



向您的项目添加 C 和 C++ 代码

搭配使用 Android Studio 2.2 或更高版本 (<https://developer.android.google.cn/studio/index.html>) 与 Android Plugin for Gradle 版本 2.2.0 或更高版本 (<https://developer.android.google.cn/studio/releases/gradle-plugin.html>) 时，您可以将 C 和 C++ 代码编译到 Gradle 与 APK 一起打包的原生库中，将这类代码添加到您的应用中。您的 Java 代码随后可以通过 Java 原生接口 (JNI) 调用您的原生库中的函数。如果您想要详细了解如何使用 JNI 框架，请阅读 Android 的 JNI 提示 (<https://developer.android.google.cn/training/articles/perf-jni.html>)。

Android Studio 用于构建原生库的默认工具是 CMake。由于很多现有项目都使用构建工具包编译其原生代码，Android Studio 还支持 ndk-build (<https://developer.android.google.cn/ndk/guides/ndk-build.html>)。如果您想要将现有的 ndk-build 库导入到您的 Android Studio 项目中，请参阅介绍如何配置 Gradle 以关联到您的原生库 (#link-gradle) 的部分。不过，如果您在创建新的原生库，则应使用 CMake。

本页面介绍的信息可以帮助您使用所需构建工具设置 Android Studio、创建或配置项目以支持 Android 上的原生代码，以及构建和运行应用。

注：如果您的现有项目使用已弃用的 `ndkCompile` 工具，则应先打开 `build.properties` 文件，并移除以下代码行，然后再将 Gradle 关联到您的原生库 (#link-gradle)：

```
// Remove this line
android.useDeprecatedNdk = true
```

实验性 Gradle 的用户注意事项：如果您是以下任意一种情况，请考虑迁移到插件版本 2.2.0 或更高版本 (<http://tools.android.com/tech-docs/new-build-system/gradle-experimental/migrate-to-stable>) 并使用 CMake 或 ndk-build 构建原生库：您的原生项目已经使用 CMake 或者 ndk-build；但是您想要使用稳定版本的 Gradle 构建系统；或者您希望支持插件工具，例如 CCache (<https://ccache.samba.org/>)。否则，您可以继续使用实验性版本的 Gradle 和

本文内容

下载 NDK 和构建工具

创建支持 C/C++ 的新项目

构建和运行示例应用

向现有项目添加 C/C++ 代码

创建新的原生源文件

创建 CMake 构建脚本

将 Gradle 关联到您的原生库

下载 NDK 和构建工具

要为您的应用编译和调试原生代码，您需要以下组件：

- *Android 原生开发工具包 (NDK)* (<https://developer.android.google.cn/ndk/index.html>)：这套工具集允许您为 Android 使用 C 和 C++ 代码，并提供众多平台库，让您管理原生 Activity 和访问物理设备组件，例如传感器和触摸输入。
- *CMake* (<https://cmake.org/>)：一款外部构建工具，可与 Gradle 搭配使用来构建原生库。如果您只计划使用 ndk-build，则不需要此组件。
- *LLDB* (<http://lldb.llvm.org/>)：一种调试程序，Android Studio 使用它来调试原生代码 (<https://developer.android.google.cn/studio/debug/index.html>)。

您可以使用 SDK 管理器 (<https://developer.android.google.cn/studio/intro/update.html#sdk-manager>) 安装这些组件：

1. 在打开的项目中，从菜单栏选择 **Tools > Android > SDK Manager**。
2. 点击 **SDK Tools** 标签。

3. 选中 **LLDB**、**CMake** 和 **NDK** 旁的复选框，如图 1 所示。

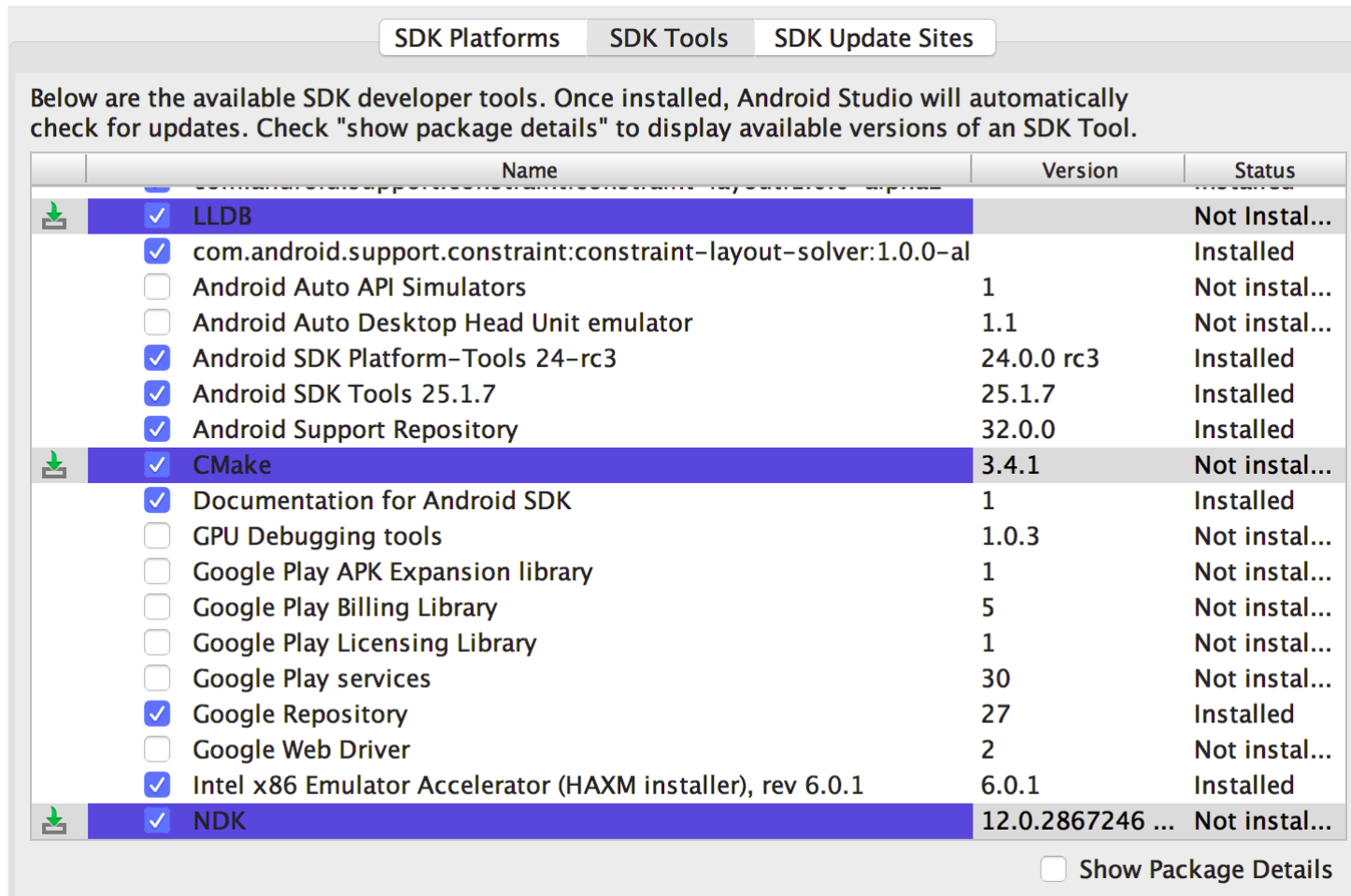


图 1. 从 SDK 管理器中安装 LLDB、CMake 和 NDK。

4. 点击 **Apply**，然后在弹出式对话框中点击 **OK**。

5. 安装完成后，点击 **Finish**，然后点击 **OK**。

创建支持 C/C++ 的新项目

This site uses cookies to store your preferences for site-specific language and display options.

OK

创建支持原生代码的项目与创建任何其他 Android Studio 项目 (<https://developer.android.google.cn/studio/projects/create-project.html>) 类似，不过前者还需要额外几个步骤：

1. 在向导的 **Configure your new project** 部分，选中 **Include C++ Support** 复选框。
2. 点击 **Next**。
3. 正常填写所有其他字段并完成向导接下来的几个部分。
4. 在向导的 **Customize C++ Support** 部分，您可以使用下列选项自定义项目：
 - **C++ Standard**：使用下拉列表选择您希望使用哪种 C++ 标准。选择 **Toolchain Default** 会使用默认的 CMake 设置。
 - **Exceptions Support**：如果您希望启用对 C++ 异常处理的支持，请选中此复选框。如果启用此复选框，Android Studio 会将 `-fexceptions` 标志添加到模块级 `build.gradle` 文件的 `cppFlags` 中，Gradle 会将其传递到 CMake。
 - **Runtime Type Information Support**：如果您希望支持 RTTI，请选中此复选框。如果启用此复选框，Android Studio 会将 `-frtti` 标志添加到模块级 `build.gradle` 文件的 `cppFlags` 中，Gradle 会将其传递到 CMake。
5. 点击 **Finish**。

在 Android Studio 完成新项目的创建后，请从 IDE 左侧打开 **Project** 窗格并选择 **Android** 视图。如图 2 中所示，Android Studio 将添加 **cpp** 和 **External Build Files** 组：

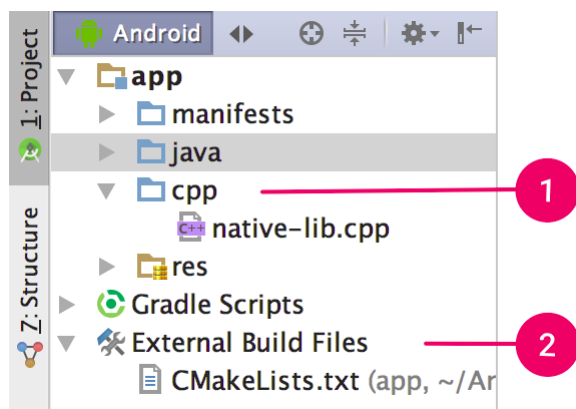


图 2. 您的原生源文件和外部构建脚本的 Android 视图组。

注：此视图无法反映磁盘上的实际文件层次结构，而是将相似文件分到一组中，简化项目导航。

- 1 在 **cpp** 组中，您可以找到属于项目的所有原生源文件、标头和预构建库。对于新项目，Android Studio 会创建一个示例 C++ 源文件 `native-lib.cpp`，并将其置于应用模块的 `src/main/cpp/` 目录中。本示例代码提供了一个简单的 C++ 函数 `stringFromJNI()`，此函数可以返回字符串“Hello from C++”。要了解如何向项目添加其他源文件，请参阅介绍如何创建新的原生源文件 (#create-sources) 的部分。
- 2 在 **External Build Files** 组中，您可以找到 CMake 或 ndk-build 的构建脚本。与 `build.gradle` 文件指示 Gradle 如何构建应用一样，CMake 和 ndk-build 需要一个构建脚本来了解如何构建您的原生库。对于新项目，Android Studio 会创建一个 CMake 构建脚本 `CMakeLists.txt`，并将其置于模块的根目录中。要详细了解此构建脚本的内容，请参阅介绍如何创建 Cmake 构建脚本 (#create-cmake-script) 的部分。

构建和运行示例应用

点击 **Run** 后，Android Studio 将在您的 Android 设备或者模拟器上构建并启动一个显示文字“Hello from C++”的应用。下面的概览介绍了构建和运行示例应用时会发生的事件：

1. Gradle 调用您的外部构建脚本 `CMakeLists.txt`。
2. CMake 按照构建脚本中的命令将 C++ 源文件 `native-lib.cpp` 编译到共享的对象库中，并命名为 `libnative-lib.so`，Gradle 随后会将其打包到 APK 中。
3. 运行时，应用的 `MainActivity` 会使用 `System.loadLibrary()` ([https://developer.android.google.cn/reference/java/lang/System.html#loadLibrary\(java.lang.String\)](https://developer.android.google.cn/reference/java/lang/System.html#loadLibrary(java.lang.String))) 加载原生库。现在，应用可以使用库的原生函数 `stringFromJNI()`。
4. `MainActivity.onCreate()` 调用 `stringFromJNI()`，这将返回“Hello from C++”并使用这些文字更新 `TextView` (<https://developer.android.google.cn/reference/android/widget/TextView.html>)。

注：Instant Run (<https://developer.android.google.cn/studio/run/index.html#instant-run>) 与使用原生代码的项目不兼容。Android Studio 会自动停用此功能。

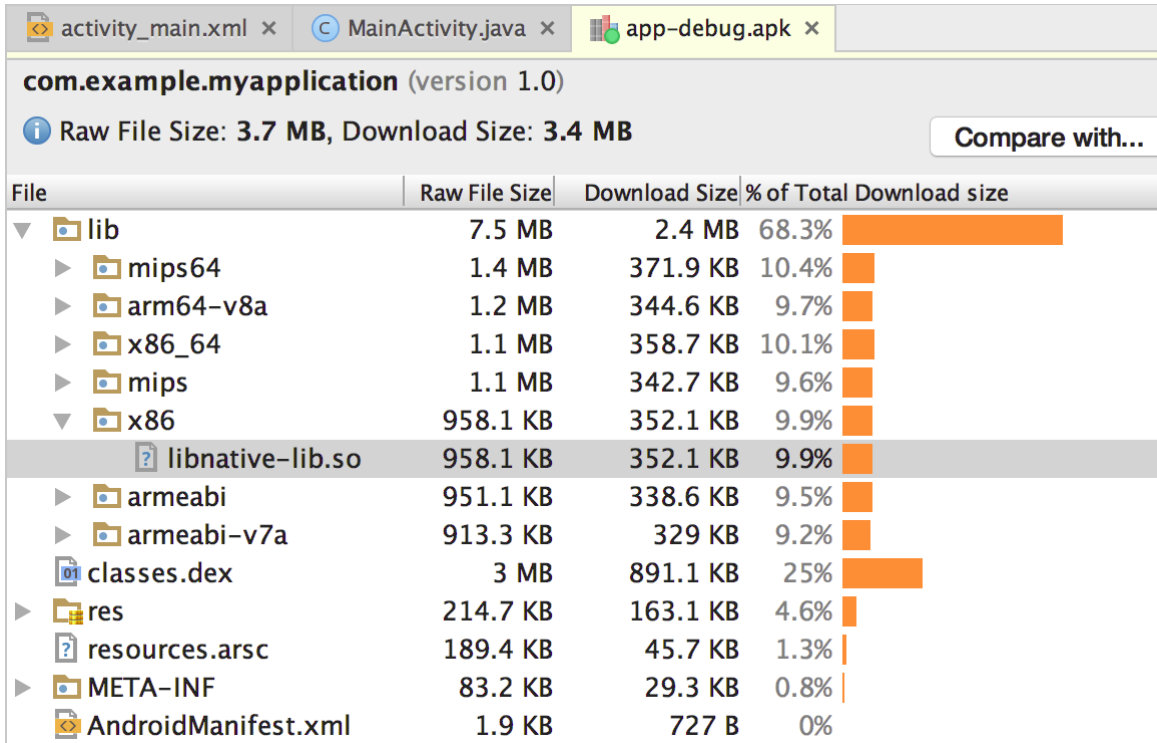
如果您想要验证 Gradle 是否已将原生库打包到 APK 中，可以使用 APK 分析器 (<https://developer.android.google.cn/studio/build/apk-analyzer.html>)：

1. 选择 **Build > Analyze APK**。

This site uses cookies to store your preferences for site-specific language and display options.

OK

3. 如图 3 中所示，您会在 APK 分析器窗口的 `lib/<ABI>/` 下看到 `libnative-lib.so`。



com.example.myapplication (version 1.0)

Raw File Size: 3.7 MB, Download Size: 3.4 MB Compare with...

File	Raw File Size	Download Size	% of Total Download size
lib	7.5 MB	2.4 MB	68.3%
mips64	1.4 MB	371.9 KB	10.4%
arm64-v8a	1.2 MB	344.6 KB	9.7%
x86_64	1.1 MB	358.7 KB	10.1%
mips	1.1 MB	342.7 KB	9.6%
x86	958.1 KB	352.1 KB	9.9%
libnative-lib.so	958.1 KB	352.1 KB	9.9%
armeabi	951.1 KB	338.6 KB	9.5%
armeabi-v7a	913.3 KB	329 KB	9.2%
classes.dex	3 MB	891.1 KB	25%
res	214.7 KB	163.1 KB	4.6%
resources.arsc	189.4 KB	45.7 KB	1.3%
META-INF	83.2 KB	29.3 KB	0.8%
AndroidManifest.xml	1.9 KB	727 B	0%

图 3. 使用 APK 分析器定位原生库。

提示：如果您想要试验使用原生代码的其他 Android 应用，请点击 **File > New > Import Sample** 并从 **Ndk** 列表中选择示例项目。


向现有项目添加 C/C++ 代码

如果您希望向现有项目添加原生代码，请执行以下步骤：

1. 创建新的原生源文件 (#create-sources)并将其添加到您的 Android Studio 项目中。
 - 如果您已经拥有原生代码或想要导入预构建的原生库，则可以跳过此步骤。

- 如果您的现有原生库已经拥有 `CMakeLists.txt` 构建脚本或者使用 `ndk-build` 并包含 `Android.mk` (https://developer.android.google.cn/ndk/guides/android_mk.html) 构建脚本，则可以跳过此步骤。

3. 提供一个指向您的 CMake 或 `ndk-build` 脚本文件的路径，将 Gradle 关联到您的原生库 (`#link-gradle`)。Gradle 使用构建脚本将源代码导入您的 Android Studio 项目并将原生库 (SO 文件) 打包到 APK 中。

配置完项目后，您可以使用 JNI 框架 (<http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html>) 从 Java 代码中访问您的原生函数。要构建和运行应用，只需点击 **Run** 。Gradle 会以依赖项的形式添加您的外部原生构建流程，用于编译、构建原生库并将其随 APK 一起打包。

创建新的原生源文件

要在应用模块的主源代码集中创建一个包含新建原生源文件的 `cpp/` 目录，请按以下步骤操作：

1. 从 IDE 的左侧打开 **Project** 窗格并从下拉菜单中选择 **Project** 视图。
2. 导航到 **您的模块** > **src**，右键点击 **main** 目录，然后选择 **New > Directory**。
3. 为目录输入一个名称（例如 `cpp`）并点击 **OK**。
4. 右键点击您刚刚创建的目录，然后选择 **New > C/C++ Source File**。
5. 为您的源文件输入一个名称，例如 `native-lib`。
6. 从 **Type** 下拉菜单中，为您的源文件选择文件扩展名，例如 `.cpp`。
 - 点击 **Edit File Types** ，您可以向下拉菜单中添加其他文件类型，例如 `.cxx` 或 `.hxx`。在弹出的 **C/C++** 对话框中，从 **Source Extension** 和 **Header Extension** 下拉菜单中选择另一个文件扩展名，然后点击 **OK**。
7. 如果您还希望创建一个标头文件，请选中 **Create an associated header** 复选框。
8. 点击 **OK**。

创建 CMake 构建脚本

如果您的原生源文件还没有 CMake 构建脚本，则需要自行创建一个并包含适当的 CMake 命令。CMake 构建脚本是一个纯文本文件，您必须将其命名为 `CMakeLists.txt`。本部分介绍了您应包含到构建脚本中的一些基本命令，用于在创建原生库时指示 CMake 应使用哪些源文件。

注：如果您的项目使用 ndk-build，则不需要创建 CMake 构建脚本。提供一个指向您的 `Android.mk` (https://developer.android.google.cn/ndk/guides/android_mk.html) 文件的路径，将 Gradle 关联到您的原生库 (`#link-gradle`)。

要创建一个可以用作 CMake 构建脚本的纯文本文件，请按以下步骤操作：

1. 从 IDE 的左侧打开 **Project** 窗格并从下拉菜单中选择 **Project** 视图。
2. 右键点击 **您的模块** 的根目录并选择 **New > File**。

注：您可以在所需的任意位置创建构建脚本。不过，在配置构建脚本时，原生源文件和库的路径将与构建脚本的位置相关。

3. 输入“CMakeLists.txt”作为文件名并点击 **OK**。

现在，您可以添加 CMake 命令，对您的构建脚本进行配置。要指示 CMake 从原生源代码创建一个原生库，请将 `cmake_minimum_required()` (https://cmake.org/cmake/help/latest/command/cmake_minimum_required.html) 和 `add_library()` (https://cmake.org/cmake/help/latest/command/add_library.html) 命令添加到您的构建脚本中：

```
# Sets the minimum version of CMake required to build your native library.
# This ensures that a certain set of CMake features is available to
# your build.

cmake_minimum_required(VERSION 3.4.1)

# Specifies a library name, specifies whether the library is STATIC or
# SHARED, and provides relative paths to the source code. You can
# define multiple libraries by adding multiple add_library() commands,
# and CMake builds them for you. When you build your app, Gradle
# automatically packages shared libraries with your APK.

add_library( # Specifies the name of the library.
            native-lib
```

This site uses cookies to store your preferences for site-specific language and display options.

OK

SHARED

```
# Provides a relative path to your source file(s).
src/main/cpp/native-lib.cpp )
```

使用 `add_library()` 向您的 CMake 构建脚本添加源文件或库时，Android Studio 还会在您同步项目后在 **Project** 视图下显示关联的标头文件。不过，为了确保 CMake 可以在编译时定位您的标头文件，您需要将 `include_directories()`

(https://cmake.org/cmake/help/latest/command/include_directories.html) 命令添加到 CMake 构建脚本中并指定标头的路径：

```
add_library(...)

# Specifies a path to native header files.
include_directories(src/main/cpp/include/)
```

CMake 使用以下规范来为库文件命名：

`lib库名称.so`

例如，如果您在构建脚本中指定“native-lib”作为共享库的名称，CMake 将创建一个名称为 `libnative-lib.so` 的文件。不过，在 Java 代码中加载此库时，请使用您在 CMake 构建脚本中指定的名称：

```
static {
    System.loadLibrary("native-lib");
}
```

注：如果您在 CMake 构建脚本中重命名或移除某个库，您需要先清理项目，Gradle 随后才会应用更改或者从 APK 中移除旧版本的库。要清理项目，请从菜单栏中选择 **Build > Clean Project**。

Android Studio 会自动将源文件和标头添加到 **Project** 窗格的 **cpp** 组中。使用多个 `add_library()` 命令，您可以为 CMake 定义要从其他源文件构建的更多库。

添加 NDK API

This site uses cookies to store your preferences for site-specific language and display options.

OK

Android NDK 提供了一套实用的原生 API 和库。通过将 NDK 库 (https://developer.android.google.cn/ndk/guides/stable_apis.html)包含到项目的 `CMakeLists.txt` 脚本文件中，您可以使用这些 API 中的任意一种。

预构建的 NDK 库已经存在于 Android 平台上，因此，您无需再构建或将其打包到 APK 中。由于 NDK 库已经是 CMake 搜索路径的一部分，您甚至不需要在您的本地 NDK 安装中指定库的位置 - 只需要向 CMake 提供您希望使用的库的名称，并将其关联到您自己的原生库。

将 `find_library()` (https://cmake.org/cmake/help/latest/command/find_library.html) 命令添加到您的 CMake 构建脚本中以定位 NDK 库，并将其路径存储为一个变量。您可以使用此变量在构建脚本的其他部分引用 NDK 库。以下示例可以定位 Android 特定的日志支持库 (https://developer.android.google.cn/ndk/guides/stable_apis.html#a3)并将其路径存储在 `log-lib` 中：

```
find_library( # Defines the name of the path variable that stores the
             # location of the NDK library.
             log-lib

             # Specifies the name of the NDK library that
             # CMake needs to locate.
             log )
```

为了确保您的原生库可以在 `log` 库中调用函数，您需要使用 CMake 构建脚本中的 `target_link_libraries()` (https://cmake.org/cmake/help/latest/command/target_link_libraries.html) 命令关联库：

```
find_library(...)

# Links your native library against one or more other native libraries.
target_link_libraries( # Specifies the target library.
                      native-lib

                      # Links the log library to the target library.
                      ${log-lib} )
```

NDK 还以源代码的形式包含一些库，您在构建和关联到您的原生库时需要使用这些代码。您可以使用 CMake 构建脚本中的 `add_library()` 命令，将源代码编译到原生库中。要提供本地 NDK 库的路径，您可以使用 `ANDROID_NDK` 路径变量，Android Studio 会自动为您定义此变量。

以下命令可以指示 CMake 构建 `android_native_app_glue.c`，后者会将 `NativeActivity`

(<https://developer.android.google.cn/reference/android/app/NativeActivity.html>) 生命周期事件和触摸输入置于静态库中并将静态库关联到 `native-lib`：

```
add_library( app-glue
            STATIC
            ${ANDROID_NDK}/sources/android/native_app_glue/android_native_app_glue.c )

# You need to link static libraries against your shared native library.
target_link_libraries( native-lib app-glue ${log-lib} )
```

添加其他预构建库

添加预构建库与为 CMake 指定要构建的另一个原生库类似。不过，由于库已经预先构建，您需要使用 `IMPORTED`

(https://cmake.org/cmake/help/latest/prop_tgt/IMPORTED.html#prop_tgt:IMPORTED) 标志告知 CMake 您只希望将库导入到项目中：

```
add_library( imported-lib
            SHARED
            IMPORTED )
```

然后，您需要使用 `set_target_properties()` (https://cmake.org/cmake/help/latest/command/set_target_properties.html) 命令指定库的路径，如下所示。

某些库为特定的 CPU 架构（或应用二进制接口 (ABI) (<https://developer.android.google.cn/ndk/guides/abis.html>））提供了单独的软件包，并将其组织到单独的目录中。此方法既有助于库充分利用特定的 CPU 架构，又能让您仅使用所需的库版本。要向 CMake 构建脚本中添加库的多个 ABI 版本，而不必为库的每个版本编写多个命令，您可以使用 `ANDROID_ABI` 路径变量。此变量使用 NDK 支持的一组默认 ABI

(<https://developer.android.google.cn/ndk/guides/abis.html#sa>)，或者您手动配置 Gradle (`#specify-abi`) 而让其使用的一组经过筛选的 ABI。例如：

```
add_library(...)
set_target_properties( # Specifies the target library.
                      imported-lib

                      # Specifies the parameter you want to define.
                      PROPERTIES IMPORTED_LOCATION
```

This site uses cookies to store your preferences for site-specific language and display options.

OK

```
# Provides the path to the library you want to import.
imported-lib/src/${ANDROID_ABI}/libimported-lib.so )
```

为了确保 CMake 可以在编译时定位您的标头文件，您需要使用 `include_directories()` 命令，并包含标头文件的路径：

```
include_directories( imported-lib/include/ )
```

注：如果您希望打包一个并不是构建时依赖项的预构建库（例如在添加属于 `imported-lib` 依赖项的预构建库时），则不需要执行以下说明来关联库。

要将预构建库关联到您自己的原生库，请将其添加到 CMake 构建脚本的 `target_link_libraries()` 命令中：


```
target_link_libraries( native-lib imported-lib app-glue ${log-lib} )
```

在您构建应用时，Gradle 会自动将导入的库打包到 APK 中。您可以使用 APK 分析器 (<https://developer.android.google.cn/studio/build/apk-analyzer.html>) 验证 Gradle 将哪些库打包到您的 APK 中。如需了解有关 CMake 命令的详细信息，请参阅 CMake 文档 (<https://cmake.org/cmake/help/latest/manual/cmake-commands.7.html>)。

将 Gradle 关联到您的原生库

要将 Gradle 关联到您的原生库，您需要提供一个指向 CMake 或 ndk-build 脚本文件的路径。在您构建应用时，Gradle 会以依赖项的形式运行 CMake 或 ndk-build，并将共享的库打包到您的 APK 中。Gradle 还使用构建脚本来了解要将哪些文件添加到您的 Android Studio 项目中，以便您可以从 **Project** 窗口访问这些文件。如果您的原生源文件没有构建脚本，则需要先创建 CMake 构建脚本 (`#create-cmake-script`)，然后再继续。

将 Gradle 关联到原生项目后，Android Studio 会更新 **Project** 窗格以在 **cpp** 组中显示您的源文件和原生库，在 **External Build Files** 组中显示您的外部构建脚本。

注：更改 Gradle 配置时，请确保通过点击工具栏中的 **Sync Project**  应用更改。此外，如果在将 CMake 或 ndk-build 脚本文件关联到 Gradle 后再对其进行更改，您应当从菜单栏中选择 **Build > Refresh Linked C++ Projects**，将 Android Studio 与您的更改同步。

您可以使用 Android Studio UI 将 Gradle 关联到外部 CMake 或 ndk-build 项目：

1. 从 IDE 左侧打开 **Project** 窗格并选择 **Android** 视图。
2. 右键点击您想要关联到原生库的模块（例如 **app** 模块），并从菜单中选择 **Link C++ Project with Gradle**。您应看到一个如图 4 所示的对话框。
3. 从下拉菜单中，选择 **CMake** 或 **ndk-build**。
 - a. 如果您选择 **CMake**，请使用 **Project Path** 旁的字段为您的外部 CMake 项目指定 **CMakeLists.txt** 脚本文件。
 - b. 如果您选择 **ndk-build**，请使用 **Project Path** 旁的字段为您的外部 ndk-build 项目指定 **Android.mk** (https://developer.android.google.cn/ndk/guides/android_mk.html) 脚本文件。如果 **Application.mk** (https://developer.android.google.cn/ndk/guides/application_mk.html) 文件与您的 **Android.mk** 文件位于相同目录下，Android Studio 也会包含此文件。

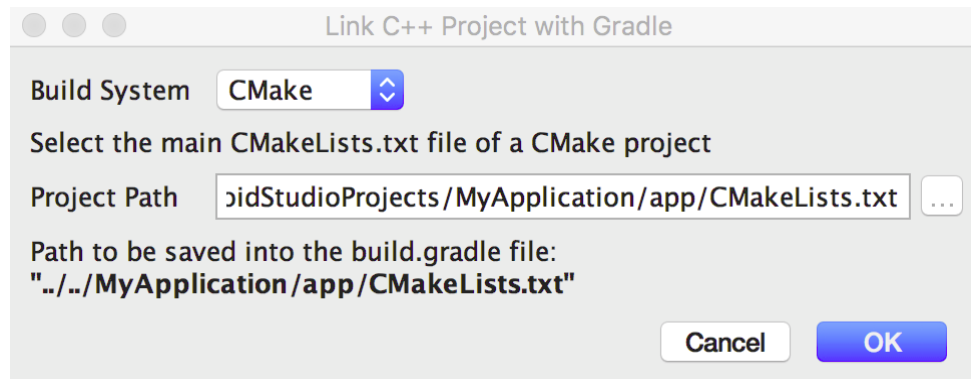


图 4. 使用 Android Studio 对话框关联外部 C++ 项目。

4. 点击 **OK**。

手动配置 Gradle

要手动配置 Gradle 以关联到您的原生库，您需要将 `externalNativeBuild {}` 块添加到模块级 `build.gradle` 文件中，并使用 `cmake {}` 或 `ndkBuild {}` 对其进行配置：

```
android {  
    ...  
}
```

This site uses cookies to store your preferences for site-specific language and display options.

OK

```
// Encapsulates your external native build configurations.
externalNativeBuild {

    // Encapsulates your CMake build configurations.
    cmake {

        // Provides a relative path to your CMake build script.
        path "CMakeLists.txt"
    }
}
}
```

注：如果您想要将 Gradle 关联到现有 ndk-build 项目，请使用 `ndkBuild {}` 块而不是 `cmake {}`，并提供 `Android.mk`

(https://developer.android.google.cn/ndk/guides/android_mk.html) 文件的相对路径。如果 `Application.mk`

(https://developer.android.google.cn/ndk/guides/application_mk.html) 文件与您的 `Android.mk` (https://developer.android.google.cn/ndk/guides/android_mk.html) 文件位于相同目录下，Gradle 也会包含此文件。

指定可选配置

您可以在模块级 `build.gradle` 文件的 `defaultConfig {}` 块中配置另一个 `externalNativeBuild {}` 块，为 CMake 或 ndk-build 指定可选参数和标志。与 `defaultConfig {}` 块中的其他属性类似，您也可以在构建配置中为每个产品风味重写这些属性。

例如，如果您的 CMake 或 ndk-build 项目定义多个原生库，您可以使用 `targets` 属性仅为给定产品风味构建和打包这些库中的一部分。以下代码示例说明了您可以配置的部分属性：

```
android {
    ...
    defaultConfig {
        ...
        // This block is different from the one you use to link Gradle
        // to your CMake or ndk-build script.
        externalNativeBuild {
```

```
// Passes optional arguments to CMake.
arguments "-DANDROID_ARM_NEON=TRUE", "-DANDROID_TOOLCHAIN=clang"

// Sets optional flags for the C compiler.
cFlags "-D_EXAMPLE_C_FLAG1", "-D_EXAMPLE_C_FLAG2"

// Sets a flag to enable format macro constants for the C++ compiler.
cppFlags "-D__STDC_FORMAT_MACROS"
}
}
}

buildTypes {...}

productFlavors {
    ...
    demo {
        ...
        externalNativeBuild {
            cmake {
                ...
                // Specifies which native libraries to build and package for this
                // product flavor. If you don't configure this property, Gradle
                // builds and packages all shared object libraries that you define
                // in your CMake or ndk-build project.
                targets "native-lib-demo"
            }
        }
    }
}

paid {
    ...
    externalNativeBuild {
        cmake {
            ...
            targets "native-lib-paid"
        }
    }
}
```

```
    }  
  }  
  
  // Use this block to link Gradle to your CMake or ndk-build script.  
  externalNativeBuild {  
    cmake {...}  
    // or ndkBuild {...}  
  }  
}
```

要详细了解配置产品风味和构建变体，请参阅配置构建变体 (<https://developer.android.google.cn/studio/build/build-variants.html>)。如需了解您可以使用 `arguments` 属性为 CMake 配置的变量列表，请参阅使用 CMake 变量 (<https://developer.android.google.cn/ndk/guides/cmake.html#variables>)。

指定 ABI

默认情况下，Gradle 会针对 NDK 支持的 ABI (<https://developer.android.google.cn/ndk/guides/abis.html#sa>) 将您的原生库构建到单独的 `.so` 文件中，并将其全部打包到您的 APK 中。如果您希望 Gradle 仅构建和打包原生库的特定 ABI 配置，您可以在模块级 `build.gradle` 文件中使用 `ndk.abiFilters` 标志指定这些配置，如下所示：

```
android {  
  ...  
  defaultConfig {  
    ...  
    externalNativeBuild {  
      cmake {...}  
      // or ndkBuild {...}  
    }  
  
    ndk {  
      // Specifies the ABI configurations of your native  
      // libraries Gradle should build and package with your APK.  
      abiFilters 'x86', 'x86_64', 'armeabi', 'armeabi-v7a',  
                  'arm64-v8a'  
    }  
  }  
}
```

This site uses cookies to store your preferences for site-specific language and display options.

OK


```
externalNativeBuild {...}  
}
```

在大多数情况下，您只需要在 `ndk {}` 块中指定 `abiFilters`（如上所示），因为它会指示 Gradle 构建和打包原生库的这些版本。不过，如果您希望控制 Gradle 应当构建的配置，并独立于您希望其打包到 APK 中的配置，请在 `defaultConfig.externalNativeBuild.cmake {}` 块（或 `defaultConfig.externalNativeBuild.ndkBuild {}` 块中）配置另一个 `abiFilters` 标志。Gradle 会构建这些 ABI 配置，不过仅会打包您在 `defaultConfig.ndk{}` 块中指定的配置。

为了进一步降低 APK 的大小，请考虑配置 ABI APK 拆分 (<https://developer.android.google.cn/studio/build/configure-apk-splits.html#configure-abi-split>)，而不是创建一个包含原生库所有版本的大型 APK，Gradle 会为您想要支持的每个 ABI 创建单独的 APK，并且仅打包每个 ABI 需要的文件。如果您配置 ABI 拆分，但没有像上面的代码示例一样指定 `abiFilters` 标志，Gradle 会构建原生库的所有受支持 ABI 版本，不过仅会打包您在 ABI 拆分配置中指定的版本。为了避免构建您不想要的原生库版本，请为 `abiFilters` 标志和 ABI 拆分配置提供相同的 ABI 列表。

