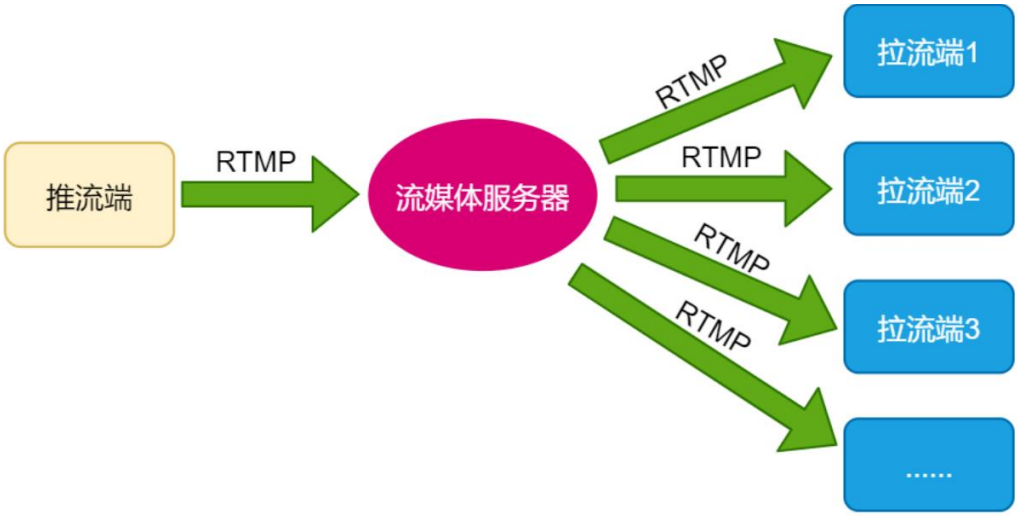
1. 视频监控简介

我们将使用 RTMP 流媒体服务来实现视频监控，RTMP 流媒体服务框架图如下所示：



推流端负责将视频数据通过 RTMP 流媒体协议传输给 RTMP 流媒体服务器，拉流端可以从流媒体服务器中通过 RTMP 协议获取到视频数据；而流媒体服务器负责接收推流端的视频数据、当有客户端（拉流端）想要获取视频数据时再将其发给相应的客户端。所以从上图可知，要想实现 RTMP 视频监控，必须要有这三部分：推流客户端、拉流客户端以及流媒体服务器。那这些需要我们自己去实现吗？当然不需要，譬如推流我们可以使用 FFmpeg 来做，拉流则可以实现 VLC 播放器来做，而流媒体客户端则使用 Nginx 来搭建即可！

2.Nginx 移植

前面也给大家提到了，我们可以使用 Nginx 来搭建 RTMP 流媒体服务器，譬如你可以在一台公网 IP 主机上搭建流媒体服务器，当

然，笔者并没有这个条件；这里我们选择在开发板上搭建流媒体服务器，并且推流端也是开发板，所以在本章的方案中，开发板既是流媒体服务器、也是推断端。既然要在开发板上搭建流媒体服务器，首先我们需要将 Nginx 移植到开发板上，事实上，我们的板子出厂系统中就已经移植好了 Nginx，并且板子在启动进入系统时会自动启动 Nginx，也就是启动流媒体服务，所以板子启动之后本身就已经是一台流媒体服务器了。当然，这里我们不管出厂系统中已经搭建好的流媒体服务，这里只是给大家说明一下，本章我们要自己动手亲自移植 Nginx、然后在板子上搭建流媒体服务。

下载 Nginx 源码

进入到 Ubuntu 系统的某个目录下，执行下面这条命令下载 Nginx 源码：

wget http://nginx.org/download/nginx-1.20.0.tar.gz

这里我们下载的是 1.20 版本，这是比较新的版本了。下载完成之后将得到一个名为 nginx-1.20.0.tar.gz 的压缩包文件。

```
sallen@sallen-virtualmachine:~/linux/nginxtools$ ls -l
总用量 2028
drwxr-xr-x 10 sallen sallen    4096 8月  27 14:39 nginx-1.20.0
-rw-rw-r--  1 sallen sallen 1061070 4月  20  2021 nginx-1.20.0.tar.gz
```

下载 nginx-rtmp-module 模块

事实上，原生的 Nginx 并不支持 RTMP，我们需要安装第三方模块 nginx-rtmp-module 插件使其支持 RTMP。

通过下面这条命令下载 nginx-rtmp-module：

git clone https://github.com/arut/nginx-rtmp-module.git

下载成功之后将得到 nginx-rtmp-module 文件夹。

```
sallen@sallen-virtualmachine:~/linux/nginxtools$ ls -l
总用量 2028
drwxr-xr-x 10 sallen sallen    4096 8月  27 14:39 nginx-1.20.0
-rw-rw-r--  1 sallen sallen 1061070 4月  20  2021 nginx-1.20.0.tar.gz
drwxrwxr-x  7 sallen sallen    4096 8月  27 14:17 nginx-rtmp-module
-rwxrw-rw-  1 sallen sallen 1001070 8月  27 15:19 test.mp4
```

交叉编译 Nginx

将下载得到的 nginx-1.20.0.tar.gz 文件进行解压：

tar -xzf nginx-1.20.0.tar.gz

```
sallen@sallen-virtualmachine:~/linux/nginxtools$ ls -l
总用量 2028
drwxr-xr-x 10 sallen sallen    4096 8月  27 14:39 nginx-1.20.0
-rw-rw-r--  1 sallen sallen 1061070 4月  20  2021 nginx-1.20.0.tar.gz
drwxrwxr-x  7 sallen sallen    4096 8月  27 14:17 nginx-rtmp-module
-rwxrw-rw-  1 sallen sallen 1001070 8月  27 15:19 test.mp4
```

解压之后生成 nginx-1.20 文件夹，进入到该目录下。在进行交叉编译之前，先对交叉编译工具进行初始

化操作： source /opt/st/stm32mp1/3.1-snapshot/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi

大家根据自己的实际安装位置填写路径。首先配置源码、接下执行 make 编译源码，最后执行 make install 安装即可！总共的步骤就这三个，但事实上在编译的过程中会出现一些问题，我们稍后再看！

配置源码

首先第一步是配置源码，在配置之前，需要进行一个简单的修改，否则配置通不过；首先打开 nginx 源

码目录下的 auto/cc/name 文件，将 21 行处的”exit 1”给注释掉！如下所示：

```
2 # Copyright (C) Igor Sysoev
3 # Copyright (C) Nginx, Inc.
4
5
6 if [ "$NGX_PLATFORM" != win32 ]; then
7
8     ngx_feature="C compiler"
9     ngx_feature_name=
10    ngx_feature_run=yes
11    ngx_feature_incs=
12    ngx_feature_path=
13    ngx_feature_libs=
14    ngx_feature_test=
15    . auto/feature
16
17    if [ $ngx_found = no ]; then
18        echo
19        echo $0: error: C compiler $CC is not found
20        echo
21    #   exit 1      前面加#号注释该行
22    fi
23
24 fi
```

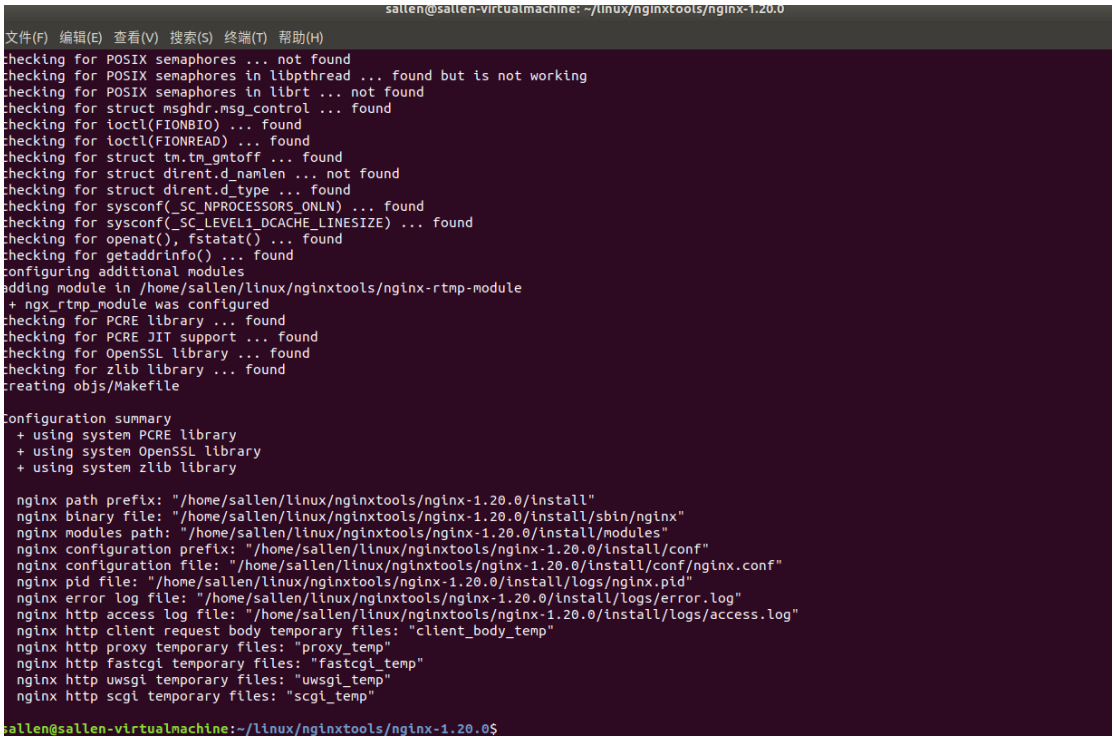
修改完成之后保存退出。接着打开 auto/types/sizeof 文件，将 15 行处的” ngx_size=”修改为” ngx_size=4”，并且将 36 行处的”$CC”修改为”gcc”，如下所示：

```
14
15 ngx_size=4      修改为ngx_size=4
16
17 cat << END > $NGX_AUTOTEST.c
18
19 #include <sys/types.h>
20 #include <sys/time.h>
21 $NGX_INCLUDE_UNISTD_H
22 #include <signal.h>
23 #include <stdio.h>
24 #include <sys/resource.h>
25 $NGX_INCLUDE_INTTYPES_H
26 $NGX_INCLUDE_AUTO_CONFIG_H
27
28 int main(void) {
29     printf("%d", (int) sizeof($ngx_type));
30     return 0;
31 }
32
33 END                          将此处的$CC修改为gcc
34
35
36 ngx_test="gcc $CC_TEST_FLAGS $CC_AUX_FLAGS \
37           -o $NGX_AUTOTEST $NGX_AUTOTEST.c $NGX_LD_OPT $ngx_feature_libs"
38
39 eval "$ngx_test >> $NGX_AUTOCONF_ERR 2>&1"
40
41
42 if [ -x $NGX_AUTOTEST ]; then
43     # ngx_size=`$NGX_AUTOTEST`
44     ngx_size=4          将ngx_size='$NGX_AUTOTEST' 修改为 ngx_size=4
45     echo " $ngx_size bytes"
46 fi
47
```

同样，修改完成之后保存退出即可！接着执行下面这条命令进行配置：

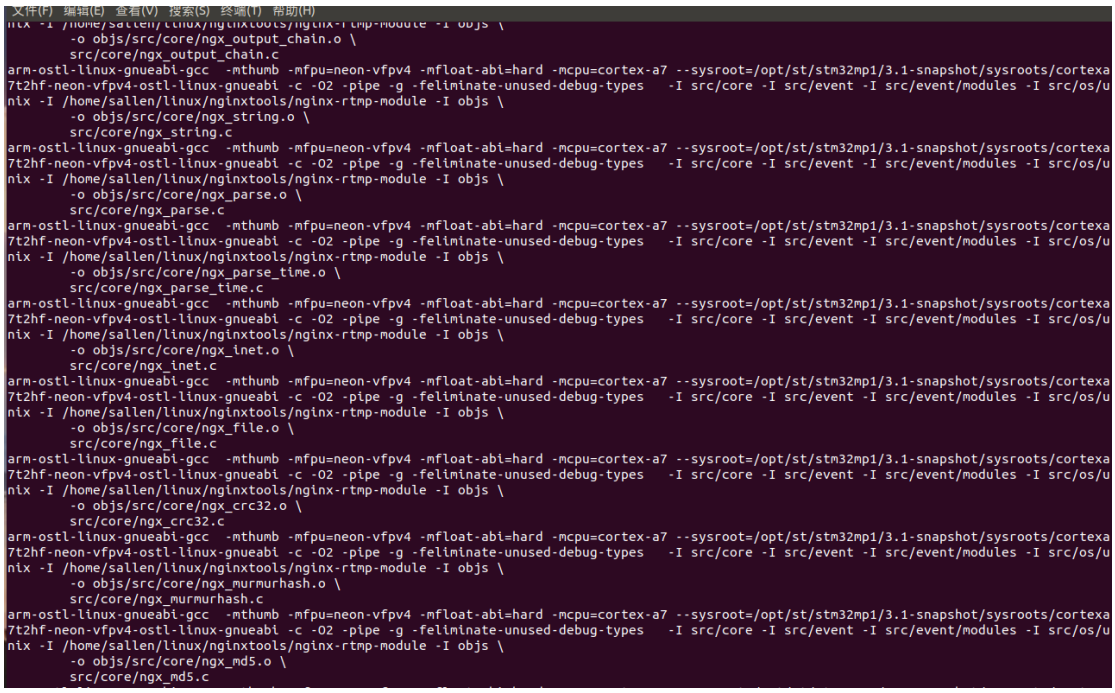
```
./configure --prefix=/home/sallen/linux/nginxtools/nginx-1.20.0/install \
--with-http_ssl_module \
--with-http_mp4_module \
--with-http_v2_module \
--without-http_upstream_zone_module \
--add-module=/home/sallen/linux/nginxtools/nginx-rtmp-module
```

上述命令中，--prefix 指定了 nginx 的安装路径，笔者为了方便直接将其安装到 nginx 源码目录下的 install 目录中；--add-module 用于添加第三方模块，譬如我们前面下载的 nginx-rtmp-module，所以--add-module 需要指向 nginx-rtmp-module 源码路径，大家根据自己的实际路径填写。

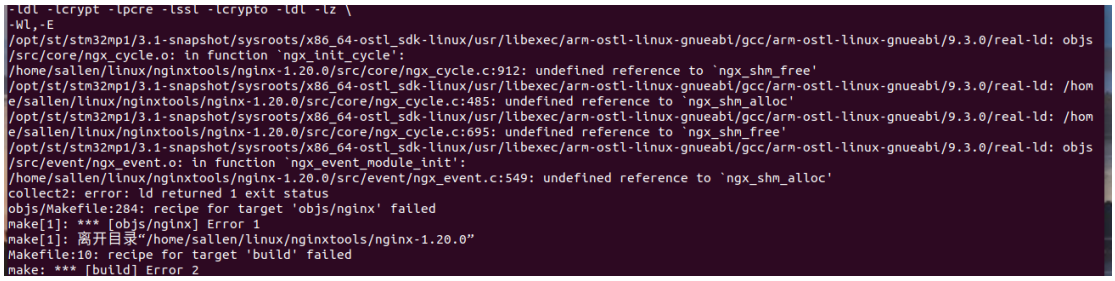


编译源码

配置完成之后，接着我们执行 make 编译：



本次编译并不会成功，将会出现如下错误打印信息：

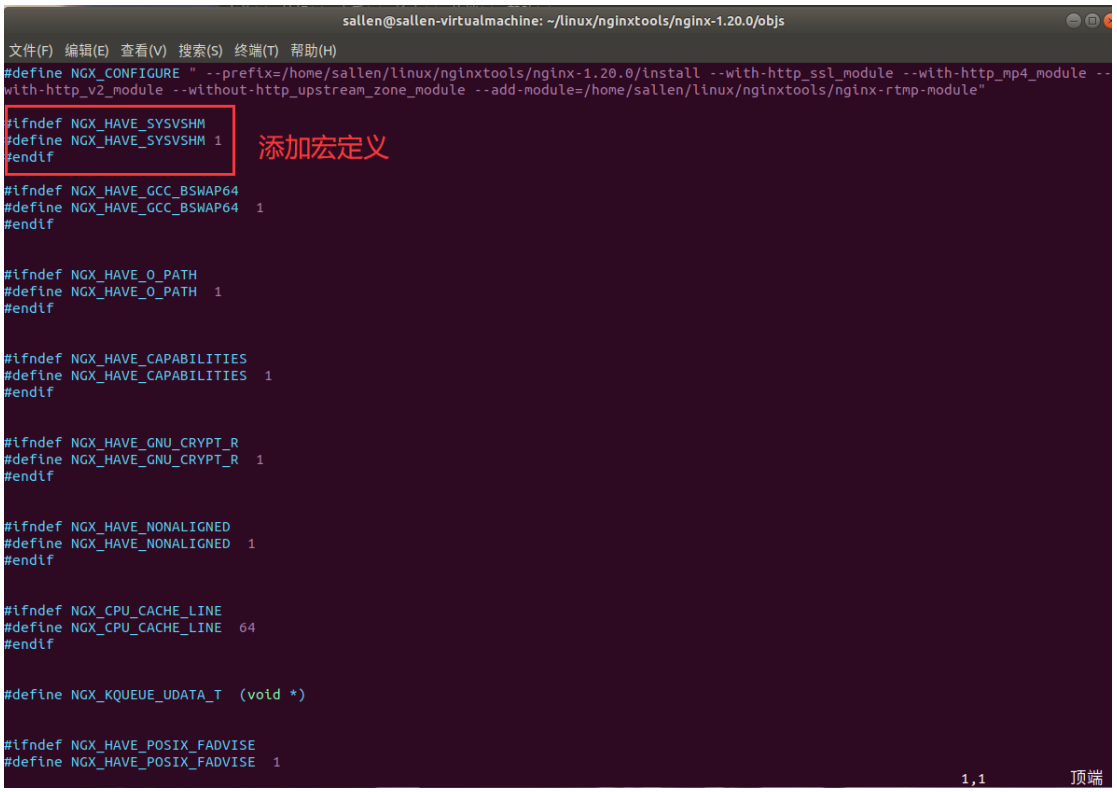


这个时候我们需要修改 nginx 源码目录下的 objs/ngx_auto_config.h 文件，将如下内容添加到该头文件中：

```
#ifndef NGX_HAVE_SYSVSHM

#define NGX_HAVE_SYSVSHM 1

#endif
```



添加完成之后保存退出，再次执行 make 编译，这样就会编译成功了。

安装

编译成功之后，接着我们进行安装，执行 make install

```
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0$ make install
make -f objs/Makefile install
make[1]: 进入目录"/home/sallen/linux/nginxtools/nginx-1.20.0"
arm-ostl-linux-gnueabi-gcc  -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=/opt/st/stm32mp1/3.1-snapshot/sysroots/cortexa
7t2hf-neon-vfpv4-ostl-linux-gnueabi -c -O2 -pipe -g -feliminate-unused-debug-types   -I src/core -I src/event -I src/event/modules -I src/os/u
nix -I /home/sallen/linux/nginxtools/nginx-rtmp-module -I objs \
        -o objs/src/core/nginx.o \
        src/core/nginx.c
arm-ostl-linux-gnueabi-gcc  -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=/opt/st/stm32mp1/3.1-snapshot/sysroots/cortexa
7t2hf-neon-vfpv4-ostl-linux-gnueabi -c -O2 -pipe -g -feliminate-unused-debug-types   -I src/core -I src/event -I src/event/modules -I src/os/u
nix -I /home/sallen/linux/nginxtools/nginx-rtmp-module -I objs \
        -o objs/src/core/ngx_log.o \
        src/core/ngx_log.c
arm-ostl-linux-gnueabi-gcc  -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=/opt/st/stm32mp1/3.1-snapshot/sysroots/cortexa
7t2hf-neon-vfpv4-ostl-linux-gnueabi -c -O2 -pipe -g -feliminate-unused-debug-types   -I src/core -I src/event -I src/event/modules -I src/os/u
nix -I /home/sallen/linux/nginxtools/nginx-rtmp-module -I objs \
        -o objs/src/core/ngx_palloc.o \
        src/core/ngx_palloc.c
arm-ostl-linux-gnueabi-gcc  -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=/opt/st/stm32mp1/3.1-snapshot/sysroots/cortexa
7t2hf-neon-vfpv4-ostl-linux-gnueabi -c -O2 -pipe -g -feliminate-unused-debug-types   -I src/core -I src/event -I src/event/modules -I src/os/u
nix -I /home/sallen/linux/nginxtools/nginx-rtmp-module -I objs \
        -o objs/src/core/ngx_array.o \
        src/core/ngx_array.c
arm-ostl-linux-gnueabi-gcc  -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=/opt/st/stm32mp1/3.1-snapshot/sysroots/cortexa
7t2hf-neon-vfpv4-ostl-linux-gnueabi -c -O2 -pipe -g -feliminate-unused-debug-types   -I src/core -I src/event -I src/event/modules -I src/os/u
nix -I /home/sallen/linux/nginxtools/nginx-rtmp-module -I objs \
        -o objs/src/core/ngx_list.o \
        src/core/ngx_list.c
arm-ostl-linux-gnueabi-gcc  -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=/opt/st/stm32mp1/3.1-snapshot/sysroots/cortexa
```

笔者将 nginx 安装到了 nginx-1.20.0 目录下的 install 目录中，进入到 install 目录下：

```
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0$ ls
auto  CHANGES  CHANGES.ru  conf  configure  contrib  html  install  LICENSE  Makefile  man  objs  README  src
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0$  cd install/
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install$ ls
conf  html  logs  sbin
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install$
```

conf 目录下有很多的配置文件，如下所示：

```
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install$ cd conf/
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install/conf$ ls
fastcgi.conf          fastcgi_params.default  mime.types          nginx.conf.default   uwsgi_params
fastcgi.conf.default  koi-utf                mime.types.default  scgi_params          uwsgi_params.default
fastcgi_params        koi-win                nginx.conf          scgi_params.default  win-utf
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install/conf$
```

其中 nginx.conf 配置文件很重要，后续我们会对该配置文件进行相应的配置。

sbin 目录下有一个可执行程序 nginx：

```
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install/sbin$ ls -lh
总用量 7.3M
-rwxrwxr-x 1 sallen sallen 7.3M 9月   7 20:57 nginx
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install/sbin$ ls -lh nginx
-rwxrwxr-x 1 sallen sallen 7.3M 9月   7 20:57 nginx
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install/sbin$
```

这个可执行文件是一个“not stripped”文件，说明该文件包含了很多的调试信息，所以导致该文件特别大 7.3MB，可以执行下面这条命令去掉调试信息、减少文件的大小：

```
arm-ostl-linux-gnueabi-strip --strip-debug nginx
```

```
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install/sbin$ arm-ostl-linux-gnueabi-strip --strip-debug nginx
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install/sbin$ ls -lh
总用量 864K
-rwxrwxr-x 1 sallen sallen 864K 9月   7 21:04 nginx
sallen@sallen-virtualmachine:~/linux/nginxtools/nginx-1.20.0/install/sbin$
```

nginx 可执行程序用于启动流媒体服务。现在我们需要将安装目录下的这些文件拷贝到开发板 Linux 系统上，再进行拷贝之前，需要先将开发板出厂系统中已经移植好的 nginx 给移除，进入到开发板 Linux 系统中，执行下面这些命令移除出厂系统自带的 nginx 程序和相应的配置文件：

```
rm -rf /usr/sbin/nginx
rm -rf /etc/nginx/*
```

接下我们将 nginx 安装目录下 sbin 中的 nginx 拷贝到开发板 Linux 系统/home/root 目录下，如下所示：

```
root@ATK-MP157:~# ls
image  nginx  projects  shell  test.mp4
root@ATK-MP157:~#
```

接着再将安装目录下的 conf、logs、html 文件夹拷贝到开发板 Linux 系统的/etc/nginx 目录下，如下所示：

```
root@ATK-MP157:/etc/nginx# ls
client_body_temp  conf  fastcgi_temp  html  logs  proxy_temp  scgi_temp  uwsgi_temp
root@ATK-MP157:/etc/nginx#
```

3.测试 nginx

我们已经将 nginx 移植到了开发板上，本小节进行测试、验证，看看 nginx 是否能够正常工作。先重启开发板，重启进入系统后，进入到/home/root 目录下，执行 nginx 程序。

查看版本信息：

```
./nginx -V
```

```
root@ATK-MP157:~# ./nginx -V
nginx version: nginx/1.20.0
built with OpenSSL 1.1.1g  21 Apr 2020
TLS SNI support enabled
configure arguments: --prefix=/home/sallen/linux/nginxtools/nginx-1.20.0/install --with-http_ssl_module --with-http_mp4_module --with-http_v2_module --without-http_upstream_zone_module --add-module=/home/sallen/linux/nginxtools/nginx-rtmp-module
root@ATK-MP157:~#
```

接下来我们要启动 nginx，执行如下命令：

./nginx -p /etc/nginx

此时 nginx 服务便在后台运行了，通过 ps 命令可查看到：

ps -aux

```
root       939  1.6  0.0      0     0 ?        I    19:57   1:21 [kworker/0:1-events]
root       963  0.0  0.0      0     0 ?        I    20:53   0:00 [kworker/1:2-events]
root       968  0.0  0.0      0     0 ?        I    21:07   0:00 [kworker/0:2-events]
root       969  0.0  0.0      0     0 ?        I    21:08   0:00 [kworker/1:0-events]
root       971  0.0  0.0      0     0 ?        I    21:14   0:00 [kworker/1:1-events]
root       973  0.0  0.0      0     0 ?        I<   21:14   0:00 [kworker/0:0H-mmc_complete]
root       978  0.0  0.0      0     0 ?        I<   21:17   0:00 [kworker/1:1H]
root       979  0.0  0.0      0     0 ?        I    21:17   0:00 [kworker/0:0-events]
root       982  0.0  0.1  12744  1776 ?        Ss   21:18   0:00 nginx: master process ./nginx -p /etc/nginx
nobody     983  0.0  0.2  13020  2456 ?        S    21:18   0:00 nginx: worker process
nobody     984  0.0  0.2  12900  2348 ?        S    21:18   0:00 nginx: cache manager process
root       985  0.0  0.2   8088  1992 ttySTM0   R+   21:18   0:00 ps -aux
root@ATK-MP157:~#
```

此时我们可以打开电脑浏览器，输入开发板的 IP 地址，如下所示：

```
root@ATK-MP157:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:04:9F:04:D2:35
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:55 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1324 (1.2 KiB)  TX bytes:1324 (1.2 KiB)

usb0      Link encap:Ethernet  HWaddr 6A:F2:50:A0:0B:43
          inet addr:192.168.7.1  Bcast:192.168.7.255  Mask:255.255.255.0
          inet6 addr: fe80::68f2:50ff:fea0:b43/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:443 errors:0 dropped:19 overruns:0 frame:0
          TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:68340 (66.7 KiB)  TX bytes:11216 (10.9 KiB)
```

如果显示出上面这个页面，表示我们的 nginx 已经正常工作了。

4.配置 nginx

后续我们会使用 FFmpeg 进行推流，将视频流通过 RTMP 推给 nginx 流媒体服务器，在此之前，我们需要对 nginx 进行配置，打开 nginx 的配置文件/etc/nginx/conf/nginx.conf，添加如下内容：

```
rtmp {
server {
listen 1935;
# 监听 1935 端口
chunk_size 4096;
application live{
allow publish 127.0.0.1;
allow play all;
live on;
# 打开直播
record off;
# 关闭 record
meta copy;
}
application hls {
live on;
```

hls on;

hls_path /tmp/hls;

hls_fragment 8s;

}

}

}

```
rtmp {
        server {
                listen 1935;
                chunk_size 4096;

                application live{
                        allow publish 127.0.0.1;
                        allow play all;
                        live on;
                        record off;
                        meta copy;
                }
                application hls {
                        live on;
                        hls on;
                        hls_path /tmp/hls;
                        hls_fragment 8s;
                }
        }
}
```

添加完成之后保存退出即可！

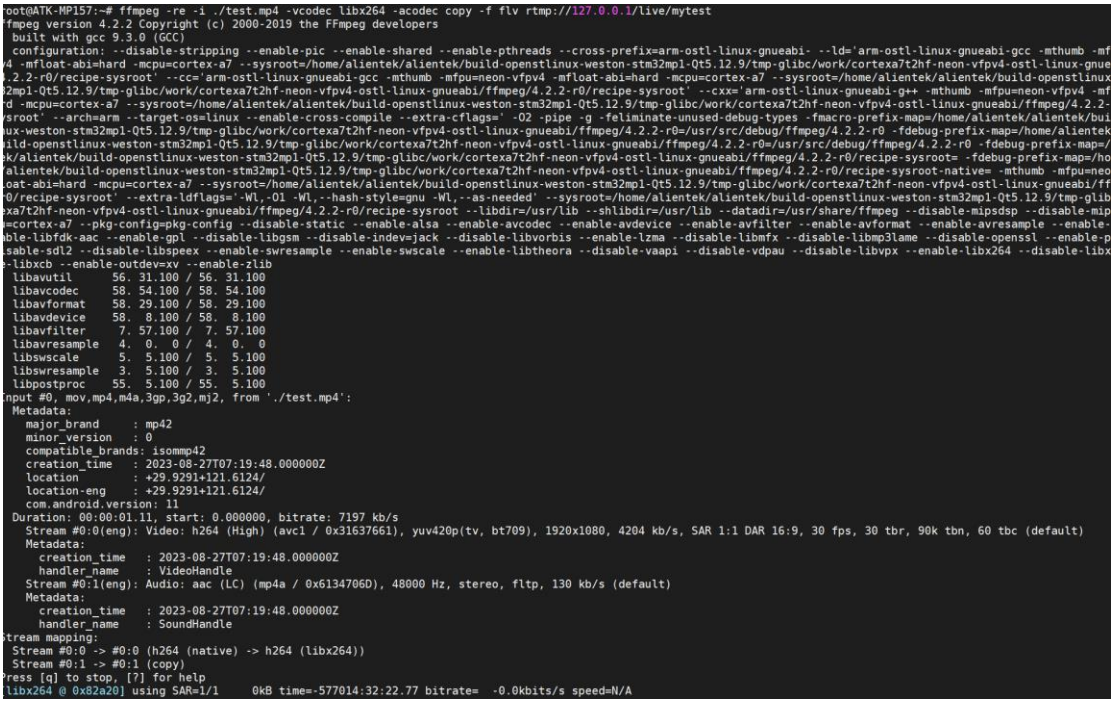
接着执行如下命令重启 nginx:

./nginx -p /etc/nginx -s reload

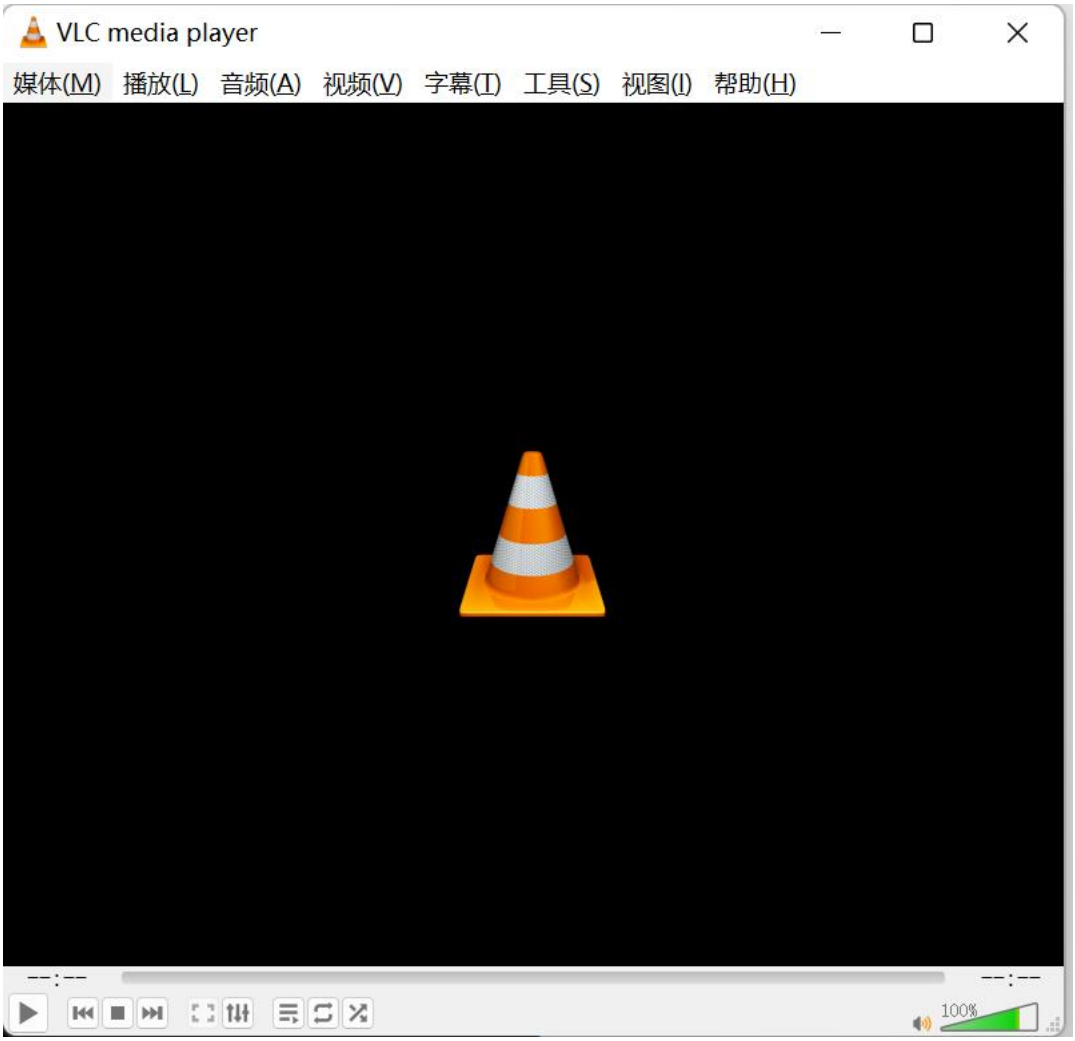
nginx 重启之后，接着我们便可使用 FFmpeg 进行推流，将视频流数据通过 RTMP 推给 nginx 流媒体服务器，执行如下命令进行推流：

ffmpeg -re -i ./test.mp4 -vcodec libx264 -acodec copy -f flv rtmp://127.0.0.1/live/mytest

简单地介绍一下这些参数，首先-i 表示输入视频数据，这里我们使用了一个 mp4 视频文件;rtmp://127.0.0.1/live/mytest 表示将视频流通过 RTMP 推给流媒体服务器,这里因为我们的服务器和推流端都是开发板，所以这个 IP 地址 127.0.0.1 指的就是本机的流媒体服务器。



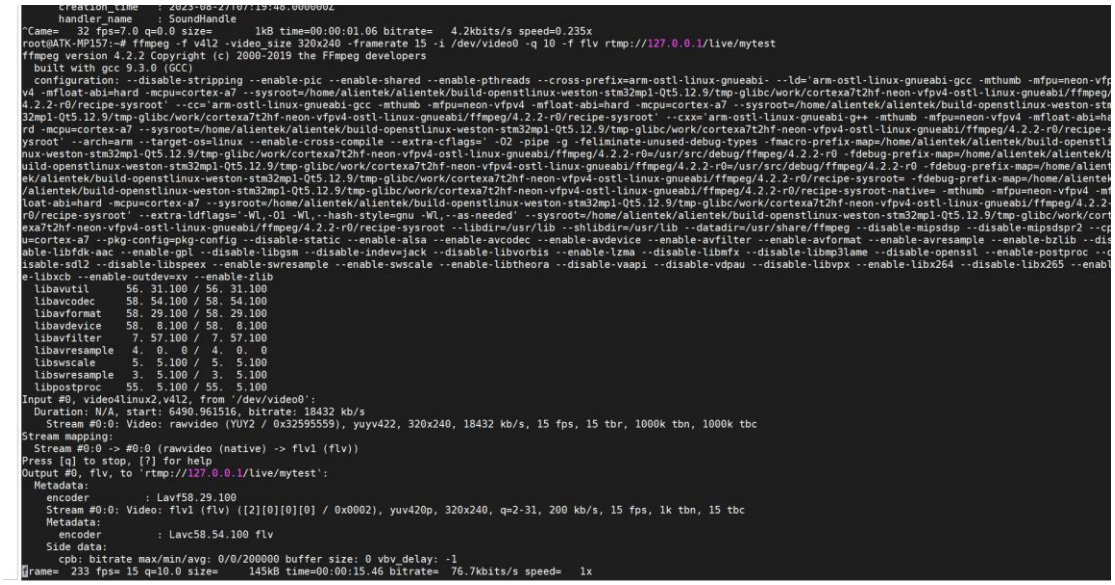
现在我们可以进行拉流了，可以将我们的 Windows 主机作为拉流端，使用 VLC 软件进行拉流，使用 VLC 软件进行拉流，VLC 软件大家自己下载、安装好。安装好之后打开 VLC，如下所示：



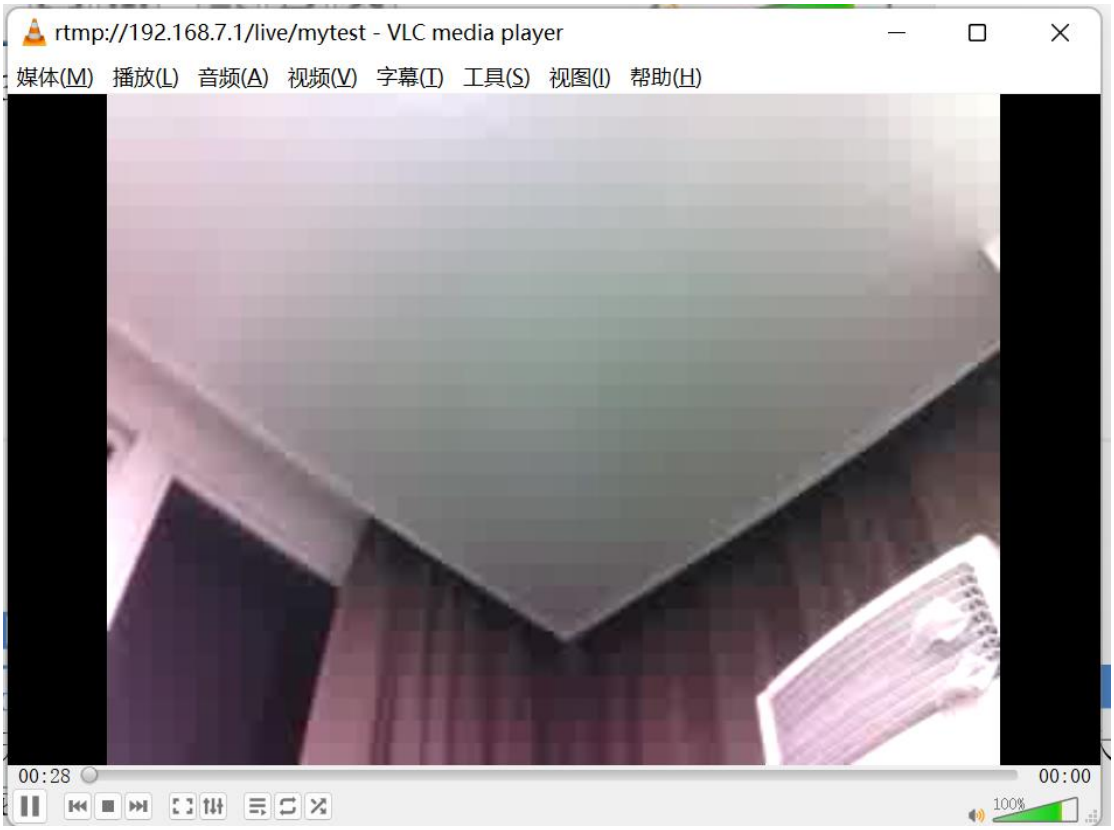
点击左上角“媒体”--->“打开网络串流”：

输入流媒体服务器的 IP 地址以及路径，笔者使用的开发板对应的 IP 地址为 192.168.7.1，点击“播放”即可从 RTMP 流媒体服务器拉取视频数据进行播放。

接下来我们再使用摄像头进行测试，使用 FFmpeg 采集摄像头视频数据将其发送给 nginx 流媒体服务器，执行如下命令：ffmpeg -f v4l2 -video_size 320x240 -framerate 15 -i /dev/video0 -q 10 -f flv rtmp://127.0.0.1/live/mytest



这里笔者的是 ov5640 摄像头进行测试，命令执行之后，接着在 Windows 下使用 VLC 拉流播放摄像头采集到的画面，实现视频摄像头监控。



测试发现，延迟太高了，导致开发板当前采集到的画面与 VLC 播放到的画面并不同步，笔者实测大概有 5、6 秒的延迟，为什么会这样呢？

开发板性能太弱了，虽然我们执行的是一条命令，但是 FFmpeg 内部却进行很多的处理，譬如对视频、音频数据的处理，由于 SoC 本就没有硬件视频解码，完全依赖于软件处理，导致会耗费相当大的时间。