ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center


# 从0开始，自顶向下地学习 V8’s build system-03

智能软件研究中心 邱吉

qiuji@iscas.ac.cn

2020/02/27

# 目 录



今天（2020-02-27）的内容

# Part-1/2回顾



## 02 Quick Glance： V8 build 过程-1

ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences

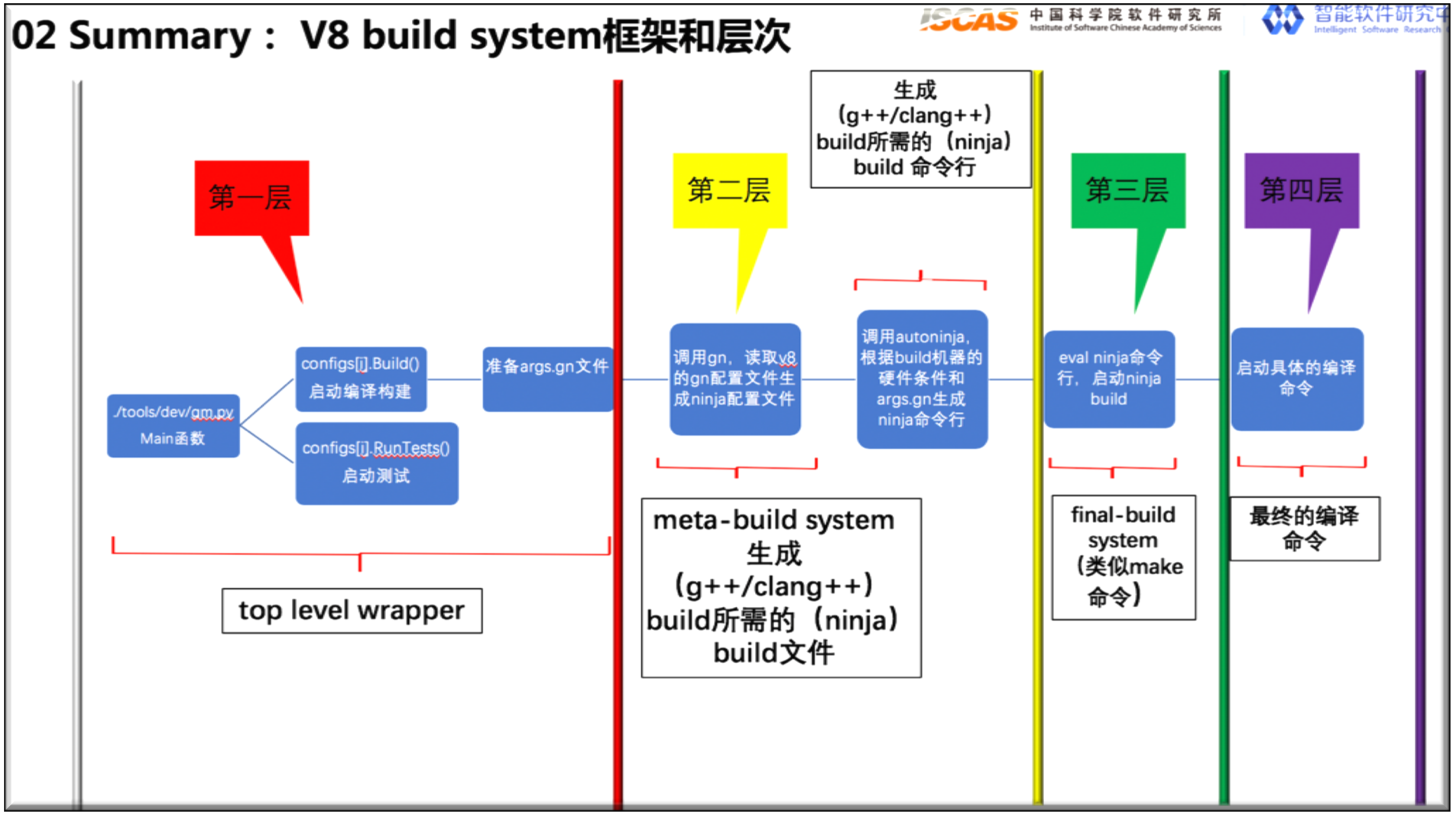
### V8 的编译拆解-总体流程

执行“*v8/tools/dev/gm.py x64.release*”后发生了什么：

```
qiuji@ubuntu-s-1vcpu-3gb-sfo2-01:~/work/v8clean/v8$ ./tools/dev/gm.py x64.release
# mkdir -p out/x64.release                                          1
# echo > out/x64.release/args.gn << EOF
is_component_build = false
is_debug = false
target_cpu = "x64"
use_goma = false                                                    2
goma_dir = "None"
v8_enable_backtrace = true
v8_enable_disassembler = true
v8_enable_object_print = true
v8_enable_verify_heap = true
EOF
# gn gen out/x64.release                                            3
Done. Made 142 targets from 83 files in 378ms
# autoninja -C out/x64.release d8                                   4
/home/qiuji/work/depot_tools/ninja -C out/x64.release d8 -j 3       5
ninja: Entering directory `out/x64.release'
[4/1403] CXX obj/torque_base/types.o                                6
```

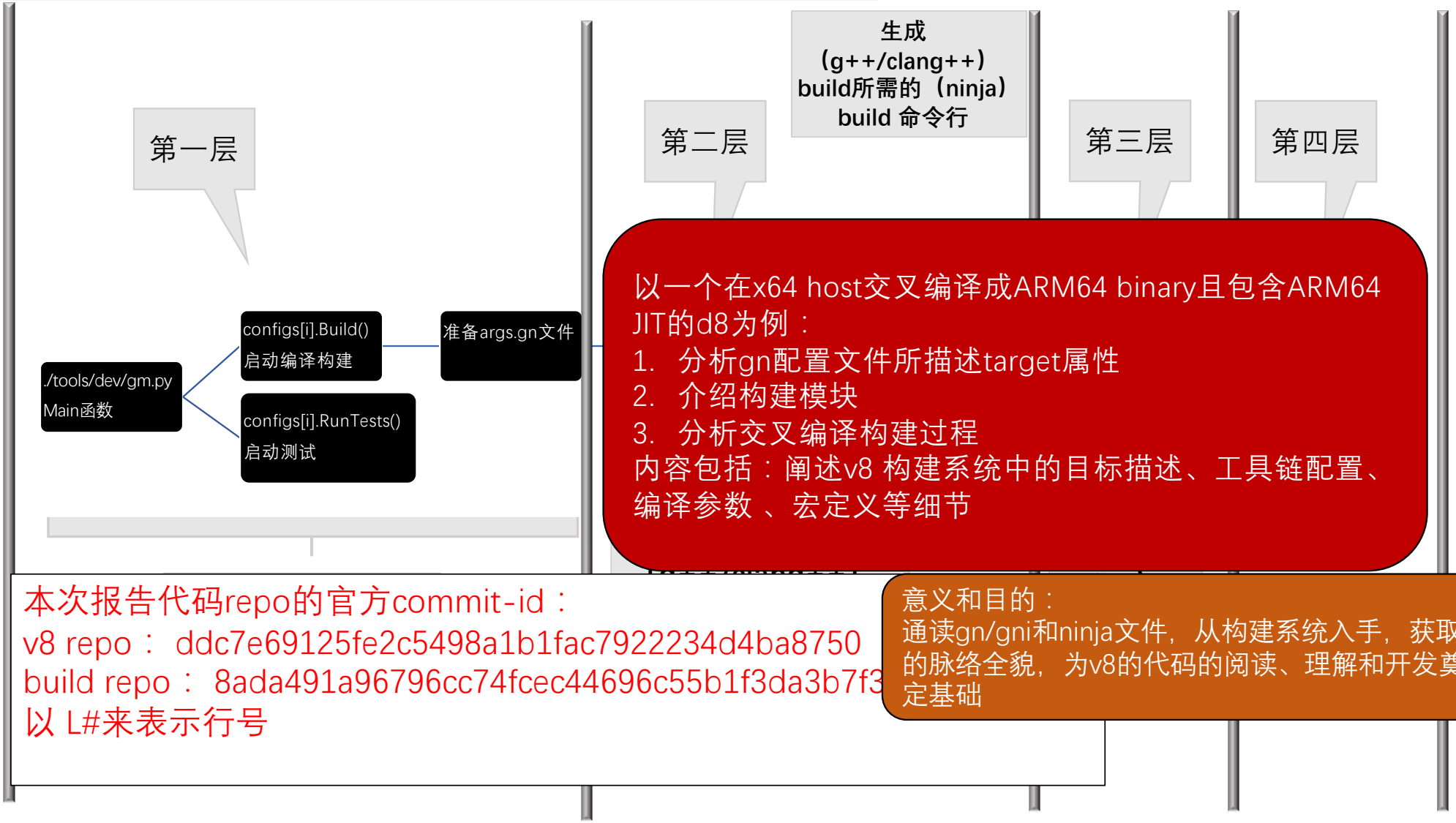
# Part-1/2回顾




02 Summary： V8 build system框架和层次
第一层
第二层
生成(g++/clang++)build所需的（ninja）build 命令行
第三层
第四层
./tools/dev/gm.py Main函数
configs[i].Build() 启动编译构建
configs[i].RunTests() 启动测试
准备args.gn文件
调用gn，读取v8的gn配置文件生成ninja配置文件
调用autoninja，根据build机器的硬件条件和args.gn生成ninja命令行
eval ninja命令行，启动ninja build
启动具体的编译命令
top level wrapper
meta-build system 生成(g++/clang++)build所需的（ninja）build文件
final-build system（类似make命令）
最终的编译命令

# Part3内容概要






以一个在x64 host交叉编译成ARM64 binary且包含ARM64 JIT的d8为例：

1. 分析gn配置文件所描述target属性
2. 介绍构建模块
3. 分析交叉编译构建过程

内容包括：阐述v8 构建系统中的目标描述、工具链配置、编译参数 、宏定义等细节

本次报告代码repo的官方commit-id：

v8 repo： ddc7e69125fe2c5498a1b1fac7922234d4ba8750

build repo： 8ada491a96796cc74fcec44696c55b1f3da3b7f3

以 L#来表示行号

意义和目的：

通读gn/gni和ninja文件，从构建系统入手，获取v8的脉络全貌，为v8的代码的阅读、理解和开发奠定基础

# GN构建配置：args.gn的配置



args.gn文件的内容：

```
is_component_build = false #编译d8可执行文件
is_debug = false #release build
target_cpu = “arm64” #交叉编译产生ARM64的d8 binary
v8_target_cpu = “arm64” #d8包含ARM64的JIT编译器
use_goma = false
goma_dir = "None"
v8_enable_backtrace = true
v8_enable_disassembler = true
v8_enable_object_print = true
v8_enable_verify_heap = true
```

1. gn gen ./out/a64demo/
2. cd ./out/a64demo/
3. ninja v8_dump_build_config
4. cat v8_build_config.json

所有gn args的内容：

{"is_full_debug": false, "is_clang": true, "is_cfi": false, "v8_target_cpu": "arm64", "target_cpu": "arm64",
"is_ubsan_vptr": false, "is_gcov_coverage": false, "dcheck_always_on": false, "is_component_build": false,
"is_asan": false, "is_android": false, "is_debug": false, "v8_enable_i18n_support": true, "v8_current_cpu": "arm64",
"v8_enable_pointer_compression": true, "v8_enable_verify_csa": false, "v8_enable_lite_mode": false, "is_msan": false,
"is_tsan": false, "current_cpu": "arm64", "v8_enable_verify_predictable": false}

# d8 target的属性-1

# gn desc . //:d8 --all --tree # . 代表当前目录 ~/out/a64demo/



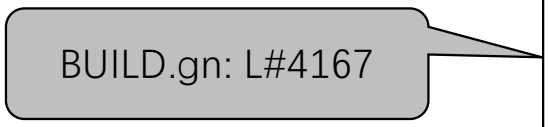
| 属性名 | 属性意义解释 | 内容 |
|---|---|---|
| Target | target名称 | //:d8 |
| type | 可执行文件/共享库。。。 | executable |
| toolchain | 编译器 | //build/toolchain/linux:clang_arm64 |
| visibility | | * |
| metadata | 描述本target的元数据 | {} |
| testonly | 本target是否只用于测试 | FALSE |
| check_includes | 当值为true时，gn check会重新检查依赖关系是否满足（类似检查依赖文件是否有更新） | TRUE |
| allow_circular_includes_from | 允许依赖成环 | |
| sources | 本target的源文件 | |
| public | 本target的公共头文件，可以传递 | |
| configs tree (in order applying) | *以树状的形式来展示本target的所有config层次 | |
| outputs | 本target的输出 | |
| arflags | ar lib flag | |
| asmflags | asm flag | |
| cflags | ... | |
| cflags_c | ... | |
| cflags_cc | ... | |
| cflags_objc | ... | |
| cflags_objcc | ... | |
| defines | 宏定义 | |
| include_dirs | 头文件目录 | |
| ldflags | 链接详细 | |
| Dependency tree | *以树状的形式来展示本target的所有依赖层次 | |
| libs | 本target依赖的系统库 | |

```
v8_executable("d8") {
```

build/config/BUILDCONFIG.GN
default_toolchain: L#230

# d8 target的属性-2



| | | |
|---|---|---|
| sources | 本target的源文件<br>(不包含依赖的源文件)<br>  | //src/d8/async-hooks-wrapper.cc<br>//src/d8/async-hooks-wrapper.h<br>//src/d8/d8-console.cc<br>//src/d8/d8-console.h<br>//src/d8/d8-js.cc<br>//src/d8/d8-platforms.cc<br>//src/d8/d8-platforms.h<br>//src/d8/d8.cc<br>//src/d8/d8.h<br>//src/d8/d8-posix.cc |
| public | 本target的公共头文件，可以传递 | [All headers listed in the sources are public.] |

# d8 target的属性-3



| | | |
|---|---|---|
| configs tree<br>(in order applying) | *以树状的形式来展示本target的所有config层次 | //build/config:feature_flags<br>//build/config/compiler:afdo<br>//build/config/compiler:afdo_optimize_size<br>//build/config/compiler:compiler<br>//build/config/linux:compiler<br>//build/config/compiler:clang_revision<br>//build/config/compiler:compiler_cpu_abi<br>//build/config/compiler:compiler_codegen<br>//build/config/compiler:compiler_deterministic<br>//build/config/compiler:compiler_arm_fpu<br>//build/config/compiler:compiler_arm_thumb<br>//build/config/compiler:chromium_code<br>//build/config/compiler:default_warnings<br>//build/config/compiler:default_include_dirs<br>//build/config/compiler:default_stack_frames<br>//build/config/compiler:default_symbols<br>//build/config/compiler:no_symbols<br>//build/config/compiler:export_dynamic<br>//build/config/compiler:no_exceptions<br>//build/config/compiler:no_rtti<br>//build/config/compiler:runtime_library<br>//build/config/c++:runtime_library<br>//build/config/posix:runtime_library<br>//build/config/linux:runtime_library<br>//build/config/compiler:thin_archive<br>//build/config/compiler:default_init_stack_vars<br>//build/config/coverage:default_coverage<br>//build/config/sanitizers:default_sanitizer_flags<br>//build/config/sanitizers:common_sanitizer_flags<br>//build/config/sanitizers:coverage_flags<br>//build/config/sanitizers:default_sanitizer_ldflags<br>//build/config/sanitizers:asan_flags<br>//build/config/sanitizers:cfi_flags<br>//build/config/sanitizers:hwasan_flags<br>//build/config/sanitizers:lsan_flags<br>//build/config/sanitizers:msan_flags<br>//build/config/sanitizers:tsan_flags<br>//build/config/sanitizers:ubsan_flags<br>//build/config/sanitizers:ubsan_no_recover<br>//build/config/sanitizers:ubsan_null_flags<br>//build/config/sanitizers:ubsan_security_flags<br>//build/config/sanitizers:ubsan_vptr_flags<br>//build/config/sanitizers:fuzzing_build_mode<br>//build/config/clang:extra_warnings<br>//build/config:release<br>//build/config:default_libs<br>//build/config:executable_config<br>//build/config/gcc:executable_config<br>//build/config/sanitizers:link_executable<br>//:features<br>//:v8_header_features<br>//:toolchain<br>//build/config/compiler:optimize_speed<br>//build/config/gcc:symbol_visibility_default<br>//:internal_config_base<br>//:external_config<br>//:v8_header_features<br>//third_party/icu:icu_config<br>//:libbase_config<br>//:libplatform_config |

- **config( “name” )是可以层层嵌套的结构体，里面有对gn variable的赋值，包括defines(宏定义)，clags*（编译选项），ldflags（链接选项）等等，嵌套的config用configs+= [] 或者configs=[]来声明**
- /build/config/BUILDCONFIG.gn：
  - default_compiler_configs
  - default_executable_configs
- ./BUILD.gn:
  - visibility = [” :*” ]的configs
  - d8 target的configs
  - d8 target的deps的configs

来源

# d8 target的属性-4



| | | |
|---|---|---|
| outputs | 本target的输出 | //out/a64demo/d8 |
| arflags | ar lib flag | … |
| asmflags | asm flag | … |
| cflags | … | … |
| cflags_c | … | … |
| cflags_cc | … | … |
| cflags_objc | … | … |
| cflags_objcc | … | … |
| defines | 宏定义 | … |
| include_dirs | 头文件目录 | … |
| ldflags | 链接选项 | … |
| libs | 本target依赖的系统库 | dl, pthread, rt |

1. outputs指出了最终输出可执行文件的路径和名字
2. 其他的flags和dirs、libs来自所有嵌套的configs结构体中，根据default args和args.gn中的参数所确定的变量值

# d8 target的属性-5



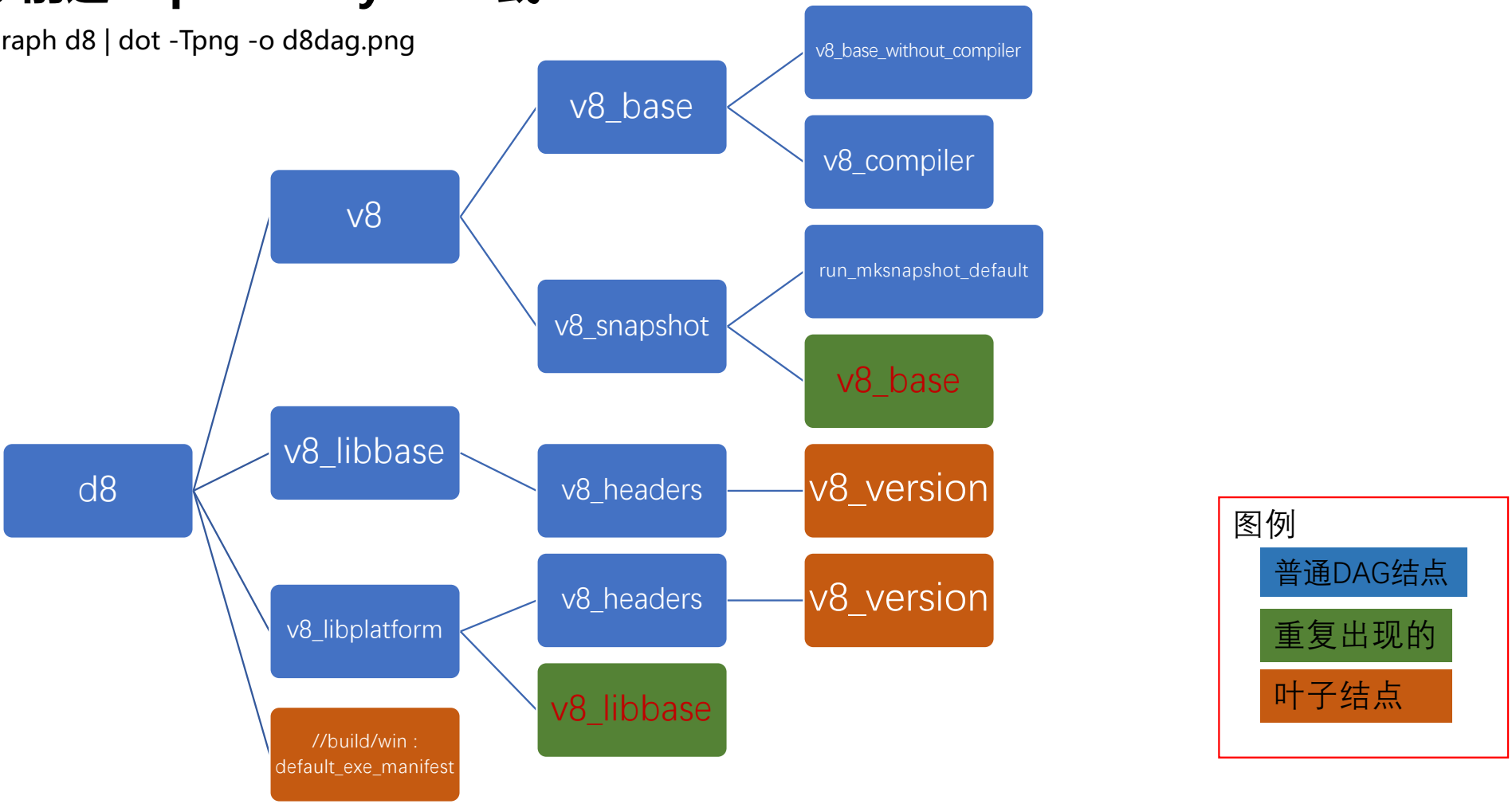
- Dependency tree-全部内容在 d8.desc.txt文件中, d8 target声明中的所有deps递归的依赖

```
//:v8 # deps@./build/BUILD.gn: L#4187， @./build/BUILD.gn: L#4142
  //:v8_base # deps@./build/BUILD.gn: L#4144， @./build/BUILD.gn: L#3395
    //:v8_base_without_compiler # deps@./build/BUILD.gn: L#3397， @./build/BUILD.gn: L#1997
      //:generate_bytecode_builtins_list
        //:bytecode_builtins_list_generator(//build/toolchain/linux:clang_x64_v8_arm64)
          //:v8_dump_build_config(//build/toolchain/linux:clang_x64_v8_arm64)
          //:v8_libbase(//build/toolchain/linux:clang_x64_v8_arm64)
            //:v8_headers(//build/toolchain/linux:clang_x64_v8_arm64)
              //:v8_version(//build/toolchain/linux:clang_x64_v8_arm64)
          //build/config:executable_deps(//build/toolchain/linux:clang_x64_v8_arm64)
            //build/config:common_deps(//build/toolchain/linux:clang_x64_v8_arm64)
              //buildtools/third_party/libc++:libc++(//build/toolchain/linux:clang_x64_v8_arm64)
                //buildtools/third_party/libc++abi:libc++abi(//build/toolchain/linux:clang_x64_v8_arm64)
          //build/win:default_exe_manifest(//build/toolchain/linux:clang_x64_v8_arm64)
```

# d8 target的构建模块组成-1

## 来源于前述Dependency tree 或

ninja -t graph d8 | dot -Tpng -o d8dag.png



# d8 target的构建模块组成-2



- v8_base_without_compiler: 构建d8跟compiler(Ignition JIT)无关的部分
  - **source:** /out/arm64demo/gen/binutils-generated/bytecodes-builtins-list.h, src/api/, src/ast/, src/builtins/, src/codegen/, src/common/,src/compiler-dispatcher/, src/date/, src/debug/, src/deoptimizer/, src/diagnostic/, src/execution, src/extension/, src/flags, src/handles, src/heap, src/ic/, src/init, src/interpreter/, src/json/, src/logging/, src/numbers/, src/objects/, src/parsing/, src/profiler/, src/regexp/, src/roots/, src/runtime/, src/sanitizer/, src/snapshot/, src/strings/, src/tasks/, src/third_party/, src/tracing/, src/trap-handler/, src/utils/, src/wasm/, src/zone/
  - **arch-specific source:** src/codegen/fooarch/, src/compiler/backend/fooarch, src/debug/fooarch, src/deoptimizer/fooarch, src/diagnotics/fooarch, src/execution/fooarch, src/regexp/fooarch, src/wasm/baseline/fooarch
  - **os-specific source:** src/trap-handler/handler*posix.*
  - **依赖模块：** torque_generated_definition, generate_bytecode_builtins_list, run_torque, v8_maybe_icu, //third_party/zlib, run_gen-regexp-special-case, v8_headers, v8_libbase, v8_libsampler, v8_shared_internal_headers, v8_version, src/inspector:inspector

# d8 target的构建模块组成-3



- v8_compiler: 构建d8跟compiler(Ignition JIT)相关的部分
  - source：v8_compiler_sources (构建 ./src/compiler/下文件)
  - 依赖模块：
    - generate_bytecode_builtins_list
    - run_torque
    - v8_maybe_icu

# d8 target的构建模块组成-4



- run_mksnapshot_default: 构建d8的snapshot_blob.bin文件，来自template(“run_mksnapshot”)
  - 依赖模块：mksnapshot( “$v8_snapshot_toolchain” )

**理解重点：**

1. snapshot是v8中一种特殊格式的二进制代码文件，可以简单理解为v8 解释器和JIT在运行过程中可以调用的AOT的代码片段/函数库
2. 主要包含在构建V8时，提前准备好的内置函数库二进制和CodeStubAssmbler二进制
3. 这些二进制，来源于由torque语言编写的tq文件，经过torque工具翻译后形成的.cc/.h文件，经过v8_generator_toolchain (target: aarch64/clang_aarch64)编译产生
4. mksnapshot是运行在构建host上的工具程序，由v8_snapshot_toolchain(target:x64/clang_x64_v8_arm64)构建

# d8 target的构建模块组成-5

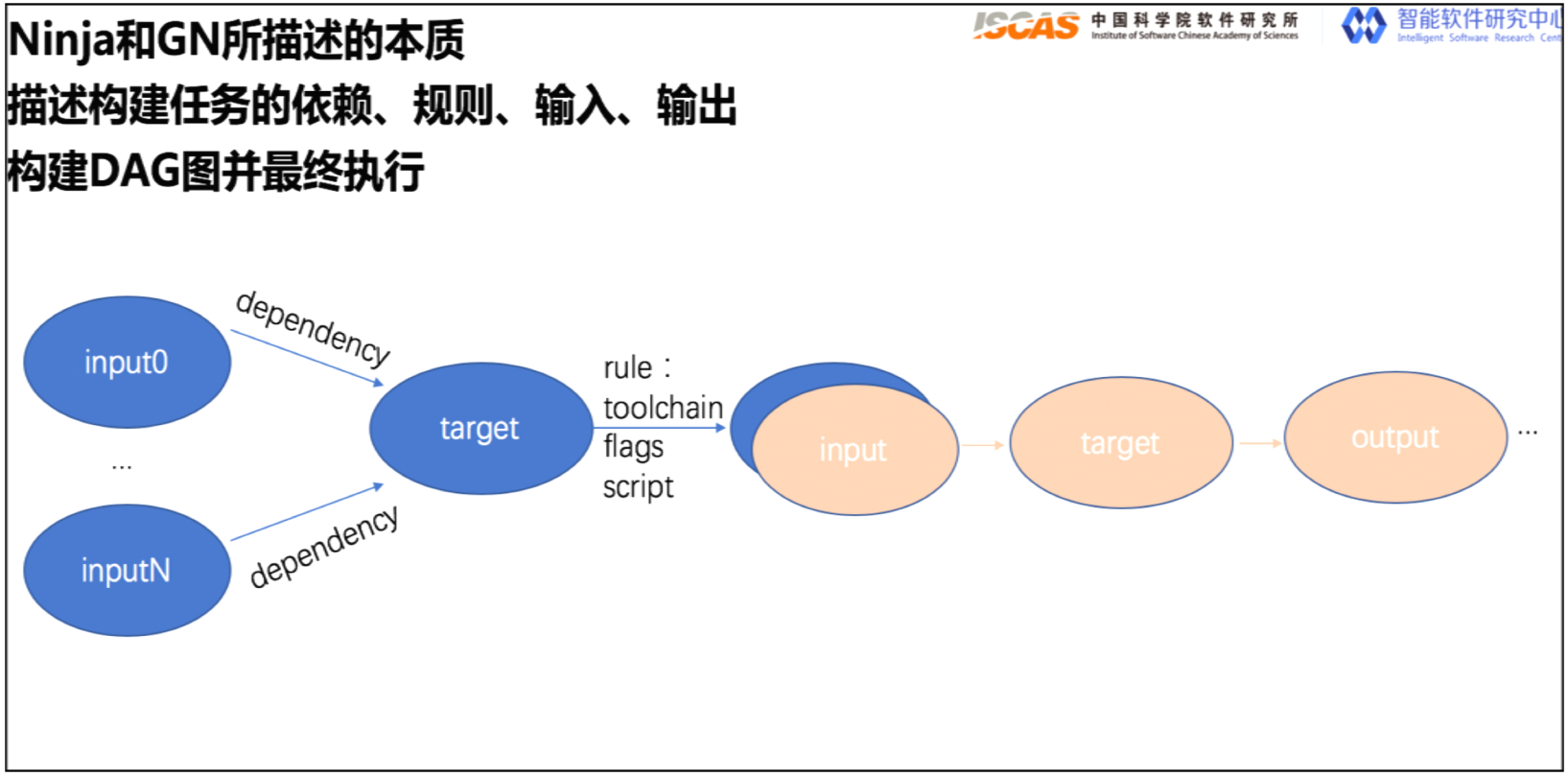


- mksnapshot
  - 源码：src/snapshot/
  - 依赖模块：
    - v8_base_without_compiler
    - v8_compiler_for_mksnapshot
    - v8_init
    - v8_libbase
    - v8_libplatform
    - v8_maybe_icu
    - //build/win:default_exe_manifest

# d8 target的交叉编译流程-1



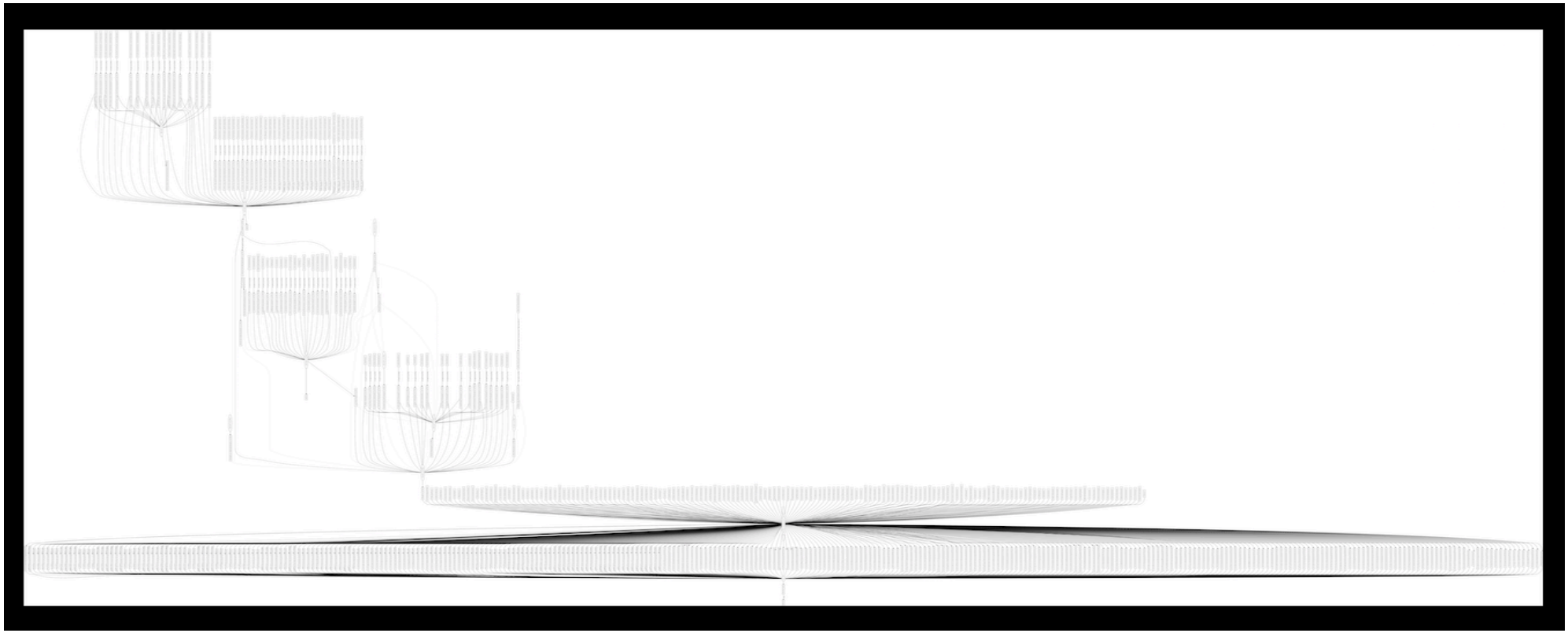
- Part2报告回顾：DAG图



# d8 target的交叉编译流程-2

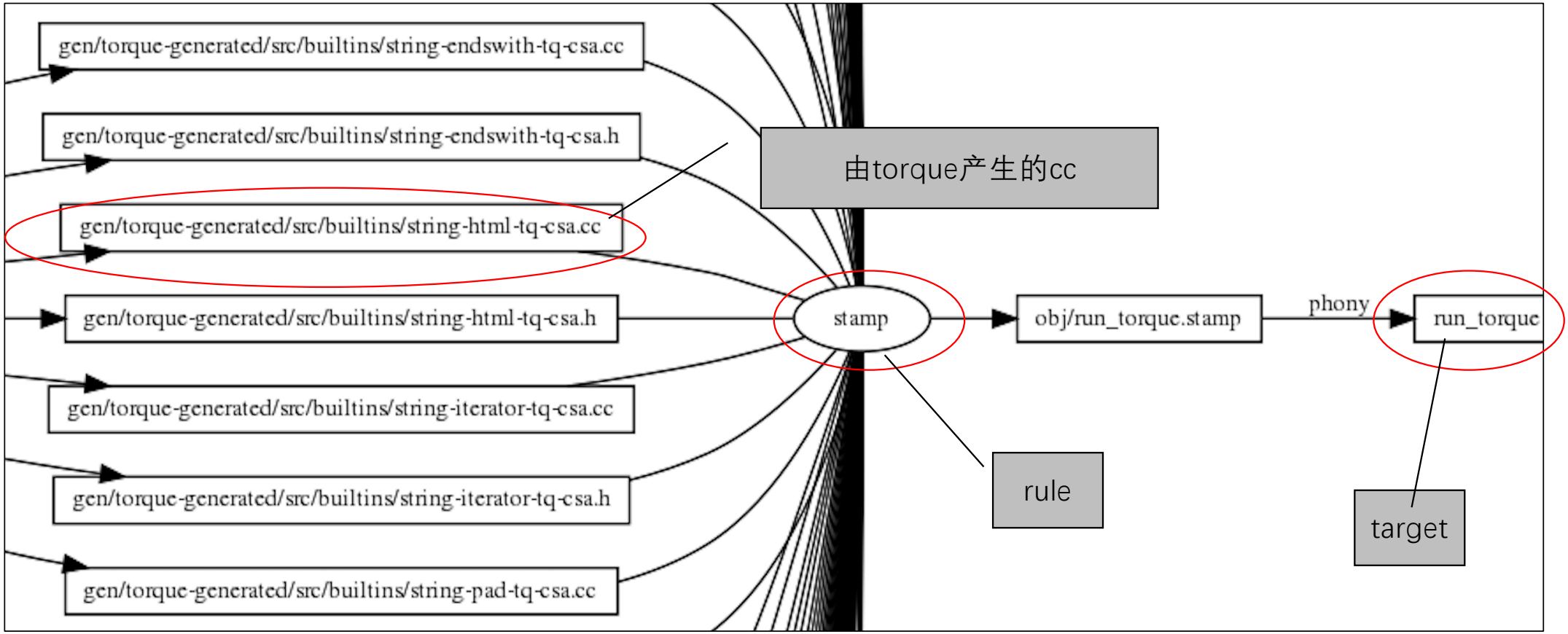


- d8的构建DAG图：ninja -t graph d8 |dot -Tpng -o d8dag.png (太复杂)
- run_torque的DAG图： ninja -t graph run_torque |dot -Tpng -o rtqdag.png



# d8 target的交叉编译流程3-run_torque DAG局部root node

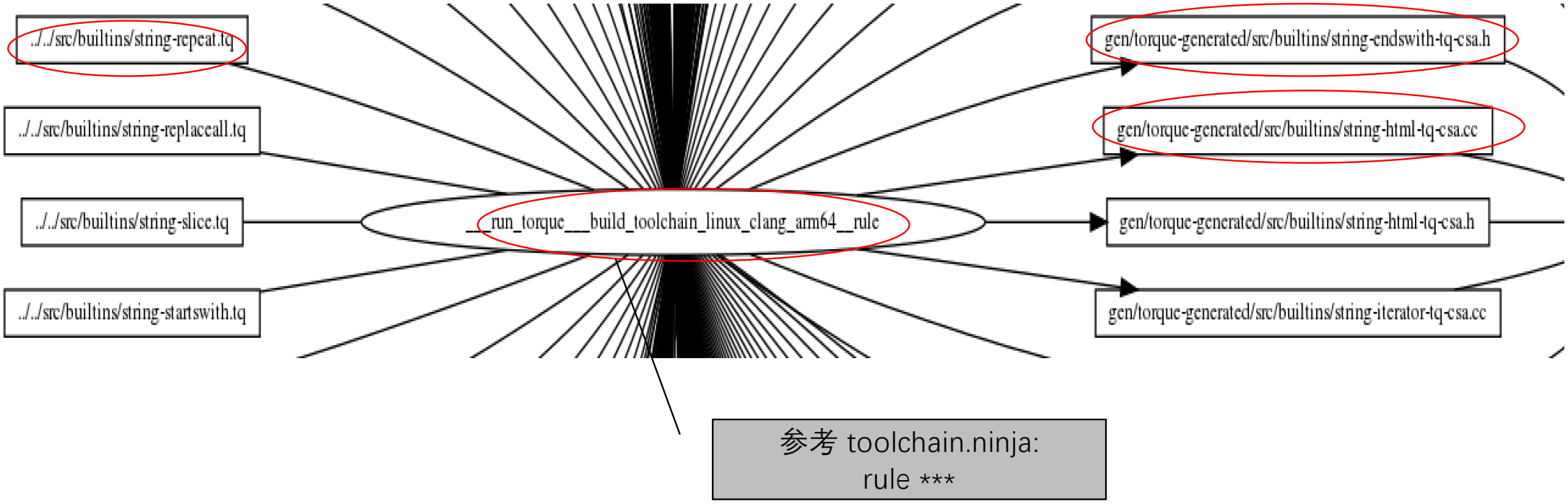




# d8 target的交叉编译流程4-run_torque DAG局部

## 中间部分：run_torque





# d8 target的交叉编译流程5-run_torque DAG局部

## 中间节点：torque



## d8 target的交叉编译流程6



- 分析方法
  - ninja -t graph target_name | dot -Tpng -o target-dag.png
  - ninja -t commands target_name 2>&1 |tee log-target-cmds.txt
  - grep 在log-target-cmds.txt 中寻找 “touch *.stamp” 的语句，分析流程
  - Tips：不推荐ninja build log，因为DAG图的执行顺序是不确定的（DAG图箭头反向后，节点的拓扑序是不确定的）

# Part3 V8的build system全景 总结

- 基于交叉编译的arm64 binary & arm64 JIT target的d8构建实例，分析root target d8：
  - 阐述如何获取和分析特定gn target的属性，包括工具链配置、编译构建参数、gn config和dependency的树状层次信息
  - 以run_torque target为例，阐述如何得到构建依赖-规则的DAG图，并说明分析构建流程的方法
- 意义和目的：读懂、读通gn/gni和ninja文件，从构建系统入手，获取v8的脉络全貌，为v8的代码的阅读、理解和开发奠定基础
- Tips： 务必结合动手实践！

# 从0开始，自顶向下地学习 V8's build system 总体总结

01 背景介绍

02 Quick Glance： V8 build 过程

part1

03 Ninja简介

04 GN简介

part2

05 V8的build system全景

part3

# Q&A

# 谢 谢

欢迎交流合作

2020/02/27