Browsing and debugging AOSP code using Android Studio

Chris Simmonds, 2net

Overview

- Importing AOSP into Android Studio
- Debugging

Creating IDEGen

- AOSP has a tool named `IDEGen` that will generate project files suitable for IntelliJ IDEA (including Android Studio)
  Details: `$AOSP/development/tools/idegen/README`

Set up the AOSP environment in the normal way:

```
$ cd aosp
$ . build/envsetup.sh
$ lunch
```

Build `IDEGen`:

```
$ m idegen
```

Creates `out/host/linux-x86/framework/idegen.jar`

Creating project files

Then, use `idegen.jar` to generate project files (ignore the "Permission denied" messages):

```
$ development/tools/idegen/idegen.sh
find: 'out/target/product/marvin/recovery/root/d': Permission denied
find: 'out/target/product/marvin/root/d': Permission denied
Read excludes: 23ms
Traversed tree: 148105ms
```

Creates an IntelliJ Project File: `android.ipr`
and an IntelliJ Module List: `android.iml`

Note: you have to rerun this step every time you add directories inside the `$AOSP` directory

Configuring Android Studio

Indexing the whole of AOSP takes a lot of resources

Increase heap to 5 GB

In Help->Edit Custom VM Options, add:

```
-Xmx5g
```

For reference, the VM Options are stored in
`$HOME/.config/Google/AndroidStudio2021.2/studio64.vmoptions`
for the current version of Studio, 2021.2 "Chipmunk"

Importing AOSP code

In Android Studio, open `$AOSP/android.ipr`

Wait for it to complete "Updating indexes" (background task, bottom right); takes a long time

Now you can use Studio to browse the code, search for classes, etc.

Some people consider this a better alternative to using vim

Debugging - JDWP

- Android uses JDWP (Java Debug Wire Protocol) for debugging
- JDWP is enabled by default for all components in userdebug and eng builds
- In user builds, enable JDWP per appliation by adding `android:debuggable="true"` to the `AndroidManifest.xml`

```
$ ps -ATp  3694
USER      PID   TID  PPID     VSZ    RSS WCHAN    ADDR S CMD
u0_a65   3694  3694   355 1409680 203024 0             0 S droid.deskclock
u0_a65   3694  3697   355 1409680 203024 0             0 S Signal Catcher
u0_a65   3694  3698   355 1409680 203024 0             0 S perfetto_hprof_
u0_a65   3694  3699   355 1409680 203024 0             0 S ADB-JDWP Connec  <----
u0_a65   3694  3700   355 1409680 203024 0             0 S Jit thread pool
[...]
```

system_server

- system_server is a Java program (not an app) that implements the core functions of the framework
- It also has a JDWP thread in userdebug and eng builds

```
$ ps -ATp  765
USER      PID   TID  PPID     VSZ    RSS WCHAN      ADDR S CMD
system    765   765   355 2427732 327936 0             0 S system_server
system    765   782   355 2427732 327936 0             0 S Signal Catcher
system    765   783   355 2427732 327936 0             0 S perfetto_hprof_
system    765   784   355 2427732 327936 0             0 S ADB-JDWP Connec  <----
[...]
```

adb jdwp

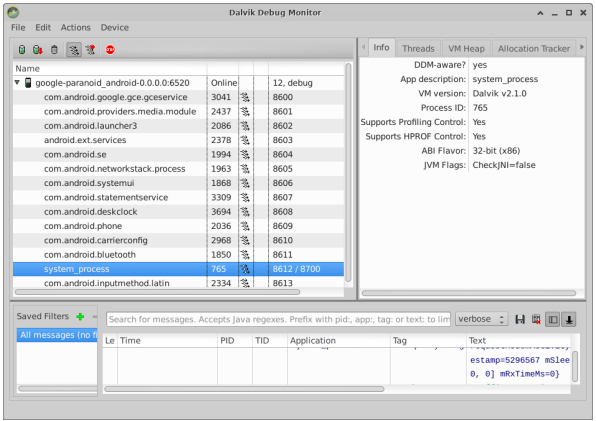
- You can get a list of PIDs which have JDWP threads:

```
$ adb jdwp
765             <---- system_server
1850
[...]
3694            <---- deskclock
```

Unfortunately, only lists the PIDs. You have to use `ps -A` to find the command

DDMS

- DDMS is the Dalvik Debug Monitor Service, an old debug tool
- Useful for debugging: lists processes with a JDWP thread, and forwards the port as localhost:8700

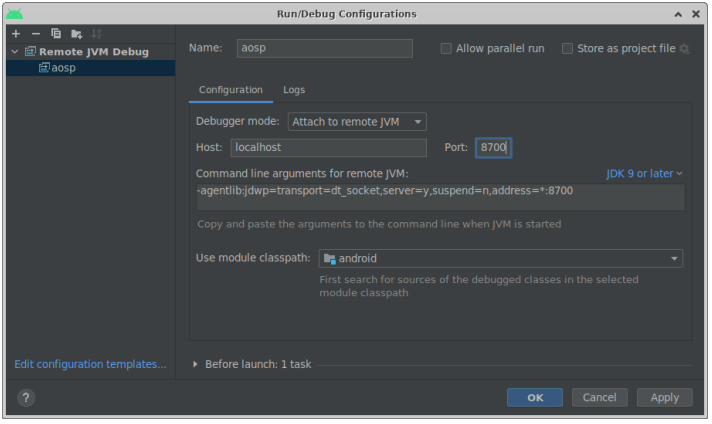


Debuggers for Android

- Any debugger that is compatible with JDWP
- For example
  - Android Studio (InteliJ IDEA)
  - JDB
- Just need to attach to the JDWP port (8700 if using DDMS)

Debugging using Android Studio

In Android Studio, create a debug configuration (Run -> Edit Configurations...)
Add a new **Remote JVM Debug**, for example called 'aosp'
Set Port to **8700**



Debugging using Android Studio

Boot target device

Run DDMS and select process: for system_server, select **system_process**

In Android Studio, Run -> Debug 'aosp'

Check that you see this in the debug window

```
Connected to the target VM, address: 'localhost:8700', transport: 'socket'
```

Debugging using JDB

- JDB is a command-line debugger, part of the JDK
- Pros: easy to set up, nothing new to install, no need for `IDEGen`, fast
- Cons: a lot of typing

More details to follow, probably at the next meetup