SRS3

一 定义

SRS 是一个流媒体集群，支持 RTMP/HLS/FLV，高效、稳定、易用。

二 安装*

```
configure -use-sys-ssl --full && make -j3
```

三 运行

```
./etc/init.d/srs stop
```

停止指令

```
./etc/init.d/srs restart
```

重启

RTMP 媒体端口监听

```
./obj/srs -c conf/rtmp.confg
```

查看实时日志

```
tail ./obj/srs.log
```

用 FFmpeg 的方式推流（略）

```
ffmpeg -re -i night.mp4 -c copy -f flv rtmp://192.168.31.53/live2/night
```

四 源码分析

1、 目录

3rdparty 第三方包，zip 格式。估计是一些公共的方法。

auto sh 脚本。服务运维使用。

conf sh 脚本。配置和修改配置。

doc 英文文档。

etc 略。编译使用。

ide 略。调试工具。

modules 空。扩展模块。HLS 音频和 MP4 音视频。

research 其他平台接入 SRS。Golong、arm、python、player

scripts 脚本。

src 主要源码。

|

|----- app API 层（主要逻辑）

|----- core 工具类通用

|----- kernel 音视频、文本输入输出、网络连接

|----- libs 略。就是普通代码

|----- main 控制台、单元测试。

|----- protool 协议的字符解析。json、avc、握手、rtmp、rtsp

|----- server 处理服务。线程消息相关。

先主要看下 RTMP 的协议和处理。

2、 代码跟踪

源码文件：/truck/src/main/src_main_server.cpp

这个 src_main_server.cpp 里面有个 int main() 入口。程序会走到 /truck/src/app/srs_app_server.cpp

srs_app_server.cpp 里面有 listen 监听和 Handler 消息。通过订阅设计模式实现。

srs_app_server.cpp 文件夹下 accept_client() 、fd2conn()接口。RTMP 媒体端口接听从这里开始。

*pconn = new SrsRtmpConn(this, stfd, ip);

这里开始 RTMP 的协议流处理。

SrsRtmpConn 类里面新建 SrsRtmpServer()类。SrsHandshakeBytes()握手后绑定当前通信接口。

SrsProtocol（）是解析协议的。

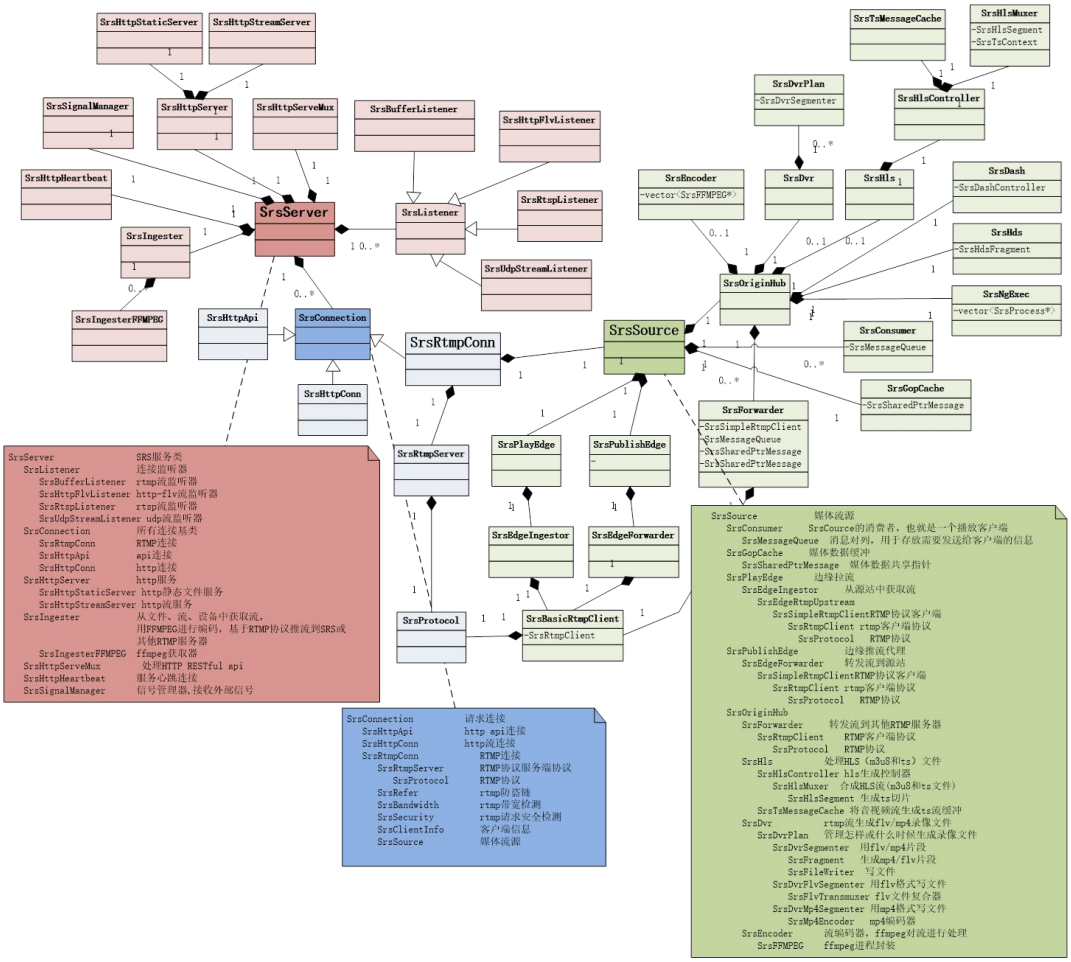
主要看下 srs_app_rtmp_conn.cpp 类，这里面是协议通信相关。

编解码的先不需要看，这个 SRS 本来只是开 RTMP 端口和会议室的。

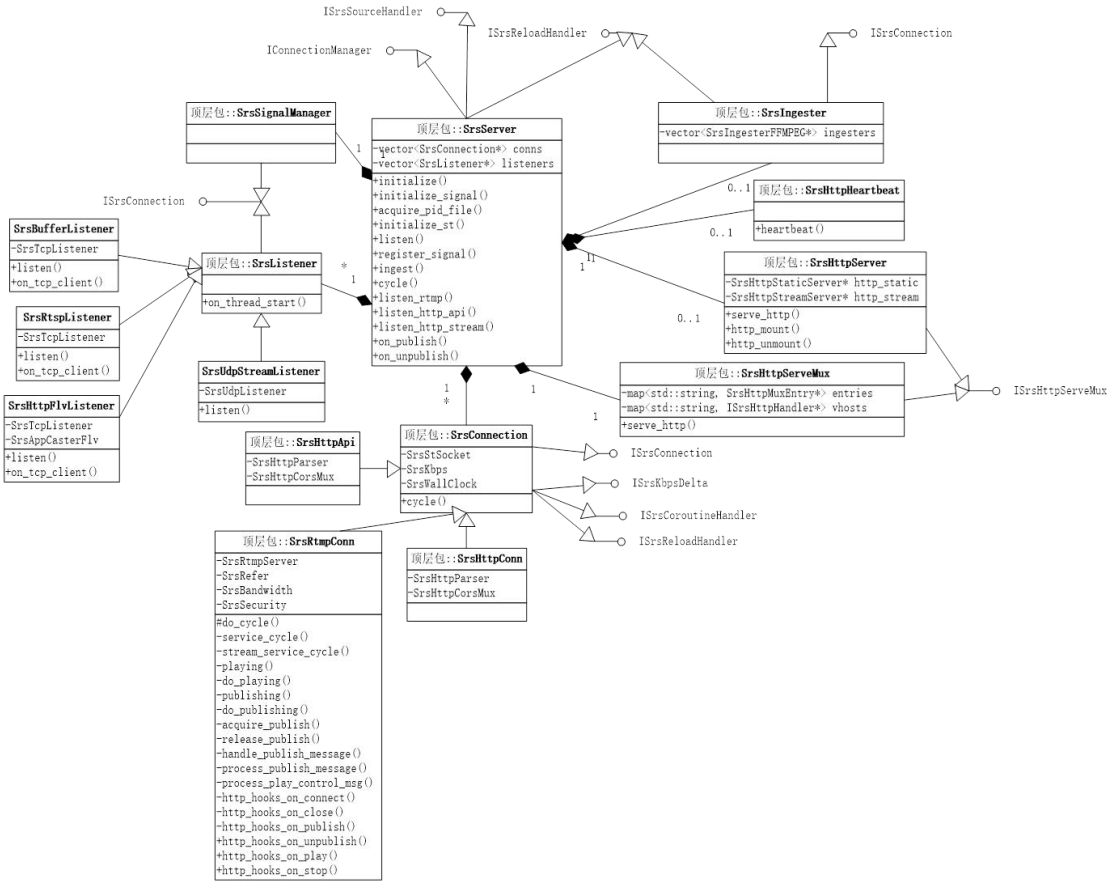
3、 二次开发技巧

自己打日志，改方法和改类的重载。梳理流程后，想怎么接入都不难。SRS 服务器对接 GB28181 的一个标准开源平台。

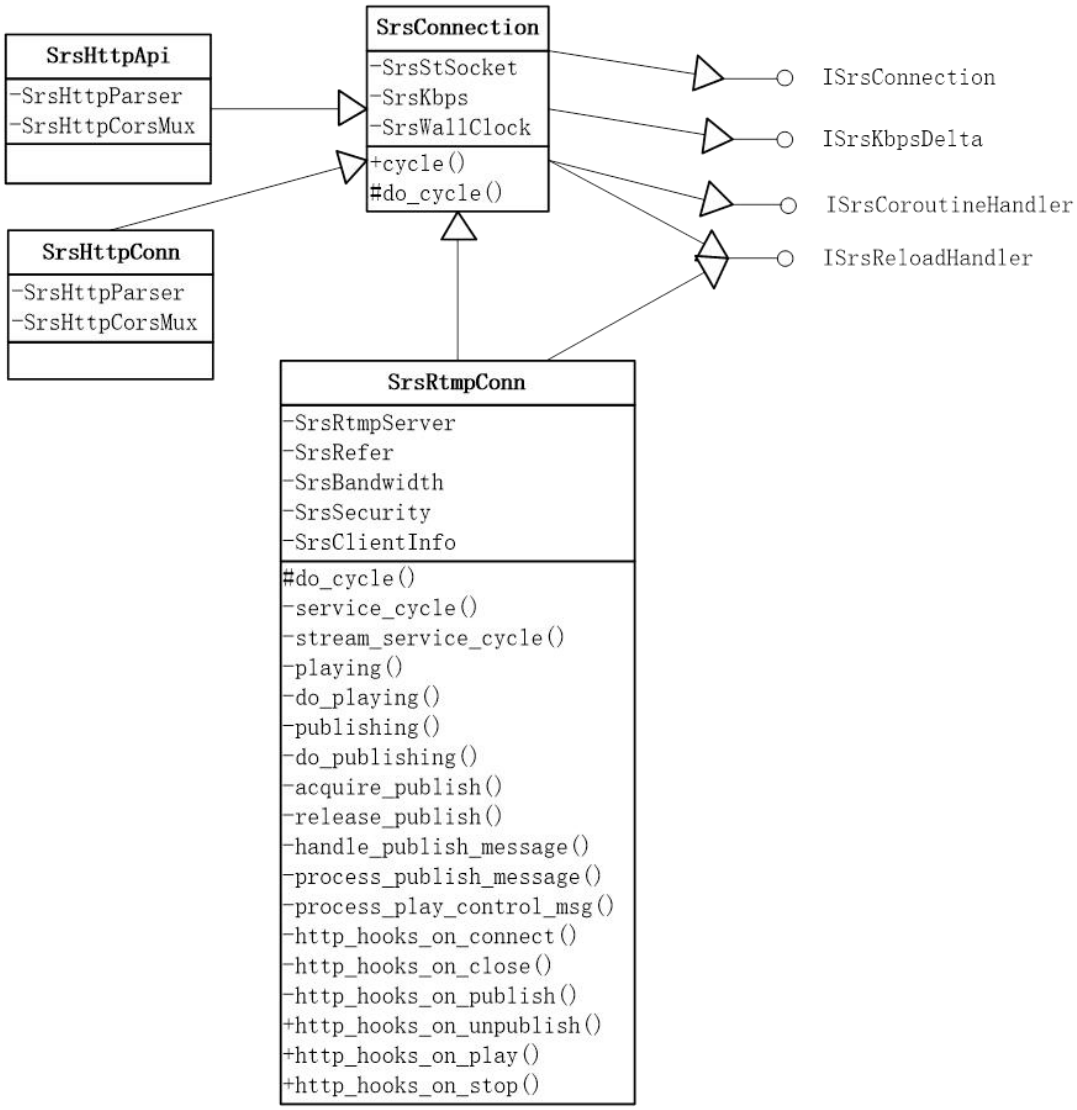
4. 官方类图



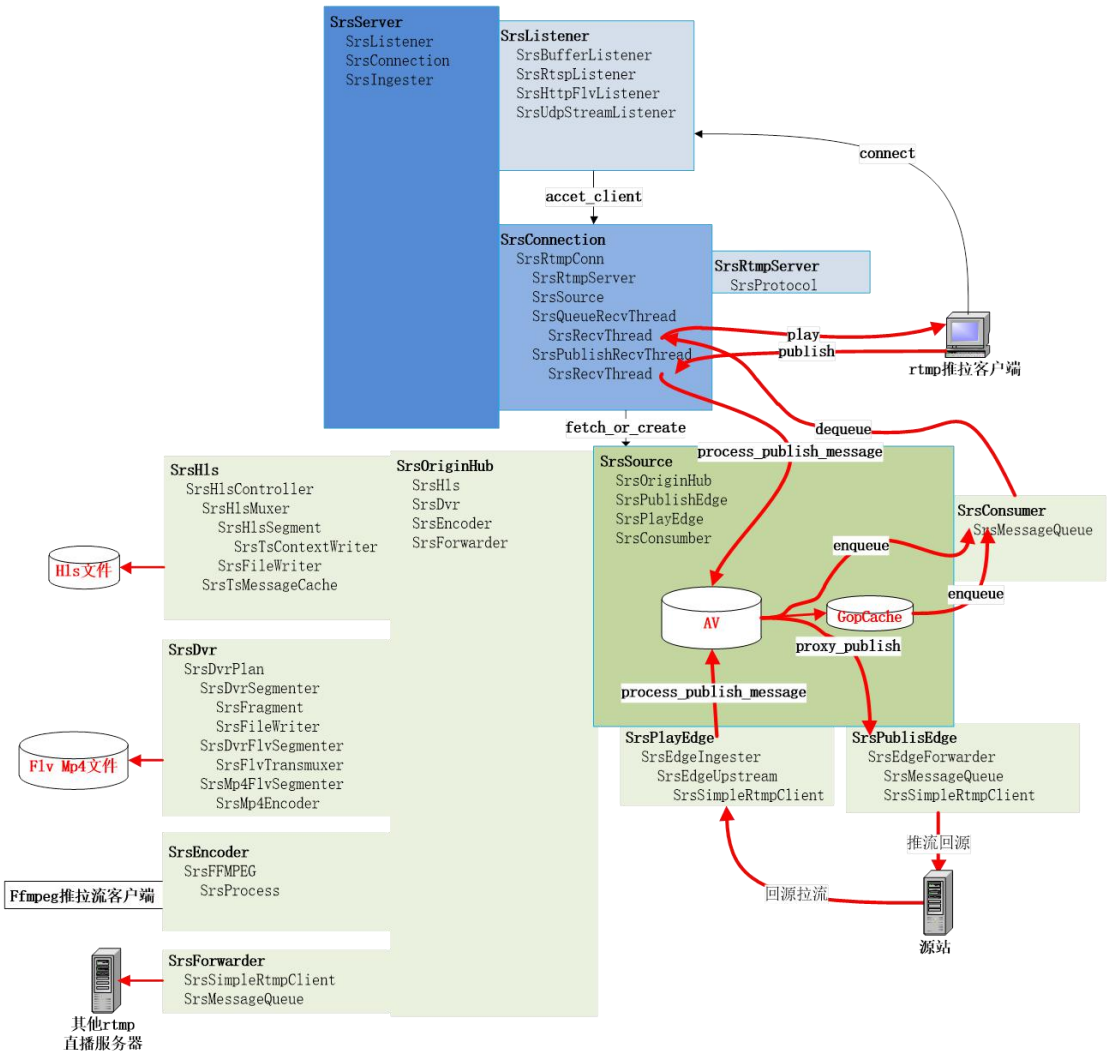
SRS server 类



SRS connect 类


SrsHttpApi
-SrsHttpParser
-SrsHttpCorsMux
SrsConnection
-SrsStSocket
-SrsKbps
-SrsWallClock
+cycle()
#do_cycle()
ISrsConnection
ISrsKbpsDelta
ISrsCoroutineHandler
ISrsReloadHandler
SrsHttpConn
-SrsHttpParser
-SrsHttpCorsMux
SrsRtmpConn
-SrsRtmpServer
-SrsRefer
-SrsBandwidth
-SrsSecurity
-SrsClientInfo
#do_cycle()
-service_cycle()
-stream_service_cycle()
-playing()
-do_playing()
-publishing()
-do_publishing()
-acquire_publish()
-release_publish()
-handle_publish_message()
-process_publish_message()
-process_play_control_msg()
-http_hooks_on_connect()
-http_hooks_on_close()
-http_hooks_on_publish()
+http_hooks_on_unpublish()
+http_hooks_on_play()
+http_hooks_on_stop()

5. SRS 类和流的问题



1. 客户端发送 RTMP 连接请求，SrsListener 接受后，创建一个 SrsConnection。每个 SrsConnection 会启动一个线程来完成相应任务。

2. SrsRtmpConn 首先按照 RTMP 协议交互成功后，根据 URL 创建流的信息。同时调用 fetch_or_create 生成一个处理音视频源的 SrsSource。

如果是推流：

创建 SrsPublishRecvThread 线程，接受客户端发过来的 RTMP 数据包。数据包由 SrsRtmpServer 来处理。如果是音视频数据，由

process_publish_message 来处理。它会通过 SrsSource 对媒体流进行处理。

（1）如果是边缘服务，SrsSource 直接将媒体 proxy publish 到源站服务

（2）否则 SrsSource 会将 publish 流放入每个 SrsConsumer 的媒体数据队列，一个 SrsConsumber 就是一个播放客户端。同事调用 SrsOriginHub 将媒体流按照配置来生成 FLV、HLS、MP4 录像文件，以及是否将流发布到其他 RTMP 服务器。最后检查如果启用 GopCache 会将流写入它的队列。对一个新的播放请求，保证首先获取一个 gop 数据，防止开始播放时黑屏和花屏。

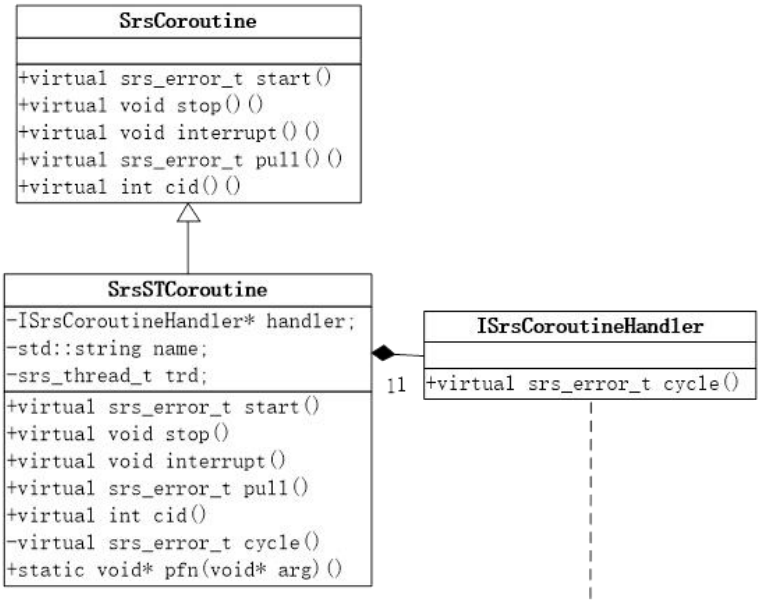
如果是拉流：

（1）如果是源站拉流同时启用源站集群，如果流不是该源站发布，则根据配置的发送 api 请求到其他源站，检查是否在其他源站发布流。如果是，发送一个 redirect，要求拉流客户重定向到指定服务器拉流。注意：RTMP 重定向信令，如果客户端直接请求的源站，要求 RTMP 客户端支持 redirect，如果 SRS 边缘回源到源站后再重定向，是可以。因为 SRS 支持 redirect。

（2）如果不走第一步，则创建一个 SrsConsumer 与 SrsQueueRecvThread 线程，创建 SrsConsumer 时，如果启用 gopcache，首先会将 gopcache 媒体数据插入 SrsConsumer 的数据队列。如果是边缘拉流，则使用 SrsPlayEdge 回源拉流。将回源拉的媒体流数据插入 SrsConsumer 的数据队列中。

（3） SrsQueueRecvThread 线程负责将 SrsConsumer 的数据队列中的媒体数据发送给客户端。SrsConsumer 队列中数据来源于 GopCache，源站 publish 的数据，以及回源拉流的数据。

6. SRS 线程模型



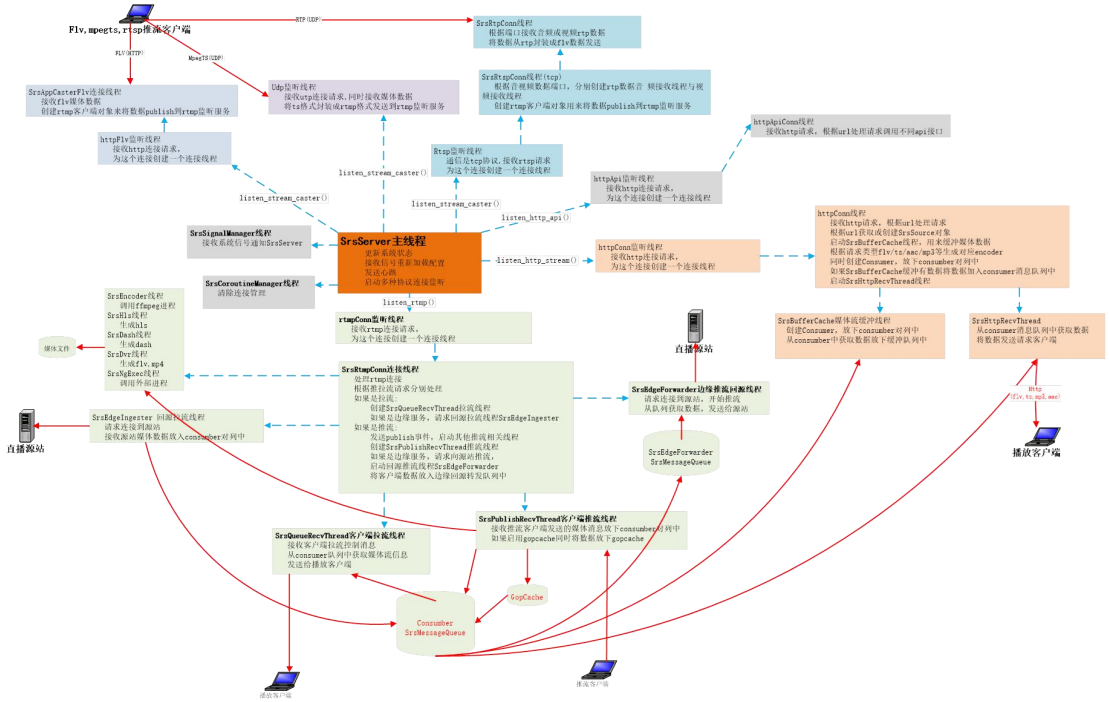
每个ST-coroutine必须实现如下两种接口之一

SrsOneCycleThread线程只执行一个任务后就正常终止，如下：

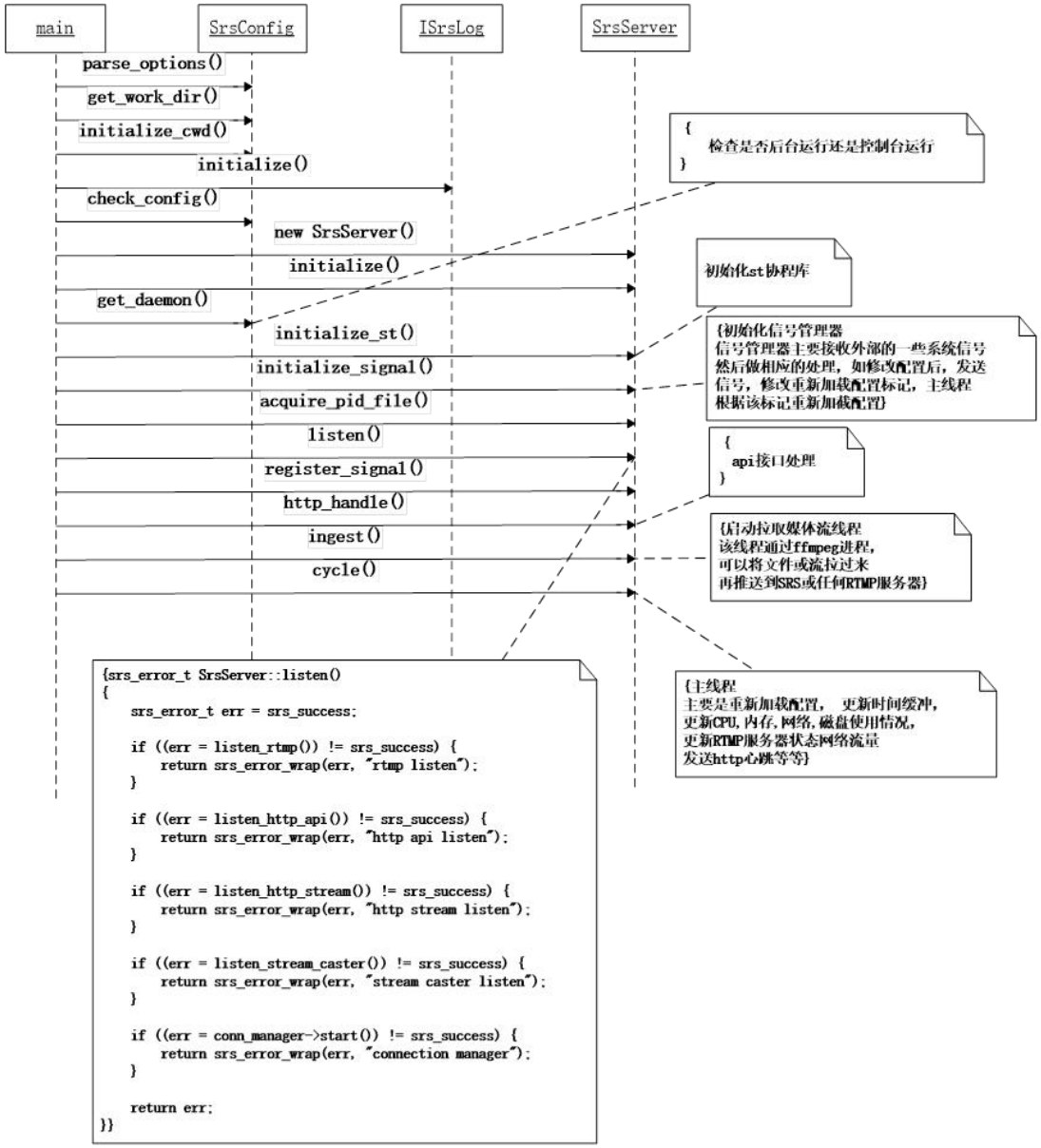
```
class SrsOneCycleThread : public ISrsCoroutineHandler {
    public: SrsCoroutine trd;
    public: virtual srs_error_t cycle() {
        // Do something, then return this cycle and thread terminated normally.
    }
};
```

线程内部有循环，如 RTMP接收线程：

```
class SrsReceiveThread : public ISrsCoroutineHandler {
    public: SrsCoroutine* trd;
    public: virtual srs_error_t cycle() {
        while (true) {
            // 检查线程是否已中断
            if ((err = trd->pull()) != srs_success) {
                return err;
            }
            // Do something, such as st_read() packets, it'll be wakeup
            // when user stop or interrupt the thread.
        }
    }
};
```


Flv, mpegts, rtsp推流客户端
RTP(UDP)
SrsRtpConn线程
根据端口接收音频或视频rtp数据
将数据从rtp封装成flv数据发送
FLV(HTTP)
Mpegts(UDP)
SrsRtpConn线程(tcp)
根据音视频数据端口，分别创建rtp数据音 频接收线程与视频接收线程
创建rtmp客户端对象用来将数据都publish到rtmp监听服务
SrsAppCasterFlv连接线程
接收flv媒体数据
创建rtmp客户端对象来将数据都publish到rtmp监听服务
Udp监听线程
接收srtp连接请求，同时接收媒体数据
将ts格式封装成rtmp格式发送到rtmp监听服务
httpApiConn线程
接收http请求，根据url处理请求调用不同api接口
httpFlv监听线程
接收http连接请求，
为这个连接创建一个连接线程
Rtsp监听线程
通信是tcp协议，接收rtsp请求
为这个连接创建一个连接线程
httpApi监听线程
接收http连接请求，
为这个连接创建一个连接线程
listen_stream_caster()
listen_stream_caster()
listen_stream_caster()
listen_http_api()
SrsSignalManager线程
接收系统信号通知SrsServer
SrsServer主线程
更新系统状态
接收信号重新加载配置
发送心跳
启动多种协议连接监听
listen_http_stream()
httpConn监听线程
接收http连接请求，
为这个连接创建一个连接线程
httpConn线程
接收http请求，根据url处理请求
根据url获取或创建SrsSource对象
启动SrsBufferCache线程，用来缓冲媒体数据
根据请求类型flv/ts/aac/mp3等生成对应encoder
同时创建Consumer，放下consumer对列中
如果SrsBufferCache缓冲有数据将数据加入consumer消息队列中
启动SrsHttpRecvThread线程
SrsCoroutineManager线程
清除连接管理
listen_rtmp()
SrsEncoder线程
调用ffmpeg进程
SrsHls线程
生成hls
SrsDash线程
生成dash
SrsDvr线程
生成flv,mp4
SrsNgExec线程
调用外部进程
媒体文件
rtmpConn监听线程
接收rtmp连接请求，
为这个连接创建一个连接线程
直播源站
SrsBufferCache媒体流缓冲线程
创建Consumer，放下consumer对列中
从consumer中获取数据放下缓冲队列中
SrsHttpRecvThread
从consumer消息队列中获取数据
将数据发送请求客户端
SrsRtmpConn连接线程
处理rtmp连接
根据推拉流请求分别处理
如果是拉流：
创建SrsQueueRecvThread拉流线程
如果是边缘服务，请求回源拉流线程SrsEdgeIngester
如果是推流：
发送publish事件，启动其他推流相关线程
创建SrsPublishRecvThread推流线程
如果是边缘服务，请求向源站推流，
启动回源推流线程SrsEdgeForwarder
将客户端数据放入边缘回源转发队列中
SrsEdgeForwarder边缘推流回源线程
请求连接到源站，开始推流
从队列获取数据，发送给源站
SrsEdgeForwarder
SrsMessageQueue
SrsEdgeIngester 回源拉流线程
请求连接到源站
接收源站媒体数据放入consumer对列中
直播源站
Http
(flv, ts, mp3, aac)
播放客户端
SrsPublishRecvThread客户端推流线程
接收推流客户端发送的媒体消息放下consumer对列中
如果启用gopcache同时将数据放下gopcache
SrsQueueRecvThread客户端拉流线程
接收客户端拉流控制消息
从consumer队列中获取媒体流信息
发送给播放客户端
GopCache
Consumer
SrsMessageQueue
播放客户端
推流客户端

7. SRS 程序启动流程



(1)首先检查解析启动命令参数，初始日志接口，检查配置文件是否正确

(2)创建 SrsServer 服务，初始化一些变量

(3)检查是否后台运行还是控制台运行

(4)初始化 st 协程库，信息号管理器

(5)如果后台运行写进程 pid 到文件

(6)监听连接：

listen_rtmp: rtmp 推流或拉流连接

listen_http_api: api 请求连接

listen_http_stream: http 拉流连接,http-flv ,http-ts，http-aac，http-mp3

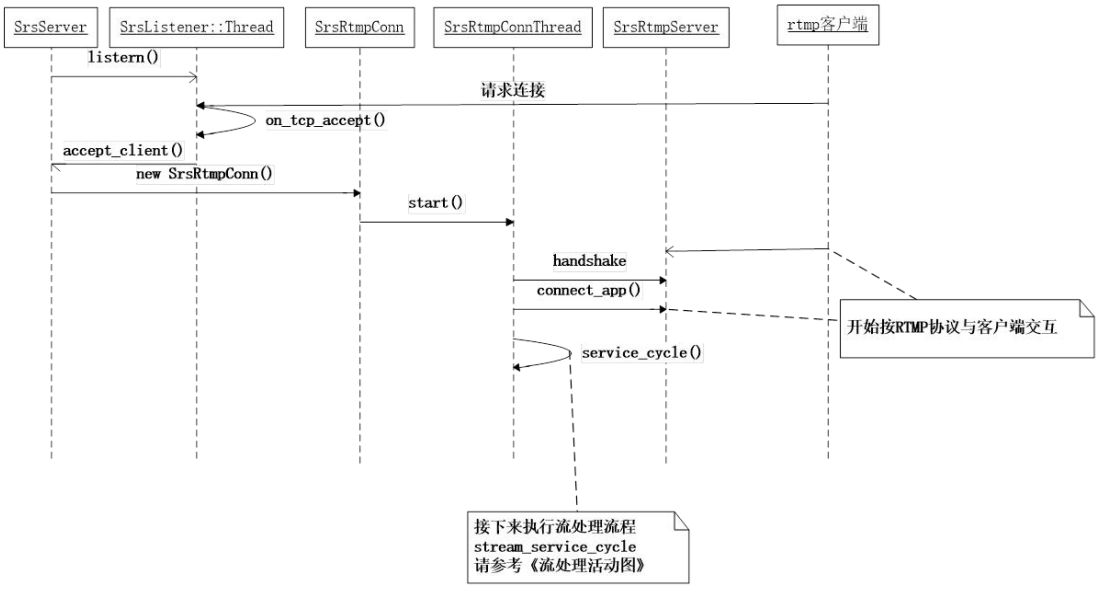
listen_stream_caster: 接收 MpegTSOverUdp 流请求，rtsp 推流请求，http-flv 推流请求

(7)初始化 http_api 接口处理

(8)启动 ingest 协程，使用 ffmpeg，拉取文件或流转发到本服务

(9)启动主线程

8.1 RTMP 监听



(1)SerServer 调用 listern 启动 rtmp 监听线程

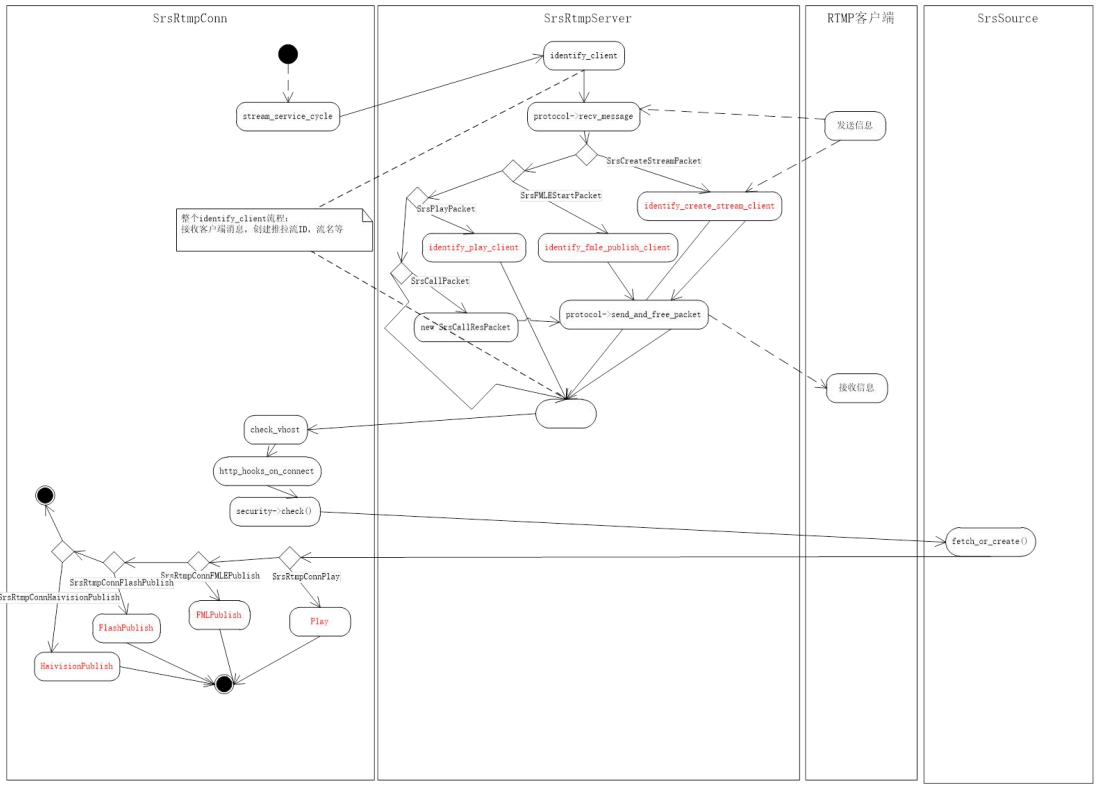
(2)客户端发送连接请求，监听线程收到请求后，发送 on_tcp_accetpt()事件

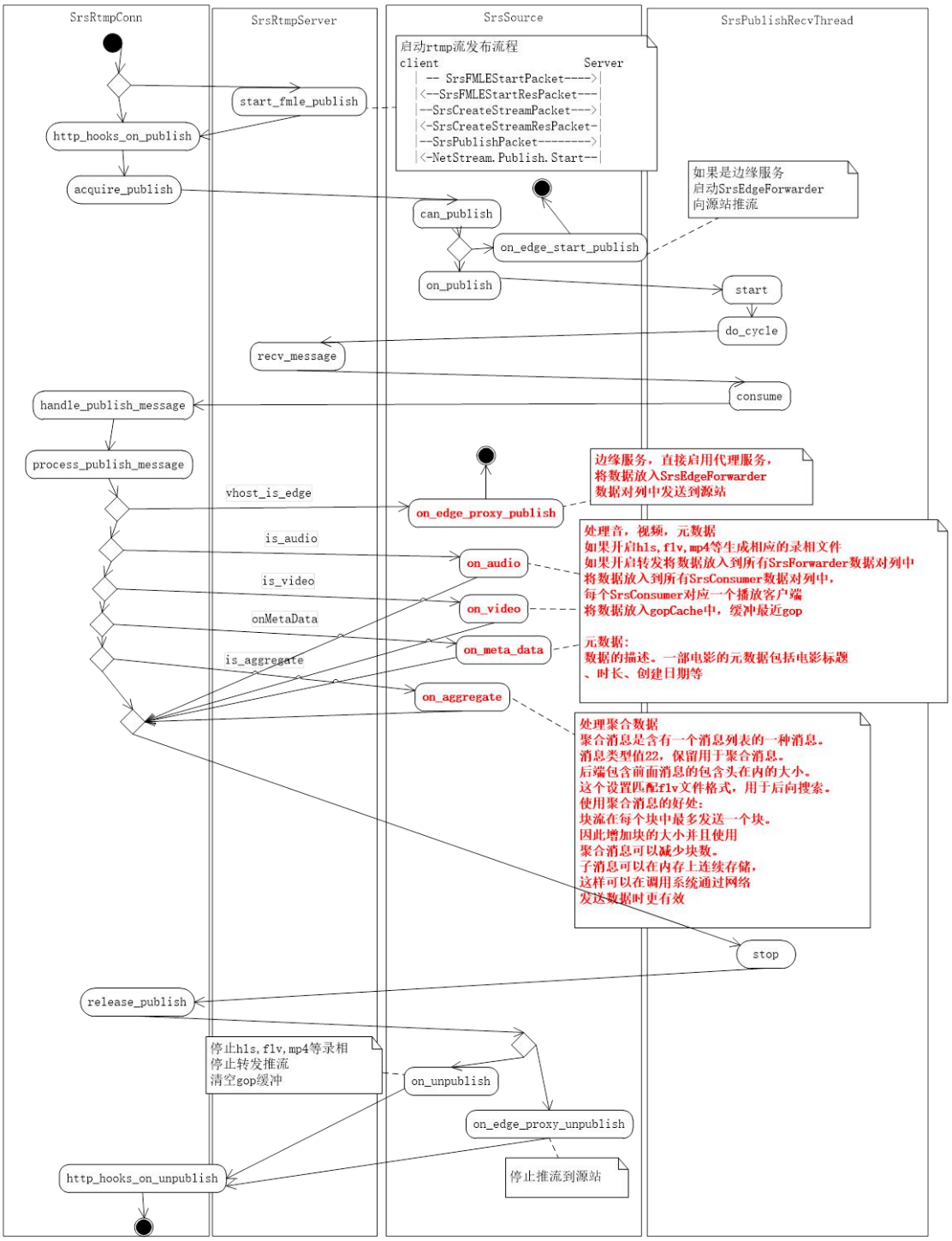
(3)SrsrServer 处理 accetp_client() 创建一个新的 SrsRtmpConn,同时启动 SrsRtmpConnThread 连接线程

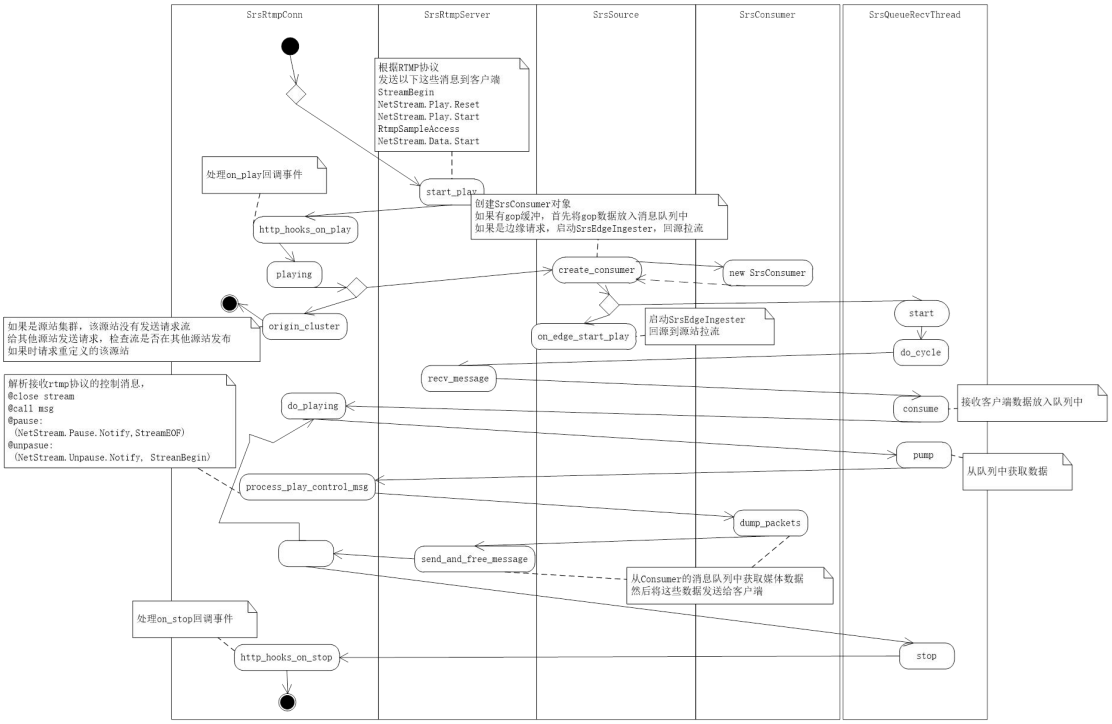
(4)SrsRtmpConnThread 收到客户端 rtmp 握手，同时根据 rtmp 连接流程创建一个 rtmp 连接

(5)连接成功之后,调用 stream_service_cycle 对 rtmp 媒体流处理

8.2 RTMP 媒体流处理流程




SrsRtmpConn
SrsRtmpServer
SrsSource
SrsPublishRecvThread
start_fmle_publish
http_hooks_on_publish
acquire_publish
can_publish
on_edge_start_publish
on_publish
start
do_cycle
recv_message
consume
handle_publish_message
process_publish_message
vhost_is_edge
on_edge_proxy_publish
is_audio
on_audio
is_video
on_video
onMetaData
on_meta_data
is_aggregate
on_aggregate
stop
release_publish
on_unpublish
on_edge_proxy_unpublish
http_hooks_on_unpublish
启动rtmp流发布流程
client Server
-- SrsFMLEStartPacket---->
<--SrsFMLEStartResPacket---
--SrsCreateStreamPacket--->
<-SrsCreateStreamResPacket-
--SrsPublishPacket-------->
<-NetStream.Publish.Start--
如果是边缘服务
启动SrsEdgeForwarder
向源站推流
边缘服务，直接启用代理服务，
将数据放入SrsEdgeForwarder
数据对列中发送到源站
处理音，视频，元数据
如果开启hls,flv,mp4等生成相应的录相文件
如果开启转发将数据放入到所有SrsForwarder数据对列中
将数据放入到所有SrsConsumer数据对列中，
每个SrsConsumer对应一个播放客户端
将数据放入gopCache中，缓冲最近gop
元数据:
数据的描述。一部电影的元数据包括电影标题
、时长、创建日期等
处理聚合数据
聚合消息是含有一个消息列表的一种消息。
消息类型值22，保留用于聚合消息。
后端包含前面消息的包含头在内的大小。
这个设置匹配flv文件格式，用于后向搜索。
使用聚合消息的好处:
块流在每个块中最多发送一个块。
因此增加块的大小并且使用
聚合消息可以减少块数。
子消息可以在内存上连续存储，
这样可以在调用系统通过网络
发送数据时更有效
停止hls,flv,mp4等录相
停止转发推流
清空gop缓冲
停止推流到源站



ALL PICTURE FROM :

https://github.com/xialixin/srs_code_note/blob/master/doc/srs_note.md