# 文件 DAC 配置

将文件系统对象和服务添加到编译环境中时，您通常需要分别为此类项目指定唯一 ID，称为 Android ID (AID)。目前，文件和服务等很多资源不需要一定使用 Android 定义的核心 AID；在很多情况下，您可以改为使用 OEM 定义的 AID。

在早期版本的 Android 中，对 AID 机制进行扩展时，是使用设备专属的 `android_filesystem_config.h` 文件来指定文件系统权能和/或自定义 OEM AID。但是，此机制不够直观，因为它不支持 OEM AID 使用好记的名称，而是要求您为用户和群组字段指定原始数字，这样一来，便无法将好记的名称与数字 AID 关联起来。

Android 8.0 及更高版本中采取了一种新的 AID 机制来扩展文件系统权能。这种新方法支持：

- 配置文件可以有多个源位置（支持可扩展的编译环境配置）。
- 在编译时对 OEM AID 值进行健全性检查。
- 生成可视需要在源文件中使用的自定义 OEM AID 标头。
- 将好记的名称与实际的 OEM AID 值相关联。支持为用户和群组指定非数字的字符串参数，即“foo”而不是“2901”。

其他改进包括从 `system/core/include/private/android_filesystem_config.h` 中移除了 `android_ids[]` 数组。 该数组现在作为完全自行生成的数组存在于 Bionic 中，并具有通过 `getpwnam()` 和 `getgrnam()` 提取数据的访问器。（此改进还有另一个作用，即使核心 AID 发生更改，生成的二进制文件也可保持稳定。）如需了解这种机制以及查看包含更多详情的 README 文件，请参阅 `build/make/tools/fs_config`。

**注意**：虽然您仍可以使用旧版 Android 中的文件系统替换方法 (#older)，但不能同时再使用新的 AID 机制。建议您尽可能使用新的机制。

## 添加 Android ID（AID）

Android 8.0 从 Android 开放源代码项目 (AOSP) 中移除了 `android_ids[]` 数组。所有适合 AID 的名称都改为在生成 Bionic `android_ids[]` 数组时从 `system/core/include/private/android_filesystem_config.h` 标头文件生成。这种机制会提取与 `AID_*` 匹配的所有 `define`，且 * 会变为小写名称。

例如，在 `private/android_filesystem_config.h` 中：

```
#define AID_SYSTEM 1000
```

会变为：

- 好记的名称：system
- uid：1000
- gid：1000

要添加新的 AOSP 核心 AID，只需将 **#define** 添加到 **android_filesystem_config.h** 标头文件中即可。AID 在编译环境中生成，并会提供给使用用户和群组参数的接口。这种机制会确认新的 AID 不在应用或 OEM 范围内；此外，它还会接受对此类范围的更改，并自动根据相应更改或新的 OEM 保留范围重新进行配置。

## 配置 AID

要启用新的 AID 机制，请在 **BoardConfig.mk** 文件中设置 **TARGET_FS_CONFIG_GEN**。此变量含有配置文件列表，使您可以根据需要附加文件。

**注意**：请勿通过旧版 Android 中早期的 **TARGET_ANDROID_FILESYSTEM_CONFIG_H** 方法使用 **TARGET_FS_CONFIG_GEN**！否则，您会收到错误提示。

按照惯例，配置文件使用名称 **config.fs**，但在实践中，您可以使用任何名称。**config.fs** 文件采用 Python ConfigParser ini 格式 (https://docs.python.org/2/library/configparser.html)，并包含大写部分（用于配置文件系统权能）和 AID 部分（用于配置 OEM 专属 AID）。

### 配置大写部分

大写部分支持在编译环境中对文件系统对象设置文件系统权能 (http://man7.org/linux/man-pages/man7/capabilities.7.html)（文件系统本身也必须支持此功能）。

由于在 Android 中以 Root 身份运行稳定的服务会导致兼容性测试套件 (CTS) (https://source.android.com/compatibility/cts/index.html?hl=zh-cn) 失败，因此在之前有关在运行进程或服务时保留权能的要求中，您需要先设置权能，然后使用 **setuid/setgid** 设置适当的 AID 以运行程或服务。借助大写部分，您可以跳过这些要求，让内核为您代劳。当控制权交给 **main()** 时，您的进程就已拥有所需的权能，因此您的服务可以使用非 Root 用户和群组（这是启动特权服务的首选方式）。

大写部分使用以下语法：

| 部分 | 值 | 定义 |
| --- | --- | --- |
| `[path]` | | 要配置的文件系统路径。以 / 结尾的路径被视为目录，否则，将被视为文件。<br><br>在不同文件中使用同一 `[path]` 指定多个部分的做法是错误的。在 Python 3.2 之前的版本中，同一文件中包含的某些部分可替换它之前的部分；而在 Python 3.2 中，系统设置了严格模式。 |
| `mode` | 八进制文件模式 | 至少为 3 位数的有效八进制文件模式。如果指定 3，则会附上前缀 0，否则系统会按原样使用模式。 |
| `user` | AID_<user> | 有效 AID 的 C 样式的 `define` 或好记的名称（例如 `AID_RADIO` 和 `radio` 皆可）。要指定自定义 AID，请参阅配置 AID 部分 (#configuring-the-aid-section)。 |
| `group` | AID_<group> | 和用户一样。 |
| `caps` | cap* | `system/core/include/private/android_filesystem_capability.h` 中所声明的名称，不含前导 `CAP_`。允许大小写混用。大写也可以是原始值：<br>• binary (0b0101)<br>• octal (0455)<br>• int (42)<br>• hex (0xFF)<br>可以使用空格隔开多个大写字母。 |

有关使用示例，请参阅使用文件系统权能 (#using-file-system-capabilities)。

## 配置 AID 部分

AID 部分包含 OEM 专属 AID，并使用以下语法：

| 部分 | 值 | 定义 |
| --- | --- | --- |
| `[AID_<name>]` | | `<name>` 可以包含大写字母、数字和下划线字符。小写版本作为好记的名称使用。生成的用于代码收录的标头文件使用确切的 `AID_<name>`。<br><br>使用同一 `AID_<name>` 指定多个部分（不区分大小写，限制条件与 `[path]` 相同）是错误的做法。 |
| `value` | <number> | 有效的 C 样式的数字字符串（十六进制、八进制、二进制和十进制）。<br><br>使用同一值选项指定多个部分或指定超出收录的 OEM 范围（在 `system/core/include/private/android_filesystem_config.h` 中指定）的值，属于错误的做法：<br>• AID_OEM_RESERVED_START(2900) - AID_OEM_RESERVED_END(2999) |

- AID_OEM_RESERVED_2_START(5000) - AID_OEM_RESERVED_2_END(5999)

有关使用示例，请参阅定义 OEM 专属 AID (#defining-an-oem-specific-aid) 和使用 OEM 专属 AID (#using-an-oem-specific-aid)。

# 用法示例

以下示例详细介绍了如何定义和使用 OEM 专属 AID，以及如何启用文件系统权能。

## 定义 OEM 专属 AID

要定义 OEM 专属 AID，请创建一个 **config.fs** 文件并设置 AID 值。例如，在 **device/x/y/config.fs** 中设置以下内容：

```
[AID_FOO]
value: 2900
```

创建好文件后，设置 **TARGET_FS_CONFIG_GEN** 变量并在 **BoardConfig.mk** 指向它。例如，在 **device/x/y/BoardConfig.mk** 中设置以下内容：

```
TARGET_FS_CONFIG_GEN += device/x/y/config.fs
```

总的来说，现在系统已经可以在新编译环境中使用您的自定义 AID 了。

## 使用 OEM 专属 AID

要通过 C 或 C++ 代码访问 AID 的 **#define** 值，请使用自动生成的标头文件，方法是：将其添加到模块的 **Android.mk** 中并纳入空的仿库。例如，在 **Android.mk** 中添加以下内容：

```
 LOCAL_STATIC_LIBRARIES := liboemaids
```

在您的 C 代码中，**#include "generated_oem_aid.h"** 并开始使用所声明的标识符。例如，在 **my_file.c** 中添加以下内容：

```
#include "generated_oem_aid.h"

…

If (ipc->uid == AID_FOO) {
```

```
  // Do something
...
```

在 Android 8.0 中，您必须配合 **oem_####** 使用 **getpwnam** 和类似函数，在通过 **getpwnam**（如 init 脚本）处理查询时也是如此。例如，在 **some/init.rc** 中使用以下内容：

```
service foo /vendor/bin/foo_service
    user: oem_2900
    group: oem_2900
```

### 使用文件系统权能

要启用文件系统权能，请在 **config.fs** 文件中创建一个大写部分。例如，在 **device/x/y/config.fs** 中添加以下部分：

```
[system/bin/foo_service]
mode: 0555
user: AID_FOO
group: AID_SYSTEM
caps: SYS_ADMIN | SYS_NICE
```

**注意**：此处也可以使用好记的名称 **foo** 和 **system**。

创建好文件后，设置 **TARGET_FS_CONFIG_GEN** 并在 **BoardConfig.mk** 中指向它。例如，在 **device/x/y/BoardConfig.mk** 中设置以下内容：

```
TARGET_FS_CONFIG_GEN += device/x/y/config.fs
```

当执行服务 **foo** 时，它会先使用权能 **CAP_SYS_ADMIN** 和 **CAP_SYS_NICE**，而不使用 **setuid** 和 **setgid** 调用。此外，**foo** 服务的 SELinux 策略也不再需要 **setuid** 和 **setgid**，因此，可以从 **foo** 的 SELinux 策略中移除这些权能。

## 配置替换（Android 6.x 到 7.x 版本）

Android 6.0 将 **fs_config** 和关联的结构定义 (**system/core/include/private/android_filesystem_config.h**) 转移到了 **system/core/libcutils/fs_config.c**。在此处，可使用安装在 **/system/etc/fs_config_dirs** 和 **/system/etc/fs_config_files** 中的二进制文件来更新或替换它们。针对目录和文件分别采用单独的匹配和解析规则（可能会使用其他全局表达式），这样一来，Android 就能够在两个不同的表中处理目录和文件。

`system/core/libcutils/fs_config.c` 中的结构定义不仅可让系统在运行时读取目录和文件，而且主机在编译时也可以使用相同的文件来构建文件系统映像，比方说 `${OUT}/system/etc/fs_config_dirs` 和 `${OUT}/system/etc/fs_config_files`。

虽然扩展文件系统时采用的替换方法已被 Android 8.0 中推出的模块化配置系统所取代，但如果需要，您仍可以使用原来的方法。以下部分将详细介绍如何生成和纳入替换文件以及如何配置文件系统。

## 生成替换文件

您可以使用 `build/tools/fs_config` 中的 `fs_config_generate` 工具生成相应的二进制文件 `/system/etc/fs_config_dirs` 和 `/system/etc/fs_config_files`。该工具使用 `libcutils` 库函数 (`fs_config_generate()`) 管理放入缓冲区内的 DAC 需求，并为头文件定义规则来规定 DAC 规则的用法。

要使用该工具，请在 `device/`*`vendor`*`/`*`device`*`/android_filesystem_config.h` 中创建头文件以用作替换文件。该文件必须使用 `system/core/include/private/android_filesystem_config.h` 中定义的 `structure fs_path_config` 格式，并对目录和文件符号进行以下结构初始化：

- 对于目录，请使用 `android_device_dirs[]`。
- 对于文件，请使用 `android_device_files[]`。

在不使用 `android_device_dirs[]` 和 `android_device_files[]` 时，您可以定义 `NO_ANDROID_FILESYSTEM_CONFIG_DEVICE_DIRS` 和 `NO_ANDROID_FILESYSTEM_CONFIG_DEVICE_FILES`（请参阅下面的示例 (#older-example)）。您还可以使用板级配置中的 `TARGET_ANDROID_FILESYSTEM_CONFIG_H` 指定强制基本名称为 `android_filesystem_config.h` 的替换文件。

## 包含替换文件

要包含文件，请确保 `PRODUCT_PACKAGES` 包含 `fs_config_dirs` 和/或 `fs_config_files`，以便它可以分别将其安装到 `/system/etc/fs_config_dirs` 和 `/system/etc/fs_config_files` 中。编译系统会在 `BoardConfig.mk` 所在的 `$(TARGET_DEVICE_DIR)` 中搜索自定义 `android_filesystem_config.h`。如果此文件位于其他位置，请设置板级配置变量 `TARGET_ANDROID_FILESYSTEM_CONFIG_H` 来指向该位置。

## 配置文件系统

要在 Android 6.0 及更高版本中配置文件系统，请执行以下操作：

1. 创建 **$(TARGET_DEVICE_DIR)/android_filesystem_config.h** 文件。
2. 将 **fs_config_dirs** 和/或 **fs_config_files** 添加到板级配置文件（例如 **$(TARGET_DEVICE_DIR)/device.mk**）中的 **PRODUCT_PACKAGES** 。

## 替换示例

此示例展示了用于替换 **system/bin/glgps** 守护进程以在 **device/*vendor*/*device*** 目录中添加唤醒锁定支持的补丁程序。请注意以下几点：

- 每个结构条目都包含模式、uid、gid、权能和名称。 已自动包含 **system/core/include/private/android_filesystem_config.h** 来提供清单 #defines（**AID_ROOT**、**AID_SHELL**、**CAP_BLOCK_SUSPEND**）。
- **android_device_files[]** 区段包含在未指定时禁止访问 **system/etc/fs_config_dirs** 的操作，其作用是在缺少目录替换内容时提供额外 DAC 保护。但此保护的强度较弱；如果有人拥有对 **/system** 的控制权，那么他通常可以执行任何操作。

```
diff --git a/android_filesystem_config.h b/android_filesystem_config.h
new file mode 100644
index 0000000..874195f
--- /dev/null
+++ b/android_filesystem_config.h
@@ -0,0 +1,36 @@
+/*
+ * Copyright (C) 2015 The Android Open Source Project
+ *
+ * Licensed under the Apache License, Version 2.0 (the "License");
+ * you may not use this file except in compliance with the License.
+ * You may obtain a copy of the License at
+ *
+ *      http://www.apache.org/licenses/LICENSE-2.0
+ *
+ * Unless required by applicable law or agreed to in writing, software
+ * distributed under the License is distributed on an "AS IS" BASIS,
+ * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
+ * implied. See the License for the specific language governing
+ * permissions and limitations under the License.
+ */
+
+/* This file is used to define the properties of the filesystem
+** images generated by build tools (eg: mkbootfs) and
+** by the device side of adb.
+*/
+
```

```
+#define NO_ANDROID_FILESYSTEM_CONFIG_DEVICE_DIRS
+/* static const struct fs_path_config android_device_dirs[] = { }; */
+
+/* Rules for files.
+** These rules are applied based on "first match", so they
+** should start with the most specific path and work their
+** way up to the root. Prefixes ending in * denotes wildcard
+** and will allow partial matches.
+*/
+static const struct fs_path_config android_device_files[] = {
+  { 00755, AID_ROOT, AID_SHELL, (1ULL << CAP_BLOCK_SUSPEND),
"system/bin/glgps" },
+#ifdef NO_ANDROID_FILESYSTEM_CONFIG_DEVICE_DIRS
+  { 00000, AID_ROOT, AID_ROOT, 0, "system/etc/fs_config_dirs" },
+#endif
+};

diff --git a/device.mk b/device.mk
index 0c71d21..235c1a7 100644
--- a/device.mk
+++ b/device.mk
@@ -18,7 +18,8 @@ PRODUCT_PACKAGES := \
     libwpa_client \
     hostapd \
     wpa_supplicant \
-    wpa_supplicant.conf
+    wpa_supplicant.conf \
+    fs_config_files

 ifeq ($(TARGET_PREBUILT_KERNEL),)
 ifeq ($(USE_SVELTE_KERNEL), true)
```

## 从早期版本迁移文件系统

当从 Android 5.x 及更低版本迁移文件系统时，请注意以下事项：

- Android 6.x 移除了部分头文件、结构和内嵌定义。
- Android 6.x 需要引用 **libcutils**，而不是直接从 **system/core/include/private/android_filesystem_config.h** 运行。依赖于 **system/code/include/private_filesystem_config.h** 的文件/目录结构或者 **fs_config** 的设备制造商私有可执行文件必须添加 **libcutils** 库依赖关系。
- Android 6.x 需要 **system/core/include/private/android_filesystem_config.h** 的设备制造商专有分支副本，该副本应包含有关现有目标的附加内容，以便移至 **device/*vendor*/*device*/android_filesystem_config.h**。

- 由于 Android 保留将 SELinux 强制访问控制 (MAC) 应用于目标系统中配置文件的权利，因此包含使用 `fs_config()` 的自定义目标可执行文件的实现必须确保具有访问权限。

---



*上次更新日期：一月 4, 2018*