环境

Linux 版本：Ubuntu 11.04 （可由 10.10 的版本进行升级）64 位系统
GCC 版本：gcc version 4.5.2
Java 版本：java version "1.6.0_26"

下载 android 源码前注意：

1、保证 Ubuntu 系统中的容量在 80G 左右，以保证足够的空间来存放 android 源码以及编译后的相关文件。

2、保证 Ubuntu 系统进行 Internet 访问。

联网方法：采用拨号进行连接。相关操作步骤如下所示：

1、虚拟机→设置→硬件→网络适配器→网络连接→桥接

2、启动 Ubuntu 系统，打开终端（在普通用户下），输入相关命令如下：

$ pppoeconf　//打开后输入上网账号跟密码，确认保存

$ sudo pon dsl-provider　//上网连接命令

经过这两个步骤后就可以进行上网了。

Android 源码编译所依赖的 **tools**

01.$ sudo apt-get update
02.$ sudo apt-get -y install git-core
03.$ sudo apt-get -y install gnupg
04.$ sudo apt-get -y install sun-java6-jdk flex
05.$ sudo apt-get -y install bison
06.$ sudo apt-get -y install gperf
07.$ sudo apt-get -y install libsdl-dev
08.$ sudo apt-get -y install libesd0-dev
09.$ sudo apt-get -y install libwxgtk2.6-dev
10.$ sudo apt-get -y install build-essential
11.$ sudo apt-get -y install zip
12.$ sudo apt-get -y install curl
13.$ sudo apt-get -y install libncurses5-dev
14.$ sudo apt-get -y install zlib1g-dev
15.$ sudo apt-get -y install valgrind

注意：（如果是 32bit 的系统的话，则要更改几个 Android.mk 文件）
01./external/clearsilver/cgi/Android.mk
02./external/clearsilver/java-jni/Android.mk
03./external/clearsilver/util/Android.mk
04./external/clearsilver/cs/Android.mk
用 gedit 打开，修改 m64 为 m32 即可

另外
将 build/core/main.mk 中的 ifneq (64,$(findstring 64,$(build_arch)))修改为：
ifneq (i686,$(findstring i686,$(build_arch)))

对于 32 位系统所出现的问题，解决方法：

Error1：

make: ***

[out/host/linux-x86/obj/STATIC_LIBRARIES/libutils_intermediates/RefBase.o] error 1

在 terminal 中输入

$ gedit frameworks/base/libs/utils/Android.mk

将 LOCAL_CFLAGS += -DLIBUTILS_NATIVE=1 $(TOOL_CFLAGS)修改为：

LOCAL_CFLAGS += -DLIBUTILS_NATIVE=1 $(TOOL_CFLAGS) –fpermissive

Error2：

make: *** [out/host/linux-x86/obj/EXECUTABLES/obbtool_intermediates/Main.o] error 1

此处编译错误是由于 ubuntu 11.10 采用了 GCC4.6.1 导致的

修改源码目录下/build/core/combo/HOST_linux-x86.mk

并将以下语句：

HOST_GLOBAL_CFLAGS

+= -D_FORTIFY_SOURCE=0

修改为：

HOST_GLOBAL_CFLAGS

+= -U_FORTIFY_SOURCE -D_FORTIFY_SOURCE=0

Android 源码下载

1、安装 **repo**

安装过程步骤如下所示：

Repo 下载安装

1、$ mkdir ~/bin //在 home 目录下创建 bin 文件夹

2、$ PATH=~/bin:$PATH //环境变量设置

3、$ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo //repo 下载

4、$ chmod a+x ~/bin/repo //权限设置，保证 repo 可执行

Repo client 初始化

1、$ mkdir Android //创建 Android 目录，用于存放下载的 android 源码

2、$ cd Android //进入到 Android 目录

3、$ repo init -u https://android.googlesource.com/platform/manifest //指定 URL 清单,指定包含在 Android 源代码的各种存储库将被安置在工作目录中

当然也可以指定 android 版本，使用如下命令：

$ repo init -u https://android.googlesource.com/platform/manifest -b android-4.0.1_r1

Android 源码文件获取

$ repo sync

注意：在 Android 源码下载过程中凡出现问题，可以进入 Android 源码下载的相关官方网站 http://source.android.com/source/downloading.html 进行查看。

Android 源码编译

环境初始化

1、$ source build/envsetup.sh

OR

$. build/envsetup.sh

2、$ lunch full-eng　　//选择编译目标

Code 编译

$ make -j4　//其中 j4 表示同时启动 4 个线程任务进行编译，可以直接使用 make，这样就是最大程度使用线程数来进行编译

总结：到这里为止，android 源码完成了编译过程，编译后，会在源码目录下的/out/target/product/generic/有三个镜像文件生成，分别是 system.img、userdata.img 以及 ramdisk.img 。

注意：1、如果编译过程没有出问题，但是在/out/target/product/generic/下 3 个镜像文件不全，可以再次进行一次命令 make，就可以解决问题了。

2、如果是编译过程中出现错误，一般导致的原因是因为缺少了相关编译环境，可以根据提示进行安装，安装后，重新进行编译。

Windows XP 下使用 Android 模拟器运行所编译好的文件。

原理：将 Android 模拟器下所加载的 system.img 替换成编译后的 system.img 即可。

1、将编译好的 system.img 从 Ubuntu 中拷贝出来

方法：1）先拷贝到 U 盘或者移动硬盘中，再从 U 盘或移动硬盘拷贝到 Windows XP 硬盘下；

2）先进行 Windows XP 跟 Ubuntu 文件夹共享，这样就可以将 system.img 拷贝到共享文件夹中；

3）传 FTP，从 FTP 上进行下载；

4）在 Ubuntu 中上网将 system.img 作为附件进行邮件发送，然后从 Windows XP 下进行下载。

2、将拷贝出来的 system.img 文件放置到 Android 模拟器下的对于文件夹中，如果 Android 是 4.0 以上的版本的话，一般是在 system-imges 文件夹中。C:\Program Files\Android\android-sdk\system-images\android-15\armeabi-v7a

3、启动模拟器就可以运行了。

注意：创建 Android 虚拟机的时候，Android 版本应该跟 Ubuntu 下编译好的 Android 版本要保持一致，否则有可能替换 system.img 后，Android 系统无法正常启动，或者无法进入 Android 启动界面。

Android 源码修改并进行编译

下面以 2 个实例来进行说明，实例 1 目的是实现 Android 系统永不休眠，实例 2 是实现 Android 系统启动后，不自动进入锁屏界面。

实例 **1**：

实现 Android 系统永不休眠其实很简单，只需要对源码下的配置文件 XML 下的参数进行修改即可。步骤如下：

1、$ cd frameworks/base/packages/SettingsProvider/res/values　//在下载的 Android 源码目录下输入此命令，进入到 values 目录下

2、$ gedit defaults.xml　//使用文本编辑器打开 defaults.xml 文件

3、将 <integer name="def_screen_off_timeout">60000</integer> 改为 <integer name="def_screen_off_timeout">-1</integer>保存后退出 //其中 60000 单位是 ms，表示 60s，就是说 60s 屏幕会进行休眠状态，改为-1 后，不再进入休眠

4、$ cd ../../../../../../　//返回到源码目录下

5、$ source ./build/envsetup.sh　//环境初始化

6、$ mmm frameworks/base/packages/SettingsProvider/　//对 SettingsProvider 模块进行编译

7、$ make snod //重新打包 system.img

到这里，模拟编译就完成了，查看效果的话，直接将 Android 模拟器下的 system.img 替换掉启动运行即可。

注意：mmm 命令可以对模块进行编译，后面必须接上模块所在目录，查看某目录是否为一模块，就看当前目录下是否有 Android.mk 文件，如果有的话，就是模块。除了可以使用 mmm 进行编译的话，还可以使用 mm，但是 mm 是在当前目录下实行编译的，也就是模块目录下使用此命令。

实例 **2**：

实现 Android 启动后不进入锁屏状态。修改 KeyguardViewMediator.java 文件下的变量，将其赋值改为 false 即可。步骤如下：

1、$ cd frameworks/base/policy/src/com/android/internal/policy/impl/　//在下载的 Android 源码目录下输入此命令，进入到 impl 目录下

2、$ gedit KeyguardViewMediator.java　//是用文本编辑器打开 java 文件

3、将 private boolean mExternallyEnabled = true;代码中的 true 改为 false，保存退出

4、$ cd ../../../../../../ ../../../　//返回到源码目录下

5、$ source ./build/envsetup.sh　//环境初始化

6、$ mmm frameworks/base/policy/　//对 policy 模块进行编译

7、$ make snod //重新打包 system.img

注意：如果在进行模块代码编译了，报出少了一个 odex 后缀名的文件的错误时，总结了下，导致报错的原因可能是 Android 版本的差异，不过没事，即使模块编译通不过，还是可以对整个源码进行编译的（直接在源码目录下执行 make 命令即可），只是时间会相对较长，同样会将修改的模块整合到 system.img 文件中。

Android 4.0 下防止用户对安装程序进行卸载

打开源码目录找到 packages/apps/PackageInstaller/src/com/android/packageinstaller/下的两个文件：

1、UninstallAppProgress.Java　//程序卸载过程

2、UninstallerActivity.java　//程序卸载界面

方法是在 UninstallerActivity.java 对应位置加上一段代码即可：

```
@Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
```

```
// Get intent information.
// We expect an intent with URI of the form package://<packageName>#<className>
// className is optional; if specified, it is the activity the user chose to uninstall
final Intent intent = getIntent();
Uri packageURI = intent.getData();
String packageName = packageURI.getEncodedSchemeSpecificPart();
if(packageName == null) {
    Log.e(TAG, "Invalid package name:" + packageName);
    showDialog(DLG_APP_NOT_FOUND);
    return;
}
if(packageName.equals("cn.android.managerapp")){
/*AlertDialog.Builder builder= new AlertDialog.Builder(this);
//builder.setIcon(android.R.drawable.ic_dialog_info);
builder.setTitle("Message");
builder.setMessage("Cannot be uninstall!");
builder.setPositiveButton("确认", null);
//builder.create().show();
builder.show();
    //showDialog(DLG_APP_NOT_FOUND);
//return;
finish();*/
new AlertDialog.Builder(this)
.setTitle("游戏控制")
.setMessage("对不起，此程序不允许卸载！")
.setPositiveButton("确定",
new DialogInterface.OnClickListener(){
public void onClick(DialogInterface dialoginterface, int i){
//这里设置点击后执行
finish();
}
}
)
.show();
}
mPm = getPackageManager();
boolean errFlag = false;
```

加上红色部分的代码，就可以对特定的安装程序进行控制了。其中 cn.android.managerapp 是指安装程序的进程名，效果就是弹出提示框，提醒用户程序不允许卸载。

Android 4.0 下防止用户对安装程序进行强行停止

打开源码目录，到 frameworks/base/services/java/com/android/server/am/ 目录下找到 ActivityManagerService.java 文件，并打开它。

可以对代码中两个地方进行控制，都可以达到强行停止的目的：

1、 在 forceStopPackage 函数中，加上一段代码

```
2、 public void forceStopPackage(final String packageName) {
3、             if (checkCallingPermission(android.Manifest.permission.FORCE_STOP_PACKAGES)
4、                     != PackageManager.PERMISSION_GRANTED) {
5、                 String msg = "Permission Denial: forceStopPackage() from pid="
6、                         + Binder.getCallingPid()
7、                         + ", uid=" + Binder.getCallingUid()
8、                         + " requires " + android.Manifest.permission.FORCE_STOP_PACKAGES;
9、                 Slog.w(TAG, msg);
10、                throw new SecurityException(msg);
11、            }
12、            final int userId = UserId.getCallingUserId();
13、            long callingId = Binder.clearCallingIdentity();
14、            try {
15、                IPackageManager pm = AppGlobals.getPackageManager();
16、                int pkgUid = -1;
17、                synchronized(this) {
18、                    try {
19、                        pkgUid = pm.getPackageUid(packageName, userId);
20、                    } catch (RemoteException e) {
21、                    }
22、                    if (packageName == -1) {
23、                        Slog.w(TAG, "Invalid packageName: " + packageName);
24、                        return;
25、                    }
26、                    //添加判断代码
27、                    if(packageName.equals("cn.android.managerapp")){
28、                        return;
29、                    }
30、                    //
31、                    forceStopPackageLocked(packageName, pkgUid);
32、                    try {
33、                        pm.setPackageStoppedState(packageName, true, userId);
34、                    } catch (RemoteException e) {
35、                    } catch (IllegalArgumentException e) {
36、                        Slog.w(TAG, "Failed trying to unstop package "
37、                                + packageName + ": " + e);
38、                    }
39、                }
40、            } finally {
41、                Binder.restoreCallingIdentity(callingId);
```

```
42、                }
43、            }
```

方法二：这是在网上看到的方法，尝试了下，编译通过了，但是在模拟器上面运行的时候出问题

2、修改 clearApplicationUserData 函数

```
44、    public boolean clearApplicationUserData(final String packageName,
45、                    final IPackageDataObserver observer, final int userId) {
46、                enforceNotIsolatedCaller("clearApplicationUserData");
47、                int uid = Binder.getCallingUid();
48、                int pid = Binder.getCallingPid();
49、                long callingId = Binder.clearCallingIdentity();
50、                try {
51、                    IPackageManager pm = AppGlobals.getPackageManager();
52、                    int pkgUid = -1;
53、                    synchronized(this) {
54、                        try {
55、                            pkgUid = pm.getPackageUid(packageName, userId);
56、                        } catch (RemoteException e) {
57、                        }
58、                        if (pkgUid == -1) {
59、                            Slog.w(TAG, "Invalid packageName:" + packageName);
60、                            return false;
61、                        }
62、                        if(packageName.equals("cn.android.managerapp")){
63、                        AlertDialog.Builder builder = new AlertDialog.Builder(mContent);
64、                        builder.setTitle();
65、                        builder.setMessage(msg);
66、                        builder.setPositiveButton("OK",new
    DialogInterface.OnClickListener(){
67、                        public void onClick(DialogInterface dialoginterface, int i){
68、                        //Click event
69、                        }
70、                        }
71、                        );
72、                        builder.create().show();
73、                        return false;
74、                        }
75、                        if (uid == pkgUid || checkComponentPermission(
76、
    android.Manifest.permission.CLEAR_APP_USER_DATA,
77、                                pid, uid, -1, true)
78、                                == PackageManager.PERMISSION_GRANTED) {
79、                            forceStopPackageLocked(packageName, pkgUid);
```

```
80、                        } else {
81、                            throw new SecurityException(pid+" does not have
    permission:"+
82、
    android.Manifest.permission.CLEAR_APP_USER_DATA+" to clear data" +
83、                                                    "for process:"+packageName);
84、                        }
85、                    }
86、
87、                    try {
88、                        //clear application user data
89、                        pm.clearApplicationUserData(packageName, observer, userId);
90、                        Intent intent = new
    Intent(Intent.ACTION_PACKAGE_DATA_CLEARED,
91、                                Uri.fromParts("package", packageName, null));
92、                        intent.putExtra(Intent.EXTRA_UID, pkgUid);
93、                        broadcastIntentInPackage("android", Process.SYSTEM_UID,
    intent,
94、                                null, null, 0, null, null, null, false, false, userId);
95、                    } catch (RemoteException e) {
96、                    }
97、                } finally {
98、                    Binder.restoreCallingIdentity(callingId);
99、                }
100、               return true;
101、           }
```


Launcher 桌面定制

方法一：将系统的 Launcher 进行删除，使用自己编写的 Launcher。

实现步骤：

1、 编写自己的 Launcher。例如：建立 Jhome 工程，修改工程下的 AndroidManifest.xml 文件，如下所示：

```
// 主屏 activity 属性设置
//  Android activity 属性设置大全 http://apps.hi.baidu.com/share/detail/44749420
<activity
        android:name="com.launcher.jhome.Laucher"
        android:launchMode="singleTask"
        android:clearTaskOnLaunch="true"
        android:stateNotNeeded="true"
        android:windowSoftInputMode="adjustPan"
        android:screenOrientation="nosensor"
        android:label="@string/title_activity_laucher" >
        <intent-filter>
```

```
                <action android:name="android.intent.action.MAIN" />
                //以下两条代码一定要加上，表示 Home Screen
                 <category android:name="android.intent.category.LAUNCHER" />
                <category android:name="android.intent.category.HOME" />
                 <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
         //应用程序列表 activity
        <activity
            android:name="com.launcher.listview.ListView"
            android:finishOnCloseSystemDialogs="true"
            android:process=":ListView"
            android:label="@string/title_activity_laucher">
             <intent-filter>
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
```

2、将写好的 Launcher 代码进行编译生成 APK，将 APK 拷贝到 Android 源码下的 /out/target/product/generic/system/app/下；

3、修改 Android 下的 Launcher 配置文件 AndroidManifest.xml，该文件在 Android 源码下 /packages/apps/Launcher2(或 Launcher，根据版本不同有所差异)，打开该目录下的 AndroidManifest.xml 文件，将以下两段代码删除或者注释掉即可。

```
<category android:name="android.intent.category.HOME" />
<category android:name="android.intent.category.DEFAULT" />
```

4、编译 Android 源码，编译好后，就可以启动模拟器进行效果查看了。

方法二：实现唯一定制的 Launcher，缺点：修改大，容易出问题。

实现步骤：

1、在 Intent.java(frameworks/base/core/java/android/content/Intent.java)中添加两行代码

```
1. @SdkConstant(SdkConstantType.INTENT_CATEGORY)
2. public static final String CATEGORY_HOME_FIRST = “android.intent.category.HOME_FIRST”;
```

2、在 frameworks/base/services/java/com/android/server/am/ActivityManagerService.java 中修改

```
1. //intent.addCategory(Intent.CATEGORY_HOME); 改成
     intent.addCategory(Intent.CATEGORY_HOME_FIRST);
2.将该文件中出现的 Intent.CATEGORY_HOME 换成 Intent.CATEGORY_HOME_FIRST
```

3、在 frameworks/base/policy/src/com/android/internal/policy/impl/ PhoneWindowManager.java 中修改

```
1.//mHomeIntent.addCategory(Intent.CATEGORY_HOME);
2.mHomeIntent.addCategory(Intent.CATEGORY_HOME_FIRST);
```

4、在 frameworks/base/services/java/com/android/server/am/ ActivityRecord.java 中修改

1. //if (Intent.ACTION_MAIN.equals(_intent.getAction()) &&
_intent.hasCategory(Intent.CATEGORY_HOME)
2. if (Intent.ACTION_MAIN.equals(_intent.getAction()) &&
_intent.hasCategory(Intent.CATEGORY_HOME_FIRST)

5、在
frameworks/base/policy/src/com/android/internal/policy/impl/
RecentApplicationsDialog.java 中修改
1. // ActivityInfo homeInfo =
//new
Intent(Intent.ACTION_MAIN).addCategory(Intent.CATEGORY_HOME)
//.resolveActivityInfo(pm, 0);
2. ActivityInfo homeInfo =
new
Intent(Intent.ACTION_MAIN).addCategory(Intent.CATEGORY_HOME_FIRST)
.resolveActivityInfo(pm, 0);

6、在 Android 源码下可以使用 grep -R ' CATEGORY_HOME ' frameworks 和命令 grep -R ' CATEGORY_HOME ' packages 找出在 frameworks 和 packages 目录下所有文件中出现 CATEGORY_HOME 的文件路径罗列出，然后对每个文件中的 CATEGORY_HOME 和 category_HOME 对应改成 CATEGORY_HOME_FIRST 和 category_HOME_FIRST；

主要有以下这些文件：

// 改 frameworks/base/docs/html/guide/components/intents-filters.jd: <td>{@code CATEGORY_HOME}

//改 frameworks/base/api/13.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/12.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/4.xml:<field name="CATEGORY_HOME"

// 改 frameworks/base/api/14.txt: field public static final java.lang.String CATEGORY_HOME = "android.intent.category.HOME";

//改 frameworks/base/api/8.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/6.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/11.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/5.xml:<field name="CATEGORY_HOME"

// 改 frameworks/base/api/16.txt: field public static final java.lang.String CATEGORY_HOME = "android.intent.category.HOME";

// 改 frameworks/base/api/15.txt: field public static final java.lang.String CATEGORY_HOME = "android.intent.category.HOME";

//改 frameworks/base/api/7.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/3.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/9.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/2.xml:<field name="CATEGORY_HOME"

//改 frameworks/base/api/10.xml:<field name="CATEGORY_HOME"

// 改 frameworks/base/api/current.txt: field public static final java.lang.String CATEGORY_HOME = "android.intent.category.HOME";

// 改 frameworks/base/api/current.txt: field public static final java.lang.String

CATEGORY_HOME_FIRST = "android.intent.category.HOME_FIRST";
//改 frameworks/base/api/1.xml:<field name="CATEGORY_HOME"
// 改 frameworks/base/policy/src/com/android/internal/policy/impl/PhoneWindowManager.java:
//mHomeIntent.addCategory(Intent.CATEGORY_HOME);
// 改 frameworks/base/policy/src/com/android/internal/policy/impl/PhoneWindowManager.java:
mHomeIntent.addCategory(Intent.CATEGORY_HOME_FIRST);
//改 frameworks/base/policy/src/com/android/internal/policy/impl/RecentApplicationsDialog.java:
//new Intent(Intent.ACTION_MAIN).addCategory(Intent.CATEGORY_HOME)
//改 frameworks/base/policy/src/com/android/internal/policy/impl/RecentApplicationsDialog.java:
new Intent(Intent.ACTION_MAIN).addCategory(Intent.CATEGORY_HOME_FIRST)
// 改 frameworks/base/services/java/com/android/server/UiModeManagerService.java:
category = Intent.CATEGORY_HOME;
// 改 frameworks/base/services/java/com/android/server/UiModeManagerService.java:
homeIntent = buildHomeIntent(Intent.CATEGORY_HOME);
// 改 frameworks/base/services/java/com/android/server/am/ActivityRecord.java:
_intent.hasCategory(Intent.CATEGORY_HOME) &&
// 改 frameworks/base/services/java/com/android/server/am/ActivityRecord.java:
_intent.hasCategory(Intent.CATEGORY_HOME_FIRST) &&
// 改 frameworks/base/services/java/com/android/server/am/ActivityManagerService.java:
//intent.addCategory(Intent.CATEGORY_HOME);
// 改 frameworks/base/services/java/com/android/server/am/ActivityManagerService.java:
intent.addCategory(Intent.CATEGORY_HOME_FIRST);
// 改 frameworks/base/services/java/com/android/server/am/ActivityManagerService.java:
/*if (intent.getCategories() != null &&
intent.getCategories().contains(Intent.CATEGORY_HOME)) {
// 改 frameworks/base/services/java/com/android/server/am/ActivityManagerService.java:if
(intent.getCategories() != null &&
intent.getCategories().contains(Intent.CATEGORY_HOME_FIRST)) {
// 改
frameworks/base/packages/SystemUI/src/com/android/systemui/recent/RecentTasksLoader.java:
homeInfo = new Intent(Intent.ACTION_MAIN).addCategory(Intent.CATEGORY_HOME)
// 改
frameworks/base/packages/SystemUI/src/com/android/systemui/recent/RecentTasksLoader.java:
.addCategory(Intent.CATEGORY_HOME).resolveActivityInfo(pm, 0);
// 改 frameworks/base/core/java/android/content/Intent.java: * <p>{@link
#ACTION_MAIN} with category {@link #CATEGORY_HOME} --
// 改 frameworks/base/core/java/android/content/Intent.java: * {@link
#CATEGORY_HOME}
// 改 frameworks/base/core/java/android/content/Intent.java: // 添 加
CATEGORY_HOME_FIRST
// 改 frameworks/base/core/java/android/content/Intent.java: public static final String
CATEGORY_HOME_FIRST = "android.intent.category.HOME_FIRST";
// 改 frameworks/base/core/java/android/content/Intent.java: public static final String
CATEGORY_HOME = "android.intent.category.HOME";

// 改 frameworks/base/tests/SmokeTest/tests/src/com/android/smoketest/ProcessErrorsTest.java: mHomeIntent.addCategory(Intent.CATEGORY_HOME);
// 改 frameworks/base/tests/StatusBar/src/com/android/statusbartest/NotificationTestList.java: intent.addCategory(Intent.CATEGORY_HOME);
// 改 packages/apps/Settings/src/com/android/settings/applications/InstalledAppDetails.java: intent.addCategory(Intent.CATEGORY_HOME);
// 改 packages/apps/Phone/src/com/android/phone/OtaUtils.java: intent.addCategory (Intent.CATEGORY_HOME);
// 改 packages/apps/Protips/src/com/android/protips/ProtipWidget.java: .addCategory(Intent.CATEGORY_HOME));

7、将写好的桌面定制 apk 放入到 out/target/product/generic/system/app/目录下
注意：写 Launcher 的 AndroidManifest.xml 文件的 Home Sreen 的 activity 需要这样设置。
<category android:name="android.intent.category.HOME_FIRST" />
<category android:name="android.intent.category.DEFAULT" />
8、将源码中与 Launcher 有关的部分全部删除掉
包括源代码
(packages/apps/Launcher)和 apk(/out/target/product/generic/system/app/Launcher.apk)
8、对修改的模块进行编译，最后再进行 make snod 重新进行 system.img 打包
注意：如果 make 出现以下错误：
1.******************************
2.You have tried to change the API from what has been previously approved.
3.
4.To make these errors go away, you have two choices:
5. 1) You can add "@hide" javadoc comments to the methods, etc. listed
6.in the
7. errors above.
8.
9. 2) You can update current.xml by executing the following commands:
10.
11. p4 edit frameworks/base/api/current.xml
12. make update-api
13.
14. To check in the revised current.xml, you will need OWNERS
15.approval.
16.******************************
如果自己添加了 Android API 那么编译系统会出现此错误， 解决办法：
先 make update-api 然后再 make -j4 直接就可编译成功