转 live555学习笔记－RTSP服务运作

2013年09月20日 17:13:27　阅读数：7315

RTSP服务运作

基础基本搞明白了，那么RTSP,RTP等这些协议又是如何利用这些基础机制运作的呢？
首先来看RTSP.

RTSP首先需建立TCP侦听socket。可见于此函数：

[cpp]
```cpp
DynamicRTSPServer* DynamicRTSPServer::createNew(UsageEnvironment& env, Port ourPort,
UserAuthenticationDatabase* authDatabase,
unsigned reclamationTestSeconds) {
int ourSocket = setUpOurSocket(env, ourPort); //建立TCP socket
if (ourSocket == -1)
return NULL;


return new DynamicRTSPServer(env, ourSocket, ourPort, authDatabase,
reclamationTestSeconds);
}
```

要帧听客户端的连接，就需要利用任务调度机制了，所以需添加一个socket handler。可见于此函数：

[cpp]
```cpp
RTSPServer::RTSPServer(UsageEnvironment& env,
int ourSocket,
Port ourPort,
UserAuthenticationDatabase* authDatabase,
unsigned reclamationTestSeconds) :
Medium(env),
fRTSPServerSocket(ourSocket),
fRTSPServerPort(ourPort),
fHTTPServerSocket(-1),
fHTTPServerPort(0),
fClientSessionsForHTTPTunneling(NULL),
fAuthDB(authDatabase),
fReclamationTestSeconds(reclamationTestSeconds),
fServerMediaSessions(HashTable::create(STRING_HASH_KEYS))
{
#ifdef USE_SIGNALS
// Ignore the SIGPIPE signal, so that clients on the same host that are killed
// don't also kill us:
signal(SIGPIPE, SIG_IGN);
#endif


// Arrange to handle connections from others:
env.taskScheduler().turnOnBackgroundReadHandling(
fRTSPServerSocket,
(TaskScheduler::BackgroundHandlerProc*) &incomingConnectionHandlerRTSP,
this );
}
```

当收到客户的连接时需保存下代表客户端的新socket，以后用这个socket与这个客户通讯。每个客户将来会对应一个rtp会话，而且各客户的RTSP请求只控制自己的rtp会话，那么最好建立一个会话类，代表各客户的rtsp会话。于是类RTSPServer::RTSPClientSession产生，它保存的代表客户的socket。下为RTSPClientSession的创建过程

```cpp
void RTSPServer::incomingConnectionHandler( int serverSocket)
{
struct sockaddr_in clientAddr;
SOCKLEN_T clientAddrLen = sizeof clientAddr;

//接受连接
int clientSocket = accept(serverSocket,
( struct sockaddr*) &clientAddr,
&clientAddrLen);

if (clientSocket < 0) {
int err = envir().getErrno();
if (err != EWOULDBLOCK) {
envir().setResultErrMsg( "accept() failed: " );
}
return ;
}

//设置socket的参数
makeSocketNonBlocking(clientSocket);
increaseSendBufferTo(envir(), clientSocket, 50 * 1024);

#ifdef DEBUG
envir() << "accept()ed connection from " << our_inet_ntoa(clientAddr.sin_addr) << "\n" ;
#endif

//产生一个sesson id

// Create a new object for this RTSP session.
// (Choose a random 32-bit integer for the session id (it will be encoded as a 8-digit hex number).  We don't bother checking for
//  a collision; the probability of two concurrent sessions getting the same session id is very low.)
// (We do, however, avoid choosing session id 0, because that has a special use (by "OnDemandServerMediaSubsession").)
unsigned sessionId;
do {
sessionId = (unsigned) our_random();
} while (sessionId == 0);

//创建RTSPClientSession，注意传入的参数
( void ) createNewClientSession(sessionId, clientSocket, clientAddr);
}
```

RTSPClientSession要提供什么功能呢？可以想象：需要监听客户端的rtsp请求并回应它，需要在DESCRIBE请求中返回所请求的流的信息，需要在SETUP请求中建立起RTP会话，需要在TEARDOWN请求中关闭RTP会话，等等...

RTSPClientSession要侦听客户端的请求，就需把自己的socket handler加入计划任务。证据如下：

```cpp
RTSPServer::RTSPClientSession::RTSPClientSession(
RTSPServer& ourServer,
unsigned sessionId,
int clientSocket,
struct sockaddr_in clientAddr) :
fOurServer(ourServer),
fOurSessionId(sessionId),
fOurServerMediaSession(NULL),
fClientInputSocket(clientSocket),
fClientOutputSocket(clientSocket),
fClientAddr(clientAddr),
fSessionCookie(NULL),
fLivenessCheckTask(NULL),
fIsMulticast(False),
fSessionIsActive(True),
fStreamAfterSETUP(False),
fTCPStreamIdCount(0),
fNumStreamStates(0),
fStreamStates(NULL),
fRecursionCount(0)
{
// Arrange to handle incoming requests:
resetRequestBuffer();
envir().taskScheduler().turnOnBackgroundReadHandling(fClientInputSocket,
(TaskScheduler::BackgroundHandlerProc*) &incomingRequestHandler,
this );
noteLiveness();
}
```

下面重点讲一下下RTSPClientSession响应DESCRIBE请求的过程：

[cpp]

```cpp
void RTSPServer::RTSPClientSession::handleCmd_DESCRIBE(
char const * cseq,
char const * urlPreSuffix,
char const * urlSuffix,
char const * fullRequestStr)
{
char * sdpDescription = NULL;
char * rtspURL = NULL;
do {
//整理一下下RTSP地址
char urlTotalSuffix[RTSP_PARAM_STRING_MAX];
if (strlen(urlPreSuffix) + strlen(urlSuffix) + 2
> sizeof urlTotalSuffix) {
handleCmd_bad(cseq);
break ;
}
urlTotalSuffix[0] = '\0' ;
if (urlPreSuffix[0] != '\0' ) {
strcat(urlTotalSuffix, urlPreSuffix);
strcat(urlTotalSuffix, "/" );
}
strcat(urlTotalSuffix, urlSuffix);


//验证帐户和密码
if (!authenticationOK( "DESCRIBE" , cseq, urlTotalSuffix, fullRequestStr))
break ;


// We should really check that the request contains an "Accept:" #####
// for "application/sdp", because that's what we're sending back #####


// Begin by looking up the "ServerMediaSession" object for the specified "urlTotalSuffix":
//跟据流的名字查找ServerMediaSession，如果找不到，会创建一个。每个ServerMediaSession中至少要包含一个
//ServerMediaSubsession。一个ServerMediaSession对应一个媒体，可以认为是Server上的一个文件，或一个实时获取设备。其包含的每个ServerMediaSubSession代表媒体中的一个Track。所以一个ServerMediaSession对应一个媒体，如果客户请求的媒体名相同，就使用已存在的ServerMediaSession，如果不同，就创建一个新的。一个流对应一个StreamState，StreamState与ServerMediaSubsession相关，但代表的是动态的，而ServerMediaSubsession代表静态的。
ServerMediaSession* session = fOurServer.lookupServerMediaSession(urlTotalSuffix);
if (session == NULL) {
handleCmd_notFound(cseq);
break ;
}


// Then, assemble a SDP description for this session:
//获取SDP字符串，在函数内会依次获取每个ServerMediaSubSession的字符串然连接起来。
sdpDescription = session->generateSDPDescription();
if (sdpDescription == NULL) {
// This usually means that a file name that was specified for a
// "ServerMediaSubsession" does not exist.
snprintf(( char *) fResponseBuffer, sizeof fResponseBuffer,
"RTSP/1.0 404 File Not Found, Or In Incorrect Format\r\n"
"CSeq: %s\r\n"
"%s\r\n" , cseq, dateHeader());
break ;
}
unsigned sdpDescriptionSize = strlen(sdpDescription);


// Also, generate our RTSP URL, for the "Content-Base:" header
// (which is necessary to ensure that the correct URL gets used in
// subsequent "SETUP" requests).
rtspURL = fOurServer.rtspURL(session, fClientInputSocket);


//形成响应DESCRIBE请求的RTSP字符串。
snprintf(( char *) fResponseBuffer, sizeof fResponseBuffer,
"RTSP/1.0 200 OK\r\nCSeq: %s\r\n"
"%s"
"Content-Base: %s/\r\n"
"Content-Type: application/sdp\r\n"
"Content-Length: %d\r\n\r\n"
"%s" , cseq, dateHeader(), rtspURL, sdpDescriptionSize,
sdpDescription);
} while (0);


delete [] sdpDescription;
delete [] rtspURL;


//返回后会被立即发送（没有把socket write操作放入计划任务中）。
}
```

fOurServer.lookupServerMediaSession(urlTotalSuffix)中会在找不到同名ServerMediaSession时新建一个，代表一个RTP流的ServerMediaSession们是被RTSPServer管理的，而不是被RTSPClientSession拥有。为什么呢？因为ServerMediaSession代表的是一个静态的流，也就是可以从它里面获取一个流的各种信息，但不能获取传输状态。不同客户可能连接到同一个流，所以ServerMediaSession应被RTSPServer所拥有。创建一个ServerMediaSession过程值得一观：

[cpp]

```cpp
static ServerMediaSession* createNewSMS(UsageEnvironment& env, char const * fileName, FILE * /*fid*/ )
{
// Use the file name extension to determine the type of "ServerMediaSession":
char const * extension = strrchr(fileName, '.' );
if (extension == NULL)
return NULL;


ServerMediaSession* sms = NULL;
Boolean const reuseSource = False;
if (strcmp(extension, ".aac" ) == 0) {
// Assumed to be an AAC Audio (ADTS format) file:
NEW_SMS( "AAC Audio" );
sms->addSubsession(
ADTSAudioFileServerMediaSubsession::createNew(env, fileName,
reuseSource));
} else if (strcmp(extension, ".amr" ) == 0) {
// Assumed to be an AMR Audio file:
NEW_SMS( "AMR Audio" );
sms->addSubsession(
AMRAudioFileServerMediaSubsession::createNew(env, fileName,
reuseSource));
} else if (strcmp(extension, ".ac3" ) == 0) {
// Assumed to be an AC-3 Audio file:
NEW_SMS( "AC-3 Audio" );
sms->addSubsession(
AC3AudioFileServerMediaSubsession::createNew(env, fileName,
reuseSource));
} else if (strcmp(extension, ".m4e" ) == 0) {
// Assumed to be a MPEG-4 Video Elementary Stream file:
NEW_SMS( "MPEG-4 Video" );
sms->addSubsession(
MPEG4VideoFileServerMediaSubsession::createNew(env, fileName,
reuseSource));
} else if (strcmp(extension, ".264" ) == 0) {
// Assumed to be a H.264 Video Elementary Stream file:
NEW_SMS( "H.264 Video" );
OutPacketBuffer::maxSize = 100000; // allow for some possibly large H.264 frames
sms->addSubsession(
H264VideoFileServerMediaSubsession::createNew(env, fileName,
reuseSource));
} else if (strcmp(extension, ".mp3" ) == 0) {
// Assumed to be a MPEG-1 or 2 Audio file:
NEW_SMS( "MPEG-1 or 2 Audio" );
// 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/>)
Boolean useADUs = False;
Interleaving* interleaving = NULL;
#ifdef STREAM_USING_ADUS
useADUs = True;
#ifdef INTERLEAVE_ADUS
unsigned char interleaveCycle[] = {0,2,1,3}; // or choose your own...
unsigned const interleaveCycleSize
= ( sizeof interleaveCycle)/( sizeof (unsigned char ));
interleaving = new Interleaving(interleaveCycleSize, interleaveCycle);
#endif
#endif
sms->addSubsession(
MP3AudioFileServerMediaSubsession::createNew(env, fileName,
reuseSource, useADUs, interleaving));
} else if (strcmp(extension, ".mpg" ) == 0) {
// Assumed to be a MPEG-1 or 2 Program Stream (audio+video) file:
NEW_SMS( "MPEG-1 or 2 Program Stream" );
MPEG1or2FileServerDemux* demux = MPEG1or2FileServerDemux::createNew(env,
fileName, reuseSource);
sms->addSubsession(demux->newVideoServerMediaSubsession());
sms->addSubsession(demux->newAudioServerMediaSubsession());
} else if (strcmp(extension, ".ts" ) == 0) {
// Assumed to be a MPEG Transport Stream file:
// Use an index file name that's the same as the TS file name, except with ".tsx":
unsigned indexFileNameLen = strlen(fileName) + 2; // allow for trailing "x\0"
char * indexFileName = new char [indexFileNameLen];
sprintf(indexFileName, "%sx" , fileName);
NEW_SMS( "MPEG Transport Stream" );
sms->addSubsession(
```

```
MPEG2TransportFileServerMediaSubsession::createNew(env,
fileName, indexFileName, reuseSource));
delete [] indexFileName;
} else if (strcmp(extension, ".wav" ) == 0) {
// Assumed to be a WAV Audio file:
NEW_SMS( "WAV Audio Stream" );
// To convert 16-bit PCM data to 8-bit u-law, prior to streaming,
// change the following to True:
Boolean convertToULaw = False;
sms->addSubsession(
WAVAudioFileServerMediaSubsession::createNew(env, fileName,
reuseSource, convertToULaw));
} else if (strcmp(extension, ".dv" ) == 0) {
// Assumed to be a DV Video file
// First, make sure that the RTPSinks' buffers will be large enough to handle the huge size of DV frames (as big as 288000).
OutPacketBuffer::maxSize = 300000;


NEW_SMS( "DV Video" );
sms->addSubsession(
DVVideoFileServerMediaSubsession::createNew(env, fileName,
reuseSource));
} else if (strcmp(extension, ".mkv" ) == 0) {
// Assumed to be a Matroska file
NEW_SMS( "Matroska video+audio+(optional)subtitles" );


// Create a Matroska file server demultiplexor for the specified file.  (We enter the event loop to wait for this to complete.)
newMatroskaDemuxWatchVariable = 0;
MatroskaFileServerDemux::createNew(env, fileName,
onMatroskaDemuxCreation, NULL);
env.taskScheduler().doEventLoop(&newMatroskaDemuxWatchVariable);


ServerMediaSubsession* smss;
while ((smss = demux->newServerMediaSubsession()) != NULL) {
sms->addSubsession(smss);
}
}


return sms;
}
```

可以看到NEW_SMS("AMR Audio")会创建新的ServerMediaSession，之后马上调用sms->addSubsession（）为这个ServerMediaSession添加一个 ServerMediaSubSession 。看起来ServerMediaSession应该可以添加多个ServerMediaSubSession,但这里并没有这样做。如果可以添加多个 ServerMediaSubsession 那么ServerMediaSession与流名字所指定与文件是没有关系的,也就是说它不会操作文件,而文件的操作是放在 ServerMediaSubsession中的。具体应改是在ServerMediaSubsession的sdpLines()函数中打开。

原文地址： http://blog.csdn.net/niu_gao/article/details/6911130

live555源代码（VC6）：http://download.csdn.net/detail/leixiaohua1020/6374387

文章标签： live555 rtsp 源代码

个人分类： Live555

所属专栏： 开源多媒体项目源代码分析