Soong 编译系统

在 Android 7.0 发布之前，Android 仅使用 GNU Make (https://www.gnu.org/software/make/) 描述和执行其编译规则。Make 编译系统得到了广泛的支持和使用，但在 Android 层面变得缓慢、容易出错、无法扩展且难以测试。Soong 编译系统 (https://android.googlesource.com/platform/build/soong/+/refs/heads/master/README.md)正好提供了 Android 编译所需的灵活性。

什么是 Soong?

Soong 编译系统 (https://android.googlesource.com/platform/build/soong/+/refs/heads/master/README.md)是在 Android 7.0 (Nougat) 中引入的，旨在取代 Make。这项工作仍在进展中。

如需了解一般说明 (https://android.googlesource.com/platform/build/+/master/Usage.txt)，请参阅 Android 开源项目 (AOSP) 中的 Android Make 编译系统 (https://android.googlesource.com/platform/build/+/master/README.md)说明；如需了解从 Make 转换到 Soong 所需执行的修改，请参阅 Android.mk 编写人员须知的编译系统变更 (https://android.googlesource.com/platform/build/+/master/Changes.md)。

如需查看示例 Soong 配置（Blueprint 或 `.bp`）文件，请参阅简单的编译配置 (/compatibility/tests/development/blueprints)；如需了解完整详情，请参阅 Soong 参考文件 (https://ci.android.com/builds/latest/branches/aosp-build-tools/targets/linux/view/soong_build.html)。

注意：在 Android 完全从 Make 转换为 Soong 之前，Make 产品配置必须指定 **`PRODUCT_SOONG_NAMESPACES`** 值。如需了解相关说明，请参阅命名空间模块 (#namespace_modules)部分。

Android.bp 文件格式

根据设计，**`Android.bp`** 文件很简单。它们不包含任何条件语句，也不包含控制流语句；所有复杂问题都由用 Go 编写的编译逻辑处理。**`Android.bp`** 文件的语法和语义尽可能类似于 Bazel BUILD 文件 (https://docs.bazel.build/versions/master/be/overview.html)。

模块

`Android.bp` 文件中的模块以模块类型 (https://ci.android.com/builds/latest/branches/aosp-build-tools/targets/linux/view/soong_build.html) 开头，后跟一组 `name: "value"`，格式的属性：

```
cc_binary {
        name: "gzip",
        srcs: ["src/test/minigzip.c"],
        shared_libs: ["libz"],
        stl: "none",
    }
```

每个模块都必须具有 `name` 属性，并且相应值在所有 文件中必须是唯一的，仅有两个例外情况是命名空间和预编译模块中的 `Android.bp` 属性值，这两个值可能会重复。

`srcs` 属性以字符串列表的形式指定用于编译模块的源文件。您可以使用模块引用语法 `":<module-name>"` 来引用生成源文件的其他模块的输出，如 `genrule` 或 `filegroup`。

如需查看有效模块类型及其属性的列表，请参阅 Soong 模块参考 (https://ci.android.com/builds/latest/branches/aosp-build-tools/targets/linux/view/soong_build.html)。

类型

变量和属性是强类型，变量根据第一项赋值动态变化，属性由模块类型静态设置。支持的类型为：

- 布尔值（`true` 或 `false`）
- 整数 (`int`)
- 字符串 (`"string"`)
- 字符串列表 (`["string1", "string2"]`)
- 映射 (`{key1: "value1", key2: ["value2"]}`)

映射可以包含任何类型的值，包括嵌套映射。列表和映射可能在最后一个值后面有尾随逗号。

glob

接受文件列表的属性（例如 `srcs`）也可以采用 glob 模式。glob 模式可以包含普通的 UNIX 通配符 `*`，例如 `*.java`。glob 模式还可以包含单个 `**` 通配符作为路径元素，与零个或多个

路径元素匹配。例如，`java/**/*.java` 同时匹配 `java/Main.java` 和 `java/com/android/Main.java` 模式。

变量

`Android.bp` 文件可能包含顶级变量赋值：

```
gzip_srcs = ["src/test/minigzip.c"],
    cc_binary {
        name: "gzip",
        srcs: gzip_srcs,
        shared_libs: ["libz"],
        stl: "none",
    }
```

变量的作用域限定在声明它们的文件的其余部分，以及所有子 Blueprint 文件。变量是不可变的，但有一个例外情况：可以使用 `+=` 赋值将变量附加到别处，但只能在引用它们之前附加。

注释

`Android.bp` 文件可以包含 C 样式的多行 `/* */` 注释以及 C++ 样式的单行 `//` 注释。

运算符

可以使用 `+` 运算符附加字符串、字符串列表和映射。可以使用 `+` 运算符对整数求和。附加映射会生成两个映射中键的并集，并附加两个映射中都存在的所有键的值。

条件语句

Soong 不支持 `Android.bp` 文件中的条件语句。但是，编译规则中需要条件语句的复杂问题将在 Go（在这种语言中，您可以使用高级语言功能，并且可以跟踪条件语句引入的隐式依赖项）中处理。大多数条件语句都会转换为映射属性，其中选择了映射中的某个值并将其附加到顶级属性。

例如，要支持特定于架构的文件，请使用以下命令：

```
cc_library {
        ...
        srcs: ["generic.cpp"],
        arch: {
            arm: {
                srcs: ["arm.cpp"],
            },
            x86: {
                srcs: ["x86.cpp"],
            },
        },
    }
```

格式设置工具

Soong 包含一个针对 Blueprint 文件的规范格式设置工具，类似于 gofmt (https://golang.org/cmd/gofmt/)。要以递归方式重新格式化当前目录中的所有 **Android.bp** 文件，请运行以下命令：

```
bpfmt -w .
```

规范格式包括缩进四个空格、多元素列表的每个元素后面有换行符，以及列表和映射末尾有逗号。

特殊模块

一些特殊模块组具有独特的特征。

默认模块

默认模块可用于在多个模块中重复使用相同的属性。例如：

```
cc_defaults {
        name: "gzip_defaults",
        shared_libs: ["libz"],
        stl: "none",
    }
```

```
    cc_binary {
        name: "gzip",
        defaults: ["gzip_defaults"],
        srcs: ["src/test/minigzip.c"],
    }
```

预编译的模块

某些预编译的模块类型允许模块与其基于源代码的对应模块具有相同的名称。例如，如果已有同名的 **cc_binary**，也可以将 **cc_prebuilt_binary** 命名为 **foo**。这让开发者可以灵活地选择要纳入其最终产品中的版本。如果编译配置包含两个版本，则预编译模块定义中的 **prefer** 标记值会指示哪个版本优先。请注意，某些预编译模块的名称不是以 **prebuilt** 开头，例如 **android_app_import**。

命名空间模块

在 Android 完全从 Make 转换为 Soong 之前，Make 产品配置必须指定 **PRODUCT_SOONG_NAMESPACES** 值。它的值应该是一个以空格分隔的列表，其中包含 Soong 导出到 Make 以使用 **m** 命令进行编译的命名空间。在 Android 完成到 Soong 的转换之后，启用命名空间的详细信息可能会发生变化。

Soong 可以让不同目录中的模块指定相同的名称，只要每个模块都在单独的命名空间中声明即可。可以按如下方式声明命名空间：

```
soong_namespace {
        imports: ["path/to/otherNamespace1", "path/to/otherNamespace2"],
    }
```

请注意，命名空间没有 name 属性；其路径会自动指定为其名称。

系统会根据每个 Soong 模块在树中的位置为其分配命名空间。每个 Soong 模块都会被视为处于 **Android.bp**（位于当前目录或最近的父级目录中的 **soong_namespace** 文件内）定义的命名空间中。如果未找到此类 **soong_namespace** 模块，则认为该模块位于隐式根命名空间中。

下面是一个示例：Soong 尝试解析由模块 M 在名称空间 N（导入命名空间 I1、I2、I3…）中声明的依赖项 D。

1. 如果 D 是 `//namespace:module` 格式的完全限定名称，系统将仅在指定的命名空间中搜索指定的模块名称。
2. 否则，Soong 将首先查找在命名空间 N 中声明的名为 D 的模块。
3. 如果该模块不存在，Soong 会在命名空间 I1、I2、I3…中查找名为 D 的模块。
4. 最后，Soong 会在根命名空间中进行查找。

Content and code samples on this page are subject to the licenses described in the Content License (/license).
Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2020-02-05.