AOSP构建、编译基础理解

构建系统

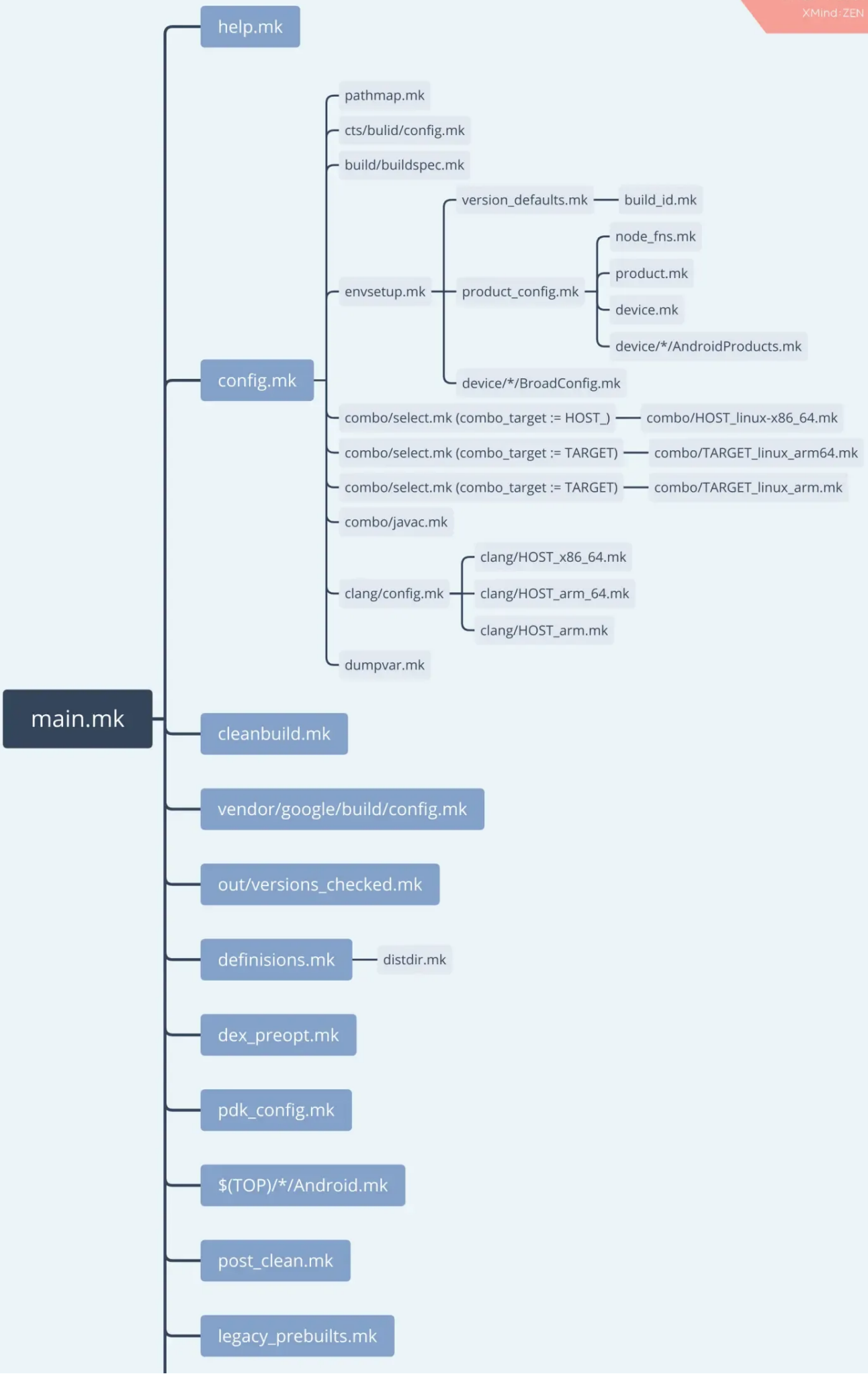
参考这篇文章，写的比较好，我就不狗尾续貂了！
android build system编译系统概述

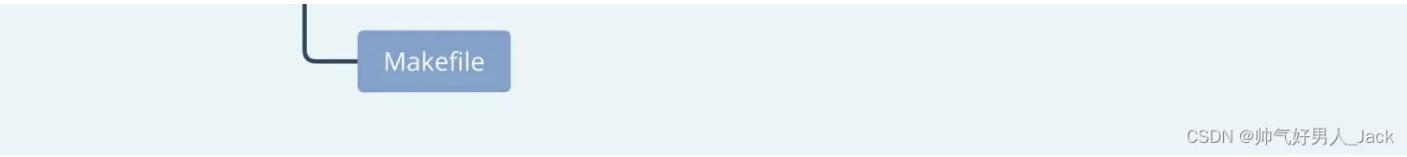
source build/envsetup.sh之后的事情

source也就是执行build/envsetup.sh里面的脚本，改脚本定义许多命令，比如lunch命令选择编译某个产品，同时它内部调用include或者inherit-product包含其他的mk文件，其他mk文件又包含了另外的mk，这样层层包含就依次展开了；

```
$(call inherit-product, device/google/common/common.mk)
或
include $(BUILD_SYSTEM)/version_defaults.mk
```

当执行了envsetup.sh后，整个系统的mk文件包含大致如下：





如果我们遇到不清楚某个mk是如何被包含到build构建时，就可以查看这个图，然后grep他的目录看看是如何被包含进去的

如何创建一个新的编译产品product?

一般是在device/company/产品ming/下，创建AndroidProducts.mk，在里面配置两个属性即可：

```
PRODUCT_MAKEFILES := \
        $(LOCAL_DIR)/makefiles.mk
COMMON_LUNCH_CHOICES := \
    product_name
```

COMMON_LUNCH_CHOICES ： 表示lunch时选择的产品名
PRODUCT_MAKEFILES：是构建此product，需要的mk配置文件
当然，不仅仅如此，要做的工作更多，需要你去编写makefiles是如何去编译的；

如何为系统添加property属性?

用如下命令即可：

```
PRODUCT_PROPERTY_OVERRIDES += \
        com.jack.perproty = 12345
```

但是在哪个文件添加呢？如上如何创建一个新的编译产品product？章节，在makefiles.mk里面添加即可，或者在其他mk，只要被makefile.mk包含的mk文件添加即可生效

文件拷贝PRODUCT_COPY_FILES

```
PRODUCT_COPY_FILES += device/qcom/media/demo.xml:system/etc/demo.xml
```

在哪个文件写入这个命令呢？同上个命令一样，被AndroidProducts.mk中编译配置mk包含的即可！

如何把apk编译进入系统?

这个很简单！在被AndroidProducts.mk包含的mk中假如：

```
PRODUCT_PACKAGES += \
    apps \
```

其中apps表示你要编译进入系统的名字；
当然，这里还没完，你还得写编译apk的mk文件，这里的apk可以分为编译apk文件和编译apk的源代码，

先说说编译apk，Android.mk和apk在同级目录，如下文件：

```
LOCAL_PATH:=$(call my-dir)
include $(CLEAR_VARS)
#JAVA_LIBRAYIES jar包 SHARED_LIBRAYIES so库 EXECUTABLES二进制可执行文件
LOCAL_MODULE_CLASS := APPS
#可以为user、eng、tests、optional，optional代表在任何版本下都编译
LOCAL_MODULE_TAGS := optional
#编译模块的名称
LOCAL_MODULE := CarSkin
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk
#可以为testkey、platform、shared、media、PRESIGNED（使用原签名），platform代表为系统
LOCAL_CERTIFICATE := PRESIGNED
#不设置或者设置为false，安装位置为system/app，如果设置为true，则安装位置为system/priv-
LOCAL_PRIVILEGED_MODULE := false
#module的后缀，可不设置
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
# 关闭预编译，不会生成OAT文件
LOCAL_DEX_PREOPT := true
# 可忽略，设置后将会安装到product分区
LOCAL_PRODUCT_MODULE := true
include $(BUILD_PREBUILT)
```

如果App要安装到data目录，则需要将上面LOCAL_PRIVILEGED_MODULE行替换为：

```
LOCAL_MODULE_PATH := $(TARGET_OUT_DATA_APPS)
```

某些特殊的apk可能需要依赖第三方lib，则可以通过PRODUCT_COPY_FILES将lib拷贝到对应的系统目录下，如system/priv-app/apk名字/lib下面

编译apk源代码

首先，需要将java源代码，依赖的第三方jar包、so拷贝到同一个目录，然后编写如下的Android.mk，

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
#递归调用java文件
LOCAL_SRC_FILES := $(call all-subdir-java-files)
LOCAL_PACKAGE_NAME := Test
#第三方jar依赖
LOCAL_PREBUILT_STATIC_JAVA_LIBRARIES := AndroidUtil:libs/AndroidUtil.jar
include $(BUILD_PACKAGE)
```

第三方引用可以参考此链接

kernel/arch/内有多种如arm、 arm64 、x86，如何选择哪种cpu架构呢?

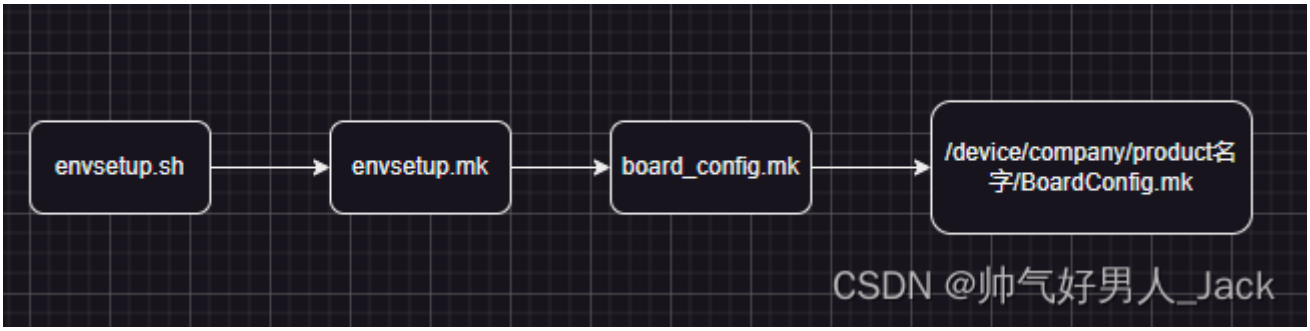
由TARGET_ARCH宏指定的，如：

```
TARGET_ARCH=arm64
```

那这个指令应该写在哪个mk文件呢？当然是在BoardConfig.mk，这个文件位于/device/company/产品名目录下

那这个文件是如何被引用到构建系统中了呢?

大致如下图的路径：



其中board_config.mk加入BoardConfig.mk的语法如下：

```
$(shell test -d device && find -L device -maxdepth 4 -path '*/$(TARGET_DEVICE)/Bo
```

这里面的TARGET_DEVICE也就是我们选择lunch时的产品名字，他的定义可以在envsetup.sh中找到；

最后言归正传，BoardConfig.mk定义了哪些东西?

```
TARGET_ARCH := arm64  #CPU架构，比较宽泛，如x86、arm等
TARGET_ARCH_VARIANT := armv8-a #CPU架构，主版本，详细定义

```

```
TARGET_CPU_VARIANT := cortex-a72       #cpu主板下的详细定义
TARGET_CPU_ABI := arm64-v8a    #CPU的指令集
TARGET_CPU_ABI2 :=
TARGET_2ND_ARCH := arm         #同上，不过是次级的
TARGET_2ND_ARCH_VARIANT := armv8-a
TARGET_2ND_CPU_VARIANT := cortex-a53
TARGET_2ND_CPU_ABI := armeabi-v7a
TARGET_2ND_CPU_ABI2 := armeabi

TARGET_SUPPORTS_32_BIT_APPS := true #是否支持32、64位的应用程序
TARGET_SUPPORTS_64_BIT_APPS := true
```

内核编译配置文件

Android内核编译一般是cd到kernel目录，执行如下：

```
make ARCH=arm64 android10_defconfig
```

输入后，会自动执行当前目录下的Makefile，android10_defconfig文件是包含了编译需要的配置参数，那这个文件在哪儿呢？在
/kernel/arch/arm64/configs下面，执行make后他就会去这个目录下找，如果要替换成其他的就换个配置文件把！

内核编译obj-$(xx) := xx.o

一般来说，如上xx代表一个变量，这个变量通常定义在kernel/arch/ARCH/configs文件中，而xx取值有可能是y或者m，替换到标题的就是obj-y := xx.o或者obj-m := xx.o

- y 表示把xx.o编译成静态模块到镜像中
- m则是xx.o编译成动态库.ko文件，在开机时，你需要insmod 命令把ko文件加载到系统中去

dts 文件解读

引用节点

在dts设备中，我们要使用如 i2s 某个节点时，直接用

```
&i2s_1
```

就可以引用i2s_1这个节点，**但是如果我们要往这个节点添加内容呢?**
一般来说，如果在同一个dts文件，直接在i2s_1这个节点添加内容即可，如果在不同文件，可以引用

追加，如下：

```
&i2s_1{
        xx : xxx
}
```

device_node的name取值是多少?

```
i2s0: i2s@16101000 {
        compatible = "qcom,i2s";
                ....
}
```

name等于compatible的i2s