**ADB简述**

目录

[1. 概述 2](#_Toc316551213)

[2. ADB架构 2](#_Toc316551214)

[2.1. ADB各模块定义 3](#_Toc316551215)

[2.2. adbd （ADB daemon） 4](#_Toc316551216)

[2.2.1. 启动流程 4](#_Toc316551217)

[2.3. adb server 7](#_Toc316551218)

[2.3.1. 启动流程 7](#_Toc316551219)

[2.4. adb client 8](#_Toc316551220)

[2.5. DDMS 11](#_Toc316551221)

[2.6. Jdwp 13](#_Toc316551222)

[3. ADB 通信 13](#_Toc316551223)

[3.1. adb client<-->adb server 14](#_Toc316551224)

[3.2. adb server<-->adbd 14](#_Toc316551225)

[3.2.1. Transport 14](#_Toc316551226)

[3.2.2. 协议 17](#_Toc316551227)

[4. ADB源代码分析相关网络资源 18](#_Toc316551228)

[5. ADB命令 18](#_Toc316551229)

[5.1. 使用adb命令 18](#_Toc316551230)

[5.2. 执行Shell命令 24](#_Toc316551231)

[5.3. 使用logcat查看日志 26](#_Toc316551232)

[5.4. 停止adb服务 31](#_Toc316551233)

# 概述

Android Debug Bridge (adb) 是一个android开发人员必会的多功能的调试工具，如它的名字一样，它在开发者和目标机器之间，架起了一座bridge。

adb的用法很简单，只要看完SDK文档关于ADB的部分，(android-sdk-linux\_86/docs/guide/developing/tools/adb.html)，再稍加练习，基本上就满上就能很熟练的使用了。

adb源码位置是：system/core/adb.

# ADB架构

adb由两个物理文件组成：

**adb/adb.exe**

运行于PC端，包括Linux、Windows、Mac OS等系统之中，通常是x86架构上（下文中，ADB指整个模块，而adb单独指这一部分）；

**adbd**

运行于Android设备的底层Linux之中，ARMv5架构上。

构建不同文件，通过传入Android.mk的$(BUILD\_SIMULATOR)变量是否为真。源码中由**ADB\_HOST**宏用来区分本地主机(adb)和目标机(adbd)。

区分不同OS，通过传入Android.mk的$(HOST\_OS)。它的有效取值包括linux、darwin、freebsd和windows。不同平台的主要差异是USB的控制方法和文件路径。

## ADB各模块定义

adb server： 计算机上的一个服务进程，进程名为adb

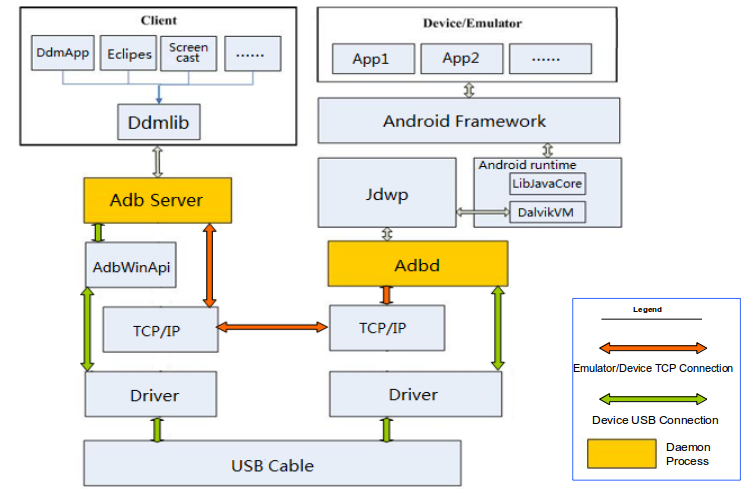
adbd (adb daemon) ： Android 手机上的一个服务进程，进程名为adbd

adb client：你可以认为是计算机上的一个终端窗口，进程名也为adb

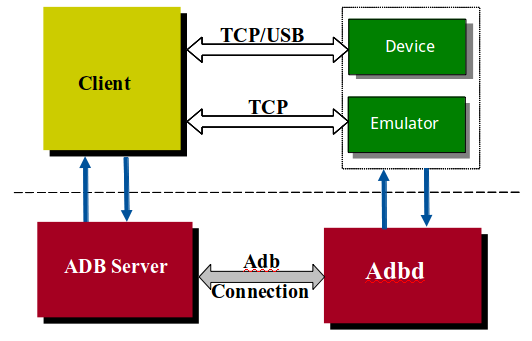
DDMS：Dalvik Debug Monitor Service

Jdwp： Java Debug Wire Protocol

**adb整体架构和数据传输图如下：**



DDMS和Jdwp不做多的了解。只看adb模块，如下图：



system/core/adb/OVERVIEW.txt文件中对它们的关系进行了描述。

system/core/adb/protocol.txt和OVERVIEW.txt描述了各模块之间通信协作的协议格式。

## adbd （ADB daemon）

运行于device/emulator的守护进程（Local service）。其作用如下：

1，用来连接device/emulator和adb server，device通过USB连接，emulator通过TCP连接。

2，为device/emulator提供服务--> adb service（adb服务，注意与adb服务端的区别）的概念，指adbd提供的功能。

在emulator/device端，adbd也创建local socket和remote socket，前者与通过jdwp与dalvik VM进层通信，后者通过TCP/USB与adb server通信。

### 启动流程

adbd在init.rc中启动。

1,adbd main:

adb.c main()

int main(int argc, char \*\*argv)

{

adb\_trace\_init();

#if ADB\_HOST

adb\_sysdeps\_init();

return adb\_commandline(argc - 1, argv + 1); / /运行PC端，用于命令发送

#else

if((argc > 1) && (!strcmp(argv[1],"recovery"))) {

**adb\_device\_banner = "recovery"; // recover模式**

recovery\_mode = 1;

}

start\_device\_log();

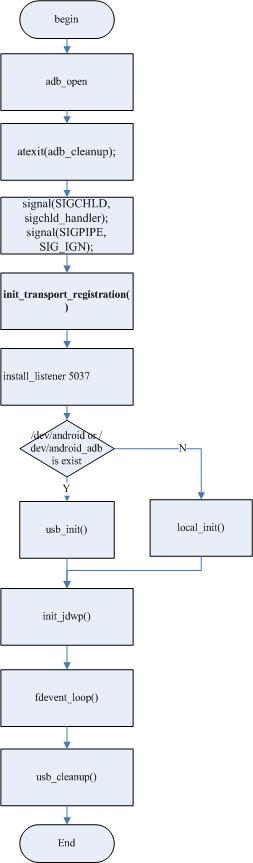
return adb\_main(0, DEFAULT\_ADB\_PORT); // 运行于emulator/device上，用于命令接收及反馈。如果执行命令“service adbd”它将会被执行。

#endif

}

2, adb\_main()

Adbd创建两个socket，一个用来控制连接，一个用来连接adb server。在该函数中初始化jdwp用来使adbd和jvm之间的交互，就是说adbd能够发送消息给app和接收app来自的消息（包括event、data等）。



在init\_transport\_registration函数中调用adb\_socketpair。

## adb server

是运行在开发用电脑上的后台进程，用于管理客户端与运行在模拟器或真机的守护进程通信。在HOST端，adb会fork(通过命令：“adb fork-server server”，见代码adb.c中的launch\_server函数)出一个守护进程（Host service），即server（给用户的调用接口是adb start-server），而父进程继续处理client请求，所有的client通过TCP端口号5037进行与server通信，而server创建local socket与remote socket，前者用于和client通信，后者用与远端（adbd）进行通信，emulator通过TCP，real device则通过usb。

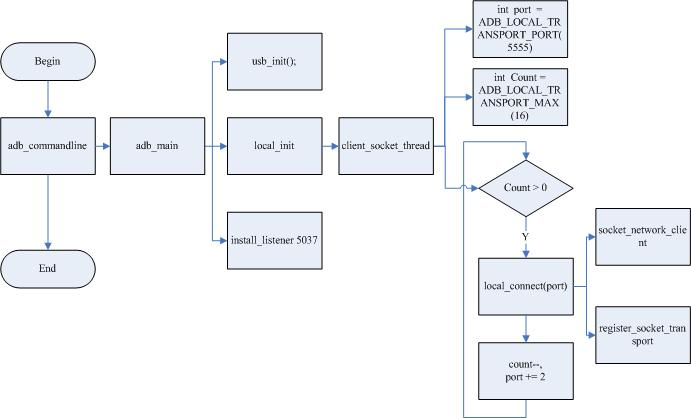
这个守护进程长期运行于后台，没有控制台界面，称之为adb server（adb服务端），其主要工作有两部分：

a，管理PC中的Android模拟器，以及通过USB线连接到PC的Android设备，负责维持运行于其中的adbd进程与自身的数据通道；

b，实现PC与设备/模拟器之间的数据拷贝。

### 启动流程

adb server由adb client启动。



## adb client

运行在开发用的电脑上，可以在命令行中运行adb命令来调用该客户端，像ADB插件和DDMS这样的Android工具也可以调用adb客户端。提供给用户的命令行工具，对用户暴露了install、push、shell等接口，与用户交互，称之为adb client（adb客户端）。其主要工作是解析这些命令的参数，做必要预处理，然后转移为指令或数据，发送给adb服务端。adb服务端再将指令数据转发到模拟器或设备中，由adbd处理，产生结果，再通过adb服务端接收回来。

说白了就是：你的指令通过adb client 交给adb server，server和daemon之间进行通讯。因此，这里建立的连接实际上是server和daemon之间的连接。（当启动adb客户端时，客户端首先检测adb服务端进程是否运行，如果没有运行，则启动服务端。当服务端启动时，它会绑定到本地的TCP5037端口，并且监听从adb客户端发来的命令——所有的adb客户端都使用5037端口与adb服务端通信。）

**三者通信整个流程是这样的：**

a、client调用某个adb命令

b、adb进程fork出一个子进程作为server

c、server查找当前连接的emulator/device

d、server接收到来自client请求

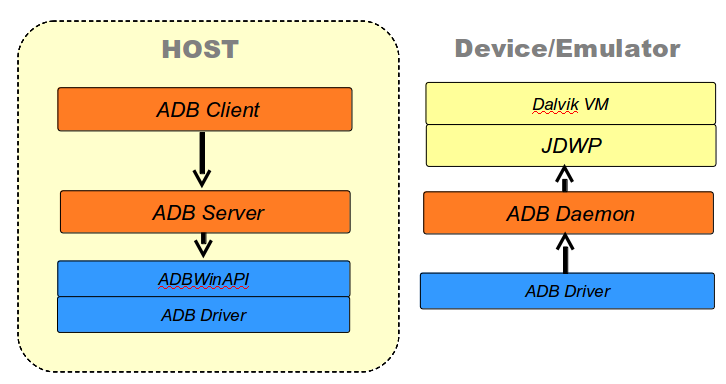
e、server处理请求，将本地处理不了的请求发给emulator/device

f、位于emulator/device的adbd拿到请求后交给对应的java虚拟机进程。

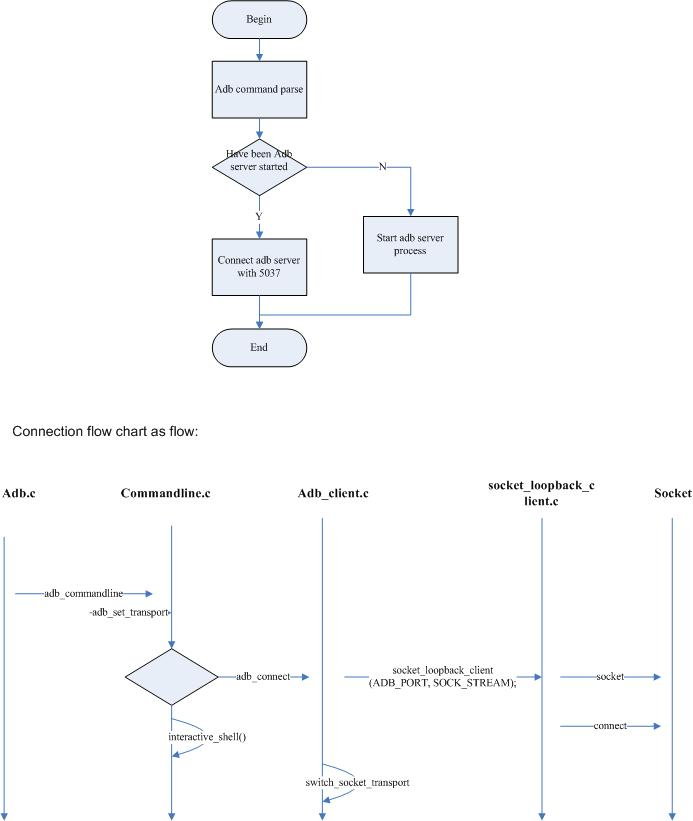
g、adbd将结果发回给server

h、server讲结果发回给client

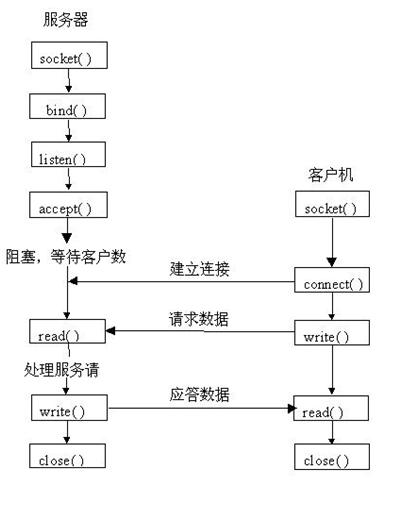
如下图：



**adb client连接adb server：**



Adb server 和 client 连接流程

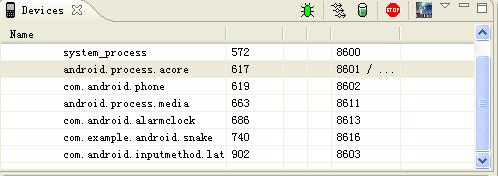


## DDMS

DDMS 的全称是Dalvik Debug Monitor Service。DDMS为IDE和Emultor(or GPhone)架起来了一座桥梁。Developer可以通过DDMS看到目标机器上运行的进程/线程状态：可以让Eclipse程序连接到开发机上运行；可以看进程的heap信息、logcat信息、进程分配内存情况；为测试设备截屏，广播状态信息、模拟电话呼叫、接收SMS、虚拟地理坐标等等。

**DDMS 的工作原理：**

DDMS将搭建起IDE与测试终端(Emulator 或者connected device)的链接，它们应用各自独立的端口监听调试器的信息，DDMS可以实时监测到测试终端的连接情况。当有新的测试终端连接后，DDMS将捕捉到终端的ID,并通过adb建立调试器，从而实现发送指令到测试终端的目的。



DDMS监听第一个终端App进程的端口为8600，APP进程将分配8601，如果有更多终端或者更多APP进程将按照这个顺序依次类推。DDMS通过8700端口（”base port”）接收所有终端的指令。

**DDMS选项卡：**

（1）Device选项卡

Device中罗列了Emulator中所有的进程，选项卡右上角那一排按钮分别为：调试进程、更新进程、更新进程堆栈信息、停止某个进程，最后一个图片按钮是抓取Emulator目前的屏幕。当你选中某个进程，并按下调试进程按钮时，如果eclipse中有这个进程的代码，那就可以进行源代码级别的调试。有点像gdb attach。图片抓取按钮可以把当前android的显示桌面抓到你的机器上，也是非常有用。

（2）Threads选项卡

显示线程统计信息

（3）Heap选项卡

显示栈信息

（4）File Explorer选项卡

显示GPhone Emulator的文件系统信息。

（5）Emulator Control选项卡

通过它可以向手机发送短信、打电话、更新手机位置信息。

总结：

Eclipse ADT目前提供的的ddms功能只是真正ddms的一小部分，你可以通过ddms.bat命令来使用所有功能。其中有一个查看进程内存分配的功能比较有用。

## Jdwp

jdwp(java debug wire protocol)是dalvik VM的一个线程，可以建立在adb（usb连接）或者tcp/ip（socket）基础上，与DDMS或debugger进行通信。

任何调试状态的vm都启动一个jdwp线程，该线程处于空闲状态直到DDMS或者调试器连接它。该线程只负责处理调试器来的请求，而vm发起的通信（例如，当vm在断点停下来的时候告知调试器）都由相应的线程发出。JDWP协议是无状态的，因此vm在调试器请求到来的时候就处理，在发生事件的时候就发送给vm。

代码位置

dalvik/vm/jdwp

frameworks/base/core/jni

java虚拟机初始化后，或者每次“zygote fork”出一个新进程时，会启动jdwp线程。

# ADB 通信

adb的通信涉及到host端的adb client和adb server之间通信（协议描述见：OVERVIEW.txt）；adb server和adbd之间的通信两部分（协议描述见：protocol.TXT）。

## adb client<-->adb server

adb client的每个命令都会包含两个部分，前一部分固定4个字节，以十六进制方式指定命令部分的长度。后一部分是真正的内容。发送命令的接口为writex，并最终调用\_fh\_socket\_write，通过send发送出去。因此这两部分至少需要发送两个tcp包。

例如想要获取adb server的版本号，client首先连接本机的TCP 5037端口，然后发送“000C”和“host:version”。

adb server对adb client回复，分为如下情况：

1、 成功，回复四字节串“OKAY”，后面跟的内容根据不同的命令而不同。

2、 失败，回复四字节串“FAIL”，然后跟四字节的十六进制长度，以及失败原因。

3、 对于host:version，回复4个字节的十六进制字串，代表server的内部版本号。

adb shell command 描述见文件：system/core/adb/SERVICES.TXT

## adb server<-->adbd

### Transport

adb server和设备或者模拟器之间的通信方式，包含如下两种情况：

**1、 USB transports。**通过USB方式和物理设备通信。

**2、 Local transports。**通过本机的TCP连接方式和模拟器通信。

Adb server通过扫描所有5555到5585范围内的奇数端口来定位所有的模拟器或设备。一旦adb server找到了adbd守护程序，它将建立一个到该端口的连接。请注意任何模拟器或设备实例会取得两个连续的端口——一个偶数端口用来相应控制台的连接，和一个奇数端口用来响应adb连接。比如说：

模拟器1，控制台：端口5554

模拟器1，Adb端口5555

模拟器2，控制台端口 5556

模拟器2，Adb端口5557

…

即如果模拟器与adb在5555端口连接，则其与控制台的连接就是5554端口。当服务端与所有的模拟器或手机设备建立连接之后，就可以使用adb命令来控制或者访问了。因为服务端管理着连接并且可以接收到从多个adb客户端的命令，所以可以从任何一个客户端或脚本来控制任何模拟器或手机设备。

**物理设备与adb server通信：**

传输类型：

transport-usb

used for switching transport to the only USB transport

transport-local

used for switching transport to the only local transport

transport-any

used for switching transport to the only transport

Host和physical device采用TCP/IP来进行通信，adb daemon需要使用5555到5585之间的奇数端口。首先看一下下面这段源代码，出自system/core/adb/adb.c adb\_main()函数

|  |
| --- |
| /\* for the device, start the usb transport if the  \*\* android usb device exists and the "service.adb.tcp.port" and  \*\* "persist.adb.tcp.port" properties are not set.  \*\* Otherwise start the network transport.  \*/  property\_get("service.adb.tcp.port", value, "");  **if** (!value[0])  property\_get("persist.adb.tcp.port", value, "");  **if** (sscanf(value, "%d", &port) == 1 && port > 0) {  // listen on TCP port specified by service.adb.tcp.port property  local\_init(port);  } **else** **if** (access("/dev/android\_adb", F\_OK) == 0) {  // listen on USB  usb\_init();  } **else** {  // listen on default port  local\_init(DEFAULT\_ADB\_LOCAL\_TRANSPORT\_PORT);  } |

分析上述代码可以发现，在adbd启动时首先检查是否设置了service.adb.tcp.port，如果设置了，就是使用TCP作为连接方式；如果没 设置，就去检查是否有adb的USB设备（dev/android\_adb)，如果有就用USB作为连接方式；如果没有USB设备，则还是用TCP作为连 接方式。

因此只需要在启动adbd之前设置service.adb.tcp.port，就可以让adbd选则TCP模式，也就可以通过网络来连接adb了。这需要修改init.rc文件。如果不想修改，也可以在系统启动之后，在控制台上执行通过命令修改，有如下要求：

1，adb工具版本至少是1.0.25或以上。

2，设置device adb调试端口号，需要root权限。

|  |
| --- |
| setprop service.adb.tcp.port 5555  stop adbd #停止adbd  start adbd #启动adbd |

3，device和PC必须在同一网段内(能互相ping通)。

建立连接：

adb connect host:5555

断开连接：

adb disconnect host:5555

### 协议

transport层用于处理消息，每个消息包含24个字节的头部，定义如下（代码见：adb.h）。

struct message {

unsigned command; // command identifier constant

unsigned arg0; // remote file descriptor

unsigned arg1; // local file descriptor

unsigned data\_length; // payload length

unsigned data\_crc32; // checksum of data payload

unsigned magic; // command ^ 0xffffffff

};

不过adb server 和 adbd传输和接收的都是apacket结构数据，定义如下：

struct apacket

{

apacket \*next;

unsigned len;

unsigned char \*ptr;

amessage msg;

unsigned char data[MAX\_PAYLOAD];

};

定义了6种消息类型：

（标识符"local-id" 和 "remote-id"是相对而言，是对于消息的收发角色而定）

CONNECT(version, maxdata, "system-identity-string")

OPEN(local-id, 0, "destination")

READY(local-id, remote-id, "")

WRITE(0, remote-id, "data")

CLOSE(local-id, remote-id, "")

SYNC(online, sequence, "")

--- amessage command 类型：（代码见：adb.h）--

#define A\_SYNC 0x434e5953

#define A\_CNXN 0x4e584e43

#define A\_OPEN 0x4e45504f

#define A\_OKAY 0x59414b4f

#define A\_CLSE 0x45534c43

#define A\_WRTE 0x45545257

描述：

A\_SYNC

A\_CNXN:send CNXN SIG to device/emulator after receive SYNC

A\_OPEN: open local service socket

A\_OKAY: ready

A\_CLSE: close socket

A\_WRTE: write to remote socket

ADB源代码分析相关网络资源

<http://blog.chinaunix.net/space.php?uid=20564848&do=blog&id=73628>

<http://blog.chinaunix.net/space.php?uid=20564848&do=blog&id=73684>

<http://blog.chinaunix.net/space.php?uid=20564848&do=blog&id=73633>

# ADB命令

## 使用adb命令

从开发用电脑的命令行或脚本文件中使用adb命令的用法是：

adb [-d|-e|-s <serialNumber>] <command>

当使用的时候，程序会调用adb客户端。因为adb客户端不需要关联到任何模拟器，所以如果有多个模拟器或手机正在运行，就需要使用-d参数指定要操作的是哪一个。

查询设备状态

了解adb服务端连接的模拟器或物理设备可以帮助更好的使用adb命令，这可以通过devices命令列举出来：

adb devices

执行结果是adb为每一个设备输出以下状态信息：

序列号(serialNumber) — 由adb创建的使用控制台端口号的用于唯一标识一个模拟器或物理设备的字符串，格式是 <设备类型>-<端口号>，例如： emulator-5554

状态(state) — 连接状态，其值是：

offline — 未连接或未响应

device —已经连接到服务商。注意这个状态并不表示Android系统已经完全启动起来，系统启动的过程中已经可以连接adb，但这个状态是正常的可操作状态。

每一个设备的输出形如：

[serialNumber] [state]

下面是 devices 命令和其执行结果：

$ adb devices

List of devices attached

emulator-5554 device

emulator-5556 device

emulator-5558 device

如果没有模拟器或物理设备在运行，该状态返回的是no device。

操作指定的模拟器或物理设备

如果有多个模拟器或手机正在运行，当使用adb命令的时候就需要指定目标设备，这可以通过使用-s选项参数实现，用法是：

adb -s <serialNumber> <command>

即可以在adb命令中使用序列号指定特定的目标，前文已经提到的devices命令可以实现查询设备的序列号信息。

例如：

adb -s emulator-5556 install helloWorld.apk

需要注意的是，如果使用了-s而没有指定设备的话，adb会报错。

安装应用程序

可以使用adb从开发用电脑中复制应用程序并且安装到模拟器或手机上，使用install命令即可，在这个命令中，必须指定待安装的.apk文件的路径：

adb install <path\_to\_apk>

如果使用了安装有ADT插件的Eclipse开发环境，就不需要直接使用adb或aapt命令来安装应用程序了，ADT插件可以自动完成这些操作。

关于创建可安装的应用的更多信息，请参见[Android Asset Packaging Tool (aapt)](http://developer.android.com/guide/developing/building/index.html#detailed-build).

转发端口

可以使用forward 命令转发端口 — 将特定端口上的请求转发到模拟器或物理设备的不同的端口上。下例是从6100端口转到7100端口：

adb forward tcp:6100 tcp:7100

也可以使用UNIX命名的socket标识：

adb forward tcp:6100 local:logd

在TCP网络编程中，Client的Socket(C)如果调用Connect()成功，就说明已经和Server端的Socket(S)连接上，可以通讯了。但是如果使用adb forward做端口映射，就不一样了。端口映射的实质是，让ADB-server作为一个switcher转发ADB-client的数据包，送给 adbd，adbd再发给设备端的对应端口。因此一旦建立了映射，就相当于ADB-server开始监听这个目标端口。而此时如果有C去尝试 Connect这个端口，是一定会成功的，因为与C连接的是ADB-server，而非真正的设备上的目标程序。这就出现了，即使Connect()成功，却完全无法知道究竟是否成功连接到S。

　　因此，判断真正连接成功的方法，只有轮询收发握手数据包。程序中约定好事先做个交互：C发送一个数据包，等待S回复;C如果收到了S的回复包，说明连通;如果接收超时，则认为没有连通。在没有连通的情况下，需要重新建立Socket，并Connect()，然后再尝试握手。

与模拟器或手机传输文件

可以使用adb的 pull 和 push 命令从模拟器或物理设备中复制文件，或者将文件复制到模拟器或物理设备中。与 install 命令不同，它仅能复制.apk文件到特定的位置， pull 和 push 命令可以复制任意文件夹和文件到模拟器或物理设备的任何位置。

从模拟器或手机中复制一个文件或文件夹（递归的）使用：

adb pull <remote> <local>

复制一个文件或文件夹（递归的）到模拟器或物理设备中使用：

adb push <local> <remote>

在这个命令中<local>和<remote>引用的是文件或文件夹的路径，在开发用电脑上的是local，在模拟器或物理设备上的是remote。

例如：

adb push foo.txt /sdcard/foo.txt

adb命令列表

下表列出了所有adb支持的命令及其说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 类别 | 命令 | 说明 | 备注 |
| 可选项 | -d | 命令仅对USB设备有效 | 如果有多个USB设备就会返回错误 |
| -e | 命令仅对运行中的模拟器有效 | 如果有多个运行中的模拟器就会返回错误 |
| -s <serialNumber> | 命令仅对adb关联的特定序列号的模拟器或手机有效(例如 "emulator-5556"). | 如果不指定设备就会返回错误 |
| 一般项 | devices | 输出所有关联的模拟器或手机设备列表 | 参见 [Querying for Emulator/Device Instances](http://developer.android.com/guide/developing/tools/adb.html#devicestatus)以获得更多信息。 |
| help | 输出adb支持的命令 |  |
| version | 输出adb的版本号 |  |
| 调试项 | logcat [<option>] [<filter-specs>] | 在屏幕上输出日志信息 |  |
| bugreport | 为报告bug，在屏幕上输出dumpsys， dumpstate和 logcat数据 |  |
| jdwp | 输出有效的JDWP进程信息 | 可以使用 forward jdwp:<pid> 转换端口以连接到指定的 JDWP 进程，例如：  adb forward tcp:8000 jdwp:472  jdb -attach localhost:8000 |
| 数据项 | install <path-to-apk> | 安装应用程序（用完整路径指定.apk文件） |  |
| pull <remote> <local> | 从开发机COPY指定的文件到模拟器或手机 |  |
| push <local> <remote> | 从模拟器或手机COPY文件到开发机 |  |
| 端口和网络项 | forward <local> <remote> | 从本地端口转换连接到模拟器或手机的指定端口 | 端口可以使用以下格式表示：    tcp:<portnum>  local:<UNIX domain socket name>  dev:<character device name>  jdwp:<pid> |
| ppp <tty> [parm]... | 通过USB运行UPP    <tty> —PPP流中的tty。例如:/dev/omap\_csmi\_ttyl。    [parm]... — 0到多个PPP/PPPD 选项, 例如 defaultroute, local, notty等等。  注意不用自动启动PPP连接 |  |
| 脚本项 | get-serialno | 输出adb对象的序列号 | 参见 [Querying for Emulator/Device Instances](http://developer.android.com/guide/developing/tools/adb.html#devicestatus)以获得更多信息。 |
| get-state | 输出adb设备的状态 |
| wait-for-device | 阻塞执行直到设备已经连接，即设备状态是 device. | 可以在其他命令前加上此项，那样的话adb就会等到模拟器或手机设备已经连接才会执行命令，例如:  注意该命令并不等待系统完全启动，因此不能追加需要在系统完全启动才能执行的命令，例如install 命令需要Android包管理器支持，但它必须在系统完全启动后才有效。下面的命令    会在模拟器或手机与adb发生连接后就执行install，但系统还没有完全启动，所以会引起错误。 |
| 服务端项 | start-server | 检测adb服务进程是否启动，如果没启动则启动它。 |  |
| kill-server | 终止服务端进程 |  |
| Shell | shell | 在目标模拟器或手机上启动远程SHELL | 参见 [Issuing Shell Commands](http://developer.android.com/guide/developing/tools/adb.html#shellcommands)以获得更多信息。 |
| shell [<shellCommand>] | 在目标模拟器或手机上执行shellCommand然后退出远程SHELL |

## 执行Shell命令

Adb提供了shell来在模拟器或物理设备上运行各种各样的命令，这些命令的二进制形式存在于这个路径中：

/system/bin/...

无论是否进入adb远程shell，都可以使用 shell 命令来执。

在未进入远程shell的情况下可以按下述格式执行单条命令：

adb [-d|-e|-s {<serialNumber>}] shell <shellCommand>

启动远程shell使用下面的格式：

adb [-d|-e|-s {<serialNumber>}] shell

退出远程shell时使用CTRL+D 或 exit 终止会话。

从远程shell检查sqlite3 数据库

通过远程shell，可以使用[sqlite3](http://www.sqlite.org/sqlite.html)命令行程序来管理由应用程序创建的SQLite数据库。 sqlite3 工具包含很多有用的命令，例如 .dump 用于输出表格的内容，.schema 用于为已经存在的表输出 SQL CREATE 语句。 并且该工具也提供了联机执行SQLite命令的能力。

使用 sqlite3时，向前文描述的那样进入模拟器的远程shell，然后使用sqlite3 命令。也可以在调用 sqlite3时指定数据库的全路径。SQLite3数据库存储在/data/data/<package\_name>/databases/路径下。

示例:

$ adb -s emulator-5554 shell

# sqlite3 /data/data/com.example.google.rss.rssexample/databases/rssitems.db

SQLite version 3.3.12

Enter ".help" for instructions

.... enter commands, then quit...

sqlite> .exit

一旦运行了 sqlite3，就可以使用 sqlite3 命令，退出并返回远程shell可以使用 exit 或 CTRL+D。

使用Monkey进行UI或应用程序测试

Monkey是运行于模拟器或手机上的一个程序，通过生成伪随机的大量的系统级的用户事件流来模拟操作，包括单击、触摸、手势等。从而为正在开发中的应用程序通过随机响应进行压力测试。

最简单使用monkey的方式是通过下面的命令行，它可以运行指定的应用程序并向其发送500个伪随机事件。

$ adb shell monkey -v -p your.package.name 500

关于monkey更多的选项及详细信息，请参见[UI/Application Exerciser Monkey](http://developer.android.com/guide/developing/tools/monkey.html)。

还有一个monkeyrunner。monkeyrunner工具提供了一个API，使用此API写出的程序可以在Android代码之外控制Android设备和模拟器。具体可见：<http://developer.android.com/guide/developing/tools/monkeyrunner_concepts.html>

其他Shell命令

    下表列出了很多有效的adb shell命令，完整的列表可以通过启动模拟器并且使用adb –help命令获取。

adb shell ls /system/bin

    帮助对于大部分命令是有效的。

|  |  |  |
| --- | --- | --- |
| Shell 命令 | 描述 | 备注 |
| dumpsys | 在屏幕上显示系统数据 | The [Dalvik Debug Monitor Service](http://developer.android.com/guide/developing/tools/ddms.html) (DDMS) 工具提供了更易于使用的智能的调试环境。 |
| dumpstate | 将状态输出到文件 |
| logcat [<option>]... [<filter-spec>]... | 输出日志信息 |
| dmesg | 在屏幕上输出核心调试信息 |
| start | 启动或重新启动模拟器或手机 |  |
| stop | 停止模拟器或手机 |  |

## 使用logcat查看日志

Android日志系统提供了从众多应用程序和系统程序中收集和查看调试信息的机制，这些信息被收集到一系统循环缓冲区中，可以 logcat 命令查看和过滤。

使用 logcat 命令

查看和跟踪系统日志缓冲区的命令logcat的一般用法是：

[adb] logcat [<option>] ... [<filter-spec>] ...

下文介绍过滤器和命令选项，详细内容可参见[Listing of logcat Command Options](http://developer.android.com/guide/developing/tools/adb.html#logcatoptions)。

可以在开发机中通过远程shell的方式使用logcat命令查看日志输出：

$ adb logcat

如果是在远程shell中可直接使用命令：

# logcat

过滤日志输出

每一条日志消息都有一个标记和优先级与其关联。

标记是一个简短的字符串，用于标识原始消息的来源 (例如"View" 来源于显示系统)。

优先级是下面的字符，顺序是从低到高：

V — 明细 (最低优先级)

D — 调试

I — 信息

W — 警告

E — 错误

F — 严重错误

S — 无记载 (最高优先级，没有什么会被记载)

通过运行logcat ，可以获得一个系统中使用的标记和优先级的列表，观察列表的前两列，给出的格式是<priority>/<tag>。

这里是一个日志输出的消息，优先级是“I”，标记是“ActivityManager”：

I/ActivityManager( 585): Starting activity: Intent { action=android.intent.action...}

如果想要减少输出的内容，可以加上过滤器表达式进行限制，过滤器可以限制系统只输出感兴趣的标记-优先级组合。

过滤器表达式的格式是tag:priority ... ，其中tag是标记， priority是最小的优先级， 该标记标识的所有大于等于指定优先级的消息被写入日志。也可以在一个过滤器表达式中提供多个这样的过滤，它们之间用空格隔开。

下面给出的例子是仅输出标记为“ActivityManager”并且优先级大于等于“Info”和标记为“MyApp”并且优先级大于等于“Debug”的日志：

adb logcat ActivityManager:I MyApp:D \*:S

上述表达式最后的 \*:S 用于设置所有标记的日志优先级为S，这样可以确保仅有标记为“View”（译者注：应该为ActivityManager，原文可能是笔误）和“MyApp”的日志被输出，使用 \*:S 是可以确保输出符合指定的过滤器设置的一种推荐的方式，这样过滤器就成为了日志输出的“白名单”。

下面的表达是显示所有优先级大于等于“warning”的日志：

adb logcat \*:W

如果在开发用电脑上运行 logcat (相对于运行运程shell而言)，也可以通过ANDROID\_LOG\_TAGS环境变量设置默认的过滤器表达式：

export ANDROID\_LOG\_TAGS="ActivityManager:I MyApp:D \*:S"

需要注意的是，如果是在远程shell或是使用adb shell logcat 命令运行logcat ， ANDROID\_LOG\_TAGS 不会导出到模拟器或手机设备上。

控制日志格式

日志消息在标记和优先级之外还有很多元数据字段，这些字段可以通过修改输出格式来控制输出结果， -v 选项加上下面列出的内容可以控制输出字段：

brief — 显示优先级/标记和原始进程的PID (默认格式)

process — 仅显示进程PID

tag — 仅显示优先级/标记

thread — 仅显示进程：线程和优先级/标记

raw — 显示原始的日志信息，没有其他的元数据字段

time — 显示日期，调用时间，优先级/标记，PID

long —显示所有的元数据字段并且用空行分隔消息内容

可以使用 -v启动 logcat来控制日志格式：

[adb] logcat [-v <format>]

例如使用 thread 输出格式：

adb logcat -v thread

注意只能在 -v 选项中指定一种格式。

Viewing Alternative Log Buffers

Android日志系统为日志消息保持了多个循环缓冲区，而且不是所有的消息都被发送到默认缓冲区，要想查看这些附加的缓冲区，可以使用-b 选项，以下是可以指定的缓冲区：

radio — 查看包含在无线/电话相关的缓冲区消息

events — 查看事件相关的消息

main — 查看主缓冲区 (默认缓冲区)

-b 选项的用法：

[adb] logcat [-b <buffer>]

例如查看radio缓冲区：

adb logcat -b radio

查看stdout和stderr

默认的，Android系统发送 stdout 和 stderr (System.out 和 System.err) 输出到 /dev/null。 在 Dalvik VM进程，可以将输出复制到日志文件，在这种情况下，系统使用 stdout 和 stderr标记写入日志，优先级是I。

要想使用这种方式获得输出，需要停止运行中的模拟器或手机，然后使用命令 setprop 来允许输出重定位，示例如下：

$ adb shell stop

$ adb shell setprop log.redirect-stdio true

$ adb shell start

系统会保留这一设置直到模拟器或手机退出，也可以在设备中增加/data/local.prop以使得这一设备成为默认配置。

**Logcat命令选项列表**

|  |  |
| --- | --- |
| 选项 | 描述 |
| -b <buffer> | 加载不同的缓冲区日志，例如 event 或radio。main 缓冲区是默认项，参见[Viewing Alternative Log Buffers](http://developer.android.com/guide/developing/tools/adb.html#alternativebuffers). |
| -c | 清空（刷新）所有的日志并且退出 |
| -d | 在屏幕上输出日志并退出 |
| -f <filename> | 将日志输出到文件<filename>，默认输出是stdout. |
| -g | 输出日志的大小 |
| -n <count> | 设置最大的循环数据<count>，默认是4，需要-r选项 |
| -r <kbytes> | 每<kbytes>循环日志文件，默认是16，需要 -f 选项 |
| -s | 设置默认的过滤器为无输出 |
| -v <format> | 设置输出格式，默认的是brief，支持的格式列表参见[Controlling Log Output Format](http://developer.android.com/guide/developing/tools/adb.html#outputformat). |

## 停止adb服务

在某些情况下，可能需要终止然后重启服务端进程，例如adb不响应命令的时候，可以通过重启解决问题。

使用kill-server可以终止服务端，然后使用其他的adb命令重启。

原文：<http://developer.android.com/guide/developing/tools/adb.html>