修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 作者 | 说明 |
| V0.01 | 2010年12月1日 | 徐申龙 | 初始版本。 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[多媒体加载 1](#_Toc280877168)

[DDMS with Android platform 3](#_Toc280877169)

[签名你的android项目 3](#_Toc280877170)

[signed prebuilt apk into the build 7](#_Toc280877171)

[Request for help to improve automated testing for the Android platform 8](#_Toc280877172)

[Contribution : Android SDK Installer 13](#_Toc280877173)

[[android-porting] Thread Local Storage and GDB server 14](#_Toc280877174)

[PowerManager 18](#_Toc280877175)

[Controlling the Embedded VM 20](#_Toc280877176)

[Process IDs of Android applications 25](#_Toc280877177)

[Re: [android-porting] Connecting a DSLR camera via PTP on Android 26](#_Toc280877178)

[[android-porting] Increase the Sampling Rate of Accelerometer 26](#_Toc280877179)

[Re: [android-porting] how to set the SeekBar more narrow? 26](#_Toc280877180)

[Re: StageFright vs. OpenCore 27](#_Toc280877181)

[your app's Android.mk file should look like this: 27](#_Toc280877182)

[Re: IPC, one-way and request-response transactions 27](#_Toc280877183)

frameworks/policies/base/phone/com/android/internal/policy/impl

KeyguardScreen.java

LockScreen.java

# 多媒体加载

例如，Android框架里所提供的MediaPlayer.java类，含指令：

public class MediaPlayer{

static {

System.loadLibrary("media\_jni");

}

}

这要求VM去载入Android的/system/lib/libmedia\_jni.so档案。载入\*.so之后，Java类与\*.so档案就汇合起来，一起执行了。

当Android的VM(Virtual Machine)执行到System.loadLibrary()函数时，首先会去执行C组件里的JNI\_OnLoad()函数。它的用途有二：

(1)告诉VM此C组件使用那一个JNI版本。如果你的\*.so档没有提供JNI\_OnLoad()函数，VM会默认该\*.so档是使用最老的JNI 1.1版本。由于新版的JNI做了许多扩充，如果需要使用JNI的新版功能，例如JNI 1.4的java.nio.ByteBuffer,就必须藉由JNI\_OnLoad()函数来告知VM。

(2)由于VM执行到System.loadLibrary()函数时，就会立即先呼叫JNI\_OnLoad()，所以C组件的开发者可以藉由JNI\_OnLoad()来进行C组件内的初期值之设定(Initialization) 。

JNI\_Onload 在函数

frameworks/base/services/jni/onload.cpp 、frameworks/base/media/jni/android\_media\_MediaPlayer.cpp 、dalvik/libcore/sql/src/main/native/sqlite\_jni.c等中实现。

frameworks/base/core/jni/AndroidRunitme.cpp 模块为libandroid\_runtime， 使用它的代码就是如下两个：

frameworks/base/core/java/com/android/internal/util/WithFramework.java:38: System.loadLibrary("android\_runtime");

frameworks/base/tools/preload/loadclass/LoadClass.java:28: System.loadLibrary("android\_runtime");

这个模块比较特殊，它是在虚拟机启动时加载的，而且没有调用JNI\_Onload注册Native API。

register\_jni\_procs

AndroidRuntime::start

AndroidRuntime::startVm startReg

-〉readLocale -〉JNI\_CreateJavaVM ->

dalvik/vm/Native.c vonLoad=dlsym(handle, “JNI\_OnLoad”);

pthread\_cond\_t onLoadCond;

OnLoadState onLoadResult;

dalvik/vm/Jni.c

development/pdk/docs/porting/debugging\_native.jd:261:I/DEBUG: #01 lr 580239f0 /android/lib/libandroid\_runtime.so

omx\_tests : opencore 测试工具

JNI\_CreateJavaVM， dalvik/vm/Jni.c

/\*

\* Create a new VM instance.

\*

\* The current thread becomes the main VM thread. We return immediately,

\* which effectively means the caller is executing in a native method.

\*/

./DeskClock/AndroidManifest.xml: android:launchMode="singleInstance"

./Calendar/AndroidManifest.xml: <activity android:name="AlertActivity" android:launchMode="singleInstance"

Phone/AndroidManifest.xml: android:launchMode="singleInstance">

./AlarmClock/AndroidManifest.xml: android:launchMode="singleInstance"

# DDMS with Android platform

1. Setup ADT plugin to work with Eclipse environment as described

http://developer.android.com/sdk/1.1\_r1/installing.html#installingplugin

2. DDMS configuration

Click on Window->Preferences; Select Android -> DDMS

Change - ADB debugger base port: 8700; Logging Level: Verbose

Click on Apply

3. Open DDMS perspective from the eclipse menu

4. Bootup your target board

# netcfg eth0 up

# netcfg eth0 dhcp

5. On the host machine run the following commands

export ADBHOST=<target board ip addr>

adb kill-server

adb start-server

adb devices

签名你的android项目

1. 导出你的项目，使用manifest.xml中的Export the unsigned apk. （假设为HelloWorld.apk）

2. 生产一个key. 使用命令行：keytool -genkey -v -keystore my-release-key.keystore -alias alias\_name -keyalg RSA -validity 10000。

my-release-key.keystore是生成的文件，

alias\_name是对这个密钥的引用，后面要用到，

最后一个数字10000是有效期，>=25才有效。

3.使用jarsigner给你的HelloWorld签名。

命令行：jarsigner -verbose -keystore my-release-key.keystore HelloWorld.apk alias\_name

4.验证是否签名成功：jarsigner -verify my\_signed.apk

如果成功，显示： jar已验证。

备注：为了保证你的密钥的安全性，不该使用 -storepass , -keypass

转载时请注明出处和作者联系方式

文章出处：http://www.limodev.cn/blog

作者联系方式：李先静 <xianjimli at hotmail dot com>

昨天看了一下Android中的签名机制，这里介绍一下Android中签名用的Key的产生方法和签名的原理。

产生Key

o 产生RSA私钥(private key)

openssl genrsa -3 -out testkey.pem 2048

-3 是算法的参数（public exponent）。

2048 是私钥长度。

testkey.pem 是输出的文件。

o 产生PKCS#10格式的认证请求。所谓认证请求就是发给认证机构认证的一个请求，它主要包括一个公钥和一些相关信息(如组织名称和联系人邮件地址)。

openssl req -new -x509 -key testkey.pem -out testkey.x509.pem -days 10000 \

-subj ‘/C=US/ST=California/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com’

如果不提供最后两个参数，openssl会提示你输入相关信息，这里的信息可以根据你自己的实际情况填写。如：

openssl req -new -x509 -key testkey.pem -out testkey.x509.pem -days 10000

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter ‘.’, the field will be left blank.

—–

Country Name (2 letter code) [GB]:CN

State or Province Name (full name) [Berkshire]:GuangDong

Locality Name (eg, city) [Newbury]:ShenZhen

Organization Name (eg, company) [My Company Ltd]:Topwise

Organizational Unit Name (eg, section) []:Broncho

Common Name (eg, your name or your server’s hostname) []:broncho.cn

Email Address []:bronchosales@gmail.com

o 把私钥的格式转换成PKCS #8（Private-Key Information Syntax Standard.）

openssl pkcs8 -in testkey.pem -topk8 -outform DER -out testkey.pk8 -nocrypt

私钥是不能让别人知道的，否则就起不到保密的作用了。私钥通常是要加密保存的，但这里指定了-nocryp，表示不加密。

Android提供了一个脚本mkkey.sh用来简化上面的步骤：

if ["$1" == ""]; then

echo "Create a test certificate key."

echo "Usage: $0 NAME"

echo "Will generate NAME.pk8 and NAME.x509.pem"

echo " /C=US/ST=California/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com"

return

fi

openssl genrsa -3 -out $1.pem 2048

openssl req -new -x509 -key $1.pem -out $1.x509.pem -days 10000 \

-subj '/C=US/ST=California/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com'

openssl pkcs8 -in $1.pem -topk8 -outform DER -out $1.pk8 -nocrypt

签名

Android提供了为jar/zip文件签名的程序signapk.jar 。

它的用法如下：

Usage: signapk publickey.x509[.pem] privatekey.pk8 input.jar output.jar

第一个参数是公钥，即前面第二步产生的testkey.x509.pem。

第二个参数是私钥，即前面第三步产生的testkey.pk8。

第三个参数是要签名的文件。

第四个参数是输出的文件（即签名后的文件）。

如：java -jar signapk.jar testkey.x509.pem testkey.pk8 update.zip update-signed.zip

现在我们来看看签名到底做了些什么：

o 先为输入的jar/zip文件中的所有文件生成SHA1数字签名(除了CERT.RSA,CERT.SF和MANIFEST.MF）

for (JarEntry entry: byName.values()) {

String name = entry.getName();

if (!entry.isDirectory() &amp;&amp; !name.equals(JarFile.MANIFEST\_NAME) &amp;&amp;

!name.equals(CERT\_SF\_NAME) &amp;&amp; !name.equals(CERT\_RSA\_NAME) &amp;&amp;

(stripPattern == null ||

!stripPattern.matcher(name).matches())) {

InputStream data = jar.getInputStream(entry);

while ((num = data.read(buffer)) &gt; 0) {

md.update(buffer, 0, num);

}

Attributes attr = null;

if (input != null) attr = input.getAttributes(name);

attr = attr != null ? new Attributes(attr) : new Attributes();

attr.putValue("SHA1-Digest", base64.encode(md.digest()));

output.getEntries().put(name, attr);

}

}

并把数字签名信息写入MANIFEST.MF

je = new JarEntry(JarFile.MANIFEST\_NAME);

je.setTime(timestamp);

outputJar.putNextEntry(je);

manifest.write(outputJar);

o 对manifest签名并写入CERT.SF

// CERT.SF

Signature signature = Signature.getInstance("SHA1withRSA");

signature.initSign(privateKey);

je = new JarEntry(CERT\_SF\_NAME);

je.setTime(timestamp);

outputJar.putNextEntry(je);

writeSignatureFile(manifest,

new SignatureOutputStream(outputJar, signature));

o 把对输出文件的签名和公钥写入CERT.RSA。

// CERT.RSA

je = new JarEntry(CERT\_RSA\_NAME);

je.setTime(timestamp);

outputJar.putNextEntry(je);

writeSignatureBlock(signature, publicKey, outputJar);

签名的作用

签名的主要目的为了检测文件是否被别人修改了。但它并不能禁止别人修改，因为你完全重新生成签名，但是你生成的签名和原来是不一样的。

build/target/product/security 默认的签名文件

/home/leo/android/platform/build/tools/releasetools/

development/tools/make\_key 脚本，生成签名文件。

‘/C=US/ST=California/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com’

/C = COUNTRY

/ST = STATE

/L Locality Name (eg, city)

/O organization

/OU organization unit

/CN common name

/emailAddress

openssl$ java -jar signapk.jar testkey.x509.pem testkey.pk8 package.apk.unsigned package.apk

signapk.jar 只是一个Java包，需要虚拟机java来运行。

比较了package.apk.unsigned 与 package.apk，变化是META-INF目录中的，增加了CERT.SF 及CERT.RSA两个文件，MANIFEST.MF 变大了。MANIFEST.MF 与CERT.SF都是文本文件，内容格式是类似的，CERT.SF形如：

Signature-Version: 1.0

Created-By: 1.0 (Android SignApk)

SHA1-Digest-Manifest: dsoVnCMNnDa/3j7/7mbjAipZUro=

Name: res/drawable-mdpi-finger/dial\_num\_2\_wht.png

SHA1-Digest: puFCQ+J8q9AYABC3neqjO15Hpdc=

MANIFEST.MF形如：

Manifest-Version: 1.0

Created-By: 1.6.0\_22 (Sun Microsystems Inc.)

Name: res/drawable-mdpi-finger/dial\_num\_2\_wht.png

SHA1-Digest: pYcwrAfArZoDm2H08VSTgGHlZ2c=

## signed prebuilt apk into the build

Please help in adding signed prebuilt third party apk into the build.

>

> What is the for the variable "LOCAL\_CERTIFICATE := " in make fike

> ie

> LOCAL\_CERTIFICATE := platform

> or

> LOCAL\_CERTIFICATE := PRESIGNED

> or

> LOCAL\_CERTIFICATE := <path to the certificate>

IMHO this was changed between 2.1 and 2.2 ..

In 2.2 you will get an build error if you add a prebuild signed app

without using PRESIGNED.

I.E. you have to sign the APP in the full code build and use PRESIGNED

when you build an image with the prebuild version.

Good luck !

If you are building the apk using the android platform build system, you can specify the certificate by:

LOCAL\_CERTIFICATE := testkey

(or just leave it empty, the default is testkey).

start --plan CTS -t

tests.api.java.lang.reflect.ParameterizedTypeTest#testStringParameterizedSuperClass

# Request for help to improve automated testing for the Android platform

We have been working with GUI test automation for Android and would

like to share our experiences and ask for help finding ways to improve

how tests can be automated for the Android platform, especially for

physical phones. The bulk of the our work has been done by Tommi

Takala, as part of research led by Mika Katara at Tampere University

of Technology on Model-Based-Testing for mobile phone platforms, most

recently for Android. The project's web site is http://tema.cs.tut.fi

which should have more content in a few weeks. The work for Android

MBT has been open-sourced by the university and is available via the

project's web site.

My role (Julian Harty) is more general to help bridge the university's

work with real-world projects and uses of their work, etc. I used to

work for Google until June 2010 and have written material on mobile

testing and test automation see http://tr.im/mobtest, worked on

WebDriver and related projects.

Below is some background information we have gathered; and a short

description of the test automation tool prototype that we developed.

Currently, Android SDK sets some limitations on our tool, but we hope

that the following discussion could help in improving the test

automation features from the GUI testing point of view.

Background:

---------------------------------------

Creating GUI test automation on Android requires a way to interpret

the contents of the GUI to verify its state and a way to inject user

input (key presses and touch screen events) to the device. The former

issues are described in section 2 and the later in section 3 below.

1. Existing automation solutions

Some open source automation tools (Positron http://code.google.com/p/autoandroid/,

Robotium http://code.google.com/p/robotium/) already exist for

Android. They both extend the Android instrumentation framework. They

are therefore limited to testing a single application. They also

generally rely on having access to the source code to use them

effectively, although Robotium has the ability to test an APK

http://code.google.com/p/robotium/wiki/RobotiumForAPKFiles

2. GUI Access

For verifying the GUI state and to search e.g. button/text locations,

we need a way to access the GUI resources. There are basically three

ways to facilitate this: Java GUI inspection, API access using

Accessibility, and optical character recognition (OCR). Below are some

details of the approaches:

2.1 Java GUI inspection (hierarchyviewer, window service)

Android SDK is delivered with the hierarchyviewer tool

http://developer.android.com/guide/developing/tools/hierarchy-viewer.html

that work over the Android Debug Bridge (adb) for debugging the GUI of

applications. With these tools, you can dump the whole hierarchical

view of the GUI containing all its components. The GUI components show

most of their attributes (text, screen coordinates, visibility, etc.).

However, due to security reasons, these features have been disabled

from the majority of devices on the market for security reasons

http://groups.google.com/group/android-developers/browse\_thread/thread/b0b0af7a316ca768?pli=1

so using these tools might be hard outside the emulator, Android Dev

Phones or R&D prototype phones. It is also unclear how the dump works

for custom GUI components.

Window service usage:

1. start window service: adb shell service call window 1 i32 4939

2. set port forwarding: adb forward tcp:4939 tcp:4939

3. connect the service with a socket

4. list the contents of the foreground window: dump -1

Other commands:

- list: list of windows

- capture <windowid> <wiewid>: Capture a view (PNG format)

2.1.2 Problems

I. Absolute coordinates:

each GUI object lists its coordinates related to the parent view. In

order to e.g. tap the object, we need to know its absolute coordinates

on the screen, which can be calculated by summing the relative

coordinates of the parent views all the way to the hierarchy root.

However, if the foreground window does not start from the top left

corner of the screen (a floating window e.g. a menu), the calculation

will give false results. There is also a bug in hierarchyviewer

related to this issue: Pixel perfect view shows the location of e.g.

menu buttons incorrectly.

To fix this problem, another adb command (adb shell dumpsys window)

could be used. The output of the command lists information about the

windows in the device, including the absolute coordinates. However, it

might be risky to rely on the output of such commands that are not

public API, as the format might change between platform releases.

II. Another problem occurs if a GUI object is partially off-screen or

behind another object, when using the hierarchy may cause problems by

giving false positive results for verifications and causing e.g. taps

to coordinates go for wrong objects.

III. A number of component attributes are not listed in the dump. For

example, the Hint property of an e.g. EditText component or the

checked attribute of a Checkbox are not available.

2.2 Accessibility

In version 1.6 Android introduced new accessibility libraries. These

libraries enable developers to install AccessibilityServices in the

device that can be used to receive notifications on various GUI

events. For example, when a menu is opened, the service receives a

notification where you can see the different menu items, or when a

button is focused, you can see the text of the button.

However, Android accessibility does not provide means to see the whole

treelike hierarchy of the GUI (the way e.g. at-spi shows it in Linux),

which would be important when searching e.g. a specific button. Going

through the GUI using a trackball for trying to find a specific

component does not sound very practical. Improving this would improve

both accessibility and test automation but could possibly compromise

security.

2.3 Optical character recognition (OCR)

Fairly reliable text recognition can be created using the Microsoft

Office Document Imaging (MODI) components. However, OCR still is still

imperfect and likely requires maintenance when GUI styles change.

Moreover, MODI requires Windows Office (2003 or later) and thus is not

exactly free and only works on Windows.

To perform OCR, we need to capture the screen of the device. The

Dalvik Debug Monitor (DDMS) delivered with the SDK has a tool to

capture the screen (most likely using adb), so this functionality can

be extracted from there. The android source also contains a command

line tool for taking screenshots.

3. User input (Key presses, touch screen, trackball, etc.)

The simplest way to inject user input is to use adb. First, a device

is connected to remote pc via USB, then the adb sendevent command can

be used to cause events on the device. Android APIs also contain

methods for sending keypresses and touch screen events in the Java

code, but these work only inside the activity that is calling these

methods, so other applications can't be affected.

The Monkey tool that runs on Android phones/emulator uses the

sendevent command for generating GUI events. The tool provides a

network interface that seems very suitable for all user input needs

for test automation. Using the Monkey's interface looks much easier

than directly using the sendevent command. Network interface

documentation can be found from README.NETWORK.txt in androidsourceroot

\development\cmds\monkey

4. Architecture:

Depending on the chosen GUI access method, the resulting architecture

of the test automation system is one of the following:

1) If only adb is used for GUI access and user input (Java GUI

inspection or screen capture), the architecture will be very simple,

as any extra client applications on the device are not required. The

test automation application will be installed on a remote pc and adb

is used to communicate with the device.

2) If accessibility is used in verification, a service application

that reads the GUI notifications and communicates with the test

automation application on a remote pc will be required on the device.

5. Test Automation tool prototype for Android

We developed a tool prototype that uses the Monkey's network interface

for user input and window service for the GUI verification. The tool

is Keyword-driven, meaning that GUI actions and verifications are

described using abstract keywords. As an example, the following list

of keywords adds a new clock Widget to the home screen and removes it

immediately:

PressKey home

SelectFromMenu 'Add'

SelectFromList 'Widgets'

SelectFromList 'Analog clock'

Drag ;AnalogClock --> id\_delete\_zone

You can check how the script is executed in the emulator:

http://www.youtube.com/watch?v=6mdgYHmTAcg

The tool also implements an interface to a model-based testing tool

(TEMA, see http://tema.cs.tut.fi) that is used to generate keyword-

based tests automatically from state-machine models. In addition,

keywords can be executed from a file or using an interactive prompt.

The keywords used in the tool are formed from the keyword name and a

list of parameters. Many keywords are directed to a specific GUI

object, so the object must be defined in the parameters. To define the

target object, the keywords accept references that use the object ID,

text content and/or the classname of the object. The hierarchical

structure of the GUI can also be used in the references, by defining

the parents of the searched object. Sometimes, using the hierarchy can

be the only way to uniquely identify an object.

A problem with the tool is that dumping the GUI from the Window

service takes a bit too much time and slows down the execution. This

makes testing rapidly changing GUIs hard. Another problem is that the

dump does not contain all the attributes of the GUI widgets. Finally,

test automation is limited to the emulator and hardware-unlocked

devices.

Conclusion

========

Test Automation is available for Android and there are several open-

source initiatives to help augment the automation in the current

platform and tools. Currently automated testing on devices is limited,

partly to protect the security of user-devices. Can you help find or

provide ways to improve the ability to automate system and application

testing for physical Android devices, ideally that would enable and

support a range of approaches?

Thanks

Julian Harty, Tommi Takala, Mika Katara

# Contribution : Android SDK Installer

I have made Android SDK Installer as described in thread

http://groups.google.com/group/android-contrib/t/99293cd39f429cfc

It made as separate plugin (for now). See http://code.google.com/p/adt-addons/.

Shortly, you can try it using the

https://adt-addons.googlecode.com/svn/trunk/installer/com.android.ide.eclipse.installer.update/

update site (ADT plugin will be also downloaded if it doesn't exist).

The source code is on http://adt-addons.googlecode.com/svn/trunk/installer/

.

The plugin work as follows:

There is the org.eclipse.ui.startup ext. point. This class checks if

Android SDK preferences is empty and if so, show the Android SDK

Installer dialog - http://www.snpe.rs/androidsdkdialog.png

The user can set an existing ADK environment, download the latest SDK

Tools and platform or ignore dialog.

The "Hello, world" Eclipse cheat-sheet is bonus.

Would you like me to contribute (adapt) this feature ?

When I write a camera demo to take picture in froyo, but the preview video and the taken picture all rotate 90°。However, the android camera app is ok. And I have analyze the parameters. They are the same.

I have try to set camera parameters, and camera.sendCommand(CAMERA\_CMD\_SET\_DISPLAY\_ORIENTATION, 4, 0). But, it has no effect.

It seems it has difference to surface. But， I do not know how to create such surface. And set it to camera.

following is my code script.

\_surfaceClient = new SurfaceComposerClient();

\_surfaceControl = \_surfaceClient->createSurface(getpid(), 0, \_videoPreWidth, \_videoPreHeight, 4, a ndroid::ISurfaceComposer::ePushBuffers);

\_surfaceControl->setPosition(70, 70);

\_surfaceClient->openTransaction();

\_surfaceControl->setLayer(100000);

\_surfaceControl->show();

\_surfaceClient->closeTransaction();

\_surface = \_surfaceControl->getSurface();

Could anyone help me out, please?

thank you very much!

This issue is resolved by invoke SurfaceComposerClient::setOrientation()

# [android-porting] Thread Local Storage and GDB server

I ended up figuring this out for myself, and I provide details here in case they are useful for others :

To get gdbserver working with TLS reg support was just a case of recompiling gdbserver with a sysroot generated from my own release.

That was fairly simple (after fixing gdb to work with my header files that seem to be different from standard in some way).

However, while doing this, I also got gdbserver to be able to debug multiple threads. This required a new function in libthread\_db.so.

I dont have time to submit this as a patch to google, so I will include it here. Feel free to submit it to google as a patch if you have the time. This is the log from my svn server. The summary is that I added td\_thr\_tls\_get\_addr to libthread\_db.so to get the tls area of a requested thread. I also made libdl build a static version of its library as well as the dynamic one so that gdbserver can be built statically against it (this is not required just to get multithreaded gdbserver working).

Hope this is of help to someone

Bob

Index: bionic/libthread\_db/include/thread\_db.h

===================================================================

--- bionic/libthread\_db/include/thread\_db.h (revision 914)

+++ bionic/libthread\_db/include/thread\_db.h (revision 915)

@@ -136,6 +136,9 @@

extern char const \*\* td\_symbol\_list(void);

+extern td\_err\_e

+td\_thr\_tls\_get\_addr(const td\_thrhandle\_t \*th, void \*map\_address,

+ size\_t offset, void \*\*address);

#ifdef \_\_cplusplus

}

#endif

Index: bionic/libthread\_db/libthread\_db.c

===================================================================

--- bionic/libthread\_db/libthread\_db.c (revision 914)

+++ bionic/libthread\_db/libthread\_db.c (revision 915)

@@ -181,3 +181,12 @@

return err;

}

+td\_err\_e

+td\_thr\_tls\_get\_addr(const td\_thrhandle\_t \*th, void \*map\_address,

+ size\_t offset, void \*\*address)

+{

+ pthread\_t pth = th->tid;

+ \*address = pthread\_get\_tls(pth);

+ return TD\_OK;

+}

+

Index: bionic/libdl/Android.mk

===================================================================

--- bionic/libdl/Android.mk (revision 914)

+++ bionic/libdl/Android.mk (revision 915)

@@ -53,6 +53,55 @@

include $(BUILD\_SHARED\_LIBRARY)

+include $(CLEAR\_VARS)

+

+# NOTE: --exclude-libs=libgcc.a makes sure that any symbols libdl.so pulls from

+# libgcc.a are made static to libdl.so. This in turn ensures that libraries that

+# a) pull symbols from libgcc.a and b) depend on libdl.so will not rely on libdl.so

+# to provide those symbols, but will instead pull them from libgcc.a. Specifically,

+# we use this property to make sure libc.so has its own copy of the code from

+# libgcc.a it uses.

+#

+# DO NOT REMOVE --exclude-libs!

+

+LOCAL\_LDFLAGS := -Wl,--exclude-libs=libgcc.a

+

+# for x86, exclude libgcc\_eh.a for the same reasons as above

+ifneq ($(TARGET\_SIMULATOR),true)

+ifeq ($(TARGET\_ARCH),x86)

+LOCAL\_LDFLAGS += -Wl,--exclude-libs=libgcc\_eh.a

+endif

+endif

+

+LOCAL\_SRC\_FILES:= libdl.c

+

+LOCAL\_MODULE:= libdl

+

+# NOTE: libdl needs \_\_aeabi\_unwind\_cpp\_pr0 from libgcc.a but libgcc.a needs a

+# few symbols from libc. Using --no-undefined here results in having to link

+# against libc creating a circular dependency which is removed and we end up

+# with missing symbols. Since this library is just a bunch of stubs, we set

+# LOCAL\_ALLOW\_UNDEFINED\_SYMBOLS to remove --no-undefined from the linker flags.

+LOCAL\_ALLOW\_UNDEFINED\_SYMBOLS := true

+LOCAL\_SYSTEM\_SHARED\_LIBRARIES :=

+

+ifeq ($(TARGET\_ARCH),sh)

+# for SuperH, additional code is necessary to handle .ctors section.

+GEN\_SOBEGIN := $(TARGET\_OUT\_STATIC\_LIBRARIES)/sobegin.o

+$(GEN\_SOBEGIN): $(LOCAL\_PATH)/arch-sh/sobegin.S

+ @mkdir -p $(dir $@)

+ $(TARGET\_CC) -o $@ -c $<

+

+GEN\_SOEND := $(TARGET\_OUT\_STATIC\_LIBRARIES)/soend.o

+$(GEN\_SOEND): $(LOCAL\_PATH)/arch-sh/soend.S

+ @mkdir -p $(dir $@)

+ $(TARGET\_CC) -o $@ -c $<

+

+LOCAL\_ADDITIONAL\_DEPENDENCIES := $(GEN\_SOBEGIN) $(GEN\_SOEND)

+endif

+

+include $(BUILD\_STATIC\_LIBRARY)

+

BUILD\_DLTEST:=0

ifeq ($(BUILD\_DLTEST),1)

Index: bionic/libc/include/pthread.h

===================================================================

--- bionic/libc/include/pthread.h (revision 914)

+++ bionic/libc/include/pthread.h (revision 915)

@@ -107,6 +107,7 @@

extern "C" {

#endif

+void\*\* pthread\_get\_tls(pthread\_t \*thread);

int pthread\_attr\_init(pthread\_attr\_t \* attr);

int pthread\_attr\_destroy(pthread\_attr\_t \* attr);

Index: bionic/libc/bionic/pthread.c

===================================================================

--- bionic/libc/bionic/pthread.c (revision 914)

+++ bionic/libc/bionic/pthread.c (revision 915)

@@ -1879,3 +1879,10 @@

}

return 0;

}

+

+void\*\* pthread\_get\_tls(pthread\_t \*thread)

+{

+ pthread\_internal\_t \* pthi = (pthread\_internal\_t \*)thread;

+ return pthi->tls;

+}

+

# PowerManager

frameworks/base/services/java/com/android/server/status/

提供了状态栏的服务，StatusBarService.java, , 当然会涉及了电池电量提示的图标。StatusBarPolicy.java 的方法 updateBattery(Intent) 没有使用BatteryManager中定义的常变量，而是直接使用了字符串，如"icon\_small"，这个对应的 BatteryManager::EXTRA\_ICON\_SMALL。

frameworks/base/core/java/android/app/StatusBarManager.java

frameworks/policies/base/phone/com/android/internal/policy/impl/PhoneWindowManager.java

frameworks/policies/base/mid/com/android/internal/policy/impl/MIDWindowManager.java

frameworks/base/core/res/res/drawable/stat\_sys\_battery.xml

frameworks/base/core/res/res/drawable/stat\_sys\_battery\_change.xml

R.string.battery\_info\_status\_charging

R.string.battery\_info\_status\_charging\_ac

R.string.battery\_info\_status\_charging\_usb

system/core/rootdir/etc/init.goldfish.rc

#fake some battery state

setprop status.battery.state slow

packages/apps/Settings/src/com/android/settings/BatteryInfo.java 设置应用里设置菜单下电池状态。

packages/apps/Settings/src/com/android/settings/deviceinfo/Status.java

状态栏下的电池图标。

两者都接收了BatteryService 的通知。

findPreference("battery-status");

findPreference("battery-level");

frameworks/base/core/java/android/preference/PreferenceActivity.java, PreferenceGroup.java, PreferenceManager.java

SystemServer.java

SystemService.java

SystemProperties.java

SystemServer.java SystemServer 的方法 init2 开了一个 ServerThread 实例，在ServerThread.run 里实例化了 PowerManagerService 、Watchdog、WallpaperManagerService、InputMethodManagerService 等服务。

frameworks/base/services/java/com/android/server/BatteryService.java

frameworks/base/services/java/com/android/server/am/BatteryStatsServices.java

getIcon -> sendIntent -> update -> BatteryService

/sys/class/power-supply/battery/status

/sys/devices/platform/goldfish-battery.o/power-supply

mBatteryStatus, mUEventObserver, onUEvent

hardware/libhardware\_legacy/power/power.c

sys/class/power\_supply/battery/

Android电源管理概念

Android电源管理

一、 相关概念

1. 出于节电的需要，一般应用在用户一段时间无操作的情况下屏幕变暗，然后进后休眠状态

2. 用户只能在”设置->声音和显示”中设置所有应用默认的屏幕亮度和进行待机的时间

3. 电源管理的实现分内核应用两部分，通过下面介绍的接口，我们可以设置应用程序的电源管理，以控制与其休眠相关的状态（是否需要进入休眠，调整cpu频率，键盘灯的开关，屏幕的亮暗等）

二、 设置电源管理常用的几种状态

PARTIAL\_WAKE\_LOCK 屏幕关，键盘灯关，不休眠

SCREEN\_MID\_WAKE\_LOCK 屏幕灰，键盘灯关，不休眠

SCREEN\_BRIGHT\_WEEK\_LOCK 屏幕亮，键盘灯关，不休眠

FULL\_WAKE\_LOCK 屏幕亮，键盘灯亮，不休眠

三、 使用电源管理注意事项

1. 可在onCreate时设置该界面的电源管理，在onDestroy时取消设置

2. 可在onResume时设置该界面的电源管理，在onPause时取消设置

3. 注意设置是以Activity为单位，不是以应用为单位

4. 注意在AndroidManifest.xml中声明该应用有设置电源管理的权限

5. 注意加锁解锁要成对出现

6. 注意多个用途最好用多个锁，不要一锁多用，以免出错

7. 注意对运行在后台和异常时对锁的处理

8. 注意在网络连接或传输时最好加锁，以免传输被中断

9. 注意加锁以保证程序逻辑

四、 代码举例

1. 源码修改

1) 引入电源管理包，以使用相关类

import android.os.PowerManager;

2) 类中加入变量

PowerManager.WakeLock mWakeLock;

3)修改onCreate

onCreate public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

PowerManager pm = (PowerManager) getSystemService(Context.POWER\_SERVICE);

mWakeLock = pm.newWakeLock(PowerManager.SCREEN\_BRIGHT\_WAKE\_LOCK, "XYTEST"); mWakeLock.acquire();

}

4) 修改onDestroy

public void onDestroy() {

super.onDestroy(); mWakeLock.release(); }

2. AndroidManifest.xml文件修改

<uses-permission android:name="android.permission.WAKE\_LOCK"/>

多屏幕支持

<http://androidappdocs.appspot.com/guide/practices/screens_support.html>

# Controlling the Embedded VM

《http://android.git.kernel.org/?p=platform/dalvik.git;a=blob\_plain;f=docs/embedded-vm-control.html;hb=HEAD》

Introduction (read this first!)

Extended JNI Checks

Assertions

Bytecode Verification and Optimization

Execution Mode

Deadlock Prediction

Stack Dumps

DEX File Checksums

Introduction (read this first!)

The Dalvik VM supports a variety of command-line arguments (use adb shell dalvikvm -help to get a summary), but it's not possible to pass arbitrary arguments through the Android application runtime. It is, however, possible to affect the VM behavior through certain system properties.

For all of the features described below, you would set the system property with setprop, issuing a shell command on the device like this:

adb shell setprop <name> <value>

The Android runtime must be restarted before the changes will take effect (adb shell stop; adb shell start). This is because the settings are processed in the "zygote" process, which starts early and stays around "forever".

You may not be able to set dalvik.\* properties or restart the system as an unprivileged user. You can use adb root or run the su command from the device shell on "userdebug" builds to become root first. When in doubt,

adb shell getprop <name>

will tell you if the setprop took.

If you don't want the property to evaporate when the device reboots, add a line to /data/local.prop that looks like:

<name> = <value>

Such changes will survive reboots, but will be lost if the data partition is wiped. (Hint: create a local.prop on your workstation, then adb push local.prop /data. Or, use one-liners like adb shell "echo name = value >> /data/local.prop" -- note the quotes are important.)

Extended JNI Checks

JNI, the Java Native Interface, provides a way for code written in the Java programming language interact with native (C/C++) code. The extended JNI checks will cause the system to run more slowly, but they can spot a variety of nasty bugs before they have a chance to cause problems.

There are two system properties that affect this feature, which is enabled with the -Xcheck:jni command-line argument. The first is ro.kernel.android.checkjni. This is set by the Android build system for development builds. (It may also be set by the Android emulator unless the -nojni flag is provided on the emulator command line.) Because this is an "ro." property, the value cannot be changed once the device has started.

To allow toggling of the CheckJNI flag, a second property, dalvik.vm.checkjni, is also checked. The value of this overrides the value from ro.kernel.android.checkjni.

If neither property is defined, or dalvik.vm.checkjni is set to false, the -Xcheck:jni flag is not passed in, and JNI checks will be disabled.

To enable JNI checking:

adb shell setprop dalvik.vm.checkjni true

You can also pass JNI-checking options into the VM through a system property. The value set for dalvik.vm.jniopts will be passed in as the -Xjniopts argument. For example:

adb shell setprop dalvik.vm.jniopts forcecopy

For more information about JNI checks, see JNI Tips.

Assertions

Dalvik VM supports the Java programming language "assert" statement. By default they are off, but the dalvik.vm.enableassertions property provides a way to set the value for a -ea argument.

The argument behaves the same as it does in other desktop VMs. You can provide a class name, a package name (followed by "..."), or the special value "all".

For example, this:

adb shell setprop dalvik.vm.enableassertions all

enables assertions in all non-system classes.

The system property is much more limited than the full command line. It is not possible to specify more than one -ea entry, and there is no way to specify a -da entry. There is presently no equivalent for -esa/-dsa.

Bytecode Verification and Optimization

The system tries to pre-verify all classes in a DEX file to reduce class load overhead, and performs a series of optimizations to improve runtime performance. Both of these are done by the dexopt command, either in the build system or by the installer. On a development device, dexopt may be run the first time a DEX file is used and whenever it or one of its dependencies is updated ("just-in-time" optimization and verification).

There are two command-line flags that control the just-in-time verification and optimization, -Xverify and -Xdexopt. The Android framework configures these based on the dalvik.vm.dexopt-flags property.

If you set:

adb shell setprop dalvik.vm.dexopt-flags v=a,o=v

then the framework will pass -Xverify:all -Xdexopt:verified to the VM. This enables verification, and only optimizes classes that successfully verified. This is the safest setting, and is the default.

You could also set dalvik.vm.dexopt-flags to v=n to have the framework pass -Xverify:none -Xdexopt:verified to disable verification. (We could pass in -Xdexopt:all to allow optimization, but that wouldn't necessarily optimize more of the code, since classes that fail verification may well be skipped by the optimizer for the same reasons.) Classes will not be verified by dexopt, and unverified code will be loaded and executed.

Enabling verification will make the dexopt command take significantly longer, because the verification process is fairly slow. Once the verified and optimized DEX files have been prepared, verification incurs no additional overhead except when loading classes that failed to pre-verify.

If your DEX files are processed with verification disabled, and you later turn the verifier on, application loading will be noticeably slower (perhaps 40% or more) as classes are verified on first use.

For best results you should force a re-dexopt of all DEX files when this property changes. You can do this with:

adb shell "rm /data/dalvik-cache/\*"

This removes the cached versions of the DEX files. Remember to stop and restart the runtime (adb shell stop; adb shell start).

(Previous version of the runtime supported the boolean dalvik.vm.verify-bytecode property, but that has been superceded by dalvik.vm.dexopt-flags.)

Execution Mode

The current implementation of the Dalvik VM includes three distinct interpreter cores. These are referred to as "fast", "portable", and "debug". The "fast" interpreter is optimized for the current platform, and might consist of hand-optimized assembly routines. In constrast, the "portable" interpreter is written in C and expected to run on a broad range of platforms. The "debug" interpreter is a variant of "portable" that includes support for profiling and single-stepping.

The VM may also support just-in-time compilation. While not strictly a different interpreter, the JIT compiler may be enabled or disabled with the same flag. (Check the output of dalvikvm -help to see if JIT compilation is enabled in your VM.)

The VM allows you to choose between "fast", "portable", and "jit" with an extended form of the -Xint argument. The value of this argument can be set through the dalvik.vm.execution-mode system property.

To select the "portable" interpreter, you would use:

adb shell setprop dalvik.vm.execution-mode int:portable

If the property is not specified, the most appropriate interpreter will be selected automatically. At some point this mechanism may allow selection of other modes, such as JIT compilation.

Not all platforms have an optimized implementation. In such cases, the "fast" interpreter is generated as a series of C stubs, and the result will be slower than the "portable" version. (When we have optimized versions for all popular architectures the naming convention will be more accurate.)

If profiling is enabled or a debugger is attached, the VM switches to the "debug" interpreter. When profiling ends or the debugger disconnects, the original interpreter is resumed. (The "debug" interpreter is substantially slower, something to keep in mind when evaluating profiling data.)

The JIT compiler can be disabled on a per-application basis by adding android:vmSafeMode="true" in the application tag in AndroidManifest.xml. This can be useful if you suspect that JIT compilation is causing your application to behave incorrectly.

Deadlock Prediction

If the VM is built with WITH\_DEADLOCK\_PREDICTION, the deadlock predictor can be enabled with the -Xdeadlockpredict argument. (The output from dalvikvm -help will tell you if the VM was built appropriately -- look for deadlock\_prediction on the Configured with: line.) This feature tells the VM to keep track of the order in which object monitor locks are acquired. If the program attempts to acquire a set of locks in a different order from what was seen earlier, the VM logs a warning and optionally throws an exception.

The command-line argument is set based on the dalvik.vm.deadlock-predict property. Valid values are off to disable it (default), warn to log the problem but continue executing, err to cause a dalvik.system.PotentialDeadlockError to be thrown from the monitor-enter instruction, and abort to have the entire VM abort.

You will usually want to use:

adb shell setprop dalvik.vm.deadlock-predict err

unless you are keeping an eye on the logs as they scroll by.

Please note that this feature is deadlock prediction, not deadlock detection -- in the current implementation, the computations are performed after the lock is acquired (this simplifies the code, reducing the overhead added to every mutex operation). You can spot a deadlock in a hung process by sending a kill -3 and examining the stack trace written to the log.

This only takes monitors into account. Native mutexes and other resources can also be the cause of deadlocks, but will not be detected by this.

Stack Dumps

Like other desktop VMs, when the Dalvik VM receives a SIGQUIT (Ctrl-\ or kill -3), it dumps stack traces for all threads. By default this goes to the Android log, but it can also be written to a file.

The dalvik.vm.stack-trace-file property allows you to specify the name of the file where the thread stack traces will be written. The file will be created (world writable) if it doesn't exist, and the new information will be appended to the end of the file. The filename is passed into the VM via the -Xstacktracefile argument.

For example:

adb shell setprop dalvik.vm.stack-trace-file /tmp/stack-traces.txt

If the property is not defined, the VM will write the stack traces to the Android log when the signal arrives.

DEX File Checksums

For performance reasons, the checksum on "optimized" DEX files is ignored. This is usually safe, because the files are generated on the device, and have access permissions that prevent modification.

If the storage on a device becomes unreliable, however, data corruption can occur. This usually manifests itself as a repeatable virtual machine crash. To speed diagnosis of such failures, the VM provides the -Xcheckdexsum argument. When set, the checksums on all DEX files are verified before the contents are used.

The application framework will provide this argument during VM creation if the dalvik.vm.check-dex-sum property is enabled.

To enable extended DEX checksum verification:

adb shell setprop dalvik.vm.check-dex-sum true

Incorrect checksums will prevent the DEX data from being used, and will cause errors to be written to the log file. If a device has a history of problems it may be useful to add the property to /data/local.prop.

Note also that the dexdump tool always verifies DEX checksums, and can be used to check for corruption in a large set of files.

Copyright © 2008 The Android Open Source Project

## Process IDs of Android applications

> Another issue is how to force the interpreter to run the "C" version.

> > What I do is:

> > - start the emulator

> > - start adb shell

> > - setprop dalvik.vm.execution-mode int:portable

> > - exit shell

> > - restart emulator

Restarting the emulator will reset all of the system properties.

What you'd like to do is "stop ; setprop X Y ; start" to restart the

Android app framework. In recent versions of the emulator this may

simply cause the system to hang.

What you can do instead is put the properties into /data/local.prop,

which is read as the system is booting. For example:

% adb shell "echo dalvik.vm.execution-mode = int:portable >> /data/

local.prop"

Then, when you restart the emulator, the property will be set and you

should get the desired behavior. To undo the change, remove /data/

local.prop and restart the emulator.

See also "Controlling the Embedded VM" in dalvik/docs:

<http://android.git.kernel.org/?p=platform/dalvik.git;a=blob_plain;f=docs/embedded-vm-control.html;hb=HEAD>

A simpler way would be to start the emulator with one or more -boot-property <name>=<value> options.

# Re: [android-porting] Connecting a DSLR camera via PTP on Android

PTP： libptp (Picture Transfer Protocol lib)

<http://sourceforge.net/projects/libptp/files/>

PTP (Picture

Transfer Protocol) or MTP (Media Transfer Protocol) on Android

# [android-porting] Increase the Sampling Rate of Accelerometer

My current application uses SensorManager of Java API to get

accelerometer data at a sampling rate of 5Hz. However I need to

increase that rate. Is it possible for me to customize sensor library

which android uses to increase this amount? Or dou you have any more

suggestions?

each application can create a sensor listener and configure the frequency (SENSOR\_DELAY\_UI,SENSOR\_DELAY\_NORMAL,SENSOR\_DELAY\_GAME,SENSOR\_DELAY\_FASTEST)

you can refer how froyo videoplayer app uses this

http://git.omapzoom.org/?p=platform/packages/apps/Gallery3D.git;a=blob;f=src/com/cooliris/media/RenderView.java;hb=refs/heads/p-froyo#l561

have u already tried this?

# Re: [android-porting] how to set the SeekBar more narrow?

seekbar of froyo video player??

GUI and video control of video players lies in packages/apps/Gallery3D/src/com/cooliris/media

seek bar and update happens at <http://git.omapzoom.org/?p=platform/packages/apps/Gallery3D.git;a=blob;f=src/com/cooliris/media/TimeBar.java#l347>

# Re: StageFright vs. OpenCore

Found the answer:

On Froyo - Local files and files streamed over HTTP will be handled by

Stagefright

- RTSP will be handled OpenCore

On GingerBread, it will be all StageFright.

# your app's Android.mk file should look like this:

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_PACKAGE\_NAME := SyncClient

LOCAL\_CERTIFICATE := platform

LOCAL\_SRC\_FILES := $(call all-java-files-under, src)

LOCAL\_MODULE\_TAGS := user eng

LOCAL\_STATIC\_JAVA\_LIBRARIES := common syncml client pim

#LOCAL\_STATIC\_JAVA\_LIBRARIES := hirr4

include $(BUILD\_PACKAGE)

include $(CLEAR\_VARS)

#LOCAL\_PREBUILT\_STATIC\_JAVA\_LIBRARIES := hirr7:\*.jar

LOCAL\_PREBUILT\_STATIC\_JAVA\_LIBRARIES := common:libs/funambol-android-common.jar syncml:libs/funambol-android-syncml.jar client:libs/funambol-se-client.jar pim:libs/funambol-se-pim.jar

include $(BUILD\_MULTI\_PREBUILT)

Default buffer size used in BufferedInputStream constructor. It would be better to be explicit if an 8k buffer is required.

# Re: IPC, one-way and request-response transactions

On Thu, Dec 16, 2010 at 12:59 PM, Laszlo <benedekl@yahoo.com> wrote:

Is there a reason why one-way and request-response (asynchronous and

synchronous) calls cannot coexist in the same aidl interface ?

The oneway keyword is defined per interface and not per method.

It can also be used per method.

Mixing one-way with synchronous calls is extremely dangerous, as it can cause calls into the interface to be handled out of the expected order.

By looking at the generated code, the only differences are the reply

and the android.os.IBinder.FLAG\_ONEWAY flag, both being handled at

method level.

Um. Sure, the stub code only has that difference. That little flag causes a \*tremendous\* difference in how the IPC is dispatched, though.

(1) Please read the documentation in the SDK on process lifecycle, which will answer your last question.

(2) ActivityManagerService has the implementation for dispatching broadcasts.

(3) Zygote has nothing special to do about broadcasts.

(4) There is nothing special about the BOOT\_COMPLETED broadcast.

# System setProperty and getProperty

Your app cannot use SystemProperties.put() unless it has one of the

few specific UID. You can see this in system/core/init/

property\_service.c:

struct {

const char \*prefix; unsigned int uid; unsigned int gid;

} property\_perms[] = {

...

{ "hw.", AID\_SYSTEM, 0 },

{ "sys.", AID\_SYSTEM, 0 },

{ "service.", AID\_SYSTEM, 0 },

{ "dhcp.", AID\_DHCP, 0 },

{ "vpn.", AID\_SYSTEM, 0 },

{ "vpn.", AID\_VPN, 0 },

{ "debug.", AID\_SHELL, 0 },

{ "log.", AID\_SHELL, 0 },

....

{ NULL, 0, 0 }

};

If you're building your own platform, there is probably a painful way

to modify property\_service.c to do what you want, but I would not

recommend it.

There is actually a little known method in the Andorid SDK that allows

any APK to create a world readable file (it can be read by any APKs).

So if you have one "main" APK that creates some shared settings, and

other APKs need to read these settings, you can try this:

SharedPreferences pref = mActivity.getSharedPreferences("foo",

Context.MODE\_WORLD\_READABLE);

SharedPreferences.Editor editor = pref.edit();

editor.putInt("junk", 0);

editor.commit();

File world\_readable\_file = mActivity.getFilesDir().getParent() + "/

shared\_prefs/foo.xml";

FileOutputStream fout = new FileOutputStream(world\_readable\_file);

fout.write(....)

Other APKs can simply read this file using

FileInputStream fin = new FileInputStream("/data/data/

com.foobar.main/shared\_prefs/foo.xml");

Hope this helps

Where in the code you find the limitation to 247 ? Somewhee in

property\_service.c ?

Yes, it's defined as PA\_COUNT\_MAX in property\_service.c.

# java.net.ConnectException: localhost/127.0.0.1:80 - Connection refused

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): java.net.ConnectException: localhost/127.0.0.1:80 - Connection refused

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at org.apache.harmony.luni.net.PlainSocketImpl.connect(PlainSocketImpl.java:254)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at org.apache.harmony.luni.net.PlainSocketImpl.connect(PlainSocketImpl.java:533)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at java.net.Socket.connect(Socket.java:1055)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at org.apache.harmony.luni.internal.net.www.protocol.http.HttpConnection.<init>(HttpConnection.java:62)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at org.apache.harmony.luni.internal.net.www.protocol.http.HttpConnectionPool.get(HttpConnectionPool.java:88)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at org.apache.harmony.luni.internal.net.www.protocol.http.HttpURLConnectionImpl.getHTTPConnection(HttpURLConnectionImpl.java:927)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at org.apache.harmony.luni.internal.net.www.protocol.http.HttpURLConnectionImpl.connect(HttpURLConnectionImpl.java:909)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at irdc.EX08\_14.EX08\_14.getDataSource(EX08\_14.java:115)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at irdc.EX08\_14.EX08\_14.access$7(EX08\_14.java:103)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at irdc.EX08\_14.EX08\_14$3.run(EX08\_14.java:86)

12-29 14:33:29.432: ERROR/DOWNLOADAPK(335): at java.lang.Thread.run(Thread.java:1096)

原因很简单，我在主机建立了Aache web服务，在浏览器上使用<http://localhost/android/EX04_14.apk>可以下载文件，但在Android模拟器为什么就不行？因为Android自身就是一个系统，localhost不再是我的开发主机，而就是Android系统本身。

12-29 16:03:45.197: ERROR/DOWNLOADAPK(287): http://192.168.0.143/android/eoeMarket.apk

12-29 16:03:45.197: ERROR/DOWNLOADAPK(287): java.io.FileNotFoundException: http://192.168.0.143/android/eoeMarket.apk

12-29 16:03:45.197: ERROR/DOWNLOADAPK(287): at org.apache.harmony.luni.internal.net.www.protocol.http.HttpURLConnectionImpl.getInputStream(HttpURLConnectionImpl.java:1162)

12-29 16:03:45.197: ERROR/DOWNLOADAPK(287): at irdc.EX08\_14.EX08\_14.getDataSource(EX08\_14.java:117)

12-29 16:03:45.197: ERROR/DOWNLOADAPK(287): at irdc.EX08\_14.EX08\_14.access$7(EX08\_14.java:103)

12-29 16:03:45.197: ERROR/DOWNLOADAPK(287): at irdc.EX08\_14.EX08\_14$3.run(EX08\_14.java:86)

12-29 16:03:45.197: ERROR/DOWNLOADAPK(287): at java.lang.Thread.run(Thread.java:1096)

Forbidden

You don't have permission to access /android/eoeMarket.apk on this server.

文件权限不足，增加权限。

# apache2

apache2不能启动的解决办法(提示could not bind to address 0.0.0.0:80)

net$ sudo service apache2

[sudo] password for leo:

\* Usage: /etc/init.d/apache2 {start|stop|restart|reload|force-reload|start-htcacheclean|stop-htcacheclean|status}

net$ sudo service apache2 status

Apache is NOT running.

net$ sudo /etc/init.d/apache2 start

\* Starting web server apache2 apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName

(98)Address already in use: make\_sock: could not bind to address 0.0.0.0:80

no listening sockets available, shutting down

Unable to open logs

[fail]

net$ sudo /etc/init.d/nginx status

\* nginx is running

原来两个http服务器都在抢80端口。

网友碰到的情况：

在ubuntu中用apt-get装好apache2后，用apache2 -k start 等命令都能正常启动、停止服务。

后来把/usr/share/doc/apache2.2-common/examples/apache2/original/httpd.conf.gz 文件解压，拷贝httpd.conf到 /etc/apache2/目录下，进行修改后再启动apache2，却再也无法启动，提示：

(98)Address already in use: make\_sock: could not bind to address 0.0.0.0:80

于是google，网上基本都是说80端口被其他进程占用了，但是无论是用 #ps auxw 还是 #netstat -ln 都查不出哪个进程占用了80端口，百思不得其解

后来在检查 acache2.conf 文件时发现里面有句话： include /etc/apache2/ports.conf

打开一看，果然，第一行就是 listen 80

注释掉，启动 apache2 OK ！！！

原因：httpd.conf和ports.conf中都有listen 80，而这两个文件在apache2启动时都会被加载，原来是apache2自身服务开启时自己占用了自己的端口 OK！！！

net$ java Reverse http://127.0.0.1/cgi/ "o my good! xusl done."

Exception in thread "main" java.io.IOException: Server returned HTTP response code: 403 for URL: http://127.0.0.1/cgi/

at sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1441)

at Reverse.main(Reverse.java:25)

/var/log/apache2/error.log

[Wed Jan 05 11:02:02 2011] [error] [client 127.0.0.1] attempt to invoke directory as script: /var/www/cgi/

net$ java Reverse http://127.0.0.1/cgi/backwards "o my good! xusl done."

o my good! xusl done. reversed is: .enod lsux !doog ym o

400 无法解析此请求。

401.1 未经授权：访问由于凭据无效被拒绝。

401.2 未经授权: 访问由于服务器配置倾向使用替代身份验证方法而被拒绝。

401.3 未经授权：访问由于 ACL 对所请求资源的设置被拒绝。

401.4 未经授权：Web 服务器上安装的筛选器授权失败。

401.5 未经授权：ISAPI/CGI 应用程序授权失败。

401.7 未经授权：由于 Web 服务器上的 URL 授权策略而拒绝访问。

403 禁止访问：访问被拒绝。

403.1 禁止访问：执行访问被拒绝。

403.2 禁止访问：读取访问被拒绝。

403.3 禁止访问：写入访问被拒绝。

403.4 禁止访问：需要使用 SSL 查看该资源。

403.5 禁止访问：需要使用 SSL 128 查看该资源。

403.6 禁止访问：客户端的 IP 地址被拒绝。

403.7 禁止访问：需要 SSL 客户端证书。

403.8 禁止访问：客户端的 DNS 名称被拒绝。

403.9 禁止访问：太多客户端试图连接到 Web 服务器。

403.10 禁止访问：Web 服务器配置为拒绝执行访问。

403.11 禁止访问：密码已更改。

403.12 禁止访问：服务器证书映射器拒绝了客户端证书访问。

403.13 禁止访问：客户端证书已在 Web 服务器上吊销。

403.14 禁止访问：在 Web 服务器上已拒绝目录列表。

403.15 禁止访问：Web 服务器已超过客户端访问许可证限制。

403.16 禁止访问：客户端证书格式错误或未被 Web 服务器信任。

403.17 禁止访问：客户端证书已经到期或者尚未生效。

403.18 禁止访问：无法在当前应用程序池中执行请求的 URL。

403.19 禁止访问：无法在该应用程序池中为客户端执行 CGI。

403.20 禁止访问：Passport 登录失败。

404 找不到文件或目录。

404.1 文件或目录未找到：网站无法在所请求的端口访问。

注意 404.1 错误只会出现在具有多个 IP 地址的计算机上。如果在特定 IP 地址/端口组合上收到客户端请求，而且没有将 IP 地址配置为在该特定的端口上侦听，则 IIS 返回 404.1 HTTP 错误。例如，如果一台计算机有两个 IP 地址，而只将其中一个 IP 地址配置为在端口 80 上侦听，则另一个 IP 地址从端口 80 收到的任何请求都将导致 IIS 返回 404.1 错误。只应在此服务级别设置该错误，因为只有当服务器上使用多个 IP 地址时才会将它返回给客户端。

$telnet localhost 80

Trying ::1...

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

GET http://localhost/ HTTP/1.0 <--------------输入此内容后，按两次回车

HTTP/1.1 200 OK

Date: Wed, 05 Jan 2011 03:54:29 GMT

Server: Apache/2.2.14 (Ubuntu)

Last-Modified: Wed, 29 Dec 2010 03:50:34 GMT

ETag: "10-b1-4988478a80f06"

Accept-Ranges: bytes

Content-Length: 177

Vary: Accept-Encoding

Connection: close

Content-Type: text/html

<html><body><h1>It works!</h1>

<p>This is the default web page for this server.</p>

<p>The web server software is running but no content has been added, yet.</p>

</body></html>

Connection closed by foreign host.