

Android7.0 Ninja编译原理

原创

2016年11月27日 06:52:04

标签：ninja

16380



5

#####

本文为极度寒冰原创，转载请注明出处

#####



引言

使在Android N的系统中，初次使用了Ninja的编译系统。对于Ninja，最初的印象是用在了Chromium open source code的编译中，在chromium的编译环境中，使用ninja -C out/Default chrome命令，就可以利用源码编译出chrome的apk。对使用者而言，抛开对原理的探究，最直观的印象莫过于可以清楚的看到自己当前编译的进度。同时，对android而言，也可以感受到编译速度的提升带来的便捷。本文将深入分析Ninja的编译原理，以及android上面的编译改变。

正因为这个改变，所以在编译android N的code的时候需要使用OpenJDK8

编译系统的内存最少需要12G，建议16G，否则会出现JVM不足的错误。

8G内存的机器可以通过增大JVM默认值的方法来解决，但是经过测试，还是会偶尔出现JVM不足的错误

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！



极度寒冰

原创

46

粉丝

28

喜欢

1

评论

22

等级：博客 4

访问量：9万+

积分：1391

排名：3万+



九十平米装修



他的最新文章


更多

登录

注册

概念简介

名词：

Ninja	 5	Blueprint	Soong
-------	--	-----------	-------



Ninja



Ninja是一个致力于速度的小型编译系统（类似于Make）；

如果把其他编译系统比做高级语言的话，Ninja就是汇编语言

主要有两个特点：

- 1、可以通过其他高级的编译系统生成其输入文件；
- 2、它的设计就是为了更快的编译；

使用Kati把makefile转换成Ninja files，然后用Ninja编译

在不久的将来，当不再用Makefile（Android.mk）时，Kati将被去掉

ninja核心是由C/C++编写的，同时有一部分辅助功能由python和shell实现。由于其开源性，所以可以利用ninja的开源代码进行各种个性化的编译定制。

共享内存映射之mmap()函数详解

Android Native Crash 堆栈转换

Android M PackageManagerService解析

android xml解析XmlPullParser的使用

文章存档

2017年9月	1篇
2016年11月	1篇
2016年3月	2篇
2016年2月	1篇
2015年4月	3篇
2015年3月	12篇

展开

他的热门文章

Android7.0 Ninja编译原理

16318

android中SELINUX规则分析和语法简介

5758

Android DownloadProvider 源码分析

4797


加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

Github地址：<https://github.com/ninja-build/ninja>

Blueprint, Soong

Blueprint和Soong是  一起把Blueprint 文件转换为Ninja文件。将来需要写Blueprint文件（Android.bp），转换为Android.soong.mk（也可以直接写），然后转换为Ninja文件（build.ninja）然后用Ninja编译。

如果Android.mk和Android.bp同时存在，Android.mk会被忽略。

如果Android.bp的同级目录下有Android.soong.mk也会被include

1.ckati可执行文件的生成

在android系统中，目前还未完全切换到Ninja编译，编译的入口仍然是make命令，如下commands以nexus为例：

```
source build/envsetup.sh
```

```
choosecombo
```

```
make -j4
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

4116

android 中chromium_org模块打log的方法

2735

Android4.4 chromium_org研究报告

2657

android browser 的几个小feature (四) kitkat上实现UaProfile的设置

2654

使用android源码编译并烧LG nexus4

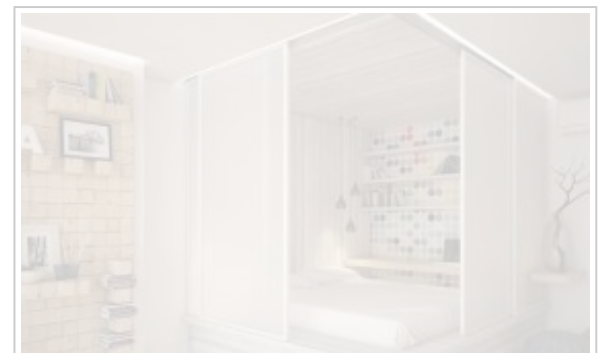
2491

android xml解析XmlPullParser的使用

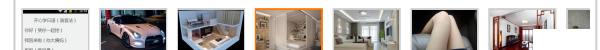
2300

Android webkit image的加载过程解析（一）

2269



45平小户型装修



登录

注册

既然是make，那就在编译中首先include到的就是build/core/main.mk了，在main.mk中，我们可以清楚的看到对Ninja的调用：

```
relaunch_with_ninja :=
```

```
ifneq $(USE_NINJA),false)
```

```
ifndef BUILDING_
```



```
_NINJA
```

5

```
relaunch_with_ninja := true
```

```
endif
```



```
endif
```



由于USE_NINJA默认没有定义，所以一定会进入到这个选项中，并且将relaunch_with_ninja置为true。这样的话，就会进入到下面的重要操作语句，去include ninja的makefile。并且在out目录下生成ninja_build的文件，显示当前是使用了ninja的编译系统。

```
ifeq $(relaunch_with_ninja),true)
```

```
# Mark this is a ninjabuild.
```

```
$(shell mkdir -p $(OUT_DIR)&& touch $(OUT_DIR)/ninja_build)
```

```
.....
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

🔔 QQ客服 🗣 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

登录

注册

```
ifndef BUILDING_WITH_NINJA
```

```
# Remove ninja build mark if it exists.
```

```
$(shell rm -f $(OUT_DIR)/ninja_build)
```

```
endif
```



在`include build/core/ninja.mk`的语句执行后，我们就可以看到真正定义ninja的地方了。由于前面简介讲了ninja是基于项目编译出来的轻便的编译工具，所以这边google肯定也对ninja进行了修改，编译，并且最终生成了一个可执行的应用程序。在simba6项目中，我们可以在prebuilts/ninja/linux-x86下面找到这个可执行的应用程序ninja。我们可以简单的运行这个ninja的命令，比如ninja -h，就可以了解到这个command的基本用法，也可以看到本版本的ninja使用的base version为1.6.0。

./ninja -h

usage: ninja [options][targets...]

if targets are unspecified, builds the 'default' target (see manual).

options:

--version print ninja version ("1.6.0")

-C DIR change to DIR before doing anything else

-f FILE specify input build file [default=build.ninja]

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

-k N keep going until N jobs fail [default=1]

-l N do not start new jobs if the load average is greater than N

-n dry run (don't run commands but act like they succeeded)

-v show all command lines while building

-d MODE enable debugging (use -d list to list modes)

-t TOOL run a subtool (use -t list to list subtools)

terminates toplevel options; further flags are passed to the tool

-w FLAG adjust warnings (use -w list to list warnings)

在声明了ninja可执行程序的路径之后，紧接着在mk中就提及了kati的定义。

KATI ?= \$(HOST_OUT_EXECUTABLES)/ckati

目标KATI是利用源码编译生成的一个可执行的程序。源码在build/kati文件夹中。这同样是一个开源的代码。

开源网站的地址为：

<https://github.com/google/kati>

我们可以clone下来最新的code，或者在源码build/kati中直接按下面的步骤来编译ckati的可执行程序。

一、添加软件源

```
sudoadd-apt-repository ppa:ubuntu-toolchain-r/test
```

```
sudo apt-get upd
```



二、安装版本的命令：

5

```
sudo apt-get insta -4.8 g++-4.8
```



三、查看本地版本



四、切换版本

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.660
```

```
sudo update-alternatives --install /usr/bin/gcc gcc/usr/bin/gcc-4.8 40
```

```
sudo update-alternatives --install /usr/bin/g++ g++/usr/bin/g++-4.6 60
```

```
sudo update-alternatives --install /usr/bin/g++ g++/usr/bin/g++-4.8 40
```

这里的4.6是你本机之前的版本。

```
sudo update-alternatives --config gcc
```

```
sudo update-alternatives --config g++
```

选择你需要的版4.8.

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

当然，android源码是利用makefile来做的，我们可以看到ninja.mk中对ckati的makefile进行了调用。

文件地址：[include build/kati/Makefile.ckati](#)

在kati的makefile中，我们可以看到真正去编译kati的过程。

```
# Rule to build ckati into KATI_BIN_PATH
H

$(KATI_BIN_PATH)ti:$(KATI_CXX_OBJS) $(KATI_CXX_GENERATED_OBJS)
S)

@mkdir -p $(dir $@)

$(KATI_LD) -std=c++11 $(KATI_CXXFLAGS) -o$@ $^ $(KATI_LIBS)

# Rule to build normal sourcefiles into object files in KATI_INTERMEDIATES_PATH
H

$(KATI_CXX_OBJS) :$(KATI_INTERMEDIATES_PATH)/%.o: $(KATI_SRC_PATH)/%.c
C

@mkdir -p $(dir $@)

$(KATI_CXX) -c -std=c++11 $(KATI_CXXFLAGS) -o $@ $<
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

Rule to build generatedsource files into object files in KATI_INTERMEDIATES_PATH

\$(KATI_CXX_GENERATED_OBJS):\$(KATI_INTERMEDIATES_PATH)/%.o: \$(KATI_INTERMEDIATES_PATH)/%.cc

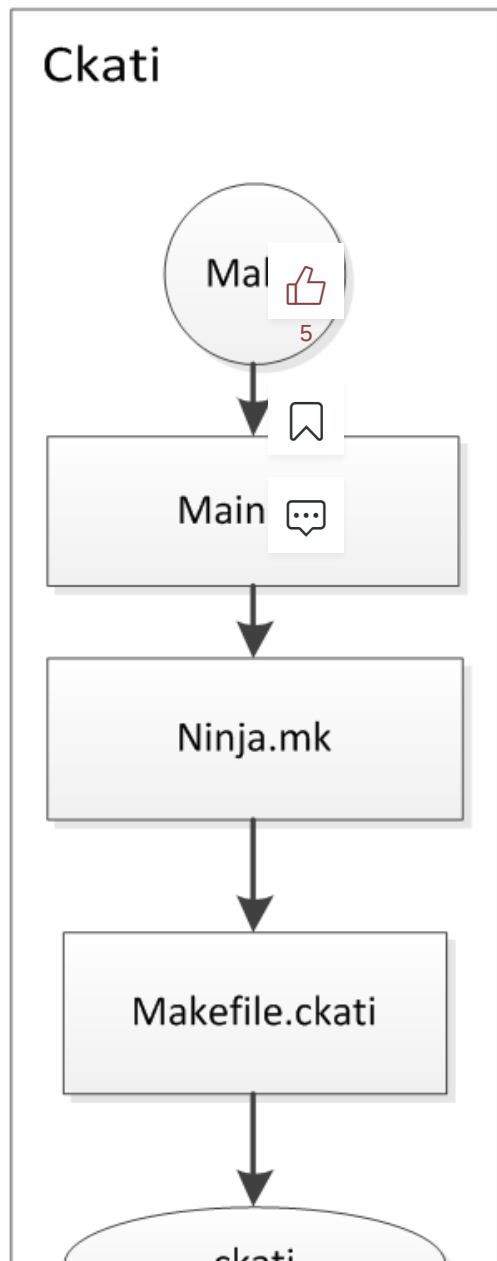
@mkdir -p \$(dir \$@)

\$(KATI_CXX)-c -std=c++11 \$(KATI_CXXFLAGS) -o \$@ \$<

这个调用简单解释一下就是编译kati是需要依赖与KATI_CXX_OBJS和 KATI_CXX_GENERATED_OBJS这两个变量。KATI_CXX_OBJS的生成依赖于 KATI_INTERMEDIATES_PATH下的.o，而.o文件的生成又依赖与KATI_INTERMEDIATES_PATH下的.cc文件。在生成了所有依赖的.o文件之后，会link成编译所需的ckati文件。具体的命令为：**\$(KATI_LD) -std=c++11\$(KATI_CXXFLAGS) -o \$@ \$^ \$(KATI_LIBS)**

这样的话，就完成了ckati可执行文件的生成。

流程图可以简单归结如下：



<http://blog.csdn.net/>

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

2. 解析并使用ninja

ckati文件生成之后，我们接着来看是如何使用的。

接着回到ninja.mk文件中，如下是具体的调用。

```
$(KATI_BUILD_NINJA) : $(KATI) $(MAKEPARALLEL)$(DUMMY_OUT_MKS) $(SOONG_ANDROID_MK) FORCE
```

```
@echo Running $(KATI) to generate build$(KATI_NINJA_SUFFIX).ninja...
$(hide)$(KATI_MAKEPARALLEL) $(KATI) --ninja --ninja_dir=$(OUT_DIR)--ninja_suffix=$(KATI_NINJA_SUFFIX) --no_ignore_dirty=$(OUT_DIR)/%--no_ignore_dirty=$(SOONG_ANDROID_MK) --ignore_dirty=$(OUT_DIR)/%.P--detect_android_echo $(KATI_FIND_EMULATOR) -f build/core/main.mk $(KATI_GOALS)--gen_all_targets BUILDING_WITH_NINJA=true SOONG_ANDROID_MK=$(SOONG_ANDROID_MK)
```

可以看到使用kati，并且将很多的参数传入到了ckati中。

在kati文件的主函数中，可以看到接受了这些参数并且进行处理。

文件地址：[build/kati/main.cc](#)

main函数：

```
int main(int argc, char*argv[]) {

    if (argc >= 2 && !strcmp(argv[1], "--realpath")) {
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录





注册

```
    return 0;

}

Init();

string orig_args;

for (int i = 0; i < a  ++){
    
    if (i)
        
        orig_args += ' ';
        
        orig_args += argv[i],
}

g_flags.Parse(argc, argv);

FindFirstMakefile();

if (g_flags.makefile == NULL)

    ERROR("*** No targets specified and nomakefile found.");

// This depends on command line flags.

if (g_flags.use_find_emulator)

    InitFindEmulator();
```

```
Quit();

return r;

}
```

argv接受到了传入的参数後，经过处理，转化为了string，传入orig_args变量，并且调用Run函数来进行后续的处理。F 数是kati程序的核心，用于各种文件的生成，流程的执行以及处理。我们这边只对重点内容进行分析。

任何的编译都脱离了环境变量的支持，在编译的第一步，肯定要对环境变量进行设置。

在run函数的开始，使用Linux标准C接口来进行了环境变量的读取和设置。

具体操作为：

```
extern "C" char**environ;

...

for (char** p = environ; *p; p++) {

    SetVar(*p, VarOrigin::ENVIRONMENT);

}
```

如果我们printf打印*p的值，可以很清楚的看到该环境变量的设置。这边只截取部分环境变量用于说明该问题：

```
*p =XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
```

```
*p =BUILD_ENV_SEQUENCE_NUMBER=10
```

```
*p =XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
```

```
*p =ANDROID_BUILD_PATHS=/data/android_N/out/host/linux-x86/bin:/data/android_N/prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9/bin:/data/android_N/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9/bin:/data/android_N/development/scripts:/data/android_N/prebuilts/devtools/tools:/data/android_N/external/selinux/prebuilts/bin:/data/android_N/prebuilts/android-  
ator/linux-x86_64:
```

```
*p = SSH_AUTH_SOCK=/tmp/keyring-941KY1/ssh
```

```
*p =MAKELEVEL=1
```

```
*p =DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
```

```
*p =SESSION_MANAGER=local/chao:@/tmp/.ICE-unix/1835,unix/chao:/tmp/.ICE-unix/1835
```

```
*p =TARGET_BUILD_APPS=
```

```
* */
```

在设置完环境变量以后，就会开始对makefile进行部分的解析。这边有个重要函数为

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```
vector<Stmt*>* stmts) {
```

```
...
```

```
}
```

函数初始定义了一些基本的变量，比如GCC，G++，SHELL，MAKE等。并且会去解析当前编译机器所拥有的cpu的核



5

并进行合理分配。

以下是一些具体初始化的变量：

```
/*
```



```
* bootstrap =
```



```
*
```

```
* CC?=cc
```

```
* CXX?=g++
```

```
* AR?=ar
```

```
* MAKE_VERSION?=3.81
```

```
* KATI?=ckati
```

```
* SHELL=/bin/sh
```

```
* .c.o:
```

```
* $(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c -o $@ $<
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```
* $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c -o $@ $<

* MAKE?=make -j2

* MAKECMDGOALS=
```

```
· CURDIR:=/data/android_N
```

并且会将这些字符串  为对应的node结构体，保存在内存变量中，方便编译的时候使用。

5

```
for (Stmt* stmt : bootstrap_ast_asts) {
```

```
printf("stmt %s = %s\n\n", stmt->DebugString().c_str(),
());
```

```
stmt->Eval(ev);
```

```
}
```

以下截取部分log：

```
/*
    stmt =AssignStmt(lhs=CC rhs=cc (cc) opstr=QUESTION_EQ dir= loc=*bootstrap*:
0)

stmt = AssignStmt(lhs=CXX rhs=g++ (g++) opstr=QUESTION_EQ dir=loc=*bootstrap*:
0)

    stmt =AssignStmt(lhs=AR rhs=ar (ar) opstr=QUESTION_EQ dir= loc=*bootstrap*:
0)
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)
[注册](#)


```
stmt =AssignStmt(lhs=MAKE_VERSION rhs=3.81 (3.81) opstr=QUESTION_EQ dir= loc=*bootstrap*:0)
```

```
stmt = AssignStmt(lhs=KATI rhs=ckati (ckati) opstr=QUESTION_EQ dir=loc=*bootstrap*:0)
```

```
stmt =AssignStmt(lhs=SHELL rhs=/bin/sh (/bin/sh) opstr=EQ dir=loc=*bootstrap*:0)
```

```
stmt =RuleStmt(expr=.cc.o: term=0 after_term=(null) loc=*bootstrap*:0)
```

```
stmt =Command Expr(SymRef(CC), ,SymRef(CFLAGS), , SymRef(CPPFLAGS), , SymRef(TARGET_ARCH), -c -o , SymRef(@), , SymRef(<)), loc=*bootstrap*:0)
```

```
stmt =RuleStmt(.cc.o: term=0 after_term=(null) loc=*bootstrap*:0)
```

```
*/
```

在环境变量，编译参数都设置成功後，就会开始GenerateNinja的重要操作。

GenerateNinja的函数定义在了ninja.cc中，以下是函数的具体实现。

```
void GenerateNinja(constvector<DepNode*>& nodes,
```

```
    Evaluator* ev,
```

```
    const string&orig_args,
```

```
    double start_time) {
```

```
    NinjaGenerator ng(ev, start_time);
```

```
}

```

函数初始化了一个 NinjaGenerator的结构体，并且继续调用 Generate的方法。

```
void Generate(const vector<DepNode*>&node
S,
const string& orig_args){
    unlink(GetNinjaStampFilename().c_str());
    PopulateNinjaNodes(nodes);
    GenerateNinja()
    GenerateShell();
    GenerateStamp(orig_args);
}

```

Generate 函数非常的重要，PopulateNinjaNodes会对前面include的makefile进行解析，并且将node进行整理。正如前面分析的link的程序会依赖.o一样，这里基本会将所依赖的.o;.a;.so进行归类，包含了所有文件下面的目录。这里举一些简单截取的例子：

```
node =out/host/linux-x86/obj/STATIC_LIBRARIES/libcutils_intermediates/strncpy.

```

```
node =out/host/linux-x86/obj/STATIC_LIBRARIES/libcutils_intermediates/threads.
o

```

```
node =out/host/linux-x86/obj/STATIC_LIBRARIES/libcutils_intermediates/dlmalloc_stub.o

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)
[注册](#)

node =out/host/linux-x86/obj/SHARED_LIBRARIES/libcryptohost_intermediates/src/crypto/evp/sign.o

node =out/host/linux-x86/obj/SHARED_LIBRARIES/libcrypto-host_intermediates/src/crypto/ex_data.o

node = out/host/linux-x86/obj/SHARED_LIBRARIES/libcrypto-host_intermediates/src/crypto/hkdf/hkdf.o



5

node = out/host/linux-x86/obj/SHARED_LIBRARIES/libcrypto-host_intermediates/src/crypto/hmac/hmac.o



....



node = out/target/common/obj/JAVA_LIBRARIES/framework_intermediates/src/core/java/android/app/IBackupAgent.java

node = out/target/common/obj/JAVA_LIBRARIES/framework_intermediates/src/core/java/android/app/InstrumentationWatcher.java

node = out/target/common/obj/JAVA_LIBRARIES/framework_intermediates/src/core/java/android/app/INotificationManager.java

node = out/target/common/obj/JAVA_LIBRARIES/framework_intermediates/src/core/java/android/app/IProcessObserver.java

node = out/target/common/obj/JAVA_LIBRARIES/framework_intermediates/src/core/java/android/app/ISearchManager.java

....

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

node = out/host/linux-x86/obj32/SHARED_LIBRARIES/libchrome_intermediates/base/base64.o

node = out/host/linux-x86/obj32/SHARED_LIBRARIES/libchrome_intermediates/base/base64url.o

node = out/host/linux-x86/obj32/SHARED_LIBRARIES/libchrome_intermediates/base/base_switches.o



5

node = out/host/linux-x86/obj32/SHARED_LIBRARIES/libchrome_intermediates/base/bind_helpers.o



node = out/host/linux-x86/obj32/SHARED_LIBRARIES/libchrome_intermediates/base/build_time.o



...

node = out/target/product/generic/obj/SHARED_LIBRARIES/libhardware_intermediates/hardware.o

node = out/target/product/generic/obj/SHARED_LIBRARIES/libhardware_intermediates/import_includes

node = out/target/product/generic/obj/lib/libandroidfw.so.to

node = out/target/product/generic/obj/lib/libandroidfw.so

node = out/target/product/generic/symbols/system/lib/libandroidfw.so ...

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```
GenerateNinja() {
```

```
....
```

```
fp_ = fopen(GetNinjaFilename().c_str(), "wb");
```

```
...
```

```
fprintf(fp_, "# Generated by kati %s\n", kGitVersion);
```

```
fprintf(fp_, "\n");
```



```
...
```



```
for (const ostream& buf : bufs) {
```



```
fprintf(fp_, "%s", kNinja().c_str());
```

```
}
```

```
...
```

```
fclose(fp_);
```

```
}
```

在GenerateNinja函数中，会创建并写入一个文件，这个文件依赖于build target的制定。比如在nexus的编译中，会在out目录下生成

Build-aosp_arm.ninja文件，

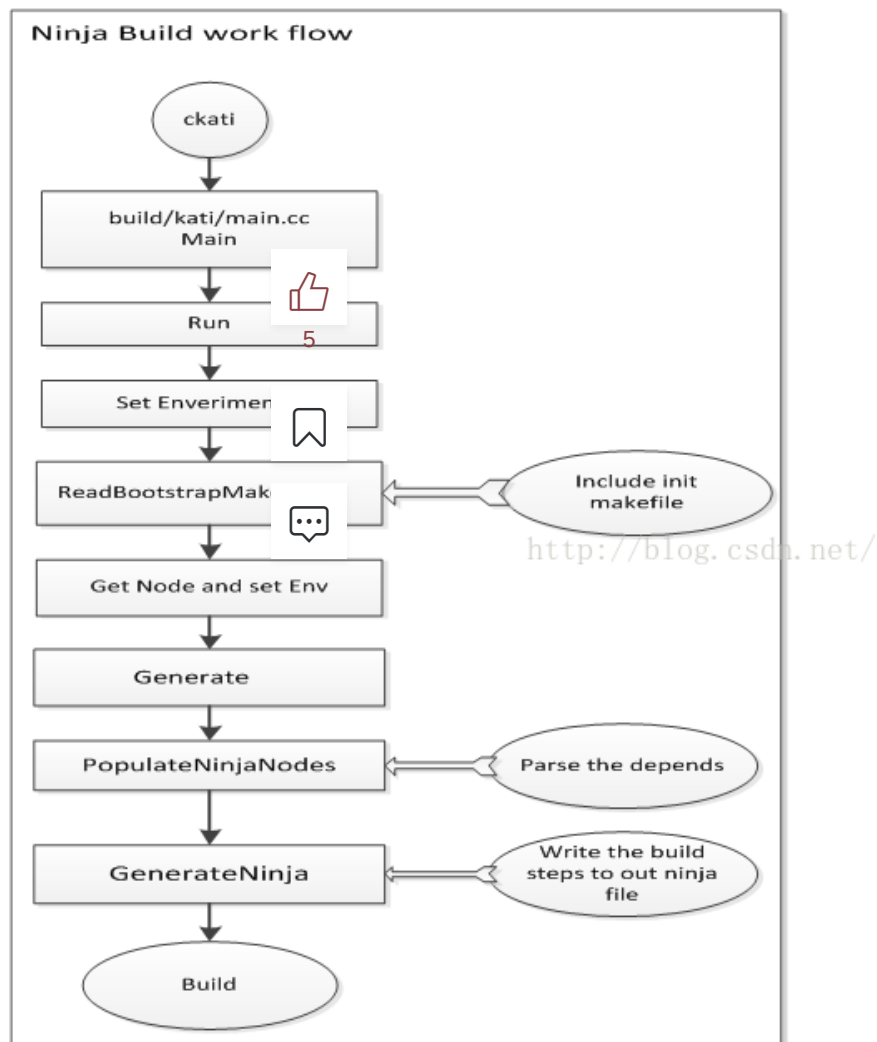
该文件会非常大，但是这个也就是编译的基础和ninja可以明确知道自己所编译的操作步数的由来。

具体的流程图：

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



3. 总结

Ninja编译带来的改变是巨大的，但是通过本文的分析，可以预见到后续的变化会更大且会一直存在。Android何时可以完全取代makefile，ninja编译时的test目录的编译其实对普通开发者来说都

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



5

目前您尚未登录，请 [登录](#) 或 [注册](#) 后进行评论

junlin132 2017-12-06 14:22

1条回复 ∨

[回复](#)

4楼

请问jack和ninja是什么关系，怎么一会说Android7.0使用jack server编译，一会又说使用ninja编译，能麻烦帮忙解答一下吗？



bzhao 2017-11-01 09:22

[回复](#)

3楼

关于ninja是如何编译xxxx.ninja 文件还请给个链接或例子！



caoyuandi 2017-08-23 09:12

[回复](#)

2楼

是不是搞错了？这个能加快速度？我觉的比之前满了不止一点点

[查看 5 条热评 ∨](#)

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)[注册](#)

最近更新了Android Nougat源码，无意间发现Android的编译系统已经发生了巨大改变，到处是“Android.bp”文件，下面就来了解一下这个bp文件到底是何方神圣。首先从Soong说起，S...

 iEearth 2017年01月24日 14:01 10566

android.bp

 prike 2017年10月24日 10:40 1610

最近更新了Android Nougat源码，无意间发现Android的编译系统已经发生了巨大改变，到处是“Android.bp”文件，下面就来了解一下这个bp文件到底是何方神圣。首先从Soong说起，S...



5分钟完成加壳 防止代码反编译，防调试

一键加密代码逻辑，驱动级别反调试,秒杀常见调试器，无法dump内存。

Android 编译系统之Android.bp

 drageon_j 2017年08月17日 17:31 1921

从Android 7.0 (N)开始, Google开始逐步使用Android.bp代替原来的Android.mk进行编译. Google称之为soong, 具体可以参考: <https://and...>

Android O 前期预研之二：HIDL相关介绍

 ljp1205 2017年09月07日 00:19 9943

在上一篇博客里，大致介绍了下Android O 中treble计划的一些背景与相关基本架构，这一篇中跟大家一起来探讨下HIDL相关的内容。Android HAL类型 在此之前的ANDROI...

mtk android.mk --> android.bp

 qq_37610155 2018年01月09日 16:14 311

本文主要介绍的是MTK的编译系统，从Android 7.0开始，MTK的编译系统已经转变成为了Android.bp，首先我们...

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

JAVA课程免费试听！

全球 90% 的大型企业都在用的编程语言



android 中Bn 和 Bp 的区别



w401229755

2014年07月23日 11:29

📖 9001

最近在研究surfaceflinger,真心被里面的类名折腾惨了。

Bn

android 中Bn 和 Bp 的区别



prike

2015年10月16日 13:57

📖 247

最近在研究surfaceflinger,真心被里面的类名折腾惨了。

Bn

n 就是native, 这是一个怎样的类? 我们继承它的原因是为了实现一个接口, 具体点说就是一个BnXXX...

Android7.0 Ninja编译原理



ztguang

2018年01月19日 23:04

📖 143

<http://blog.csdn.net/ChaoY1116/article/details/53063082> #####...

Android7.0 Ninja编译原理



ksjay_1943

2016年12月12日 08:39

📖 1007

本文为极度寒冰原创, 转载请注明出处

#####...

Android frameworks中Bn*和Bp*的区别



ameyume

2012年04月26日 18:05

📖 4626

Q:What do "Bn*" and "Bp*" stand for in frameworks/base/include/utils/Interface.h ? I understand th...

加入CSDN, 享受更精准的内容推荐, 与500万程序员共同成长!

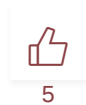
[登录](#)
[注册](#)

首先解释一下BP文件：后缀名为BP的文件是卫星对地面遥感数据的ASCII文本文件（google，then ull get it）我的海深BP文件是保密文件所以不能公开，sorry我用的是window...

开发一个app多少钱

开发一个app大概需要多少钱呢

百度广告



Android Treble架构解析



xiaosayidao 2017年12月21日 16:39 1890

本文主要介绍Treble架构下的HAL&HIDL&Binder相关技术原理。Treble的详细资料文档，请参考Treble 官方文档。1. Treble 简介 Android 8.0 的...

手机开发中的AP与BP的概念



macong01 2013年11月12日 16:34 4211

手机的AP和BP：AP：ApplicationProcessor，即应用芯片 BP：BasebandProcessor，即基带芯片 AP上面则运行了我们通常的操作系统和应用软件，如Androi...

Android AP及BP



liu_jun_y 2013年07月18日 21:19 1986

大多数的手机都含有两个处理器。操作系统、用户界面和应用程序都在Application Processor(AP)（应用处理器）上执行，AP一般采用ARM芯片的CPU。而手机射频通讯控制软件，则运行在另...

android binder机制中的BN跟BP



lucky_liuxiang 2014年06月10日 14:22 1926

android binder机制中的BN和BP 看到android的binder机制，有点不太理解。BP(binder proxy)和BN(binder native)是通过binder来通信的。B...

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

BufferedReader的readLine方法阻塞的处理

BufferedReader的read方法和readLine方法在任何情况下都是阻塞的。readLine方法每次读一行，相对于一个字符/字节地读取、转换、返回来说，它有一个缓冲区，读满缓冲区才返回；一般情...



u010595903 2016年06月15日 10:46 6179

Unity3D程序加密，可有效防止反编译

无需手动加密Assembly代码，自动编译mono，防止反编译



java中split()特殊符号"." "|" "*" "\\" "]"



myfmyfmyfmyf 2014年07月09日 09:49 18902

关于点的问题是用string.split("[.]") 解决。关于竖线的问题用 string.split("\\")解决。关于星号的问题用 string.split("*")解决。关...

android系统开发 AP 和 BP 简要说明



zhanghao_Hulk 2013年01月05日 14:40 11479

手机的AP和BP根据上下文可以指代硬件和软件两种意思。 1) 大多数的手机都含有两个处理器。操作系统、用户界面和应用程序都在Application Processor(AP)上执行，AP一般...

Android7.0 Ninja编译原理



makeyourprogress 2017年06月21日 09:48 290

引言 使在Android N的系统上，初次使用了Ninja的编译系统。对于Ninja，最初的印象是用在了Chromium open source code的编译中，在chromium的编译...

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

从官方的定义，ninja大大缩短了android系统的编译周期，android 7.0即nougat上已经默认使用，禁用方式：`export USE_NINJA = false` 从build/...



5



加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册