**使用配置文件引导的优化**

Android编译系统支持在具有[blueprint](https://android.googlesource.com/platform/build/soong/" \t "_blank)构建规则的Android native 模块上使用Clang的[配置文件引导优化（PGO）](https://clang.llvm.org/docs/UsersManual.html" \l "profile-guided-optimization" \t "_blank)。本文描述Clang PGO如何持续生成和更新用于PGO的配置文件，以及如何将PGO与编译系统集成（使用用例）。

## 关于Clang PGO

Clang可以使用两种类型的配置文件执行配置文件引导优化：

* **基于检测的配置文件**是从检测的目标程序生成的。这些配置文件很详细，并且会产生高运行时开销。
* **基于采样的配置文件**通常通过采样硬件计数器来生成。它们会产生低运行时开销，并且无需对二进制文件进行任何检测或修改即可收集。它们没有基于检测的配置文件详细。

所有配置文件应该从应用程序的典型行为的代表性工作负载生成。虽然Clang同时支持基于AST的（-fprofile-instr-generate）和基于LLVM IR的（-fprofile-generate)，Android仅支持基于LLVM IR的基于检测的PGO。

构建配置文件集合需要以下标志：

* -fprofile-generate用于基于IR的仪器。使用此选项，后端使用加权最小生成树方法来减少检测点的数量并优化它们在低权重边缘的位置（对于链接步骤也使用此选项）。Clang驱动程序自动将运行时配置（libclang\_rt.profile-\*arch\*-android.a）传递给链接器。该库包含在程序退出时将配置文件写入磁盘的例程。
* -gline-tables-only 用于基于采样的配置文件收集，以生成最少的调试信息

配置文件可使用-fprofile-instr-use=\*pathname\*或-fprofile-instr-use=\*pathname\*来用于PGO，分别对应基于检测的配置文件和基于采样的配置文件。

**注：**当对代码进行更改时，如果Clang无法再使用配置文件数据，则会生成 -Wprofile-instr-out-of-date警告。

## 使用PGO

使用PGO涉及以下步骤：

1. 通过传递-fprofile-generate给编译器和链接器来构建带有检测的库/可执行文件 。
2. 通过在检测二进制文件上运行代表性工作负载来收集配置文件
3. 使用该llvm-profdata实用程序对配置文件进行后处理（有关详细信息，请参阅[处理LLVM配置文件](https://source.android.com/devices/tech/perf/pgo" \l "handling-llvm-profile-files" \t "_blank)）。
4. 通过传递-fprofile-use=<>.profdata给编译器和链接器来将配置文件应用于PGO 。

对于Android中的PGO，应该离线收集配置文件并与代码一起检查以确保可重现的构建。配置文件可以用作代码演变，但必须定期重新生成（或者每当Clang警告配置文件是陈旧的时）。

### 收集配置文件

Clang可以使用通过运行基准测试收集的配置文件，使用库的检测构建，或者在运行基准测试时通过采样硬件计数器。目前，Android不支持使用基于采样的配置文件集合，因此您必须使用经过检测的构建来收集配置文件：

1. 确定基准和由该基准共同行使的一组库。
2. 添加pgo属性到基准和库（详细信息如下）。
3. 使用以下方法生成带有这些库的检测副本的Android构建：

make ANDROID\_PGO\_INSTRUMENT=benchmark

\*benchmark\*是一个占位符，用于标识在构建期间检测的库集合。实际的代表性输入（以及可能与被基准测试的库链接的另一个可执行文件）并非特定于PGO，超出了本文档的范围。

1. 在设备上Flash或同步已检测的构建。
2. 运行基准测试以收集配置文件。
3. 使用该llvm-profdata工具（下面讨论）对配置文件进行后处理，并准备好将其签入源树。

### 在构建期间使用配置

在Android树中检查配置文件toolchain/pgo-profiles。该名称应与库profile\_file的pgo属性的子属性中指定的名称匹配 。构建库时，构建系统会自动将配置文件传递给Clang。该ANDROID\_PGO\_DISABLE\_PROFILE\_USE 环境变量可以被设置为**true**来暂时禁用PGO和衡量其性能优势。

要指定其他特定于产品的配置文件目录，请将它们附加到BoardConfig.mk里的make变量PGO\_ADDITIONAL\_PROFILE\_DIRECTORIES中。如果指定其他路径，这些路径配置文件覆盖在toolchain/pgo-profiles中的路径。

使用dist目标来make生成发布映像时，构建系统会将缺失的配置文件的名称写入$DIST\_DIR/pgo\_profile\_file\_missing.txt。您可以检查此文件以查看意外删除的配置文件（静默禁用PGO）。

## 在Android.bp文件中启用PGO

要在Android.bp文件中为native 模块启用PGO ，只需指定pgo属性即可。此属性具有以下子属性：

| **属性** | **描述** |
| --- | --- |
| instrumentation | PGO使用检测则设置为true。默认是 false。 |
| sampling | **目前不受支持。**PGO使用采样设置为true。默认是false。 |
| benchmarks | 字符串列表。如果在ANDROID\_PGO\_INSTRUMENT构建选项中指定了列表中的任何基准，则构建此模块用于分析。 |
| profile\_file | 用于PGO的配置文件（相对于toolchain/pgo-profile）。构建通过添加此文件至$DIST\_DIR/pgo\_profile\_file\_missing.txt 来警告此文件不存在，除非将enable\_profile\_use属性设置为false **或**将ANDROID\_PGO\_NO\_PROFILE\_USE构建变量设置为true |
| enable\_profile\_use | 若在构建期间不应使用配置文件则设置为false。可以在引导期间使用以启用配置文件收集或暂时禁用PGO。默认是true。 |
| cflags | 在检测的构建期间使用的其他标志的列表。 |

带PGO的模块示例：

cc\_library {

name: "libexample",

srcs: [

"src1.cpp",

"src2.cpp",

],

static: [

"libstatic1",

"libstatic2",

],

shared: [

"libshared1",

]

pgo: {

instrumentation: true,

benchmarks: [

"benchmark1",

"benchmark2",

],

profile\_file: "example.profdata",

}

}

如果基准benchmark1和benchmark2 行使代表行为库libstatic1， libstatic2或者libshared1，则这些库的pgo属性也包含基准。Android.bp中的defaults模块可include一个一系列库的共通pgo定义，以避免多个模块重复相同的构建规则。

为一个架构选择不同的配置文件或选择性地禁用PGO，需要指定每个体系结构的profile\_file， enable\_profile\_use以及cflags属性。例如（架构目标以**粗体**显示）：

cc\_library {

name: "libexample",

srcs: [

"src1.cpp",

"src2.cpp",

],

static: [

"libstatic1",

"libstatic2",

],

shared: [

"libshared1",

],

pgo: {

instrumentation: true,

benchmarks: [

"benchmark1",

"benchmark2",

],

}

target: {

android\_arm: {

pgo: {

profile\_file: "example\_arm.profdata",

}

},

android\_arm64: {

pgo: {

profile\_file: "example\_arm64.profdata",

}

}

}

}

要在基于检测的分析期间解析对分析运行时库的引用，请将构建标志 -fprofile-generate传递给链接器。使用PGO检测的静态库，所有共享库以及直接依赖于静态库的任何二进制文件也必须为PGO进行检测。但是，此类共享库或可执行文件不需要使用PGO配置文件，并且其enable\_profile\_use属性可以被设置为false。除此限制外，您可以将PGO应用于任何静态库，共享库或可执行文件。

## 处理LLVM配置文件

执行一个检测库或可执行文件在/data/local/tmp中生成一个名为default\_\*unique\_id\*\_0.profraw的配置文件 （其中unique\_id是此库唯一的数字哈希值）。如果此文件已存在，则分析运行时会在编写配置文件时将新配置文件与旧配置文件合并。要更改配置文件的位置，请在运行时设置LLVM\_PROFILE\_FILE环境变量。

[llvm-profdata](https://llvm.org/docs/CommandGuide/llvm-profdata.html)实用程序用于将.profraw文件（并可能合并多个.profraw文件）转换为.profdata 文件：

llvm-profdata merge -output=profile.profdata <.profraw and/or .profdata files>

然后\*profile.profdata\* 可被签入源码树以便在构建时使用。

如果在基准测试期间加载了多个检测二进制文件/库，则每个库都会生成一个具有唯一ID 的独立.profraw文件。通常，所有这些文件都可以合并为单个 .profdata文件并用于PGO构建。如果库由另一个基准测试执行，则必须使用两个基准测试的配置文件优化该库。在这种情况下，show 选项llvm-profdata是有用的：

llvm-profdata merge -output=default\_unique\_id.profdata default\_unique\_id\_0.profraw

llvm-profdata show -all-functions default\_unique\_id.profdata

要将unique\_id映射到单个库，请在每个unique\_id的show输出中搜索该库唯一的函数名称。

## 案例研究：ART的PGO

案例研究将ART作为一个相关的例子; 但是，它并不能准确描述为ART或其相互依赖性分析的实际库集。

ART中的dex2oat预编译器依赖于 libart-compiler.so，而后者依赖于 libart.so。ART运行时主要在 libart.so中实现。编译器和运行时的基准将是不同的：

| **基准** | **配置库** |
| --- | --- |
| dex2oat | dex2oat（可执行）， libart-compiler.so，libart.so |
| art\_runtime | libart.so |

1. 将以下pgo属性添加到dex2oat， libart-compiler.so：

pgo: {

instrumentation: true,

benchmarks: ["dex2oat",],

profile\_file: "dex2oat.profdata",

}

1. 将以下pgo属性添加到libart.so：

pgo: {

instrumentation: true,

benchmarks: ["art\_runtime", "dex2oat",],

profile\_file: "libart.profdata",

}

1. 使用以下方法为dex2oat和 art\_runtime基准创建检测构建：

make ANDROID\_PGO\_INSTRUMENT=dex2oat

make ANDROID\_PGO\_INSTRUMENT=art\_runtime

或者，使用以下方法创建一个具有所有库检测的单个检测构建：

make ANDROID\_PGO\_INSTRUMENT=dex2oat,art\_runtime

(or)

make ANDROID\_PGO\_INSTRUMENT=ALL

第二个命令构建**所有**启用PGO的模块以进行性能分析。

1. 运行基准测试dex2oat和 art\_runtime以获得：
   * 来自dex2oat （dex2oat\_exe.profdata， dex2oat\_libart-compiler.profdata和dexeoat\_libart.profdata）的三个.profraw文件，使用[处理LLVM配置文件](https://source.android.com/devices/tech/perf/pgo" \l "handling-llvm-profile-files" \t "_blank)中描述的方法进行标识。
   * 单个art\_runtime\_libart.profdata文件。
2. 为dex2oat可执行文件和 libart-compiler.so使用生成一个通用的profdata文件：

llvm-profdata merge -output=dex2oat.profdata \

dex2oat\_exe.profdata dex2oat\_libart-compiler.profdata

1. 通过合并两个基准测试中的配置文件来获取libart.so的配置文件：

llvm-profdata merge -output=libart.profdata \

dex2oat\_libart.profdata art\_runtime\_libart.profdata

libart.so的两个配置文件的原始计数可能是不同的，因为基准测试用例的数量和运行的持续时间不同。在这种情况下，您可以使用加权合并：

llvm-profdata merge -output=libart.profdata \

-weighted-input=2,dex2oat\_libart.profdata \

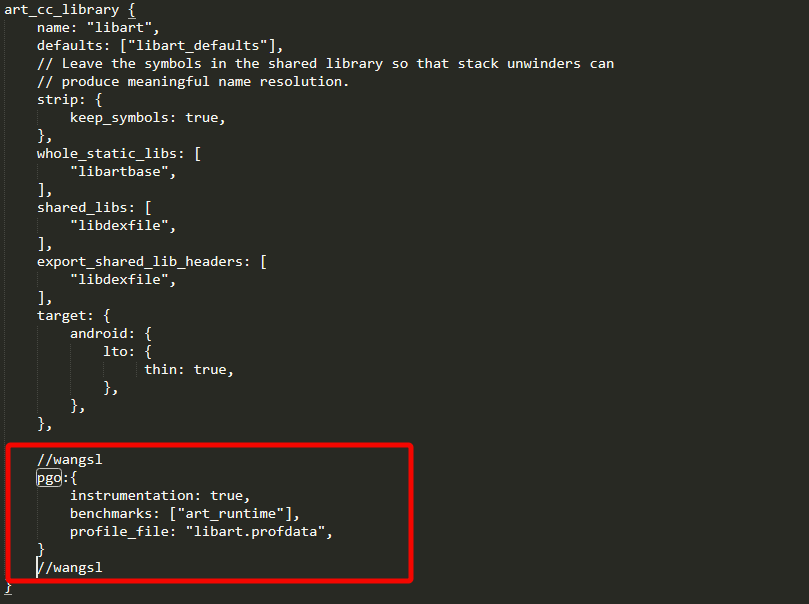
-weighted-input=1,art\_runtime\_libart.profdata

上面的命令为 dex2oat配置文件赋予两倍的权重。实际权值应根据域知识或实验确定。

1. 将dex2oat.profdata和 libart.profdata配置文件签入到toolchain/pgo-profiles以便构建时使用。

# 备忘：RM500上加入PGO

1. 加入pgo控制



1. 重新编译

make ANDROID\_PGO\_INSTRUMENT=art\_runtime

1. 烧机，跑monkey等操作，重启手机
2. data/local/tmp下会自动生成prof文件
3. 系统安装llvm-dev tool后

llvm-profdata merge -output=libart.profdata XXXX

XXX为data/local/tmp下导出的文件

1. 将新生成的libart.profdata放入toolchain\pgo-profiles\art
2. 重新编译版本

参考：

<https://www.jianshu.com/p/bd2fe89e2025>

<https://source.android.google.cn/devices/tech/perf/pgo>