为了让大家少踩笔者踩过的坑，目前将工作中搭建rtmp推流服务器的步骤总结如下:

可直接使用打包后的 下载链接 省去下面的配置
默认推流地址 rtmp://你的ip地址:1935/live/xxx

1.下载 nginx 1.7.11.3 Gryphon

下载完成后解压， 将解压后的目录命名为nginx_1.7.11.3_Gryphon

> （此处注意千万不要命名为nginx-1.7.11.3-Gryphon，笔者初次搭建rtmp推流服务器时，将解压后的目录命名为nginx-1.7.11.3-Gryphon，出现局域网内的其他电脑都无法访问rtmp服务器的问题，也是花了一天的时间填坑，将主文件名改为nginx_1.7.11.3_Gryphon时，其他电脑才能正常访问。为神马会如此，笔者也是疑问，好歹问题无意间解决了）

2.下载服务器状态检查程序 stat.xsl（注：直接clone到nginx-1.7.11.3-Gryphon目录下）

此时的目录结构如下图所示:

名称	修改日期	类型	大小
ajp_temp	2014/4/27 19:11	文件夹	
conf	2019/4/12 15:52	文件夹	
contrib	2019/4/12 15:44	文件夹	
docs	2019/4/12 15:44	文件夹	
html	2019/4/12 15:44	文件夹	
logs	2019/4/12 16:01	文件夹	
nginx-rtmp-module	2019/4/12 15:49	文件夹	
temp	2019/4/12 16:01	文件夹	
FAQ nginx-win version.txt	2015/3/14 21:28	文本文档	8 KB
lua51.dll	2014/3/31 18:35	应用程序扩展	336 KB
nginx.exe	2015/3/19 20:37	应用程序	3,088 KB
nginx_basic.exe	2015/3/19 18:26	应用程序	1,920 KB
Readme nginx-win version.txt	2015/3/19 20:43	文本文档	34 KB
start.bat	2019/4/12 16:06	Windows 批处理...	1 KB
Tweak-Optimize tcpip parameters for nginx connections.reg	2013/9/7 12:08	注册表项	1 KB

3.配置文件 conf\nginx-win-rtmp.conf 内容如下:

```
#user  nobody;
# multiple workers works !
worker_processes  2;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;
#worker_rlimit_nofile 100000;   #更改worker进程的最大打开文件数限制
                                                    #如果没设置的话，这个值为操作系统的限制.
                                                    #设置后你的操作系统和Nginx可以处理比
“ulimit -a”更多的文件
                                                    #所以把这个值设高，这样nginx就不会有
“too many open files”问题了
```

```
events {
    worker_connections  8192;#设置可由一个worker进程同时打开的最大连接数
                              #如果设置了上面提到的worker_rlimit_nofile, 我们可以将这
个值设得很高
    # max value 32768, nginx recycling connections+registry optimization =
    #   this.value * 20 = max concurrent connections currently tested with one
worker
    #   C1000K should be possible depending there is enough ram/cpu power
    # multi_accept on;
}

rtmp {
    server {
        listen 1935;#监听端口,若被占用,可以更改
        chunk_size 4000;#上传flv文件块儿的大小
        application live { #创建一个叫live的应用
             live on;#开启live的应用
             allow publish 127.0.0.1;#
             allow play all;
        }
    }
}

http {
    #include      /nginx/conf/naxsi_core.rules;
    include       mime.types;
    default_type  application/octet-stream;

    #log_format  main  '$remote_addr:$remote_port - $remote_user [$time_local]
"$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    #access_log  logs/access.log  main;

#     # loadbalancing PHP
#     upstream myLoadBalancer {
#         server 127.0.0.1:9001 weight=1 fail_timeout=5;
#         server 127.0.0.1:9002 weight=1 fail_timeout=5;
#         server 127.0.0.1:9003 weight=1 fail_timeout=5;
#         server 127.0.0.1:9004 weight=1 fail_timeout=5;
#         server 127.0.0.1:9005 weight=1 fail_timeout=5;
#         server 127.0.0.1:9006 weight=1 fail_timeout=5;
#         server 127.0.0.1:9007 weight=1 fail_timeout=5;
#         server 127.0.0.1:9008 weight=1 fail_timeout=5;
#         server 127.0.0.1:9009 weight=1 fail_timeout=5;
#         server 127.0.0.1:9010 weight=1 fail_timeout=5;
#         least_conn;
#     }

    sendfile        off;
    #tcp_nopush     on;

    server_names_hash_bucket_size 128;

## Start: Timeouts ##
```

```
    client_body_timeout   10;
    client_header_timeout 10;
    keepalive_timeout     30;
    send_timeout          10;
    keepalive_requests    10;
## End: Timeouts ##

    #gzip  on;

    server {
        listen       8088;
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        ## Caching Static Files, put before first location
        #location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
        #    expires 14d;
        #    add_header Vary Accept-Encoding;
        #}

# For Naxsi remove the single # line for learn mode, or the ## lines for full WAF
mode
        location / {
            #include    /nginx/conf/mysite.rules; # see also http block naxsi
include line
            ##SecRulesEnabled;
              ##DeniedUrl "/RequestDenied";
              ##CheckRule "$SQL >= 8" BLOCK;
              ##CheckRule "$RFI >= 8" BLOCK;
              ##CheckRule "$TRAVERSAL >= 4" BLOCK;
              ##CheckRule "$XSS >= 8" BLOCK;
            root   html;
            index  index.html index.htm;
        }

# For Naxsi remove the ## lines for full WAF mode, redirect location block used
by naxsi
        ##location /RequestDenied {
        ##    return 412;
        ##}

## Lua examples !
#         location /robots.txt {
#           rewrite_by_lua '
#             if ngx.var.http_host ~= "localhost" then
#               return ngx.exec("/robots_disallow.txt");
#             end
#           ';
#         }

        #error_page  404              /404.html;

        # redirect server error pages to the static page /50x.html
        #
        error_page   500 502 503 504  /50x.html;
```

```
        location = /50x.html {
            root   html;
        }

        # proxy the PHP scripts to Apache listening on 127.0.0.1:80
        #
        #location ~ \.php$ {
        #    proxy_pass   http://127.0.0.1;
        #}

        # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
        #
        #location ~ \.php$ {
        #    root           html;
        #    fastcgi_pass   127.0.0.1:9000; # single backend process
        #    fastcgi_pass   myLoadBalancer; # or multiple, see example above
        #    fastcgi_index  index.php;
        #    fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
        #    include        fastcgi_params;
        #}

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        #    deny  all;
        #}
    }


    # another virtual host using mix of IP-, name-, and port-based configuration
    #
    #server {
    #    listen       8000;
    #    listen       somename:8080;
    #    server_name  somename  alias  another.alias;

    #    location / {
    #        root   html;
    #        index  index.html index.htm;
    #    }
    #}


    # HTTPS server
    #
    #server {
    #    listen       443 ssl spdy;
    #    server_name  localhost;

    #    ssl                  on;
    #    ssl_certificate      cert.pem;
    #    ssl_certificate_key  cert.key;

    #    ssl_session_timeout  5m;

    #    ssl_prefer_server_ciphers On;
    #    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
```

```
    #    ssl_ciphers
ECDH+AESGCM:ECDH+AES256:ECDH+AES128:ECDH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL
:!eNULL:!MD5:!DSS:!EXP:!ADH:!LOW:!MEDIUM;

    #    location / {
    #        root   html;
    #        index  index.html index.htm;
    #    }
    #}

}

nginx-win-rtmp.conf
```

4.启动服务器

```
nginx.exe -c conf\nginx-win-rtmp.conf
1
```


5.使用推流地址

推流地址:rtmp://IP:监听端口/应用名/home，步骤3配置文件配置出的推流地址为rtmp://192.168.xxxx.xxxx:1935/live/home

通过此推流地址，便可以成功推流

6.其他ngnix常用命令:

（1）终止服务器

nginx.exe -s stop

stop是快速停止nginx，可能并不保存相关信息；

nginx.exe -s quit

quit是完整有序的停止nginx，并保存相关信息。

（2）重新载入Nginx

nginx.exe -s reload

当配置信息修改，需要重新载入这些配置时使用此命令。

（3）重新打开日志文件

nginx.exe -s reopen

1.2答疑

RTSP特点

RTSP（Real Time Streaming Protocol）是用来控制声音或影像的多媒体串流协议，并允许同时多个串流需求控制， 服务器端可以自行选择使用TCP或UDP来传送串流内容，它的语法和运作跟HTTP 1.1类似，但并不特别强调时间同步，所以比较能容忍网络延迟

特点

> 允许大量的网络延迟，
>
> 丢帧的现象很少出现

应用

> 主要用在监控领域，

RTP特点

特点

> 延时性比较低
>
> 网络拥堵时 会主动丢帧，达到发送消息最新
>
> 可以控制对方解码流程，操作流程
>
> RTP实时传输协议，广泛应用于流媒体传输应用场景，

应用场景

> 简单多播音频会议（Simple Multicast Audio Conference）
>
> 音频和视频会议（Audioand Video Conference）
>
> 混频器和转换器（MixersandTranslators）

RTCP控制协议

实时传输控制协议(**R**eal-**t**ime **C**ontrol**P**rotocol，RTCP)与RTP共同定义在1996年提出的RFC 1889中，是和 RTP一起工作的控制协议。**RTCP单独运行在低层协议上，由低层协议提供数据与控制包的复用。在RTP会话期间，每个会话参与者周期性地向所有其他参与者发送RTCP控制信息包**，如下图所示。**对于RTP会话或者广播，通常使用单个多目标广播地址，属于这个会话的所有RTP和RTCP信息包都使用这个多目标广播地址，通过使用不同的端口号可把RTP信息包和RTCP信息包区分开来。**

特点

> 在rtp基础上 控制 和确定 RTP用户源
>
> 主要控制rtp数据包大小，转发目标
>
> rtp和rtcp 一般是结合一起用，rtp类似于 okhttp rtcp类似于Retrofit

流媒体概述：

所谓流媒体是指采用流式传输的方式在 Internet 播放的媒体格式。
流媒体又叫流式媒体，它是指商家用一个视频传送服务器把节目当成数据包发出，传送到网络上。
用户通过解压设备对这些数据进行解压后，节目就会像发送前那样显示出来。
流媒体以流的方式在网络中传输音频、视频和多媒体文件的形式。
流媒体文件格式是支持采用流式传输及播放的媒体格式。
流式传输方式是将视频和音频等多媒体文件经过特殊的压缩方式分成一个个压缩包，
由服务器向用户计算机连续、实时传送。在采用流式传输方式的系统中，用户不必像非流式播放那样等到整个文件
全部下载完毕后才能看到当中的内容，而是只需要经过几秒钟或几十秒的启动延时即可在用户计算机上利用
相应的播放器对压缩的视频或音频等流式媒体文件进行播放，剩余的部分将继续进行下载，直至播放完毕。

RTP :(Real-time Transport Protocol)

是用于Internet上针对多媒体数据流的一种传输层协议.RTP 协议和 RTP 控制协议 RTCP 一起使用，
而且它是建立在 UDP 协议上的.
RTP 不像http和ftp可完整的下载整个影视文件，它是以固定的数据率在网络上发送数据，客户端也是按照这种速度观看影视文件，当
影视画面播放过后，就不可以再重复播放，除非重新向服务器端要求数据。

RTCP:Real-time Transport Control Protocol 或 RTP Control Protocol或简写 RTCP)

实时传输控制协议,是实时传输协议(RTP)的一个姐妹协议.

注:--:RTP 协议和 RTP控制协议(RTCP) 一起使用，而且它是建立在UDP协议上的

RTSP:(Real Time Streaming Protocol)

实时流媒体会话协议,SDP(会话描述协议)，RTP(实时传输协议)。

是用来控制声音或影像的多媒体串流协议,RTSP 提供了一个可扩展框架，使实时数据，如音频与视频的受控、点播成为可能。
媒体数据使用rtp,rtcp协议。
一般使用udp 作为传输层。适合IPTV场景。
数据源包括现场数据与存储在剪辑中的数据。该协议目的在于控制多个数据发送连接，为选择发送通道，如UDP、多播UDP与TCP提供途
径，并为选择基于RTP上发送机制提供方法
传输时所用的网络通讯协定并不在其定义的范围内，服务器端可以自行选择使用TCP或UDP来传送串流内容，比较能容忍网络延迟.

--->:RTSP 与 RTP 最大的区别在于：RTSP 是一种双向实时数据传输协议，它允许客户端向服务器端发送请求，如回放、快进、倒退等操作。当
然，RTSP 可基于 RTP 来传送数据，还可以选择 TCP、UDP、组播 UDP 等通道来发送数据，具有很好的扩展性。它时一种类似与http协议
的网络应用层协议.

WebRTC:

web端实现流媒体的协议。google刚推出WebRTC的时候巨头们要么冷眼旁观，要么抵触情绪很大。使用RTP协议传输。

RTMP(Real Time Messaging Protocol)

Macromedia 开发的一套视频直播协议，现在属于 Adobe。和 HLS 一样都可以应用于视频直播，基于TCP不会丢失。
// 区别是 RTMP 基于 flash 无法在 iOS 的浏览器里播放，但是实时性比 HLS 要好。
实时消息传送协议是 Adobe Systems 公司为 Flash 播放器和服务器之间音频、视频和数据传输 开发的开放协议.
// iOS 代码里面一般常用的是使用 RTMP 推流，可以使用第三方库 librtmp-iOS 进行推流，librtmp 封装了一些核心的 API 供使用者调用
RTMP 协议也要客户端和服务器通过“握手”来建立 RTMP Connection，然后在Connection上传输控制信息。RTMP 协议传输时会对数据格式化，而实际传输的时候为了更好地实现多路复用、分包和信息的公平性，发送端会把Message划分为带有 Message ID的Chunk，每个Chunk可能是一个单独的Message，
也可能是Message的一部分，在接受端会根据Chunk中包含的data的长度，message id和message的长度把chunk还原成完整的Message，从而实现信息的收发。

HLS:HTTP Live Streaming(HLS)

是苹果公司(Apple Inc.)实现的基于HTTP的流媒体传输协议，

可实现流媒体的 直播 和 点播 ,主要应用在iOS系
统，为iOS设备(如iPhone、iPad)提供音视频直播和点播方案。
HLS 点播，基本上就是常见的分段HTTP点播，不同在于，它的分段非常小。
相对于常见的流媒体直播协议，例如RTMP协议、RTSP 协议、MMS 协议等，HLS 直播最大的不同在于，直播客户端获取到的，并不是一个完
整的数据流。
HLS 协议在服务器端将直播数据流存储为连续的、很短时长的媒体文件(MPEG-TS格式)，而客户端则不断的下载并播放这些小文件，
因为服务器端总是会将最新的直播数据生成新的小文件，这样客户端只要不停的按顺序播放从服务器获取到的文件，就实现了直播。
由此可见，基本上可以认为，HLS 是以>>点播的技术方式来实现直播<<。由于数据通过 HTTP 协议传输，所以完全不用考虑防火墙或者代理的问
题,而且分段文件的时长很短，客户端可以很快的选择和切换码率，以适应不同带宽条件下的播放。不过HLS的这种技术特点，决定了它的
延迟一般总是会高于普通的流媒体直播协议。
// iOS和 Android 都天然支持这种协议，配置简单，直接使用 video 标签即可

VLS： 是一种流服务器，专门用来解决流的各种问题，它也具有一些 VLC 的特征。 videolan 作为服务器可以输出http，rtp，rtsp的流。

-

MediaCodec

MediaCodec是安卓自带的视频编解码接口，由于使用的是硬解码，其效率相对FFMPEG高出来不少。而MediaCodec就很好拓展，我们可以根据流媒体的协议和设备硬件本身来自定义硬件解码，代表播放器就是Google的ExoPlayer。

ExoPlayer的开源地址： https://github.com/google/ExoPlayer

基本流程：

- 1.创建和配置MediaCodec对象
- 2.进行以下循环：
- 如果一个输入缓冲区准备好：
- 读取部分数据，复制到缓冲区
- 如果一个输出缓冲区准备好：
- 复制到缓冲区
- 3.销毁MediaCodec对象

接口如下：

```
//根据视频编码创建解码器，这里是解码AVC编码的视频


MediaCodec mediaCodec
=MediaCodec.createDecoderByType(MediaFormat.MIMETYPE_VIDEO_AVC);


//创建视频格式信息


MediaFormat mediaFormat = MediaFormat.createVideoFormat(mimeType, width,
height);


//配置


mediaCodec.configure(mediaFormat, surfaceView.getHolder().getSurface(), null,
0);


mediaCodec.start();


//停止解码，此时可以再次调用configure()方法


mediaCodec.stop();
```

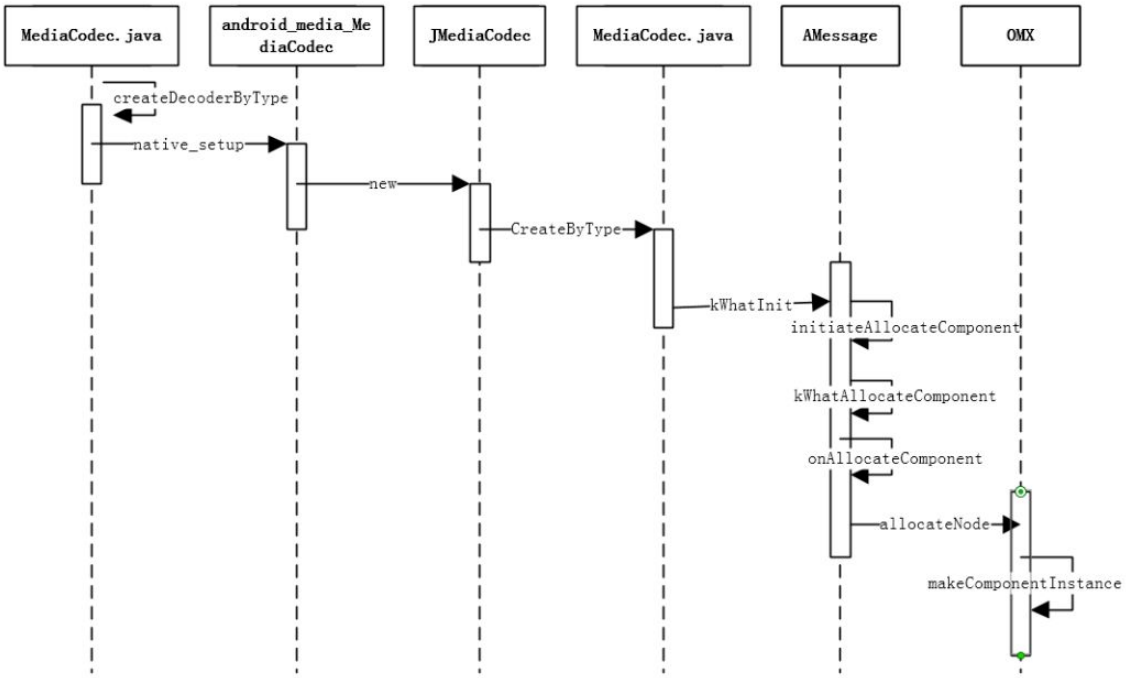
```
//释放内存

mediaCodec.release();

//一下是循环解码接口

getInputBuffers：获取需要编码数据的输入流队列，返回的是一个ByteBuffer数组

queueInputBuffer：输入流入队列

dequeueInputBuffer：从输入流队列中取数据进行编码操作

getOutputBuffers：获取编解码之后的数据输出流队列，返回的是一个ByteBuffer数组

dequeueOutputBuffer：从输出队列中取出编码操作之后的数据

releaseOutputBuffer：处理完成，释放ByteBuffer数据
```

用MediaCodec来实现的话，代码中主要通过上面的接口实现了媒体的播放过程。

实例可以参考：

https://blog.csdn.net/u014653815/article/details/81084161

下图是MediaCodec调用createDecoderByType创建过程。



由上图可知，MediaCodec并不是真正的codec，真正codec是在openMax。

OpenMax是一个多媒体应用程序的框架标准，通过使媒体加速组件能够在开发、集成和编程环节中实现跨多操作系统和处理器硬件平台，提供全面的流媒体编解码器和应用程序便携化。Android的多媒体引擎OpenCore和StageFright都可以使用OpenMax作为插件，主要用于编解码（Codec）处理。