Android Developers

Wrap Shell Script

When debugging and profiling apps with native code, it's often useful to be able to bundle command-line developer tools in your APK and run them automatically on the device when the app launches.

In this document

Beginning in Android 8.0, you can opt to create and package a user-defined shell script that the system automatically runs in a fresh process when you launch a debuggable app. This feature lets you write a script to invoke native developer tools to perform such operations as:

- Tracing system calls with strace (https://linux.die.net/man/1/strace).
- Ensuring memory integrity with malloc debug (https://android.googlesource.com/platform/bionic/+/master/libc/malloc_debug/README.md), Address Sanitizer (https://github.com/google/sanitizers/wiki/AddressSanitizerOnAndroidO) (ASAN), and Valgrind (http://valgrind.org/).
- Enabling app-specific native memory tracking via malloc debug (https://android.googlesource.com/platform/bionic/+/master/libc/malloc_debug/README.md).
- Running Simpleperf (https://developer.android.google.cn/ndk/guides/simpleperf.html) against a specific app process, if you are profiling Java code.

Using the wrap shell script

To run native developer tools on the device using the wrap shell script, follow these steps:

1. Compile a custom debuggable APK that packages the following:
    - The native developer tools you want to use, and

- A shell script with instructions to run the native developer tools (named `wrap.sh`). See Creating the wrap shell script (#creating_the_wrap_shell_script) and Packaging wrap.sh (#packaging_wrapsh) for more details.

2. Install the debuggable APK on a device. Depending on the Android platform version on the device, you may need to modify access control over processes for the wrap shell script to run.
   - **Android 8.1 (API level 27) or higher**: No such modification is needed.
   - **Android 8.0 (API level 26)**: The device must have SELinux enforcement disabled (https://source.android.google.cn/security/selinux/validate#switching_to_permissive). You can disable SELinux enforcement via `adb` (https://developer.android.google.cn/studio/command-line/adb.html) on userdebug or eng builds. Supported devices with these permission levels are:
     - An Android emulator with a userdebug system image, or
     - A Google device flashed with an AOSP eng build or root image.
3. Launch the app normally on your device, either via the app launcher or from the command line.
4. It is strongly recommended to re-enable SELinux enforcement on your device when done.

Creating the wrap shell script

You can create a shell script named `wrap.sh` to customize your runtime environment. In your script, you can set environment variables and modify runtime arguments. The script should follow MirBSD Korn shell (https://www.mirbsd.org/mksh.htm) (mksh) syntax.

When you launch a debuggable APK that contains `wrap.sh`, the system executes the script and grants it control over the app process. The system passes the command to start the requested app as an argument to the `wrap.sh` invocation. Note that your script is responsible for starting the app.

The following snippet shows how to write a simple `wrap.sh` file that just starts the app:

```
#!/system/bin/sh
"$@"
```

To start the malloc debug (https://android.googlesource.com/platform/bionic/+/master/libc/malloc_debug/README.md) tool using `wrap.sh`, you would include the following line:

```
#!/system/bin/sh
LIBC_DEBUG_MALLOC_OPTIONS=backtrace logwrapper $@
```

Packaging wrap.sh

To take advantage of `wrap.sh`, you must build a debuggable APK. Make sure that the `android:debuggable="true"`setting is configured in the `<application>` (https://developer.android.google.cn/guide/topics/manifest/application-element.html) element in your Android manifest.

You must package the `wrap.sh` script with the native libraries of the app. If your app does not contain native libraries, add the lib directory manually to your project directory. For all architectures that your app supports, you must provide a copy of the wrap shell script under their respective native-library architecture directories.

The following project directory example shows the file layout to support both the x86 and ARMv8 architectures:

```
# App Directory
|- AndroidManifest.xml
|- …
|- lib
   |- x86
      |- ...
      |- wrap.sh
   |- arm64-v8a
      |- ...
      |- wrap.sh
```

Make sure that your wrap.sh script is executable (that is, its executable bit is set). For example:

```
$ ls -l lib/x86/wrap.sh
-rwxr-x--x 1 user user 0 Jan 01  1980 lib/x86/wrap.sh
```

在微信上关注 Google Developers

Follow @AndroidDev on Twitter

Follow Android Developers on Google+

Check out Android Developers on YouTube