

01-SRS流媒体服务器开发

版权归零声学院所有，侵权必究

[SRS环境搭建](#)

[SRS源码目录](#)

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>

SRS环境搭建

srs官网：<https://github.com/ossrs/srs>

码云的源速度快：<https://gitee.com/winlinvip/srs.oschina.git>

github的源速度慢：<https://github.com/ossrs/srs.git>

选择当前最新的release版本3.0

第一步，获取SRS。详细参考[GIT获取代码](#)

```
git clone https://gitee.com/winlinvip/srs.oschina.git srs.3.0-20200720
cd srs.3.0-20200720
#使用当前最新的3.0版本
git checkout 3.0release
cd trunk
```

第二步，编译SRS。详细参考[Build](#)

```
./configure && make
```

第三步，编写SRS配置文件。详细参考[RTMP分发](#)，[Delivery HLS](#)，[Delivery HTTP FLV](#)

编辑 `conf/srs.conf`，服务器启动时指定该配置文件(srs的conf文件夹有该文件)。

```

1 listen                1935;
2 max_connections       1000;
3 srs_log_tank          file;
4 srs_log_file          ./objs/srs.log;
5 http_api {
6     enabled           on;
7     listen            1985;
8 }
9 http_server {
10    enabled            on;
11    listen             8081; # http监听端口, 注意自己配置的端口
12    dir               ./objs/nginx/html;
13 }
14 stats {
15     network          0;
16     disk             sda sdb xvda xvdb;
17 }
18 vhost __defaultVhost__ {
19     # hls darren
20     hls {
21         enabled       on;
22         hls_path       ./objs/nginx/html;
23         hls_fragment   10;
24         hls_window     60;
25     }
26     # http-flv darren
27     http_remux {
28         enabled       on;
29         mount         [vhost]/[app]/[stream].flv;
30         hstrs         on;
31     }
32 }

```

第四步，启动SRS。

```
./objs/srs -c conf/srs.conf
```

```
1 ubuntu@VM-0-13-ubuntu:~/0voice/media/srs.3.0-20200720/trunk$ ./objs/s
```

```

rs -c conf/srs.conf
2 后台运行结果
3 [2020-07-20 17:34:48.061][Trace][30433][0] XCORE-SRS/3.0.141(0uXuli)
4 [2020-07-20 17:34:48.061][Trace][30433][0] config parse complete
5 [2020-07-20 17:34:48.061][Trace][30433][0] write log to file ./objs/srs.log
6 [2020-07-20 17:34:48.061][Trace][30433][0] you can: tailf ./objs/srs.log
7 [2020-07-20 17:34:48.061][Trace][30433][0] @see: https://github.com/ossrs/srs/wiki/v1_CN_SrsLog

```

确认是否已经正常启动

```

1 ubuntu@VM-0-13-ubuntu:~/0voice/media/srs.3.0-20200720/trunk$ sudo ps -ef | grep srs
2 ubuntu 30435 1 0 17:34 pts/0 00:00:00 ./objs/srs -c conf/srs.conf

```

显示到ubuntu 30435 1 0 17:34 pts/0 00:00:00 ./objs/srs -c conf/srs.conf

安全退出正在运行的srs

sudo kill -SIGQUIT srs_pid

默认是后台启动的方式，如果是要方便GDB调试则需要修改配置文件为前台启动。

```

1 listen 1935;
2 max_connections 1000;
3 #srs_log_tank file;
4 #srs_log_file ./objs/srs.log;
5 # 前台运行
6 daemon off;
7 # 打印到终端控制台
8 srs_log_tank console;
9 http_api {
10     enabled on;
11     listen 1985;
12 }

```

```

13 http_server {
14     enabled          on;
15     listen           8081; # http监听端口
16     dir              ./objs/nginx/html;
17 }
18 stats {
19     network          0;
20     disk             sda sdb xvda xvdb;
21 }
22 vhost __defaultVhost__ {
23     # hls darren
24     hls {
25         enabled       on;
26         hls_path       ./objs/nginx/html;
27         hls_fragment   10;
28         hls_window     60;
29     }
30     # http-flv darren
31     http_remux {
32         enabled        on;
33         mount          [vhost]/[app]/[stream].flv;
34         hstrs          on;
35     }
36 }

```

执行方法: `./objs/srs -c conf/srs.conf`

在终端运行, log也在终端显示

```

1 [2020-07-20 17:46:33.586][Trace][1533][0] system default latency(ms)
  : mw(0-350) + mr(0-350) + play-queue(0-30000)
2 [2020-07-20 17:46:33.586][Warn][1533][0][0] SRS/3.0.141 is beta
3 [2020-07-20 17:46:33.586][Trace][1533][0] http flv live stream, vhos
  t=__defaultVhost__, mount=[vhost]/[app]/[stream].flv
4 [2020-07-20 17:46:33.586][Trace][1533][0] http: root mount to ./objs
  /nginx/html
5 [2020-07-20 17:46:33.586][Trace][1533][0] st_init success, use epoll
6 [2020-07-20 17:46:33.586][Trace][1533][380] server main cid=380, pid

```

```

=1533, ppid=2337, asprocess=0
7 [2020-07-20 17:46:33.586][Trace][1533][380] write pid=1533 to ./objs
  /srs.pid success!
8 [2020-07-20 17:46:33.586][Trace][1533][380] RTMP listen at tcp://0.0
  .0.0:1935, fd=7
9 [2020-07-20 17:46:33.586][Trace][1533][380] HTTP-API listen at tcp:/
  /0.0.0.0:1985, fd=8
10 [2020-07-20 17:46:33.586][Trace][1533][380] HTTP-Server listen at tc
  p://0.0.0.0:8081, fd=9
11 [2020-07-20 17:46:33.586][Trace][1533][380] signal installed, reload
  =1, reopen=10, fast_quit=15, grace_quit=3
12 [2020-07-20 17:46:33.586][Trace][1533][380] http: api mount /console
  to ./objs/nginx/html/console

```

SRS源码目录

trunk目录

3rdparty auto conf configure doc etc ide modules research scripts src usr

```

ubuntu@VM-0-13-ubuntu:~/srs.3.0-20200720/trunk$ ls
3rdparty auto conf configure doc etc ide modules research scripts src usr

```

src下的源码

app core kernel libs main protocol service utest

```

ubuntu@VM-0-13-ubuntu:~/srs.3.0-20200720/trunk/src$ ls
app core kernel libs main protocol service utest

```

app应用.

```

├── app
│   ├── srs_app_async_call.cpp
│   ├── srs_app_async_call.hpp    可以用来执行异步任务，通过execute()函数 push任务，然后
在cycle()执行
│   ├── srs_app_bandwidth.cpp
│   ├── srs_app_bandwidth.hpp    提供带宽测试接口

```

	——	srs_app_caster_flv.cpp	
	——	srs_app_caster_flv.hpp	支持POST一个flv流到服务器，类似相当于RTMP的publish
	——	srs_app_config.cpp	
	——	srs_app_config.hpp	读取配置文件
	——	srs_app_conn.cpp	
	——	srs_app_conn.hpp	srs的基本连接，每个连接对应一个协程，所有的连接都被管理
	——	srs_app_coworkers.cpp	
	——	srs_app_coworkers.hpp	SrsCoWorkers For origin cluster
	——	srs_app_dash.cpp	
	——	srs_app_dash.hpp	SrsDash 流媒体DASH业务 The MPEG-DASH encoder, transmux RTMP to DASH.
	——	srs_app_dvr.cpp	
	——	srs_app_dvr.hpp	SrsDvr 录制RTMP流程flv或者mp4文件
	——	srs_app_edge.cpp	
	——	srs_app_edge.hpp	SrsEdgeRtmpUpstream 边缘节点业务，比如从源站拉流到边缘，边缘回溯到源站
	——	srs_app_empty.cpp	
	——	srs_app_empty.hpp	没有内容
	——	srs_app_encoder.cpp	
	——	srs_app_encoder.hpp	SrsEncoder 可以使用多个ffmpeg来转换指定的流，最终调用SrsFFMPEG来转流
	——	srs_app_ffmpeg.cpp	
	——	srs_app_ffmpeg.hpp	SrsFFMPEG 使用ffmpeg来转换流
	——	srs_app_forward.cpp	
	——	srs_app_forward.hpp	SrsForwarder 将流转发到其他服务器
	——	srs_app_fragment.cpp	
	——	srs_app_fragment.hpp	SrsFragment 表示一个分片，如HLS分片、DVR分片或DASH分片。它是一个媒体文件，例如FLV或MP4，有持续时间。
	——	srs_app_hds.cpp	
	——	srs_app_hds.hpp	SrsHds 将RTMP转成Adobe HDS流
	——	srs_app_heartbeat.cpp	
	——	srs_app_heartbeat.hpp	SrsHttpHeartbeat HTTP心跳
	——	srs_app_hls.cpp	
	——	srs_app_hls.hpp	SrsHls HLS业务，Transmux RTMP stream to HLS(m3u8 and ts).
	——	srs_app_hourglass.cpp	
	——	srs_app_hourglass.hpp	SrsHourGlass 滴答tick的处理程序
	——	srs_app_http_api.cpp	
	——	srs_app_http_api.hpp	SrsHttpApi HTTP业务API
	——	srs_app_http_client.cpp	

	——	srs_app_http_client.hpp	没有内容
	——	srs_app_http_conn.cpp	
	——	srs_app_http_conn.hpp	SrsHttpConn, HTTP连接, 继承于SrsConnection
	——	srs_app_http_hooks.cpp	
	——	srs_app_http_hooks.hpp	SrsHttpHooks HTTP钩子, HTTP回调API
	——	srs_app_http_static.cpp	
	——	srs_app_http_static.hpp	SrsHttpStaticServer HTTP静态服务器实例, 为HTTP静态文件和FLV/MP4视频点播服务
	——	srs_app_http_stream.cpp	
	——	srs_app_http_stream.hpp	SrsHttpStreamServer HTTP直播流服务, 支持FLV/TS/MP3/AAC流
	——	srs_app_ingest.cpp	
	——	srs_app_ingest.hpp	SrsIngester摄取文件/流/设备, 用FFMPEG编码(可选), 通过RTMP推送到SRS(或其他RTMP服务器)
	——	srs_app_listener.cpp	
	——	srs_app_listener.hpp	SrsTcpListener SrsUdpListener TCP/UDP监听器
	——	srs_app_log.cpp	
	——	srs_app_log.hpp	SrsFastLog 日志
	——	srs_app_mpegts_udp.cpp	
	——	srs_app_mpegts_udp.hpp	SrsMpegtsOverUdpThe mpegts over udp stream caster
	——	srs_app_ng_exec.cpp	
	——	srs_app_ng_exec.hpp	SrsNgExec
	——	srs_app_pithy_print.cpp	
	——	srs_app_pithy_print.hpp	SrsPithyPrint 收集信息, 然后打印
	——	srs_app_process.cpp	
	——	srs_app_process.hpp	SrsProcess启动和停止进程, 当被终止时调用cycle重新启动进程
	——	srs_app_recv_thread.cpp	
	——	srs_app_recv_thread.hpp	SrsHttpRecvThread HTTP数据读取, SrsPublishRecvThread推流数据读取, SrsQueueRecvThread从队列读取; SrsRecvThread封装的协程
	——	srs_app_refer.cpp	
	——	srs_app_refer.hpp	SrsRefer
	——	srs_app_reload.cpp	
	——	srs_app_reload.hpp	ISrsReloadHandler 重新读取配置文件的处理
	——	srs_app_rtmp_conn.cpp	
	——	srs_app_rtmp_conn.hpp	SrsRtmpConn RTMP连接
	——	srs_app_rtsp.cpp	
	——	srs_app_rtsp.hpp	SrsRtpConn RTSP连接, SrsRtspCaster RTSP业务
	——	srs_app_security.cpp	
	——	srs_app_security.hpp	SrsSecurity 安全限制, 主要是限制url

```

|   |—— srs_app_server.cpp
|   |—— srs_app_server.hpp    SrsServer SRS服务，对应的rtmp、rtsp、http-flv等业务在这里
启动
|   |—— srs_app_source.cpp
|   |—— srs_app_source.hpp    SrsSource 对应一个源，支持多个SrsConsumer来拉流，
SrsSourceManager管理源， SrsMetaCache用于源缓存Meta数据， SrsConsumer源的消费者，
SrsGopCache GOP缓存
|   |—— srs_app_statistic.cpp
|   |—— srs_app_statistic.hpp  SrsStatistic流统计
|   |—— srs_app_st.cpp
|   |—— srs_app_st.hpp        SrsSTCoroutine协程相关
|   |—— srs_app_thread.cpp
|   |—— srs_app_thread.hpp    SrsCoroutineManager协程管理
|   |—— srs_app_utility.cpp
|   |—— srs_app_utility.hpp  工具类： SrsPlatformInfo、 SrsNetworkDevices、 SrsMemInfo、
SrsDiskStat等等
|   |—— core
|   |—— srs_core_autofree.cpp
|   |—— srs_core_autofree.hpp  通过栈上的方式构建自动释放堆申请的对象，这个设计还是非常值
得我们学习
|   |—— srs_core.cpp
|   |—— srs_core.hpp          版本相关的一些信息
|   |—— srs_core_mem_watch.cpp
|   |—— srs_core_mem_watch.hpp 内存监测接口
|   |—— srs_core_performance.cpp
|   |—— srs_core_performance.hpp 性能测试相关
|   |—— srs_core_time.cpp
|   |—— srs_core_time.hpp      时间单位相关
|   |—— srs_core_version3.cpp
|   |—— srs_core_version3.hpp  版本信息
|—— kernel // 音视频格式相关的
|   |—— srs_kernel_aac.cpp
|   |—— srs_kernel_aac.hpp    SrsAacTransmuxer 合成AAC音频流，带ADTS header
|   |—— srs_kernel_balance.cpp
|   |—— srs_kernel_balance.hpp SrsLbRoundRobin负载均衡，用于边缘节点拉流和其他多个服
务器的功能
|   |—— srs_kernel_buffer.cpp
|   |—— srs_kernel_buffer.hpp SrsBuffer读取字节的实用类
|   |—— srs_kernel_codec.cpp

```


| |—— srs_kernel_codec.hpp 编码器相关，包括视频和音频，非常核心的文件；SrsFlvVideo用来检测FLV的video tag对应内容；SrsFlvAudio用来检测FLV的audio tag对应内容；SrsMaxNbSamples 256表示video最大的NALUS个数，audio最大的packet数量；SrsFrame存储帧，SrsAudioFrame 存储AAC帧，SrsVideoFrame存储视频帧；SrsFormat编码器格式，包含了一个或者多个流，比如为RTMP format时，包含一个视频和一个音频帧。先猜测推流时的数据实例是保存在SrsFormat？

| |—— srs_kernel_consts.cpp

| |—— srs_kernel_consts.hpp SRS的常量定义，比如播放的标记#define

SRS_CONSTS_LOG_PLAY "PLA"；发布的标记#define SRS_CONSTS_LOG_CLIENT_PUBLISH "CPB"；SRS_CONSTS_HTTP_XXX等HTTP响应码；SRS_CONSTS_RTSP_XXX响应码等等。

| |—— srs_kernel_error.cpp

| |—— srs_kernel_error.hpp 返回值常量定义，ERROR_XXX；SrsCplxError 异常类

| |—— srs_kernel_file.cpp

| |—— srs_kernel_file.hpp 文件的读写，SrsFileWriter文件写入器，SrsFileReader文件读取器

| |—— srs_kernel_flv.cpp

| |—— srs_kernel_flv.hpp FLV SrsFlvDecoder解析，SrsFlvTransmuxer将RTMP转成FLV流；

SrsSharedPtrMessage对应RTMP的消息

| |—— srs_kernel_io.cpp

| |—— srs_kernel_io.hpp IO读写接口类

| |—— srs_kernel_log.cpp

| |—— srs_kernel_log.hpp 日志相关

| |—— srs_kernel_mp3.cpp

| |—— srs_kernel_mp3.hpp SrsMp3Transmuxer将RTMP转成MP3流

| |—— srs_kernel_mp4.cpp

| |—— srs_kernel_mp4.hpp SrsMp4Encoder MP4复用器；

| |—— srs_kernel_stream.cpp

| |—— srs_kernel_stream.hpp SrsSimpleStream用vector实现的简单的字节append类，主要在hls和http中使用，将来需要进行改进。

| |—— srs_kernel_ts.cpp

| |—— srs_kernel_ts.hpp SrsTsTransmuxer将RTMP流转成http-ts流，该文件实现了ts格式相关的接口

| |—— srs_kernel_utility.cpp

| |—— srs_kernel_utility.hpp 工具函数，比如bool srs_string_ends_with(std::string str, std::string flag)

|—— libs

| |—— srs_lib_bandwidth.cpp

| |—— srs_lib_bandwidth.hpp SrsBandwidthClient srs-librtmp 客户端带宽统计

| |—— srs_librtmp.cpp

| |—— srs_librtmp.hpp srs提供的客户端rtmp库

| |—— srs_lib_simple_socket.cpp

| |—— srs_lib_simple_socket.hpp SimpleSocketStream rtmp客户端的socket封装

```

|—— main
|   |—— srs_main_ingest_hls.cpp    拉取hls发布到rtmp流媒体服务器
|   |—— srs_main_mp4_parser.cpp    MP4 box解析
|   |—— srs_main_server.cpp        srs流媒体服务器主入口
|—— protocol    流媒体协议相关的协议都在这里
|   |—— srs_http_stack.cpp
|   |—— srs_http_stack.hpp        HTTP协议
|   |—— srs_protocol_amf0.cpp
|   |—— srs_protocol_amf0.hpp     Amf0解析
|   |—— srs_protocol_format.cpp
|   |—— srs_protocol_format.hpp   SrsRtmpFormat继承了SrsFormat， 代表RTMP格式
|   |—— srs_protocol_io.cpp
|   |—— srs_protocol_io.hpp       协议数据读取的IO封装接口，比如ISrsProtocolReadWrite
|   |—— srs_protocol_json.cpp
|   |—— srs_protocol_json.hpp     json类
|   |—— srs_protocol_kbps.cpp
|   |—— srs_protocol_kbps.hpp     比特率统计相关
|   |—— srs_protocol_stream.cpp
|   |—— srs_protocol_stream.hpp   流读取，从ISrsReader读取数据到buffer里面
|   |—— srs_protocol_utility.cpp
|   |—— srs_protocol_utility.hpp  协议工具函数
|   |—— srs_raw_avc.cpp
|   |—— srs_raw_avc.hpp          SrsRawH264Stream H264裸流解析，SrsRawAacStream AAC
裸流解析
|   |—— srs_rtmp_handshake.cpp
|   |—— srs_rtmp_handshake.hpp    RTMP握手，包括SrsSimpleHandshake和
SrsComplexHandshake
|   |—— srs_rtmp_msg_array.cpp
|   |—— srs_rtmp_msg_array.hpp    SrsMessageArray消息数组
|   |—— srs_rtmp_stack.cpp
|   |—— srs_rtmp_stack.hpp        RTMP协议栈
|   |—— srs_rtsp_stack.cpp
|   |—— srs_rtsp_stack.hpp        RTSP协议栈
|—— service
|   |—— srs_service_conn.cpp
|   |—— srs_service_conn.hpp      ISrsConnection HTTP/RTMP/RTSP等对象的连接接口；
IConnectionManager管理连接接口
|   |—— srs_service_http_client.cpp
|   |—— srs_service_http_client.hpp SrsHttpClient HTTP客户端
|   |—— srs_service_http_conn.cpp

```

```

|   |—— srs_service_http_conn.hpp      HTTP连接 SrsHttpParser, SrsHttpMessage,
SrsHttpResponseWriter, SrsHttpResponseReader
|   |—— srs_service_log.cpp
|   |—— srs_service_log.hpp      SrsConsoleLog日志相关
|   |—— srs_service_rtmp_conn.cpp
|   |—— srs_service_rtmp_conn.hpp    SrsBasicRtmpClient RTMP客户端类
|   |—— srs_service_st.cpp
|   |—— srs_service_st.hpp      对st-thread协程的封装
|   |—— srs_service_utility.cpp
|   |—— srs_service_utility.hpp    service组件的工具类
|—— utest
|   |—— srs_utest_amf0.cpp
|   |—— srs_utest_amf0.hpp
|   |—— srs_utest_app.cpp
|   |—— srs_utest_app.hpp
|   |—— srs_utest_avc.cpp
|   |—— srs_utest_avc.hpp
|   |—— srs_utest_config.cpp
|   |—— srs_utest_config.hpp
|   ..... 还有其他utest文件, 这里忽略
8 directories, 203 files

```