1 一、什么是交叉编译

在一种计算机环境中运行的编译程序，能编译出在另外一种环境下运行的代码，我们就称这种编译器支持交叉编译。这个编译过程就叫交叉编译。

> 简单地说，就是在一个平台上生成另一个平台上的可执行代码。

这里需要注意的是所谓平台，实际上包含两个概念：

> 1. 体系结构（Architecture）、
> 2. 操作系统（OperatingSystem）。

同一个体系结构可以运行不同的操作系统；同样，同一个操作系统也可以在不同的体系结构上运行。

举例来说:

> 我们常说的x86 Linux平台实际上是Intel x86体系结构和Linux for x86操作系统的统称；而x86 WinNT平台实际上是Intel x86体系结构和Windows NT for x86操作系统的简称。

要进行交叉编译，我们需要在主机平台上安装对应的交叉编译工具链（crosscompilation tool chain），然后用这个交叉编译工具链编译我们的源代码，最终生成可在目标平台上运行的代码。常见的交叉编译例子如下：

1、在Windows PC上，利用ADS（ARM 开发环境），使用armcc编译器，则可编译出针对ARM CPU的可执行代码。

2、在Linux PC上，利用arm-linux-gcc编译器，可编译出针对Linux ARM平台的可执行代码。

3、在Windows PC上，利用cygwin环境，运行arm-elf-gcc编译器，可编译出针对ARM CPU的可执行代码。

1.1、为什么要使用交叉编译

有时是因为目的平台上不允许或不能够安装我们所需要的编译器，而我们又需要这个编译器的某些特征；

有时是因为目的平台上的资源贫乏，无法运行我们所需要编译器；

有时又是因为目的平台还没有建立，连操作系统都没有，根本谈不上运行什么编译器。

1.2 、本地编译和交叉编译的比较

本地编译：本地编译可以理解为，在当前编译平台下，编译出来的程序只能放到当前平台下运行。平时我们常见的软件开发，都是属于本地编译。比如，我们在 x86 平台上，编写程序并编译成可执行程序。这种方式下，我们使用 x86 平台上的工具，开发针对 x86 平台本身的可执行程序，这个编译过程称为本地编译。

交叉编译：交叉编译可以理解为，在当前编译平台下，编译出来的程序能运行在体系结构不同的另一种目标平台（该平台自己不能干，所以让其它平台来干）上，但是编译平台本身却不能运行该程序。比如，我们在 x86 平台上，编写程序并编译成能运行在 ARM 平台的程序，编译得到的程序在 x86 平台上是不能运行的，必须放到 ARM 平台上才能运行。

2 Clang 交叉编译

添加NDK编译环境变量

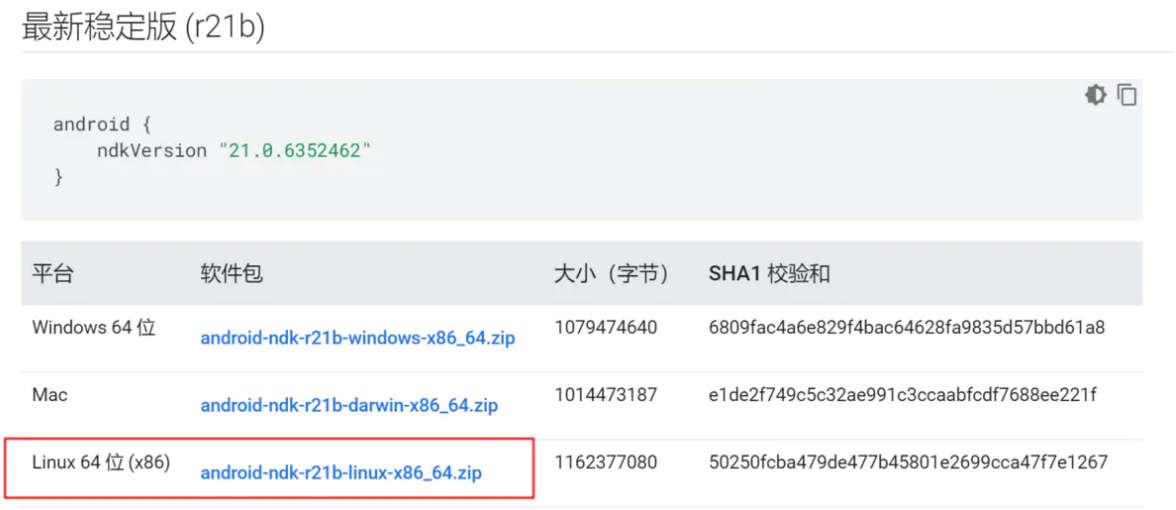
2.1 下载NDK

> wget https://dl.google.com/android/repository/android-ndk-r21d-linux-x86_64.zip

2.2 安装NDK

从下面的链接下载NDK，并解压：
https://developer.android.google.cn/ndk/downloads/

这里下载了 android-ndk-r21b，解压到 /home/temp/programs/android-ndk-r21b



最新稳定版 (r21b)

```
android {
    ndkVersion "21.0.6352462"
}
```

平台	软件包	大小（字节）	SHA1 校验和
Windows 64 位	android-ndk-r21b-windows-x86_64.zip	1079474640	6809fac4a6e829f4bac64628fa9835d57bbd61a8
Mac	android-ndk-r21b-darwin-x86_64.zip	1014473187	e1de2f749c5c32ae991c3ccaabfcdf7688ee221f
Linux 64 位 (x86)	android-ndk-r21b-linux-x86_64.zip	1162377080	50250fcba479de477b45801e2699cca47f7e1267

2.3 添加系统环境变量

> vim etc/profile
>
> export PATH=$PATH:/root/ndk/android-ndk-r21d
> export SYSROOT="$NDK/toolchains/llvm/prebuilt/linux-x86_64/sysroot/"
> export ANDROID_GCC="$NDK/toolchains/llvm/prebuilt/linux-x86_64/bin/x86_64-linux-android24-clang"

2.4 生效环境变量

source /etc/profile

```
$NDK/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android26-clang++ main.cpp -o hello
```

这里NDK用的是r19及以上的版本。

2.5. 写main.cpp文件

```
#include <stdio.h>

int main()
{
    printf("hello world\n");
    return 0;
}
```

2.6. 写交叉编译脚本 generate.sh

由于命令比较短，也可直接在命令行里写。
新建generate.sh，并给执行权限： `chmod +x generate.sh`

```
export NDK=/home/temp/programs/android-ndk-r21b

$NDK/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android26-clang++
main.cpp -o hello
```

使用了NDK 默认安装的工具链，按照官网说明，NDK在r21之后，NDK 默认安装的工具链可供使用。
可以不需要使用 make_standalone_toolchain.py 脚本生成独立工具链来使用。
这样使用自带的工具链就比较方便，不用再配置`sysroot`等编译选项。

其中NDK为自己解压的目录。
编译器要选择自己手机的架构，这里用的是arm64，所以是aarch64-linux-android。
编译器要选择android的api版本，这里用的是anroid 8.0.0，对应api是26。
Android NDK从r13起，默认使用Clang进行编译。

交叉编译出可执行程序`hello`

```
./generate.sh
```

2.7 放到手机上执行

2.8 push到手机

```
adb push hello /data/local/tmp
```

2.9给hello执行权限

```
adb shell
cd /data/local/tmp
chmod +x hello
```

2. 9.1执行hello

```
./hello
```

可以看到输出

```
hello world
```

3 X264

x264是一个开源的H.264/MPEG-4 AVC视频编码函数库，是最好的有损视频编码器之一。 它将作为我们直播数据的视频编码库。

> FFmpeg中同样实现了H.264的编码，同时FFmpeg也能够集成X264。本次我们将直接使用X264来进行视 频编码而不是FFmpeg

X264 主页

下载源码:(前提是已经安装git并存在环境变量)

```
git clone https://code.videolan.org/videolan/x264.git
```

并不是所有的库都已经存在configure文件，可能只存在configure.ac和makefile.am。这种情况需要借助autoconf来生成configure。

```
cd x264
ls
```

发现已经帮助我们生成好了configuration文件

如果存在configure，这时候我们第一反应一定是执行help

```
./configure --help
```

- `--prefix` 设置编译结果输出目录，一般规范都会存在这个参数
- `--exec-prefix` 参数表示我们可以借助这个对编译器（gcc/clang)设置选项 「类似javac设置-classpath」
- `--disable-cli` 这个配置是关闭编译命令行工具，在Android，是自己编译代码，用不到，也可以不管，倒库的是后不用就好
- `--enable-shared & --enable-static` 这两个参数是指我们编译成静态库.a 还是动态库.so

```
静态库-> xxxx.c 生成xxxx.a 里面函数
编译时，从xxxx.a 中找到这个函数，与xxxx.c 一起生成一个 a.so
最终 可能 .a 库 10M

动态库-> xxxx.a 生成xxxx.so 里面函数
编译时，只会找有没有这个函数，有就行不做其他操作
运行时，当a.so  执行到这个函数，在动态去找对应的函数
最终，会有两个so .so 6M   a.so 5M. 加起来会稍微大一点


多个库情况，a.so  和 b.so 都需要x264
那么这个时候我们应该选择动态库。 静态库会编译两份，动态库是用到时候去找可以公用一份。
```

- `--enable-debug & --enable-gprof & --enable-strip` 给编译器传递参数 相当 gcc -g / clang -g
- `--enable-pic` 如果编译Android使用的动态库,使用PIC 指令。有的--with-pic. 一些脚本没有提供 那么我们可以加上gcc -fPIC 通过 CFLAGS 变量 CFLAGS="-g -fPIC -xxx" 开启
  x264 还给我们提供了跨平台的参数Configuration options:
  `--cross-prefix` --cross-prefix=前缀- 那么我们相当于使用"gcc xxx.c" 就会用 "前缀-gcc xxxx.c" 编译
  `--sysroot` 查找库，有点类似-L 使用。但是有区别
  `--extra-cflags` 作为传递给编译器的参数，所以就算有些库没有--extra-cflags配置，我们也可以自己创建变量cFLAGS传参

```
比如指定了搜索路径/abc/
会在 指定的路径/abc/usr/lib/libxxx.so(libxxx.a)
/abc/usr/include/xxx.h

gcc -Lxxx -Ixxx
gcc -Labc
会在 /abc/libxxx.so(libxxx.a)
```

根据上面的分析我们形成配置脚本, 对于android 我们需要用ndk编译。但是ndk19 以上移除了gcc，高版本我们需要用clang编译工具。在toolchains\llvm 下面可以找到
`/Users/xxx/Android/.../sdk/ndk-bundle/toolchains/llvm/prebuilt/darwin-x86_64/bin/` 类似路径，我的是Macbook路径
我们如何指定clang 编译器。那么需要给编译器变量CC 的编译器变量

printf -> 这种实现在哪？ 标准类库中 好比（java->jdk/rt.jar -> java 官方提供的库）
那么需要我们 c/c++ -> NDK 中的头文件与库 才能给Android中使用

4.1.1 linux脚本

> $NDK/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android26-clang++ main.cpp -o hello

```
#!/bin/bash
```

```
export TOOLCHAIN=$NDK/toolchains/llvm/prebuilt/linux-x86_64
export API=21

function build
{
  ./configure \
    --prefix=$PREFIX \
    --disable-cli \
    --enable-static \
    --enable-pic \
    --host=$HOST \
    --cross-prefix=$CROSS_PREFIX \
    --sysroot=$NDK/toolchains/llvm/prebuilt/linux-x86_64/sysroot \

        make clean
        make -j8
        make install
}
#armeabi-v7a
PREFIX=./armeabi-v7a
HOST=armv7a-linux-android
export TARGET=armv7a-linux-androideabi
export CC=$TOOLCHAIN/bin/$TARGET$API-clang
export CXX=$TOOLCHAIN/bin/$TARGET$API-clang++
export CROSS_PREFIX=$TOOLCHAIN/bin/arm-linux-androideabi-

build
```

如需要编译arm64-v8a架构版本，则修改以下变量：

```
#arm64-v8a
PREFIX=./android/arm64-v8a
HOST=aarch64-linux-android
export TARGET=aarch64-linux-android
export CC=$TOOLCHAIN/bin/$TARGET$API-clang
export CXX=$TOOLCHAIN/bin/$TARGET$API-clang++
CROSS_PREFIX=$TOOLCHAIN/bin/aarch64-linux-android-
```

4.1.2 mac 脚本

```
#!/bin/bash
# NDK目录
NDK_ROOT=/Users/xxx/Android/android_SDK/sdk/ndk/21.1.6352462
#编译后安装位置 pwd表示当前目录
PREFIX=`pwd`/android/armeabi-v7a
#目标平台版本,我们将兼容到android-21
API=21
#编译工具链目录
TOOLCHAIN=$NDK_ROOT/toolchains/llvm/prebuilt/darwin-x86_64

#小技巧，创建一个AS的NDK工程，执行编译，
#然后在 app/.cxx/cmake/debug(release)/自己要编译的平台/ 目录下自己观察 build.ninja与
rules.ninja
```

```
#虽然x264提供了交叉编译配置:--cross-prefix，如--corss-prefix=/NDK/arm-linux-androideabi-
#那么则会使用 /NDK/arm-linux-androideabi-gcc 来编译
#然而ndk19开始gcc已经被移除，由clang替代。
# 小常识:一般的库都会使用$CC 变量来保存编译器，我们自己设置CC变量的值为clang。

export CC=$TOOLCHAIN/bin/armv7a-linux-androideabi$API-clang
export CXX=$TOOLCHAIN/bin/armv7a-linux-androideabi$API-clang++


#--extra-cflags会附加到CFLAGS 变量之后，作为传递给编译器的参数，所以就算有些库没有--extra-cflags配置，我们也可以自己创建变量cFLAGS传参
FLAGS="--target=armv7-none-linux-androideabi21 --gcc-toolchain=${TOOLCHAIN}  -g -DANDROID -fdata-sections -ffunction-sections -funwind-tables -fstack-protector-strong -no-canonical-prefixes -D_FORTIFY_SOURCE=2 -march=armv7-a -mthumb -Wformat -Werror=format-security   -Oz -DNDEBUG  -fPIC "


# echo ${FLAGS}


# prefix: 指定编译结果的保存目录 `pwd`: 当前目录
./configure --prefix=${PREFIX} \
--disable-cli \
--enable-static \
--enable-pic=no \
--host=arm-linux \
--cross-prefix=${TOOLCHAIN}/bin/arm-linux-androideabi- \
--sysroot=${TOOLCHAIN}/sysroot \
--extra-cflags="$cleFLAGS"

make install
```

FAAC

FAAC是一个MPEG-4和MPEG-2的AAC编码器，我们将使用它作为音频编码库。在Linux/Mac中下载源码:

```
wget https://nchc.dl.sourceforge.net/project/faac/faac-src/faac-1.29/faac-1.29.9.2.tar.gz
# 解压
tar xvf  faac-1.29.9.2.tar.gz
# 进入facc目录
cd faac-1.29.9.2
```

类似x2264我们编译FAAC 了

CameraX

CameraX 是 Android Jetpack 的新增功能，通过该功能，向应用添加相机功能变得更加容易。该库提供了很多兼容性修复程序和解决方法，有助于在很多设备上打造一致的开发者体验。关于CameraX的使用在官网有详细文档及 Example

我们需要获取摄像头的数据自行编码，需要使用分析图片功能。CameraX获取数据格式为YUV_420_888，此数据 被包含在Image中，从Image取出YUV数据，