Chromium 可以采用gn和gyp两种方式编译，目前chromium团队已经放弃gyp编译，转到了gn编译这种方式上面，因此，我们有必要了解和学习gn的编译体系。

1. GN的语法

GN是一门类似于CMake的脚本语言，主要采用函数式编程的方式去定义一个编译脚本，主要包括以下几方面的函数。

1. GN的定义函数

action: Declare a target that runs a script a single time.

action\_foreach: Declare a target that runs a script over a set of files.

bundle\_data: [iOS/OS X] Declare a target without output.

copy: Declare a target that copies files.

create\_bundle: [iOS/OS X] Build an OS X / iOS bundle.

executable: Declare an executable target.

group: Declare a named group of targets.

loadable\_module: Declare a loadable module target.

shared\_library: Declare a shared library target.

source\_set: Declare a source set target.

static\_library: Declare a static library target.

target: Declare an target with the given programmatic type.

1. 构建文件函数

assert: Assert an expression is true at generation time.

config: Defines a configuration object.

declare\_args: Declare build arguments.

defined: Returns whether an identifier is defined.

exec\_script: Synchronously run a script and return the output.

foreach: Iterate over a list.

forward\_variables\_from: Copies variables from a different scope.

get\_label\_info: Get an attribute from a target's label.

get\_path\_info: Extract parts of a file or directory name.

get\_target\_outputs: [file list] Get the list of outputs from a target.

getenv: Get an environment variable.

import: Import a file into the current scope.

pool: Defines a pool object.

print: Prints to the console.

process\_file\_template: Do template expansion over a list of files.

read\_file: Read a file into a variable.

rebase\_path: Rebase a file or directory to another location.

set\_default\_toolchain: Sets the default toolchain name.

set\_defaults: Set default values for a target type.

set\_sources\_assignment\_filter: Set a pattern to filter source files.

split\_list: Splits a list into N different sub-lists.

template: Define a template rule.

tool: Specify arguments to a toolchain tool.

toolchain: Defines a toolchain.

write\_file: Write a file to disk.

1. 参数设置函数

all\_dependent\_configs: [label list] Configs to be forced on dependents.

allow\_circular\_includes\_from: [label list] Permit includes from deps.

arflags: [string list] Arguments passed to static\_library archiver.

args: [string list] Arguments passed to an action.

asmflags: [string list] Flags passed to the assembler.

assert\_no\_deps: [label pattern list] Ensure no deps on these targets.

bundle\_deps\_filter: [label list] A list of labels that are filtered out.

bundle\_executable\_dir: Expansion of {{bundle\_executable\_dir}} in create\_bundle

bundle\_plugins\_dir: Expansion of {{bundle\_plugins\_dir}} in create\_bundle.

bundle\_resources\_dir: Expansion of {{bundle\_resources\_dir}} in create\_bundle.

bundle\_root\_dir: Expansion of {{bundle\_root\_dir}} in create\_bundle.

cflags: [string list] Flags passed to all C compiler variants.

cflags\_c: [string list] Flags passed to the C compiler.

cflags\_cc: [string list] Flags passed to the C++ compiler.

1. GN编译脚本说明

GN脚本以BUILD.gn命名，放置在需要编译的源码处，例如：

executable("hello\_world") {

sources = [

"hello\_world.cc",

]

}

我们创建了一个在/tools/gn/tutorial/BUILD.gn的编译脚本，然后将这个模块添加到根目录编译group中，例如：

group("root") {

deps = [

...

"//url",

"//tools/gn/tutorial:hello\_world",

]

}

然后在根目录执行命令，例如：

gn gen out/Default

ninja -C out/Default hello\_world

out/Default/hello\_world

当然我们也可以不添加模块至根目录编译脚本中，可以直接执行编译命令，例如：

ninja -C out/Default tools/gn/tutorial:hello\_world

这样就可以直接编译该模块。

编译一个依赖于静态库的可执行程序脚本如下：

首先创建一个编译脚本，用于编译静态库hello

static\_library("hello") {

sources = [

"hello.cc",

]

}

然后创建一个可执行程序的编译脚本，并且添加依赖为hello静态库

executable("say\_hello") {

sources = [

"say\_hello.cc",

]

deps = [

":hello",

]

}

最后执行编译命令：

ninja -C out/Default say\_hello

out/Default/say\_hello

这样就成功编译了一个依赖于静态库的可执行程序。

1. chromium编译说明

1.SystemWebView.apk应用的编译脚本路径为：

chromium/src/android\_webview/BUILD.gn