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-02

智能软件研究中心 邱吉

qiuji@iscas.ac.cn

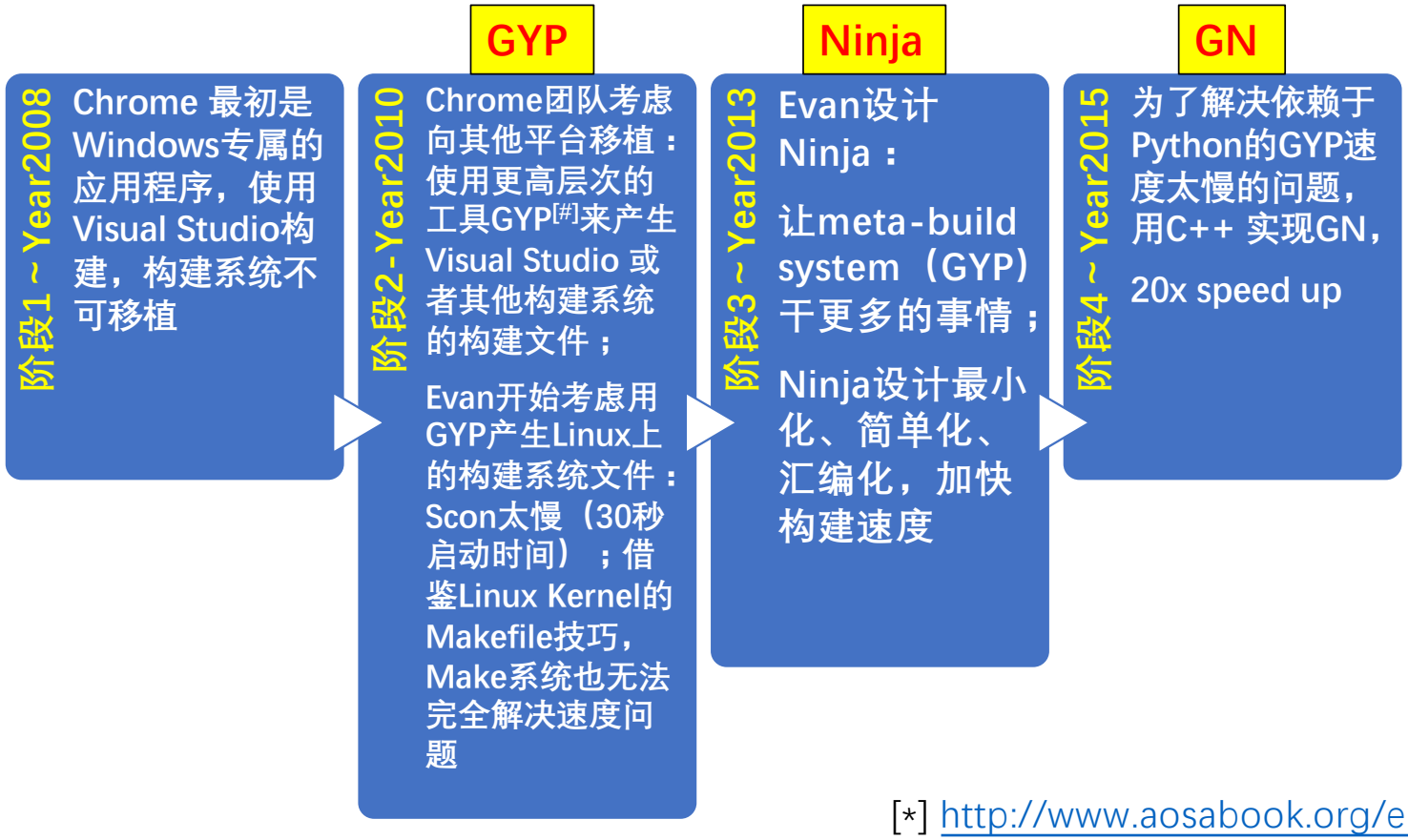
2020/02/15

目 录

今天（2020-02-15）的内容

背景介绍-GYP, Ninja和GN都是源于Chrome

以下内容来源于Ninja作者Evan Martin, The Performance of Open Source Application, Chapter 3 ” [*]

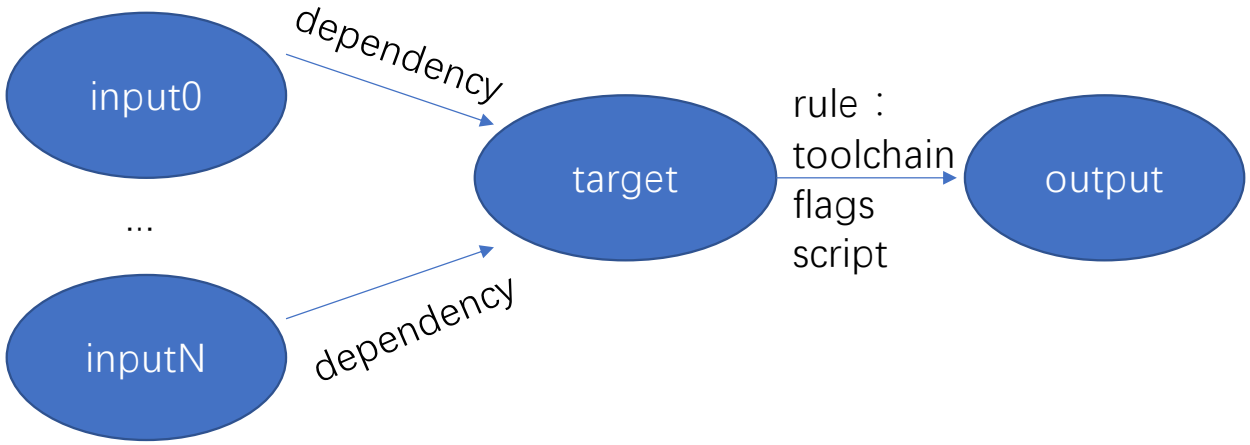


[*] http://www.aosabook.org/en/posa/ninja.html
[#] https://en.wikipedia.org/wiki/GYP_(software)

Ninja和GN所描述的本质

描述构建任务的依赖、规则、输入、输出

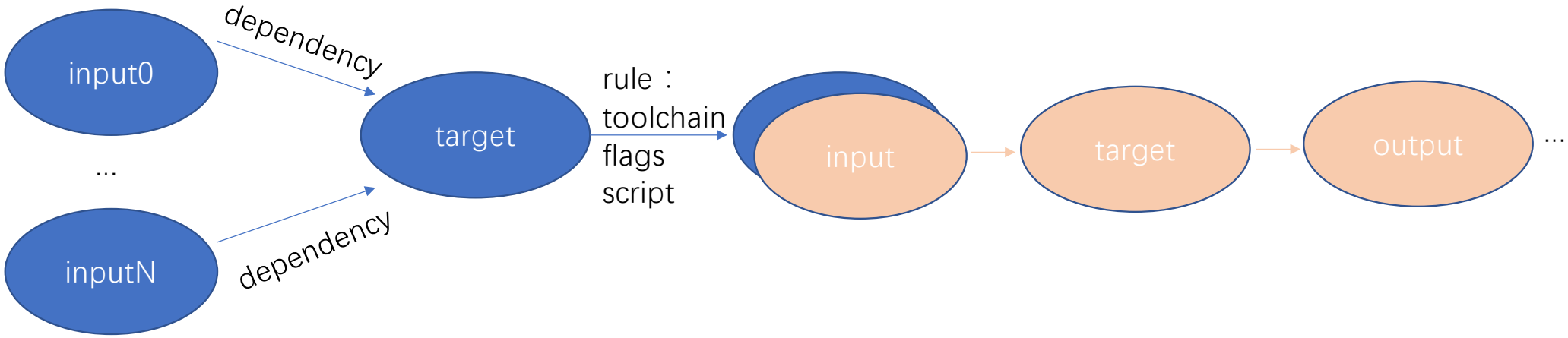
构建DAG图并最终执行



Ninja和GN所描述的本质

描述构建任务的依赖、规则、输入、输出

构建DAG图并最终执行



Ninja概览

- Ninja是什么？
  - 一种build system，包括领域专用语言（DSL）和执行系统
  - 对照系是Make系统和Makefile
  - 来自Google程序员Evan Martin，为了解决Chrome的Windows Visual Studio工程移植到Linux上以后的构建系统速度问题
- Ninja的特点和设计初衷
  - 汇编级别的构建DSL，语言元素简单、没有条件判断（对应Make是高级语言级别）
  - 设计初衷是由其他更高级的meta-build system来产生build文件，而不是手写
  - 速度快
- Ninja的应用范围
  - Google Chrome
  - 部分Android
  - LLVM

Ninja的适用范围

- 对规模较小的工程的build速度影响很小
- 主要用于改进build速度太慢的工程，换言之，大规模工程
- 配合meta-build system使用效果更佳
  - 给大规模工程手写Ninja文件是不太现实的
  - GN: Google Chrome, V8, node.js
  - CMake: LLVM

Ninja的安装和运行

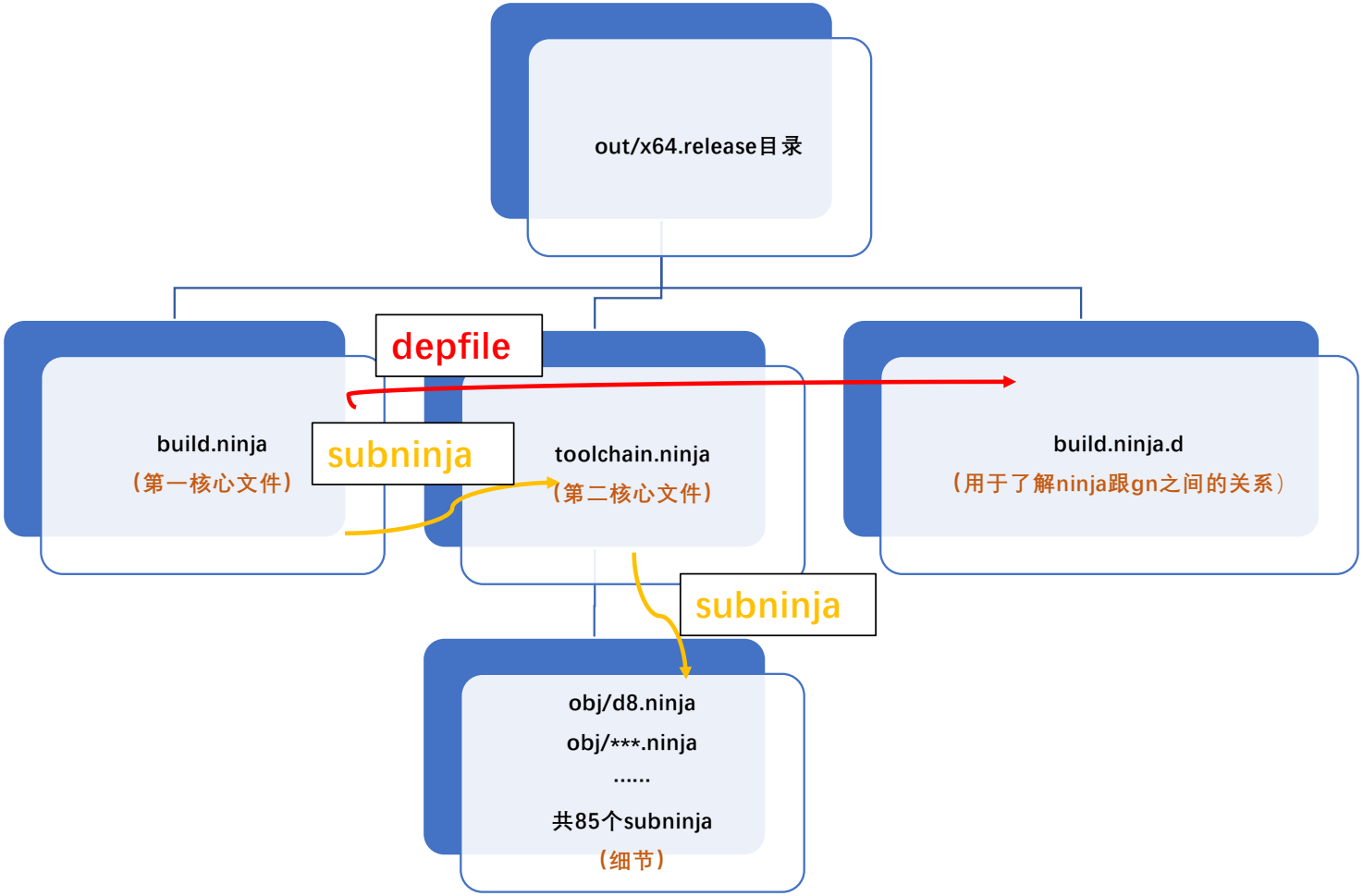
- Build from scratch
- 官网直接下载二进制
- 用system package manager安装，apt-get

https://ninja-build.org

- 对于V8来说，直接获取depot_tools repo 即可
- 设置PATH到安装目录，直接运行 “ninja”

V8-Ninja实战-1-overview

- 运行gn gen out/x64.release后生成了如下层次的ninja文件



V8-Ninja实战2-语法示例

- build. ninja语法示例：

```
cflags = -Wall

rule cc
  command = gcc $cflags -c $in -o $out

build foo.o: cc foo.c
```

- 执行ninja命令后，实际效果是执行了如下的命令行：
  - gcc -Wall -c foo.c -o foo.o

V8-Ninja实战3-核心语法要素

- 变量 variables：
  - 两类：ninja系统默认的global变量（in/out）， 用户自定义的变量（cflags）
  - 使用 “变量名 = 字符串” 的形式来定义/赋值
  - 使用$或者${}来引用
  - 要注意作用域
- Rule语句:
  - 定一个在bash里面执行的单个或多个命令所组成的命令行的简称，可以引用变量
- Build语句:
  - 说明一个Build动作：要素是target名字，Rule，输入，输出（target）
- Default语句:
  - ninja命令启动时，如果没有指定某个输出（target），就构建Default target
- Phony规则:
  - 给phony后的target取个别名，并在执行时不打印phony规则

V8-Ninja实战4-命令行参数简介


```
$ ninja -help
usage: ninja [options] [targets...]

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

options:
  --version  print ninja version ("1.8.2")

  -C DIR   change to DIR before doing anything else
  -f FILE  specify input build file [default=build.ninja]

  -j N     run N jobs in parallel [default=6, derived from CPUs available]
  -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)
```

以下debug的选项，可以在ninja build过程中，给log增加额外的信息

```
$ ninja -d list
debugging modes:
  stats        print operation counts/timing info
  explain      explain what caused a command to execute ()
  keepdepfile  don't delete depfiles after they're read by ninja
  keeprsp      don't delete @response files on success
multiple modes can be enabled via -d FOO -d BAR
```

以下tools的选项，可以输出ninja文件所包含的依赖、input/output等等

```
$ ninja -t list
ninja subtools:
    browse  browse dependency graph in a web browser
     clean  clean built files
  commands  list all commands required to rebuild given targets
      deps  show dependencies stored in the deps log
     graph  output graphviz dot file for targets
     query  show inputs/outputs for a path
   targets  list targets by their rule or depth in the DAG
    compdb  dump JSON compilation database to stdout
 recompact  recompacts ninja-internal data structures
```

V8-Ninja实战5-实际运用

- V8的编译
  - ninja -C out/x64.release d8 -j 1: 在v8 src目录下，按照out/x64.release目录下的ninja文件规则来编译d8，只启动一个job
  - ninja -v -C out/x64.release d8 -j 1: 编译时verbose所有的command
  - ninja -C out/x64.release v8_hello_world: 编译一个可执行demo程序
- Ninja extra tools的使用（利用它们探索.ninja文件和编译过程的关系）
  - ninja -t query out/x64.release d8: 输出target d8的所有直接的input和output
  - ninja -t clean out/x64.release d8: 相当于make clean，去除编译d8所产生的所有中间文件（stamp、obj、.a等）
  - ninja -t targets rule cxx/link/…: 打印某个rule所产生的所有target的列表
  - ninja -t commands d8: 产生d8所需要的所有命令，跟-v所产生的log不同，这log是有顺序的，可以重新执行（*debug和移植中很有用）

推荐将本页中的命令作为课后的实践练习，结合Ninja的文档一起学习

GN概览

- GN是什么？
  - 一种meta-build system，也包括特殊的领域专用语言（DSL）
  - 用于产生Ninja文件
  - 用于给Google Chrome产生构建配置
- GN的设计初衷
  - 具有较小灵活性的构建任务描述语言：同样的任务，不同的人员描述应该是相同的

GN的安装和运行

- 安装：
  - 获取depot_tools repo
  - 获取chromium buildtools
    - https://chromium.googlesource.com/chromium/src/buildtools.git
- 运行：
  - 设置PATH到depot_tools安装目录
  - 建立out目录，如v8/out/foo
  - 在v8/out/foo建立args.gn文件
  - 运行 “gn gen out/foo/”

GN-reference文档要点导读1

- 完全参考文档：
  - https://gn.googlesource.com/gn/+/master/docs/reference.md
- commands：介绍gn的命令行中可用的命令
  - gn gen: 产生ninja文件
  - gn args: “gn args --list ./out/x64.release”
  - gn ls: “gn ls ./out/x64.release ”：列举出当前目录下BUILD.gn文件中的所有target
  - gn desc:
    - “gn desc ./out/x64.release //:d8 ”：列举target //:d8的所有信息，包括type, toolchain, sources, configs, outputs, arflags, asmflags, cflags, defines, Direct dependencies, libs
    - “gn desc ./out/x64.release //:d8 deps --all --tree”:列举target //:d8的所有依赖信息，使用树状的层次结构来展示
  - gn path: “gn path ./out/x64.release //:d8 //:torque_base”：展示target //:d8和target：//torque_base之间的依赖路径，用于了解两个 targets 之间的关系

build 参数来自于该目录的args.gn

推荐将本页中的命令作为课后的实践练习，结合Ninja的文档一起学习

GN-reference文档要点导读2

- Target declarations：介绍如何声明gn的构建目标（动作）
  - group：声明一组targets，并赋予名字
  - executable：声明一个产生可执行文件的target
  - source_set: 声明一个源文件集合，用于某一类targets的构建
  - shared_library:声明一个产生共享库的target
  - ……
- Buildfile functions: GN内置的函数
  - config：定义一个configuration的对象
  - declare_args: 声明一个或一组构建参数
  - template：定义一个模版规则
  - toolchain：定义一个工具链
  - get_label_info: 从一个label获得相关信息
  - import：在当前作用域import一个文件
  - print：打印信息

GN-Ninja-V8实战-执行流程1

- 参考文档和运行gn gen-v out/x64.release后观察执行流程：
  - 读取执行目录下的.gn文件

```
# This file is used by the GN meta build system to find the root of the source
# tree and to set startup options. For documentation on the values set in this
# file, run "gn help dotfile" at the command line.

import("//build/dotfile_settings.gni")

# The location of the build configuration file.

buildconfig = "//build/config/BUILDCONFIG.gn"

# These are the targets to check headers for by default. The files in targets
# matching these patterns (see "gn help label_pattern" for format) will have
# their includes checked for proper dependencies when you run either
# "gn check" or "gn gen --check".

check_targets = []

# These are the list of GN files that run exec_script. This whitelist exists
# to force additional review for new uses of exec_script, which is strongly
# discouraged except for gypi_to_gn calls.

exec_script_whitelist = build_dotfile_settings.exec_script_whitelist +
[ "//build_overrides/build.gni" ]
```

Chrome源码树中的构建配置文件，设置全局的参数、变量、默认值

GN-Ninja-V8实战-执行流程2

- 参考文档和运行gn gen -v out/x64.release后观察执行流程：
  - 读取当前执行目录下的.gn文件，如果找不到，就依次向上一层目录寻找，直到找到一个.gn为止，该.gn文件所在目录就是“source root”，作为接下来执行过程中//的相对位置
  - 读取//build/config/BUILDCONFIG.gn和out/x64.release目录下的args.gn 文件，确定参数
  - 读取并执行source root目录中的BUILD.gn
  - 递归地读取其他目录下的BUILD.gn来解析所有依赖
  - 当解析完成一个target的所有依赖时，写入一个.ninja文件
  - 当解析完成所有targets的依赖时，写入build.ninja文件

推荐将本页中的命令作为课后的实践练习，结合Ninja的文档一起学习

目 录

下节报告的内容，作为准备，可以先将本次内容中推荐的命令行进行实践，并分析log

Q&A

谢 谢

欢迎交流合作

2020/02/15