

3-SRS 4.0 RTMP推拉流转发原理

0 SRS 4.0流媒体服务器入门系列

1 RTMP端口监听逻辑

1.1 listen监听逻辑

1.2 accept有新socket到来

`::accept` 断点

`SrsServer::accept_client`断点

1.3 new SrsRtmpConn新建一个连接对象

2 RTMP推流、拉流怎么创建连接

2.1 推流

2.1.1 判断是不是推流SrsRtmpServer::identify_client

2.1.2 进入推流循环SrsRtmpConn::do_publishing

2.2 拉流

2.2.1 判断是不是推流SrsRtmpServer::identify_client

2.2.2 进入拉流循环SrsRtmpConn::do_playing

3 服务器怎么读取RTMP推流数据

3.1 SrsSource::on_video_imp调用栈

3.2 分发数据SrsConsumer::enqueue

4 服务器怎么给RTMP拉流端转发数据

5 哪些配置文件会影响RTMP的延迟

5.1 推流相关的延迟

`SrsConfig::get_mr_enabled`和`get_mr_sleep`

`SrsPublishRecvThread::on_start`

`SrsPublishRecvThread::on_stop`

`SrsPublishRecvThread::on_read`

5.2 拉流相关的延迟

`SrsConfig::get_mw_sleep`

`SrsConfig::get_mw_msgs`

`SrsConfig::get_queue_length`

go cacache延迟

SrsGopCache::cache保存Gop

SrsGopCache::dump 拷贝Gop

6 总结

版权归零声学院所有，侵权必究

音视频高级教程 – Darren老师：QQ326873713

课程链接：<https://ke.qq.com/course/468797?tuin=137bb271>

网页版本课件：<https://www.yuque.com/docs/share/386eddf9-201e-4375-ad30-44cd72de3bbb?#> 《3-SRS 4.0 RTMP推拉流转发原理》

0 SRS 4.0流媒体服务器入门系列

结合SRS官方Wiki以及本人对SRS的理解，推出《SRS 4.0流媒体服务器入门系列》，包括内容：

1. [SRS 4.0 开发环境搭建](#)
2. [SRS 4.0 配置支持WebRTC推拉流](#)
3. [SRS 4.0 RTMP推拉流转发原理，包括延迟分析](#)
4. SRS 4.0 支持WebRTC一对一通话，包括信令原理讲解(待录制)
5. SRS 4.0 支持WebRTC多人通话，包括信令原理讲解(待录制)
6. SRS 4.0 RTMP to WebRTC原理分析(待录制)
7. SRS 4.0 WebRTC to RTMP 原理分析(待录制)
8. SRS 4.0 配置支持GB28181推流(待录制)
9.其他待规划

每篇文章配合对应的视频，[本文档配套视频：3-SRS 4.0 RTMP推拉流转发原理](#)，大家可以扫码关注“音视频流媒体技术”公众号，获取后续的更新。

云服务器：阿里云Ubuntu 16.04

服务器：SRS(Simple Realtime Server，支持RTMP、HTTP-FLV、HLS、WebRTC)

推流端：ffmpeg + OBS

拉流端：ffplay +VLC + srs播放器

SRS官网: <http://www.ossrs.net/>

微信官方公众号 srs-server



SRS开源服务器

微信扫描二维码, 关注我的公众号

这一节主要讲述srs推拉转发原理以及和配置文件的关系, 重点内容:

- RTMP端口监听逻辑
- RTMP推流怎么创建连接
- RTMP拉流怎么创建连接
- 服务器怎么读取RTMP推流数据
- 服务器怎么给RTMP拉流端转发数据
- 哪些配置文件会影响RTMP的延迟

本文不纠结细节, 只是理清整个RTMP推拉流脉络以及延迟的问题, 并且不能单靠本文理解, 还要结合对应的视频去理解。

我们主要通过gdb调试的方式分析RTMP推拉流逻辑, 为了方便debug, 需要修改配置srs.conf配置文件

```
listen          1935;
max_connections 1000;
srs_log_tank     console;
#srs_log_file    ./objs/srs.log;
daemon          off;
```

PS: 不只是SRS使用gdb调试分析源码有效, 在分析nginx、redis源码时该方法同样有效。

1 RTMP端口监听逻辑

RTMP基于TCP连接, 必然存在:

- listen监听端口

- accept接收一个套接字中已建立的连接
- 然后new一个RTMP连接对象关联accept得到的fd。

我们可以从该逻辑出发探索SRS 是如何监听和创建RTMP连接。

1.1 listen监听逻辑

通过在配置文件对应的接口打断点分析SRS代码逻辑屡试不爽，比如我们知道RTMP监听端口对应配置文件的配置项为：

```
main config for srs.
# @see full.conf for detail config.
listen 1935;
max_connections 1000;
```

，我们可以在srs_app_config.cpp(所有的配置文件读取解析都在该文件)

文件的vector<string> SrsConfig::get_listens()接口打断点。

最终发现RTMP监听是通过SrsTcpListener::listen()调用srs_tcp_listen，SrsTcpListener类并创建Coroutine处理accept。

可以根据该调用栈去分析源码：

```
#0 SrsConfig::get_listens[abi:cxx11]() (this=0xf5d3a0) at src/app/srs_app_config.cpp:4124#1
0x00000000004f78de in SrsServer::listen_rtmp (this=0xf5daa0) at
src/app/srs_app_server.cpp:1326
#2 0x00000000004f4681 in SrsServer::listen (this=0xf5daa0) at
src/app/srs_app_server.cpp:906
#3 0x000000000061008e in SrsServerAdapter::run (this=0xf5de30) at
src/app/srs_app_hybrid.cpp:177
#4 0x0000000000610828 in SrsHybridServer::run (this=0xf592e0) at
src/app/srs_app_hybrid.cpp:279
#5 0x00000000006c246b in run_hybrid_server () at src/main/srs_main_server.cpp:483
#6 0x00000000006c1ff0 in run_directly_or_daemon () at src/main/srs_main_server.cpp:418
#7 0x00000000006c096c in do_main (argc=3, argv=0x7fffffffe4e8) at
src/main/srs_main_server.cpp:210
#8 0x00000000006c0b4e in main (argc=3, argv=0x7fffffffe4e8) at
src/main/srs_main_server.cpp:221
```

进一步阅读SrsServer::listen_rtmp()函数。

```

1321: srs_error_t SrsServer::listen_rtmp()
1322: {
1323:     srs_error_t err = srs_success;
1324:
1325:     // stream service port.
1326:     std::vector<std::string> ip_ports = _srs_config->get_listens(); 读取监听端
1327:     srs_assert((int)ip_ports.size() > 0); 口
1328:
1329:     close_listeners(SrsListenerRtmpStream);
1330:
1331:     for (int i = 0; i < (int)ip_ports.size(); i++) {
1332:         SrsListener* listener = new SrsBufferListener(this, SrsListenerRtmpStream);
1333:         listeners.push_back(listener); 监听的业务类
1334:
1335:         int port; string ip;
1336:         srs_parse_endpoint(ip_ports[i], ip, port);
1337:
1338:         if ((err = listener->listen(ip, port)) != srs_success) {
1339:             srs_error_wrap(err, "rtmp listen %s:%d", ip.c_str(), port);
1340:         }
1341:     }
1342:
1343:     return err;
1344: } « end listen_rtmp »

```

1.2 accept有新socket到来

每个监听都会绑定一个Coroutine处理accept，当有新socket连接到来，则触发accept，通过一层层回调调用到SrsServer::accept_client。

::accept 断点

```

#0  accept () at ../sysdeps/unix/syscall-template.S:84#1 0x00000000006c5068 in st_accept
(fd=0xf881e0, addr=0x0, addrlen=0x0, timeout=18446744073709551615) at io.c:282
#2  0x00000000004da66c in srs_accept (stfd=0xf881e0, addr=0x0, addrlen=0x0, timeout=-1)
at src/protocol/srs_service_st.cpp:436
#3  0x00000000005eeacf in SrsTcpListener::cycle (this=0xf8c6b0) at
src/app/srs_app_listener.cpp:278
#4  0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0xf8c790) at
src/app/srs_app_st.cpp:270
#5  0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0xf8c790) at
src/app/srs_app_st.cpp:285
#6  0x00000000006c311d in _st_thread_main () at sched.c:363
#7  0x00000000006c399a in st_thread_create (start=0x7fff6f07b78 <main_arena+88>,
arg=0x7fff6f07b78 <main_arena+88>,
joinable=0, stk_size=9998) at sched.c:694

```

可以分析到是在 SrsTcpListener::cycle进行accept。

```

269: srs_error_t SrsTcpListener::cycle()
270: {
271:     srs_error_t err = srs_success;
272:
273:     while (true) {
274:         if ((err = trd->pull()) != srs_success) {
275:             return srs_error_wrap(err, "tcp listener");
276:         }
277:
278:         srs_netfd_t fd = srs_accept(lfd, NULL, NULL, SRS_ETIME_NO_TIMEOUT); 等待新的fd到来
279:         if (fd == NULL) {
280:             return srs_error_new(ERROR_SOCKET_ACCEPT, "accept at fd=%d", srs_netfd_fileno(lfd));
281:         }
282:
283:         if ((err = srs_fd_closeexec(srs_netfd_fileno(fd))) != srs_success) {
284:             return srs_error_wrap(err, "set closeexec");
285:         }
286:
287:         if ((err = handler->on_tcp_client(fd)) != srs_success) { 回调对应的业务处理函数
288:             return srs_error_wrap(err, "handle fd=%d", srs_netfd_fileno(fd));
289:         }
290:     }
291:
292:     return err;
293: } « end cycle »
294:

```

我们可以在srs_app_listener.cpp:287行打上断点，当有RTMP客户端请求连接时触发该断点，然后进入handler->on_tcp_client。

最终RTMP业务回调到SrsServer::accept_client函数，这里我就不绕弯子，直接在SrsServer::accept_client也断点。

SrsServer::accept_client断点

可以从以下堆栈显然得出对于RTMP业务，最终是：

SrsTcpListener::cycle -> SrsBufferListener::on_tcp_client -> SrsServer::accept_client

```

#0  SrsServer::accept_client (this=0xf5daa0, type=SrsListenerRtmpStream, stfd=0x1043540)
   at src/app/srs_app_server.cpp:1574
#1  0x00000000004ee490 in SrsBufferListener::on_tcp_client (this=0xf8bc90, stfd=0x1043540)
   at src/app/srs_app_server.cpp:134
#2  0x00000000005eeb87 in SrsTcpListener::cycle (this=0xf8bec0) at
src/app/srs_app_listener.cpp:287
#3  0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0xf8bfa0) at
src/app/srs_app_st.cpp:270
#4  0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0xf8bfa0) at
src/app/srs_app_st.cpp:285
#5  0x00000000006c311d in _st_thread_main () at sched.c:363
#6  0x00000000006c399a in st_thread_create (start=0x7ffff6f07b78 <main_arena+88>,
   arg=0x7ffff6f07b78 <main_arena+88>, joinable=0, stk_size=9998) at sched.c:694

```

1.3 new SrsRtmpConn新建一个连接对象

不管RTMP是推流还是拉流，每个连接在服务器都对应一个SrsRtmpConn对象。SrsRtmpConn持有连接对应的fd。在SrsRtmpConn::SrsRtmpConn(SrsServer* svr, srs_netfd_t c, string cip, int cport)构造函数断点。

堆栈如下所示：

```
#0 SrsRtmpConn::SrsRtmpConn (this=0x1043570, svr=0xf5daa0, c=0x1043540,
  cip="\340\313\371\000\000\000\000\000\016\000\000\000\000\000\000\062\062\0
60.202.216.45\000\000\220+\004\001\000\000\000\000\060+\004\001\000\000\000\000\
220+\004\001\000\000\000\000\000\000\257\337Xu\347\377A\001", '\000' <repeats 15 times>,
"\200\314\371\000\000\000\000\000\206\206M\000\000\000\000\000\260\314\371\000\
000\000\000\200\315\371\000\000\000\000\000\000p\323\365\000\000\000\000\000h\3
23\365", '\000' <repeats 13 times>, "(", '\000' <repeats 15 times>,
"\060+\004\001\000\000\000\000\000\030\214\234\000\000\000\000\000\210\314\371\000\0
00\000\000\000\b\000\000\000\000\000\000\000b6b8j39u\000\000\000\000\000\000\00
0\000"... , cport=17852, __in_chrg=<optimized out>,
  __vtt_parm=<optimized out>) at src/app/srs_app_rtmp_conn.cpp:108
#1 0x00000000004f9972 in SrsServer::fd_to_resource (this=0xf5daa0,
  type=SrsListenerRtmpStream, stfd=0x1043540,
  pr=0xf9cd58) at src/app/srs_app_server.cpp:1646
#2 0x00000000004f937f in SrsServer::accept_client (this=0xf5daa0,
  type=SrsListenerRtmpStream, stfd=0x1043540)
  at src/app/srs_app_server.cpp:1579
#3 0x00000000004ee490 in SrsBufferListener::on_tcp_client (this=0xf8bc90, stfd=0x1043540)
  at src/app/srs_app_server.cpp:134
#4 0x00000000005eeb87 in SrsTcpListener::cycle (this=0xf8bec0) at
src/app/srs_app_listener.cpp:287
#5 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0xf8bfa0) at
src/app/srs_app_st.cpp:270
#6 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0xf8bfa0) at
src/app/srs_app_st.cpp:285
#7 0x00000000006c311d in _st_thread_main () at sched.c:363
#8 0x00000000006c399a in st_thread_create (start=0x7ffff6f07b78 <main_arena+88>,
  arg=0x7ffff6f07b78 <main_arena+88>, joinable=0, stk_size=9998) at sched.c:694
```

2 RTMP推流、拉流怎么创建连接

RTMP推流和拉流创建连接对象的方式是一致的，都是创建了SrsRtmpConn。

```
1603: srs_error_t SrsServer::fd_to_resource(SrsListenerType type, srs_netfd_t stfd, ISrsStartableConneciton** pr)
1645: {
1646:     if (type == SrsListenerRtmpStream) { 创建对RTMP连接对象
1647:         *pr = new SrsRtmpConn(this, stfd, ip, port);
1648:     } else if (type == SrsListenerHttpApi) {
1649:         *pr = new SrsHttpApi(false, this, stfd, http_api_mux, ip, port);
1650:     } else if (type == SrsListenerHttpsApi) {
1651:         *pr = new SrsHttpApi(true, this, stfd, http_api_mux, ip, port);
1652:     } else if (type == SrsListenerHttpStream) {
1653:         *pr = new SrsResponseOnlyHttpConn(false, this, stfd, http_server, ip, port);
1654:     } else if (type == SrsListenerHttpsStream) {
1655:         *pr = new SrsResponseOnlyHttpConn(true, this, stfd, http_server, ip, port);
1656:     } else {
1657:         srs_warn("close for no service handler. fd=%d, ip=%s:%d", fd, ip.c_str(), port);
1658:         srs_close_stfd(stfd);
1659:         return err;
1660:     }
}
```

每个SrsRtmpConn都绑定一个SrsCoroutine，具体的业务处理在SrsCoroutine的循环进行，对于RTMP而言最终的循环为SrsRtmpConn::cycle。

对于推流和拉流都是在SrsRtmpConn::cycle，只是最终：

- 推流SrsRtmpConn::do_publishing
- 拉流SrsRtmpConn::do_playing

具体是在SrsRtmpConn::stream_service_cycle根据RTMP交换协议决定的流程走向。

要启动协程，需要调用：[SrsRtmpConn::start\(\)](#)

PS：熟悉协程的原理，对于理解SRS源码非常有帮助。

2.1 推流

2.1.1 判断是不是推流SrsRtmpServer::identify_client

在SrsRtmpConn::stream_service_cycle调用SrsRtmpServer::identify_client判断是不是推流。从这里也可以看出来SrsRtmpConn不会直接和socket数据打交道。

Breakpoint 8, SrsRtmpServer::identify_fmle_publish_client (this=0x104d3c0, req=0x10efbc0, type=@0x11059c8: SrsRtmpConnUnknown, stream_name="") at

src/protocol/srs_rtmp_stack.cpp:3009

```
3009         srs_error_t err = srs_success;
```

(gdb) bt

```
#0  SrsRtmpServer::identify_fmle_publish_client (this=0x104d3c0, req=0x10efbc0, type=@0x11059c8: SrsRtmpConnUnknown, stream_name="") at
```

src/protocol/srs_rtmp_stack.cpp:3009


```

#1 0x0000000000493078 in SrsRtmpServer::identify_client (this=0x104d3c0, stream_id=1,
    type=@0x11059c8: SrsRtmpConnUnknown, stream_name="", duration=@0x10efba8: -1)
    at src/protocol/srs_rtmp_stack.cpp:2579
#2 0x000000000050c6f5 in SrsRtmpConn::stream_service_cycle (this=0x104ccc0) at
src/app/srs_app_rtmp_conn.cpp:472
#3 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x104ccc0) at
src/app/srs_app_rtmp_conn.cpp:420
#4 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x104ccc0) at
src/app/srs_app_rtmp_conn.cpp:233
#5 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x104ccc0) at
src/app/srs_app_rtmp_conn.cpp:1474
#6 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10cbb30) at
src/app/srs_app_st.cpp:270
#7 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10cbb30) at
src/app/srs_app_st.cpp:285
#8 0x00000000006c311d in _st_thread_main () at sched.c:363
#9 0x00000000006c399a in st_thread_create (start=0x542b2c
<SrsFastCoroutine::pfn(void*)>, arg=0x10cbb30,
    joinable=1, stk_size=65536) at sched.c:694

```

调用[SrsProtocol::decode_message\(SrsCommonMessage* msg, SrsPacket** ppacket\)](#) 去判断对应数据是属于哪一种类型消息。这里具体的细节需要去阅读RTMP手册。

2.1.2 进入推流循环SrsRtmpConn::do_publishing

SrsRtmpConn::do_publishing 不实际读取推流过来的数据，具体读取推流数据是通过[SrsPublishRecvThread](#)类，我们在《3 怎么读取RTMP推流数据》小节继续讲解。

```

Breakpoint 5, SrsRtmpConn::do_publishing (this=0x105f740, source=0x104bdc0,
rtrd=0x1072480)  at src/app/srs_app_rtmp_conn.cpp:879
879 {
(gdb) bt
#0 SrsRtmpConn::do_publishing (this=0x105f740, source=0x104bdc0, rtrd=0x1072480)
    at src/app/srs_app_rtmp_conn.cpp:879
#1 0x00000000005100d8 in SrsRtmpConn::publishing (this=0x105f740, source=0x104bdc0)
    at src/app/srs_app_rtmp_conn.cpp:860
#2 0x000000000050d5ac in SrsRtmpConn::stream_service_cycle (this=0x105f740) at
src/app/srs_app_rtmp_conn.cpp:566

```

```

#3 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x105f740) at
src/app/srs_app_rtmp_conn.cpp:420
#4 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x105f740) at
src/app/srs_app_rtmp_conn.cpp:233
#5 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x105f740) at
src/app/srs_app_rtmp_conn.cpp:1474
#6 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x1060020) at
src/app/srs_app_st.cpp:270
#7 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x1060020) at
src/app/srs_app_st.cpp:285
#8 0x00000000006c311d in _st_thread_main () at sched.c:363
#9 0x00000000006c399a in st_thread_create (start=0x542b2c
<SrsFastCoroutine::pfn(void*)>, arg=0x1060020,
joinable=1, stk_size=65536) at sched.c:694

```

2.2 拉流

2.2.1 判断是不是推流SrsRtmpServer::identify_client

和RTMP推流判断同理，请参考《2.1.1 判断是不是推流SrsRtmpServer::identify_client》

2.2.2 进入拉流循环SrsRtmpConn::do_playing

```

Breakpoint 16, SrsRtmpConn::do_playing (this=0x11011c0, source=0x1044580,
consumer=0x1045de0, rtrd=0x113f0f0) at src/app/srs_app_rtmp_conn.cpp:713
713 {
(gdb) bt
#0 SrsRtmpConn::do_playing (this=0x11011c0, source=0x1044580, consumer=0x1045de0,
rtrd=0x113f0f0)
    at src/app/srs_app_rtmp_conn.cpp:713
#1 0x000000000050ebc2 in SrsRtmpConn::playing (this=0x11011c0, source=0x1044580)
    at src/app/srs_app_rtmp_conn.cpp:699
#2 0x000000000050d4d2 in SrsRtmpConn::stream_service_cycle (this=0x11011c0)
    at src/app/srs_app_rtmp_conn.cpp:556
#3 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x11011c0) at
src/app/srs_app_rtmp_conn.cpp:420
#4 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x11011c0) at
src/app/srs_app_rtmp_conn.cpp:233

```

```
#5 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x11011c0) at
src/app/srs_app_rtmp_conn.cpp:1474
#6 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x1102780) at
src/app/srs_app_st.cpp:270
#7 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x1102780) at
src/app/srs_app_st.cpp:285
#8 0x00000000006c311d in _st_thread_main () at sched.c:363
#9 0x00000000006c399a in st_thread_create (start=0x542b2c
<SrsFastCoroutine::pfn(void*)>, arg=0x1102780,
joinable=1, stk_size=65536) at sched.c:694
```

服务器怎么把推流的音视频消息转发给拉流端，在《4 服务器怎么给RTMP拉流端转发数据》小节继续讲解。

3 服务器怎么读取RTMP推流数据

通过**SrsPublishRecvThread**读取推流数据。通过SrsSource和SrsConsumer的关系转发RTMP音视频数据。

推流数据转发给拉流

- SrsSource::on_video_imp(SrsSharedPtrMessage* msg)
- SrsSource::on_audio_imp(SrsSharedPtrMessage* msg)

拉流通过SrsConsumer::enqueue缓存音视频数据

3.1 SrsSource::on_video_imp调用栈

```
#0 SrsSource::on_video_imp (this=0x105fb30, msg=0x7ffff7fc5a70) at
src/app/srs_app_source.cpp:2333#1 0x000000000051fb11 in SrsSource::on_video
(this=0x105fb30, shared_video=0x10aac50)
at src/app/srs_app_source.cpp:2309
#2 0x0000000000511798 in SrsRtmpConn::process_publish_message (this=0x1043570,
source=0x105fb30,
msg=0x10aac50) at src/app/srs_app_rtmp_conn.cpp:1079
#3 0x0000000000511598 in SrsRtmpConn::handle_publish_message (this=0x1043570,
source=0x105fb30,
msg=0x10aac50) at src/app/srs_app_rtmp_conn.cpp:1051
#4 0x00000000005d364c in SrsPublishRecvThread::consume (this=0x105dbc0,
msg=0x10aac50)
at src/app/srs_app_recv_thread.cpp:396
```

```
#5 0x00000000005d1e35 in SrsRecvThread::do_cycle (this=0x105dbd0) at
src/app/srs_app_recv_thread.cpp:150
#6 0x00000000005d1c92 in SrsRecvThread::cycle (this=0x105dbd0) at
src/app/srs_app_recv_thread.cpp:119
#7 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10aa9b0) at
src/app/srs_app_st.cpp:270
#8 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10aa9b0) at
src/app/srs_app_st.cpp:285
#9 0x00000000006c311d in _st_thread_main () at sched.c:363
#10 0x00000000006c399a in st_thread_create (start=0x10aa990, arg=0x10aa990, joinable=0,
stk_size=5849344)
    at sched.c:694
```

3.2 分发数据SrsConsumer::enqueue

通过调用SrsConsumer::enqueue，给每个拉流端对应的SrsConsumer queue发送音视频消息。比如这里拉流客户端刚连接时，先把SrsSource对应的metadata以及Gop cache发送给该客户端对应的SrsConsumer queue。

Breakpoint 4, SrsConsumer::enqueue (this=0x10bda70, shared_msg=0x10aaf40, atc=false, ag=SrsRtmpJitterAlgorithmFULL) at src/app/srs_app_source.cpp:465

```
465     srs_error_t err = srs_success;
```

(gdb) bt

```
#0 SrsConsumer::enqueue (this=0x10bda70, shared_msg=0x10aaf40, atc=false,
ag=SrsRtmpJitterAlgorithmFULL)
```

```
    at src/app/srs_app_source.cpp:465
```

```
#1 0x000000000051b77a in SrsMetaCache::dumps (this=0x1060590, consumer=0x10bda70,
atc=false,
```

```
    ag=SrsRtmpJitterAlgorithmFULL, dm=true, ds=true) at src/app/srs_app_source.cpp:1569
```

```
#2 0x0000000000521465 in SrsSource::consumer_dumps (this=0x105fb30,
consumer=0x10bda70, ds=true, dm=true,
```

```
    dg=true) at src/app/srs_app_source.cpp:2620
```

```
#3 0x000000000050ea4c in SrsRtmpConn::playing (this=0x10db880, source=0x105fb30)
    at src/app/srs_app_rtmp_conn.cpp:685
```

```
#4 0x000000000050d4d2 in SrsRtmpConn::stream_service_cycle (this=0x10db880)
    at src/app/srs_app_rtmp_conn.cpp:556
```

```
#5 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x10db880) at
src/app/srs_app_rtmp_conn.cpp:420
```

```
#6 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x10db880) at
src/app/srs_app_rtmp_conn.cpp:233
```

```
#7 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x10db880) at
src/app/srs_app_rtmp_conn.cpp:1474
#8 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10c0160) at
src/app/srs_app_st.cpp:270
#9 0x0000000000542b4c in SrsFastCoroutine::pfm (arg=0x10c0160) at
src/app/srs_app_st.cpp:285
#10 0x00000000006c311d in _st_thread_main () at sched.c:363
#11 0x00000000006c399a in st_thread_create (start=0x542b2c <SrsFastCoroutine::pfm(void*)>,
arg=0x10c0160,
joinable=1, stk_size=65536) at sched.c:694
```

我们可以多触发几次 SrsConsumer::enqueue，就可以知道后续转发音视频消息的调用栈如《3.1 SrsSource::on_video_imp调用栈》小节所示。
比如转发视频消息：

```
2332: srs_error_t SrsSource::on_video_imp(SrsSharedPtrMessage* msg)
2333: {
2334:     srs_error_t err = srs_success;
2335:
2336:     bool is_sequence_header = SrsFlvVideo::sh(msg->payload, msg->size);
2337:
2338:     // whether consumer should drop for the duplicated sequence header.
2339:     bool drop_for_reduce = false;
2340:     if (is_sequence_header && meta->previous_vsh() && _srs_config->get_reduce_sequence_header(req->vhost)) {
2341:         if (meta->previous_vsh()->size == msg->size) {
2342:             drop_for_reduce = srs_bytes_equals(meta->previous_vsh()->payload, msg->payload, msg->size);
2343:             srs_warn("drop for reduce sh video, size=%d", msg->size);
2344:         }
2345:     }
2346:
2347:     // cache the sequence header if h264
2348:     // donot cache the sequence header to gop_cache, return here.
2349:     if (is_sequence_header && (err = meta->update_vsh(msg)) != srs_success) {
2350:         return srs_error_wrap(err, "meta update video");
2351:     }
2352:
2353:     // Copy to hub to all utilities.
2354:     if ((err = hub->on_video(msg, is_sequence_header)) != srs_success) {
2355:         return srs_error_wrap(err, "hub consume video");
2356:     }
2357:
2358:     // For bridger to consume the message.
2359:     if (bridger_ && (err = bridger->on_video(msg)) != srs_success) {
2360:         return srs_error_wrap(err, "bridger consume video");
2361:     }
2362:
2363:     // copy to all consumer
2364:     if (!drop_for_reduce) {
2365:         for (int i = 0; i < (int)consumers.size(); i++) {
2366:             SrsConsumer* consumer = consumers.at(i);
2367:             if ((err = consumer->enqueue(msg, atc, jitter_algorithm)) != srs_success) {
2368:                 return srs_error_wrap(err, "consume video");
2369:             }
2370:         }
2371:     }
2372: }
```

发送给每个拉流端对应的
SrsConsumer queue

转发音频同理见：srs_error_t SrsSource::on_audio_imp(SrsSharedPtrMessage* msg) 函数。

4 服务器怎么给RTMP拉流端转发数据

要回到SrsRtmpConn::do_playing函数

1. consumer->dump_packets(&msgs, count) 从SrsConsumer queue读取音视频消息
2. rtmp->send_and_free_messages(msgs.msgs, count, info->res->stream_id) 把音视频消息发给拉流客户端

5 哪些配置文件会影响RTMP的延迟

主要是merge read、merge write以及gop cache影响延迟。

- merge read默认是关闭的
- merge write默认开启
- gop cache默认开启

低延迟配置可以参考：realtime.conf配置文件

```
listen          1935;
max_connections 1000;
daemon          off;
srs_log_tank     console;
vhost __defaultVhost__ {
    tcp_nodelay    on // 开启tcp nodelay
    min_latency    on; // 开启最小延迟，默认关闭

    play {
        gop_cache    off; // 关闭gop cache，默认开启
        queue_length 10; // consumer queue的时长 单位是秒
        mw_latency    100; // 合并写入的延迟ms
    }

    publish {
        mr off; // 关闭合并读取，默认关闭
    }
}
```

注：更多的配置参数详解请参考full.conf，该配置文件是对配置项最权威的解释。

5.1 推流相关的延迟

可以在以下函数打上断点分析：

- bool SrsConfig::get_mr_enabled(string vhost)
- srs_utime_t SrsConfig::get_mr_sleep(string vhost)
- SrsPublishRecvThread::on_start

SrsConfig::get_mr_enabled和get_mr_sleep

```
#0 SrsConfig::get_mr_enabled (this=0xf5d3a0, vhost="__defaultVhost__") at
src/app/srs_app_config.cpp:5690#1 0x00000000005d2c14 in
SrsPublishRecvThread::SrsPublishRecvThread (this=0x7ffff7f82700,
    rtmp_sdk=0x1060560, _req=0x10bca20, mr_sock_fd=9, tm=0, conn=0x10c4690,
    source=0x1060390,
    parent_cid=..., __in_chrg=<optimized out>, __vtt_parm=<optimized out>)
    at src/app/srs_app_recv_thread.cpp:303
#2 0x0000000000510098 in SrsRtmpConn::publishing (this=0x10c4690, source=0x1060390)
    at src/app/srs_app_rtmp_conn.cpp:859
#3 0x000000000050d5ac in SrsRtmpConn::stream_service_cycle (this=0x10c4690)
    at src/app/srs_app_rtmp_conn.cpp:566
#4 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x10c4690) at
src/app/srs_app_rtmp_conn.cpp:420
#5 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x10c4690) at
src/app/srs_app_rtmp_conn.cpp:233
#6 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x10c4690) at
src/app/srs_app_rtmp_conn.cpp:1474
#7 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10ab1c0) at
src/app/srs_app_st.cpp:270
#8 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10ab1c0) at
src/app/srs_app_st.cpp:285
#9 0x00000000006c311d in _st_thread_main () at sched.c:363
```

SrsConfig::get_mr_sleep也是同一个位置读取。

SrsConfig::get_realtime_enabled也是同一个位置读取。

SrsPublishRecvThread::on_start

```

432: void SrsPublishRecvThread::on_start()
433: {
434:     // we donot set the auto response to false,
435:     // for the main thread never send message.
436:
437: #ifdef SRS_PERF_MERGED_READ
438:     if (mr) {
439:         // set underlayer buffer size 设置socket可以
440:         set_socket_buffer(mr_sleep); 缓存的数据时间
441:
442:         // disable the merge read
443:         // @see https://github.com/ossrs/srs/issues/241
444:         rtmp->set_merge_read(true, this);
445:     }
446: #endif
447: }

```

```

#0  SrsPublishRecvThread::on_start (this=0x1072600) at
src/app/srs_app_recv_thread.cpp:438#1  0x00000000005d1c7b in SrsRecvThread::cycle
(this=0x1072610) at src/app/srs_app_recv_thread.cpp:117
#2  0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10c1890) at
src/app/srs_app_st.cpp:270
#3  0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10c1890) at
src/app/srs_app_st.cpp:285
#4  0x00000000006c311d in _st_thread_main () at sched.c:363

```

SrsPublishRecvThread::on_stop

```

Breakpoint 5, SrsPublishRecvThread::on_stop (this=0x1072600) at
src/app/srs_app_recv_thread.cpp:456
srs_cond_signal(error);
(gdb) bt
#0  SrsPublishRecvThread::on_stop (this=0x1072600) at src/app/srs_app_recv_thread.cpp:456
#1  0x00000000005d1d15 in SrsRecvThread::cycle (this=0x1072610) at
src/app/srs_app_recv_thread.cpp:126
#2  0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10c1890) at
src/app/srs_app_st.cpp:270
#3  0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10c1890) at
src/app/srs_app_st.cpp:285
#4  0x00000000006c311d in _st_thread_main () at sched.c:363
#5  0x00000000006c399a in st_thread_create (start=0x1072240, arg=0x1072300, joinable=0,
stk_size=6151579)
    at sched.c:694

```

SrsPublishRecvThread::on_read


```

467: #ifndef SRS_PERF_MERGED_READ
468: void SrsPublishRecvThread::on_read(ssize_t nread)
469: {
470:     if (!mr || realtime) {
471:         return;
472:     }
473:
474:     if (nread < 0 || mr_sleep <= 0) {
475:         return;
476:     }
477:
478:     /**
479:      * to improve read performance, merge some packets then read,
480:      * when it on and read small bytes, we sleep to wait more data.,
481:      * that is, we merge some data to read together.
482:      * @see https://github.com/ossrs/srs/issues/241
483:      */
484:     if (nread < SRS_MR_SMALL_BYTES) { 读到的数据量少的时候才会
485:         srs_usleep(mr_sleep);          休眠
486:     }
487: } « end on_read »
488: #endif

```

```

#0 SrsPublishRecvThread::on_read (this=0x105e3c0, nread=218373) at
src/app/srs_app_recv_thread.cpp:470#1 0x00000000004af887 in SrsFastStream::grow
(this=0x10442c0, reader=0x1043e90, required_size=1)
    at src/protocol/srs_protocol_stream.cpp:190
#2 0x000000000048a0cc in SrsProtocol::read_basic_header (this=0x10441b0,
fmt=@0x7ffff7fc5b8b: 0 '\000',
    cid=@0x7ffff7fc5b8c: 0) at src/protocol/srs_rtmp_stack.cpp:1007
#3 0x0000000000489e0c in SrsProtocol::recv_interlaced_message (this=0x10441b0,
pmsg=0x7ffff7fc5bf0)
    at src/protocol/srs_rtmp_stack.cpp:912
#4 0x0000000000487891 in SrsProtocol::recv_message (this=0x10441b0,
pmsg=0x7ffff7fc5ca8)
    at src/protocol/srs_rtmp_stack.cpp:401
#5 0x0000000000490708 in SrsRtmpServer::recv_message (this=0x1044180,
pmsg=0x7ffff7fc5ca8)
    at src/protocol/srs_rtmp_stack.cpp:2303
#6 0x00000000005d1dff in SrsRecvThread::do_cycle (this=0x105e3d0) at
src/app/srs_app_recv_thread.cpp:149
#7 0x00000000005d1c92 in SrsRecvThread::cycle (this=0x105e3d0) at
src/app/srs_app_recv_thread.cpp:119
#8 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10ab1c0) at
src/app/srs_app_st.cpp:270
#9 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10ab1c0) at
src/app/srs_app_st.cpp:285
#10 0x00000000006c311d in _st_thread_main () at sched.c:363

```

5.2 拉流相关的延迟

SrsConfig::get_mw_sleep 默认350ms

SrsConfig::get_mw_msgs 默认是8帧消息

mw_msgs和mw_sleep是与的关系，即是要两个条件同时成立。

SrsConfig::get_mw_sleep

```
733:     mw_sleep = _srs_config->get_mw_sleep(req->vhost);
734:     skt->set_socket_buffer(mw_sleep); // 根据时长设置socket buffer
735:     // initialize the send_min_interval
736:     send_min_interval = _srs_config->get_send_min_interval(req->vhost);
737:
704:
765: #ifdef SRS_PERF_QUEUE_COND_WAIT
766:     // wait for message to incoming.
767:     // @see https://github.com/ossrs/srs/issues/251
768:     // @see https://github.com/ossrs/srs/issues/257
769:     consumer->wait(mw_msgs, mw_sleep);
770: #endif
```

```
#0 SrsConfig::get_mw_sleep (this=0xf5d3a0, vhost="__defaultVhost__", is_rtc=false) at
src/app/srs_app_config.cpp:5732
#1 0x000000000050f38c in SrsRtmpConn::do_playing (this=0x10d1da0, source=0x1060390,
consumer=0x10cdb20,
    rtrd=0x105e430) at src/app/srs_app_rtmp_conn.cpp:733
#2 0x000000000050ebc2 in SrsRtmpConn::playing (this=0x10d1da0, source=0x1060390)
    at src/app/srs_app_rtmp_conn.cpp:699
#3 0x000000000050d4d2 in SrsRtmpConn::stream_service_cycle (this=0x10d1da0)
    at src/app/srs_app_rtmp_conn.cpp:556
#4 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:420
#5 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:233
#6 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:1474
#7 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10d1530) at
src/app/srs_app_st.cpp:270
#8 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10d1530) at
src/app/srs_app_st.cpp:285
#9 0x000000000006c311d in _st_thread_main () at sched.c:363
```

SrsConfig::get_mw_msgs

```
#0 SrsConfig::get_mw_msgs (this=0xf5d3a0, vhost="__defaultVhost__", is_rtc=false) at
src/app/srs_app_config.cpp:5762
#1 0x000000000050f38c in SrsRtmpConn::do_playing (this=0x10d1da0, source=0x1060390,
consumer=0x10cdb20,
    rtrd=0x105e430) at src/app/srs_app_rtmp_conn.cpp:732
#2 0x000000000050ebc2 in SrsRtmpConn::playing (this=0x10d1da0, source=0x1060390)
    at src/app/srs_app_rtmp_conn.cpp:699
#3 0x000000000050d4d2 in SrsRtmpConn::stream_service_cycle (this=0x10d1da0)
    at src/app/srs_app_rtmp_conn.cpp:556
#4 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:420
#5 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:233
#6 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:1474
#7 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10d1530) at
src/app/srs_app_st.cpp:270
#8 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10d1530) at
src/app/srs_app_st.cpp:285
#9 0x000000000006c311d in _st_thread_main () at sched.c:363
```

SrsConfig::get_queue_length

```
2597: srs_error_t SrsSource::consumer_dumps(SrsConsumer* consumer, bool ds, bool dm, bool dg)
2598: {
2599:     srs_error_t err = srs_success;
2600:
2601:     srs_utime_t queue_size = _srs_config->get_queue_length(req->vhost);
2602:     consumer->set_queue_size(queue_size); // 设置queue的最大缓存时长
```

```
#0 SrsConfig::get_queue_length (this=0xf5d3a0, vhost="__defaultVhost__") at
src/app/srs_app_config.cpp:5515
#1 0x0000000000521188 in SrsSource::consumer_dumps (this=0x1060390,
consumer=0x10cdb20, ds=true, dm=true,
    dg=true) at src/app/srs_app_source.cpp:2601
#2 0x000000000050ea4c in SrsRtmpConn::playing (this=0x10d1da0, source=0x1060390)
    at src/app/srs_app_rtmp_conn.cpp:685
#3 0x000000000050d4d2 in SrsRtmpConn::stream_service_cycle (this=0x10d1da0)
```

```

    at src/app/srs_app_rtmp_conn.cpp:556
#4 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:420
#5 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:233
#6 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:1474
#7 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10d1530) at
src/app/srs_app_st.cpp:270
#8 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10d1530) at
src/app/srs_app_st.cpp:285
#9 0x000000000006c311d in _st_thread_main () at sched.c:363

```

go cacache延迟

SrsConfig::get_gop_cache

```

541: bool enabled_cache = _srs_config->get_gop_cache(req->vhost);
542: srs_trace("source url=%s, ip=%s, cache=%d, is_edge=%d, source_id=%s/%s",
543:         req->get_stream_url().c_str(), ip.c_str(), enabled_cache, info->edge, source_id);
544: source->set_cache(enabled_cache);

```

是否缓存Gop cache

```

#0 SrsConfig::get_gop_cache (this=0xf5d3a0, vhost="__defaultVhost__") at
src/app/srs_app_config.cpp:5385#1 0x000000000050d1b6 in
SrsRtmpConn::stream_service_cycle (this=0x10d1da0)
    at src/app/srs_app_rtmp_conn.cpp:541
#2 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:420
#3 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:233
#4 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:1474
#5 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10d1530) at
src/app/srs_app_st.cpp:270
#6 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10d1530) at
src/app/srs_app_st.cpp:285
#7 0x000000000006c311d in _st_thread_main () at sched.c:363

```

SrsGopCache::cache保存Gop

缓存最新的Gop，这里实际上包括音频。

```

#0 SrsGopCache::cache (this=0x1060760, shared_msg=0x7fff7f3fa70) at
src/app/srs_app_source.cpp:619#1 0x00000000005201a1 in SrsSource::on_video_imp
(this=0x1060390, msg=0x7fff7f3fa70)
    at src/app/srs_app_source.cpp:2379
#2 0x0000000000051fb11 in SrsSource::on_video (this=0x1060390, shared_video=0x10df640)
    at src/app/srs_app_source.cpp:2309
#3 0x00000000000511798 in SrsRtmpConn::process_publish_message (this=0x10c4690,
source=0x1060390,
    msg=0x10df640) at src/app/srs_app_rtmp_conn.cpp:1079
#4 0x00000000000511598 in SrsRtmpConn::handle_publish_message (this=0x10c4690,
source=0x1060390,
    msg=0x10df640) at src/app/srs_app_rtmp_conn.cpp:1051
#5 0x000000000005d364c in SrsPublishRecvThread::consume (this=0x7fff7f82700,
msg=0x10df640)
    at src/app/srs_app_recv_thread.cpp:396
#6 0x000000000005d1e35 in SrsRecvThread::do_cycle (this=0x7fff7f82710)
    at src/app/srs_app_recv_thread.cpp:150
#7 0x000000000005d1c92 in SrsRecvThread::cycle (this=0x7fff7f82710) at
src/app/srs_app_recv_thread.cpp:119
#8 0x00000000000542ab6 in SrsFastCoroutine::cycle (this=0x10abff0) at
src/app/srs_app_st.cpp:270
#9 0x00000000000542b4c in SrsFastCoroutine::pfn (arg=0x10abff0) at
src/app/srs_app_st.cpp:285
#10 0x000000000006c311d in _st_thread_main () at sched.c:363

```

SrsGopCache::dump 拷贝Gop

拷贝gop

```

Breakpoint 11, SrsGopCache::dump (this=0x1060760, consumer=0x10cdb20, atc=false,
jitter_algorithm=SrsRtmpJitterAlgorithmFULL) at src/app/srs_app_source.cpp:685
685 {
(gdb) bt
#0 SrsGopCache::dump (this=0x1060760, consumer=0x10cdb20, atc=false,
    jitter_algorithm=SrsRtmpJitterAlgorithmFULL) at src/app/srs_app_source.cpp:685
#1 0x000000000005214e1 in SrsSource::consumer_dumps (this=0x1060390,
consumer=0x10cdb20, ds=true, dm=true,
    dg=true) at src/app/srs_app_source.cpp:2625

```

#2 0x000000000050ea4c in SrsRtmpConn::playing (this=0x10d1da0, source=0x1060390)
at src/app/srs_app_rtmp_conn.cpp:685
#3 0x000000000050d4d2 in SrsRtmpConn::stream_service_cycle (this=0x10d1da0)
at src/app/srs_app_rtmp_conn.cpp:556
#4 0x000000000050c304 in SrsRtmpConn::service_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:420
#5 0x000000000050adc5 in SrsRtmpConn::do_cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:233
#6 0x0000000000513af5 in SrsRtmpConn::cycle (this=0x10d1da0) at
src/app/srs_app_rtmp_conn.cpp:1474
#7 0x0000000000542ab6 in SrsFastCoroutine::cycle (this=0x10d1530) at
src/app/srs_app_st.cpp:270
#8 0x0000000000542b4c in SrsFastCoroutine::pfn (arg=0x10d1530) at
src/app/srs_app_st.cpp:285
#9 0x00000000006c311d in _st_thread_main () at sched.c:363

6 总结

rtmp监听

推流对应连接

推拉流怎么去读取和发送数据

延迟的问题