MediaServer项目工程中，则是通过DynamicRTSPServer类来处理，其首先是查找文件是否存在，若文件不存在，则判断ServerMediaSession（即smsExists）是否存在，如果存在则将其remove（调用removeServerMediaSession方法）；若文件存在，则根据文件名创建一个ServerMediaSession（调用createNewSMS方法，若在该方法中找不到对应的文件扩展名，则将返回NULL）。

如果通过lookupServerMediaSession返回的是NULL，则向客户端发送响应消息并将fSessionIsActive置为FALSE；否则，为该session组装一个SDP描述信息（调用generateSDPDescription方法，该方法在ServerMediaSession类中），组装完成后，生成一个RTSP URL（调用rtspURL方法，该方法在RTSPServer类中）。

  **SETUP→handleCmd\_SETUP**

handleCmd\_SETUP方法中，有两个关键的名词，一个是urlPreSuffix，代表了session name（即stream name）；一个是urlSuffix，代表了subsession name（即track name），后面经常用到的streamName和trackId分别与这两个名词有关。

接下来会创建session's state，包括incrementReferenceCount等。紧接着，会针对确定的subsession（track）查找相应的信息。接着，在request string查找一个"Transport:" header，目的是为了从中提取客户端请求的一些参数（调用parseTransportHeader方法，该方法在RTSPServer类中），如clientsDestinationAddressStr、ClientRTPPortNum等。

再接着是getStreamParameters（该方法在ServerMediaSession类中被定义为纯虚函数并在OnDemandServerMediaSubsession类中被重定义），然后通过fIsMulticast和streamingMode来组装不同的响应消息。

  **PLAY→handleCmd\_PLAY：**处理播放请求，具体的实现流程请参见后面的步骤。

  **PAUSE→handleCmd\_PAUSE：**处理暂停请求，在执行了该请求后，最终会调用StopPlaying方法，并将fAreCurrentlyPlaying置为FALSE。

  **TEARDOWN→handleCmd\_TEARDOWN：**处理停止请求，将fSessionIsActive置为FALSE。

  **GET\_PARAMETER→handleCmd\_GET\_PARAMETER：**该方法主要是维持客户端与服务器通信的生存状态，just for keep alive。

  **SET\_PARAMETER→handleCmd\_SET\_PARAMETER：**该方法未针对SET\_PARAMETER作实现，使用该方法会调用handleCmd\_notSupported方法，并将最终引发与客户端断开连接。

  **Step 5：**根据Step 4的不同指令进行消息响应（调用send方法），该消息响应是即时的。

  **Step 6：**处理客户端发送“SETUP”指令后即开始播放的特殊情况。

  **Step 7：**将RequestBuffer进行重置，以便于为之后到来的请求做好准备。

  **Step 8：**检查fSessionIsActive是否为FALSE，如果是则删除当前的ClientSession。

# 处理PLAY的基本流程：

  **Step 1：**对rtspURL及相关header的处理，涉及较多的细节。

  **Step 2：**根据不同的header对流进行缩放比例或定位的处理。

如果为sawScaleHeader，则进行缩放比例的处理（调用setStreamScale方法，该方法在OnDemandServerMediaSubsession类中实现）。

如果为sawRangeHeader，则进行寻找流的处理（即是对流进行定位，调用seekStream方法，该方法在OnDemandServerMediaSubsession类中实现；同时，该方法的调用是在初始播放前及播放过程中由于用户拖动播放进度条而产生的系列请求）。

在OnDemandServerMediaSubsession类中，seekStream方法中调用了seekStreamSource方法，该方法在对应的媒体格式文件的FileServerMediaSubsession类中实现（如针对WAV格式，则在WAVAudioFileServerMediaSubsession类中实现；针对MP3格式，则在MP3AudioFileServerMediaSubsession类中实现）。

同理，OnDemandServerMediaSubsession类中的setStreamScale方法中所调用的setStreamSourceScale方法亦是类似的实现机制。

  **Step 3：**开始进行流式播放（调用startStream方法，该方法在OnDemandServerMediaSubsession类中实现）。

  **Step 3.1：**根据clientSessionId从fDestinationsHashTable中查找到destinations（包括了客户端的IP地址、RTP端口号、RTCP端口号等信息）。

  **Step 3.2：**调用startPlaying方法，在该方法中根据RTPSink或UDPSink分别调用startPlaying方法。

如果是调用RTPSink的startPlaying方法，则接着会调用MediaSink类中的startPlaying方法，并在该方法中调用MultiFramedRTPSink类中的continuePlaying方法，之后便是buildAndSendPacket了。这里已经来到重点了，即是关于不断读取Frame并Send的要点。在MultiFramedRTPSink类中，通过buildAndSendPacket、packFrame、afterGettingFrame、afterGettingFrame1、sendPacketIfNecessary和sendNext构成了一个循环圈，数据包的读取和发送在这里循环进行着。特别注意的是sendPacketIfNecessary方法中的后面代码（nextTask() = envir().taskScheduler().scheduleDelayedTask(uSecondsToGo, (TaskFunc\*)sendNext, this);），通过Delay amount of time后，继续下一个Task，并回过来继续调用buildAndSendPacket方法。

在packFrame方法中，正常情况下，需要调用getNextFrame方法（该方法在FramedSource类中，并且对不同媒体格式的Frame的获取出现在FramedSource类的getNextFrame方法中，通过调用doGetNextFrame方法来实现）来获取新的Frame。

如果是调用UDPSink的startPlaying方法，则接着会调用MediaSink类中的startPlaying方法，并在该方法中调用BasicUDPSink类中的continuePlaying方法。在这之后由若干个方法构成了一个循环圈：continuePlaying1、afterGettingFrame、afterGettingFrame1、sendNext。并在afterGettingFrame1方法中实现了packet的发送（fGS->output(envir(), fGS->ttl(), fOutputBuffer, frameSize);）。

**Step 3.3：**针对RTPSink创建RTCP instance（RTP与RTCP的配套使用决定了其必须这么做，否则可能就跟直接使用UDP发送数据包没什么两样了^\_^），创建RTCP instance时，将incomingReportHandler句柄作为BackgroundHandlerProc，以便于处理RTCP的报告，并开始startNetworkReading。这里RTP/RTCP的使用方式有两种，一种建立在TCP之上，一种建立在UDP之上。

# 穿透防火墙

在[http://www.live555.com](http://www.live555.com/)上的相关文档中提到穿透防火墙的问题，方法是开启一个HTTP的tunnel，然后我们可以在liveMedia库中找到一个RTSPOverHTTPServer的类，该类解决了这样的问题。