

## 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]
01. DynamicRTSPServer* DynamicRTSPServer::createNew(UsageEnvironment& env, Port ourPort,
02. UserAuthenticationDatabase* authDatabase,
03. unsigned reclamationTestSeconds) {
04.     int ourSocket = setUpOurSocket(env, ourPort); //建立TCP socket
05.     if (ourSocket == -1)
06.         return NULL;
07.
08.
09.     return new DynamicRTSPServer(env, ourSocket, ourPort, authDatabase,
10.         reclamationTestSeconds);
11. }
```

要侦听客户端的连接，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]
01. RTSPServer::RTSPServer(UsageEnvironment& env,
02.     int ourSocket,
03.     Port ourPort,
04.     UserAuthenticationDatabase* authDatabase,
05.     unsigned reclamationTestSeconds) :
06.     Medium(env),
07.     fRTSPServerSocket(ourSocket),
08.     fRTSPServerPort(ourPort),
09.     fHTTPServerSocket(-1),
10.     fHTTPServerPort(0),
11.     fClientSessionsForHTTP Tunneling(NULL),
12.     fAuthDB(authDatabase),
13.     fReclamationTestSeconds(reclamationTestSeconds),
14.     fServerMediaSessions(HashTable::create(STRING_HASH_KEYS))
15. {
16.     #ifndef USE_SIGNALS
17.         // Ignore the SIGPIPE signal, so that clients on the same host that are killed
18.         // don't also kill us:
19.         signal(SIGPIPE, SIG_IGN);
20.     #endif
21.
22.
23.     // Arrange to handle connections from others:
24.     env.taskScheduler().turnOnBackgroundReadHandling(
25.         fRTSPServerSocket,
26.         (TaskScheduler::BackgroundHandlerProc*) &incomingConnectionHandlerRTSP,
27.         this );
28. }
```

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

```

01. void RTSPServer::incomingConnectionHandler( int serverSocket)
02. {
03.     struct sockaddr_in clientAddr;
04.     SOCKLEN_T clientAddrLen = sizeof clientAddr;
05.
06.     //接受连接
07.     int clientSocket = accept(serverSocket,
08.     ( struct sockaddr*) &clientAddr,
09.     &clientAddrLen);
10.
11.     if (clientSocket < 0) {
12.         int err = envir().getErrno();
13.         if (err != EWOULDBLOCK) {
14.             envir().setResultErrMsg( "accept() failed: " );
15.         }
16.         return ;
17.     }
18.
19.     //设置socket的参数
20.     makeSocketNonBlocking(clientSocket);
21.     increaseSendBufferTo(envir(), clientSocket, 50 * 1024);
22.
23.     #ifdef DEBUG
24.     envir() << "accept()ed connection from " << our_inet_ntoa(clientAddr.sin_addr) << "\n" ;
25.     #endif
26.
27.     //产生一个session id
28.
29.     // Create a new object for this RTSP session.
30.     // (Choose a random 32-bit integer for the session id (it will be encoded as a 8-dig
31.     // it hex number). We don't bother checking for
32.     // a collision; the probability of two concurrent sessions getting the same session
33.     // id is very low.)
34.     // (We do, however, avoid choosing session id 0, because that has a special use (by
35.     // "OnDemandServerMediaSubsession").)
36.     unsigned sessionId;
37.     do {
38.         sessionId = (unsigned) our_random();
39.     } while (sessionId == 0);
40.
41.     //创建RTSPClientSession, 注意传入的参数
42.     ( void ) createNewClientSession(sessionId, clientSocket, clientAddr);
43. }

```

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：

```

01. RTSPServer::RTSPClientSession::RTSPClientSession(
02.     RTSPServer& ourServer,
03.     unsigned sessionId,
04.     int clientSocket,
05.     struct sockaddr_in clientAddr) :
06.     fOurServer(ourServer),
07.     fOurSessionId(sessionId),
08.     fOurServerMediaSession(NULL),
09.     fClientInputSocket(clientSocket),
10.     fClientOutputSocket(clientSocket),
11.     fClientAddr(clientAddr),
12.     fSessionCookie(NULL),
13.     fLivenessCheckTask(NULL),
14.     fIsMulticast(False),
15.     fSessionIsActive(True),
16.     fStreamAfterSETUP(False),
17.     fTCPStreamIdCount(0),
18.     fNumStreamStates(0),
19.     fStreamStates(NULL),
20.     fRecursionCount(0)
21. {
22.     // Arrange to handle incoming requests:
23.     resetRequestBuffer();
24.     envir().taskScheduler().turnOnBackgroundReadHandling(fClientInputSocket,
25.     (TaskScheduler::BackgroundHandlerProc*) &incomingRequestHandler,
26.     this );
27.     noteLiveness();
28. }

```

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

```

01. void RTSPServer::RTSPClientSession::handleCmd_DESCRIBE(
02.     char const * cseq,
03.     char const * urlPreSuffix,
04.     char const * urlSuffix,
05.     char const * fullRequestStr)
06. {
07.     char * sdpDescription = NULL;
08.     char * rtspURL = NULL;
09.     do {
10.         //整理一下RTSP地址
11.         char urlTotalSuffix[RTSP_PARAM_STRING_MAX];
12.         if (strlen(urlPreSuffix) + strlen(urlSuffix) + 2
13.             > sizeof urlTotalSuffix) {
14.             handleCmd_bad(cseq);
15.             break ;
16.         }
17.         urlTotalSuffix[0] = '\0' ;
18.         if (urlPreSuffix[0] != '\0' ) {
19.             strcat(urlTotalSuffix, urlPreSuffix);
20.             strcat(urlTotalSuffix, "/" );
21.         }
22.         strcat(urlTotalSuffix, urlSuffix);
23.
24.
25.         //验证帐户和密码
26.         if (!authenticationOK( "DESCRIBE" , cseq, urlTotalSuffix, fullRequestStr))
27.             break ;
28.
29.
30.         // We should really check that the request contains an "Accept:" #####
31.         // for "application/sdp", because that's what we're sending back #####
32.
33.
34.         // Begin by looking up the "ServerMediaSession" object for the specified "urlTotalSu
35.         // ffix":
36.         //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
37.         // 含一个
38.         //ServerMediaSubsession。一个ServerMediaSession对应一个媒体, 可以认为是Server上的一个文件,
39.         // 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ServerMedi
40.         // aSession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
41.         // 建一个新的。一个流对应一个StreamState, StreamState与ServerMediaSubsession相关, 但代表的是动
42.         // 态的, 而ServerMediaSubsession代表静态的。
43.         ServerMediaSession* session = fOurServer.lookupServerMediaSession(urlTotalSuffix);
44.         if (session == NULL) {
45.             handleCmd_notFound(cseq);
46.             break ;
47.         }
48.
49.
50.         // Then, assemble a SDP description for this session:
51.         //获取SDP字符串, 在函数内会依次获取每个ServerMediaSubSession的字符串然后连接起来。
52.         sdpDescription = session->generateSDPDescription();
53.         if (sdpDescription == NULL) {
54.             // This usually means that a file name that was specified for a
55.             // "ServerMediaSubsession" does not exist.
56.             snprintf(( char *) fResponseBuffer, sizeof fResponseBuffer,
57.                 "RTSP/1.0 404 File Not Found, Or In Incorrect Format\r\n"
58.                 "CSeq: %s\r\n"
59.                 "%s\r\n" , cseq, dateHeader());
60.             break ;
61.         }
62.         unsigned sdpDescriptionSize = strlen(sdpDescription);
63.
64.
65.         // Also, generate our RTSP URL, for the "Content-Base:" header
66.         // (which is necessary to ensure that the correct URL gets used in
67.         // subsequent "SETUP" requests).
68.         rtspURL = fOurServer.rtspURL(session, fClientInputSocket);
69.
70.
71.         //形成响应DESCRIBE请求的RTSP字符串。
72.         snprintf(( char *) fResponseBuffer, sizeof fResponseBuffer,
73.             "RTSP/1.0 200 OK\r\nCSeq: %s\r\n"
74.             "%s"
75.             "Content-Base: %s\r\n"
76.             "Content-Type: application/sdp\r\n"
77.             "Content-Length: %d\r\n\r\n"
78.             "%s" , cseq, dateHeader(), rtspURL, sdpDescriptionSize,
79.             sdpDescription);
80.         } while (0);
81.
82.         delete [] sdpDescription;
83.         delete [] rtspURL;
84.
85.
86.         //返回后会被立即发送 (没有把socket write操作放入计划任务中)。
87.     }

```

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]
01. static ServerMediaSession* createNewSMS(UsageEnvironment& env, char const * fileName
    , FILE * /*fid*/ )
02. {
03.     // Use the file name extension to determine the type of "ServerMediaSession":
04.     char const * extension = strrchr(fileName, '.');
05.     if (extension == NULL)
06.         return NULL;
07.
08.
09.     ServerMediaSession* sms = NULL;
10.     Boolean const reuseSource = False;
11.     if (strcmp(extension, ".aac" ) == 0) {
12.         // Assumed to be an AAC Audio (ADTS format) file:
13.         NEW_SMS( "AAC Audio" );
14.         sms->addSubsession(
15.             ADTSAudioFileServerMediaSubsession::createNew(env, fileName,
16.                 reuseSource));
17.     } else if (strcmp(extension, ".amr" ) == 0) {
18.         // Assumed to be an AMR Audio file:
19.         NEW_SMS( "AMR Audio" );
20.         sms->addSubsession(
21.             AMRAudioFileServerMediaSubsession::createNew(env, fileName,
22.                 reuseSource));
23.     } else if (strcmp(extension, ".ac3" ) == 0) {
24.         // Assumed to be an AC-3 Audio file:
25.         NEW_SMS( "AC-3 Audio" );
26.         sms->addSubsession(
27.             AC3AudioFileServerMediaSubsession::createNew(env, fileName,
28.                 reuseSource));
29.     } else if (strcmp(extension, ".m4e" ) == 0) {
30.         // Assumed to be a MPEG-4 Video Elementary Stream file:
31.         NEW_SMS( "MPEG-4 Video" );
32.         sms->addSubsession(
33.             MPEG4VideoFileServerMediaSubsession::createNew(env, fileName,
34.                 reuseSource));
35.     } else if (strcmp(extension, ".264" ) == 0) {
36.         // Assumed to be a H.264 Video Elementary Stream file:
37.         NEW_SMS( "H.264 Video" );
38.         OutPacketBuffer::maxSize = 100000; // allow for some possibly large H.264 frames
39.         sms->addSubsession(
40.             H264VideoFileServerMediaSubsession::createNew(env, fileName,
41.                 reuseSource));
42.     } else if (strcmp(extension, ".mp3" ) == 0) {
43.         // Assumed to be a MPEG-1 or 2 Audio file:
44.         NEW_SMS( "MPEG-1 or 2 Audio" );
45.         // To stream using 'ADUs' rather than raw MP3 frames, uncomment the following:
46.         // #define STREAM_USING_ADUS 1
47.         // To also reorder ADUs before streaming, uncomment the following:
48.         // #define INTERLEAVE_ADUS 1
49.         // (For more information about ADUs and interleaving,
50.         //  see <http://www.live555.com/rtp-mp3/>)
51.         Boolean useADUs = False;
52.         Interleaving* interleaving = NULL;
53.         #ifndef STREAM_USING_ADUS
54.             useADUs = True;
55.             #ifndef INTERLEAVE_ADUS
56.                 unsigned char interleaveCycle[] = {0,2,1,3}; // or choose your own...
57.                 unsigned const interleaveCycleSize
58.                     = ( sizeof interleaveCycle ) / ( sizeof unsigned char );
59.                 interleaving = new Interleaving(interleaveCycleSize, interleaveCycle);
60.             #endif
61.         #endif
62.         sms->addSubsession(
63.             MP3AudioFileServerMediaSubsession::createNew(env, fileName,
64.                 reuseSource, useADUs, interleaving));
65.     } else if (strcmp(extension, ".mpg" ) == 0) {
66.         // Assumed to be a MPEG-1 or 2 Program Stream (audio+video) file:
67.         NEW_SMS( "MPEG-1 or 2 Program Stream" );
68.         MPEG1or2FileServerDemux* demux = MPEG1or2FileServerDemux::createNew(env,
69.             fileName, reuseSource);
70.         sms->addSubsession(demux->newVideoServerMediaSubsession());
71.         sms->addSubsession(demux->newAudioServerMediaSubsession());
72.     } else if (strcmp(extension, ".ts" ) == 0) {
73.         // Assumed to be a MPEG Transport Stream file:
74.         // Use an index file name that's the same as the TS file name, except with ".tsx":
75.         unsigned indexFileNameLen = strlen(fileName) + 2; // allow for trailing "x\0"
76.         char * indexFileName = new char [indexFileNameLen];
77.         sprintf(indexFileName, "%sx" , fileName);
78.         NEW_SMS( "MPEG Transport Stream" );
79.         sms->addSubsession(
```

```

80. MPEG2TransportFileServerMediaSubsession::createNew(env,
81. fileName, indexFileName, reuseSource));
82. delete [] indexFileName;
83. } else if (strcmp(extension, ".wav" ) == 0) {
84. // Assumed to be a WAV Audio file:
85. NEW_SMS( "WAV Audio Stream" );
86. // To convert 16-bit PCM data to 8-bit u-law, prior to streaming,
87. // change the following to True:
88. Boolean convertToULaw = False;
89. sms->addSubsession(
90. WVAudioFileServerMediaSubsession::createNew(env, fileName,
91. reuseSource, convertToULaw));
92. } else if (strcmp(extension, ".dv" ) == 0) {
93. // Assumed to be a DV Video file
94. // First, make sure that the RTPSinks' buffers will be large enough to handle the hu
95. ge size of DV frames (as big as 288000).
96. OutPacketBuffer::maxSize = 300000;
97.
98. NEW_SMS( "DV Video" );
99. sms->addSubsession(
100. DVVideoFileServerMediaSubsession::createNew(env, fileName,
101. reuseSource));
102. } else if (strcmp(extension, ".mkv" ) == 0) {
103. // Assumed to be a Matroska file
104. NEW_SMS( "Matroska video+audio+(optional)subtitles" );
105.
106.
107. // Create a Matroska file server demultiplexor for the specified file. (We enter th
108. e event loop to wait for this to complete.)
109. newMatroskaDemuxWatchVariable = 0;
110. MatroskaFileServerDemux::createNew(env, fileName,
111. onMatroskaDemuxCreation, NULL);
112. env.taskScheduler().doEventLoop(&newMatroskaDemuxWatchVariable);
113.
114. ServerMediaSubsession* smss;
115. while ((smss = demux->newServerMediaSubsession()) != NULL) {
116. sms->addSubsession(smss);
117. }
118. }
119.
120.
121. return sms;
122. }

```

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](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](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com