# TASK

*<!--* ***todo search crash?*** *-->*

*<item name="android:editTextColor">@\*android:attr/textColorPrimary</item>-->*

<https://blog.csdn.net/jiwancang/article/details/45061883>

service子模块的usb怎么整

<https://blog.csdn.net/hfreeman2008/article/details/52778992>

WindowPhone 主题原理加入

不生效呢

android:theme="@style/Rm500.Theme.Settings"

蓝牙和gps模式，貌似不行嗷

java.lang.IllegalArgumentException: DrawerLayout must be measured with MeasureSpec.EXACTLY.

at android.support.v4.widget.DrawerLayout.onMeasure(DrawerLayout.java:1042)

# SettingsProvider

Content Provider几乎已经成为了Android系统中两个或多个应用(系统应用和普通应用)之间共享数据库的最直接的方法。

在Android源码中，有两个目录是专门存放Content Provider。

一个是/源码目录/frameworks/providers，允许任何Adroid应用访问，包括：ApplicationProvider、CalendarProvider、ContactsProvider、DownloadProvider、MedialProvider、PartnerBookmarksProvider、TelephonyProvider、TvProvider、UserDictionaryProvider；

另一个是/源码目录/frameworks/base/packages，通常只允许系统级别的Android应用读写数据，**普通Android应用只能读数据**。包含：ExternalStorageProvider、SettingsProvider

具体到Setting Content Provider 是所有系统设置的管理者。在M(Android5.0)版本之前，SettingsProvider中系统设置是存储在settings.db[数据库](http://lib.csdn.net/base/mysql)中；但是在L(Android6.0)之后，SettingsProvider中系统设置改为由xml存储在data分区。

Android M (6.0)之后的谷歌对SettingsProvider进行了重构，所以Settings数据库的路径有了变化   
在手机目录/data/system/users/0路径下，以xml的形式存储数据，有settings\_global.xml, settings\_system.xml, settings\_secure.xml   
其中不同用户的数据库放不同的用户路径下，如果没有创建新用户，则在/data/system/users/0下。   
修改数据库中的字段使用命令   
adb shell settings get global captive\_portal\_server   
adb shell settings put global captive\_portal\_server 0

注：这里虽然手机目录中数据库存储是以xml的形式，但是代码中操作时操作的仍然是数据库。系统会先生成Settings.db数据库，然后将其中的内容保存到xml*文件中，最后删除Settings.db数据库(SettingsProvider的功能，后续会讲到)。   
这里为何先生成数据库后又删除，主要是为了兼容之前的版本。   
为什么以xml的形式保存字段：节省资源，简单，安全(xml保存的路径相对于之前的数据库路径比较安全，大概是考虑到多用户吧)*

？？？？？？？？？？？？？？

如Wifi状态，蓝牙是否开启，当前设备所处的模式等等，很多很多类似的全局属性控制的属性值。最重要的是global表，该表包含两个字段：name和value.其中name表示系统信息的key，value表示该系统设置项所对应的选项值。数据库结构如下：



## Android M之前的SettingsProvider

进入**/data/data/com.**[**android**](http://lib.csdn.net/base/android)**.providers.settings/databases**

# cd /data/data/com.android.providers.settings/databases

# sqlite3 settings.db

sqlite3 有可能提示找不到，这是由于在当前版本中sqlite3程序没有被编译到system中，这里我们只需要手动push一个sqlite3就好了。Cs果然没有

在使用sqlite3打开settings.db数据库之后，可以使用.help来查看有关sqlite3的一些帮助信息。这里常用的如下所示：



源码

**frameworks/base/packages/SettingsProvider/src/com/Android/providers/settings/DatabaseHelper.**[**Java**](http://lib.csdn.net/base/java)   
**frameworks/base/packages/SettingsProvider/res/values/defaults.xml**

**如何添加属性：**系统属性的设置，我们可以通过修改源码，编译生成SettingsProvider.apk并安装到系统中，就能生效。

### [自定义Android系统默认设置](http://www.cnblogs.com/l2rf/p/5102230.html)

无法在普通app上通过以下方式使用，即使申请了android.permission.WRITE\_SETTINGS等权限也不行。会报SecurityException。

Settings.Global.*putString*(getContentResolver(),  
 **"myKey"**,**"v1"**);  
String v1 = Settings.Global.*getString*(getContentResolver(),  
 **"myKey"**);

### 修改方法默认值方法1

**adb shell settings**,有关其详细用法，可以参照如下提示：

$ adb shell settings get system ringtone

content://media/internal/audio/media/59

$ adb shell settings set system ringtone "content://media/internal/audio/media/60"

Invalid command: set

usage: settings [--user NUM] get namespace key

settings [--user NUM] put namespace key value

settings [--user NUM] delete namespace key

'namespace' is one of {system, secure, global}, case-insensitive

If '--user NUM' is not given, the operations are performed on the owner user.

$ adb shell settings put system ringtone "content://media/internal/audio/media/60"

$ adb shell settings get system ringtone

content://media/internal/audio/media/60

### 修改方法默认值方法2,标准方法

如果要在系统中添加默认设置，需要如下步骤（以带物理键盘的系统中默认打开软键盘为例）：

1、在defaults.xml中为新加的数据定义一个默认值

<integer name="show\_ime\_with\_hard\_keyboard">1</integer>

2、在DatabaseHelper.java中的loadSecureSettings(SQLiteDatabase db)方法中加入新字段的添加代码

loadStringSetting(stmt, Settings.Secure.SHOW\_IME\_WITH\_HARD\_KEYBOARD,

R.string.show\_ime\_with\_hard\_keyboar);

3、重新编译

## Android L 的SettingsProvider

在Android L版本中，Google团队为了优化系统性能、提高安全性等等一些列需求，重构了SettingsProvider模块，在L版本中，不在使用数据库来存储系统设置，而是通过xml将系统设置存储在了**/data/system/user/** 目录下



有关L版本中SettingsProvider的修改更新方法同之前的相同

## 源码

### manifest

|  |
| --- |
| <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="com.android.providers.settings"  coreApp="true"  android:sharedUserId="android.uid.system">  <application android:allowClearUserData="false"  android:label="@string/app\_label"  android:process="system"  android:backupAgent="SettingsBackupAgent"  android:killAfterRestore="false"  android:icon="@mipmap/ic\_launcher\_settings">    <!-- todo add: android:neverEncrypt="true" -->  <provider android:name="SettingsProvider" android:authorities="settings"  android:multiprocess="false"  android:exported="true"  android:writePermission="android.permission.WRITE\_SETTINGS"  android:singleUser="true"  android:initOrder="100" />  </application>  </manifest> |

通过如上文件内容，我们大概可以得出几个结论：   
1. 拥有system用户权限。标签的coreApp属性是true,且android:sharedUserId=”android.uid.system”意味着SettingProvder与Settings同样拥有system用户的权限，且通过android:process=”system”可以看出，SettingProvider运行在**系统进程中**。   
2. provider标签中android:authorities=”settings”可以知道，访问SettingsProvider的Uri的开头部分一定是“content://settings”;

比如

CONTENT\_URI = Uri.parse("content://settings/global");

3. provider标签中android:writePermission=”android.permission.WRITE\_SETTINGS”可以知道，在执行写操作时需要指定WRITE\_SETTINGS权限。

### Src

在SettingsProvider的src源代码中，共有四个类：   
1. DatabaseHelper类。数据库操作类，实现对setting.db的读写功能，继承自SQLiteOpenHelper类   
2. SettingsBackupAgent类。用于备份和恢复系统设置等功能的类，继承自BackupAgentHelper类   
3. SettingsHelper类。Settings的帮助类   
4. SettingsProvider类。继承自ContentProvider类的最核心的文件   
在这四个类中，最最重要的就是SettingsProvider，其次我关心的可能就是DatabaseHelper类是如何执行数据库语句的。

#### DatabaseHelper

首先，来分析DatabaseHelper类，在DatabaseHelper类中，就是对数据库及表的操作。主要是表的创建，数据库版本更新。onCreate方法中负责创建表，执行表创建语句，然后进行表的初始化，也就是系统的初始设置；onUpgrade方法最长，需要从第一个版本到最新版本一个一个判断，升级。我们看一下具体的初始设置：

|  |
| --- |
| //onCreate中的初始化设置调用  // Load initial volume levels into DB  loadVolumeLevels(db);  // Load inital settings values  loadSettings(db);  //下面是loadSettings方法：  private void loadSettings(SQLiteDatabase db) {  loadSystemSettings(db);  loadSecureSettings(db);  // The global table only exists for the 'owner' user  if (mUserHandle == UserHandle.USER\_OWNER) {  loadGlobalSettings(db);  }  }  //我们选择loadGlobalSettings进行查看，下面分别是设置boolean值，String值，int值  // --- Previously in 'system'  loadBooleanSetting(stmt, Settings.Global.AIRPLANE\_MODE\_ON,  R.bool.def\_airplane\_mode\_on);  loadStringSetting(stmt, Settings.Global.AIRPLANE\_MODE\_RADIOS,  R.string.def\_airplane\_mode\_radios);  loadIntegerSetting(stmt, Settings.Global.WIFI\_SLEEP\_POLICY,  R.integer.def\_wifi\_sleep\_policy); |

通过上面的代码我们看到，最后其实就是引用（或者说是读取）了integer或者bool再或者string中的一些具体的字段，设置到数据库中。我们要想查看这些默认的值，可以到res/values中进行查看。和本地化无关的存放在res/values/default.xml文件中，和本地化有关的（主要指string,涉及语言）则放在相应的语言分包下面；另外，我们还能注意到的是，sqlite不支持boolean值，如果要保存boolean值，是通过integer来保存的。一般1为true,0为false。

#### SettingsProvider

看onCreate（）方法，创建表和各表的索引，然后调用了loadBookmarks(db); loadVolumeLevels(db); loadSettings(db); 三个方法，loadBookmarks(db);是存系统几个常用的应用的intent ( 如，通讯录app，emailApp, 短信app,)，loadVolumeLevels(db); 初始化的是与声音相关的配制， loadSettings(db);方法就是从defaults.xml文件中读取默认值存入相应的表中，验证了之前 的猜测。

loadSettings(db); 里面的三个方法者是从defaults.xml文件中读取默认值存入相应的表中，loadSystemSettings(db)是存入System表中，loadSecureSettings(db)是存入 secure表中。

下面来分析最最核心的SettingsProvider的代码，核心就是insert,delete,update,query四个方法，我们依次看四个方法：

insert方法：insert方法中调用了insertForUser(Uri url, ContentValues initialValues, int desiredUserHandle) 方法。在insert方法中在执行insert方法之前，调用了检查写权限的方法checkWritePermissions，更新db都需要android.Manifest.permission.WRITE\_SECURE\_SETTINGS权限（需要系统签名）和android.Manifest.permission.WRITE\_、SETTINGS。读取则不需要任何权限。

|  |
| --- |
| **private void** checkWritePermissions(SqlArguments args) {  **if** ((TABLE\_SECURE.equals(args.table) || TABLE\_GLOBAL.equals(args.table)) &&  getContext().checkCallingOrSelfPermission(  android.Manifest.permission.WRITE\_SECURE\_SETTINGS) !=  PackageManager.PERMISSION\_GRANTED) {  **throw new** SecurityException(  String.format(**"Permission denial: writing to secure settings requires %1$s"**,  android.Manifest.permission.WRITE\_SECURE\_SETTINGS));  } } |

delete和update方法中也调用了如上的权限检查方法，然后再执行相应的数据库操作方法

query方法：query方法中调用了queryForUser()方法进而执行query查询返回Cursor结果对象。

## settings.db 详解

globl

主意：

正常的

root@zs600b:/system/priv-app # ls -al | grep Set

drwxr-xr-x root root 2017-07-06 11:32 Settings

drwxr-xr-x root root 2017-07-06 11:30 SettingsProvider

异常：

root@zs600b:/system/priv-app # ls -al | grep Set

drwxr-xr-x root root 2017-08-29 21:42 Settings

drwxr-xr-x root root 2017-08-29 21:40 SettingsProvider

-rw-r--r-- root root 126118 2017-08-31 21:35 SettingsProvider.apk

Log：

09-01 10:10:20.509 I/PackageManager( 486): Package com.android.providers.settings at /system/priv-app/SettingsProvider.apk ignored: updated version 22 better than this 22

09-01 10:10:20.509 W/PackageManager( 486): Code path for hidden system pkg : com.android.providers.settings changing from /system/priv-app/SettingsProvider to /system/priv-app/SettingsProvider.apk

09-01 10:10:20.509 W/PackageManager( 486): Failed to parse /system/priv-app/SettingsProvider.apk: null

root@zs600b:/system/priv-app/SettingsProvider/arm # ls -al

-rw-r--r-- root root 254440 2017-08-29 21:40 SettingsProvider.odex

09-01 10:47:21.478 W/art ( 489): Failed to find OatDexFile for DexFile /system/priv-app/SettingsProvider/SettingsProvider.apk ( canonical path /system/priv-app/SettingsProvider/SettingsProvider.apk) with checksum 0x6ade6c23 in OatFile /system/priv-app/SettingsProvider/arm/SettingsProvider.odex

09-01 10:47:21.504 I/dex2oat ( 549): Error opening the directory: /system/lib/plugins

09-01 10:47:21.505 I/dex2oat ( 549): /system/bin/dex2oat --runtime-arg -classpath --runtime-arg --instruction-set=arm --instruction-set-features=div --runtime-arg -Xrelocate --boot-image=/system/framework/boot.art --dex-file=/system/priv-app/SettingsProvider/SettingsProvider.apk --oat-fd=76 --oat-location=/data/dalvik-cache/arm/system@priv-app@SettingsProvider@SettingsProvider.apk@classes.dex --runtime-arg -Xms64m --runtime-arg -Xmx512m

adb install –r

INSTALL\_FAILED\_ALREADY\_EXISTS

means you uninstall before install -r remove purpose

## 实例分析

在framework/base/core/java定义成另一个了

Settings.Secure.putInt(getContext().getContentResolver(), Settings.Secure.LOCATION\_MODE, mode);

## And7插入流程分析

<https://blog.csdn.net/tangzhihai0421/article/details/71637287>

[**测试**](http://lib.csdn.net/base/softwaretest)的手机上是/data/system/users/0，该目录的settings\_global.xml，settings\_secure.xml和settings\_system.xml三个xml文件就是SettingsProvider中的数据文件

从getSettingsFile中可以看到SettingsProvider三个表global，system和secure的文件，它们的路径是相同的，通过Environment.getUserSystemDirectory确立

frameworks/base/core/java/android/os/Environment.java

**[java]** [view plain](http://blog.csdn.net/firedancer0089/article/details/70683085) [copy](http://blog.csdn.net/firedancer0089/article/details/70683085)

[在CODE上查看代码片](https://code.csdn.net/snippets/2354289)

1. **public** **static** File getUserSystemDirectory(**int** userId) {
2. **return** **new** File(**new** File(getSystemSecureDirectory(), "users"), Integer.toString(userId));
3. }

**[java]** [view plain](http://blog.csdn.net/firedancer0089/article/details/70683085) [copy](http://blog.csdn.net/firedancer0089/article/details/70683085)

[在CODE上查看代码片](https://code.csdn.net/snippets/2354289)

1. **public** **static** File getSystemSecureDirectory() {
2. **if** (isEncryptedFilesystemEnabled()) { //加密文件系统分支，默认是关闭的
3. **return** **new** File(SECURE\_DATA\_DIRECTORY, "system");
4. } **else** {
5. **return** **new** File(DATA\_DIRECTORY, "system");
6. }
7. }

**[java]** [view plain](http://blog.csdn.net/firedancer0089/article/details/70683085) [copy](http://blog.csdn.net/firedancer0089/article/details/70683085)

[在CODE上查看代码片](https://code.csdn.net/snippets/2354289)

1. **private** **static** **final** File DATA\_DIRECTORY
2. = getDirectory("ANDROID\_DATA", "/data");

依据ANDROID\_DATA环境变量获取路径，获取不到的话就默认为/data。

获取到DATA\_DIRECTORY后，后续还要添加"/ " + "system" + "/" + "users" + "/" + userId的值

例如我[**测试**](http://lib.csdn.net/base/softwaretest)的手机上是/data/system/users/0，该目录的settings\_global.xml，settings\_secure.xml和settings\_system.xml三个xml文件就是SettingsProvider中的数据文件。

这里虽然手机目录中数据库存储是以xml的形式，但是代码中操作时操作的仍然是数据库。系统会先生成Settings.db数据库，然后将其中的内容保存到xml文件中，最后删除Settings.db数据库(SettingsProvider的功能，后续会讲到)。   
这里为何先生成数据库后又删除，主要是为了兼容之前的版本。   
为什么以xml的形式保存字段：节省资源，简单，安全(xml保存的路径相对于之前的数据库路径比较安全，大概是考虑到多用户吧)

### 脚本命令settings put/get

settings get system ringtone

settings get global ringtone

root@gl300k:/ # settings put secure icon\_blacklist ethernet

root@gl300k:/ # settings get secure icon\_blacklist

ethernet

### /data/system/users/0 # cat settings\_ global.xml

B

<setting id="384" name="dji\_RECORDING" value="0" package="com.kpad.ui" />

<setting id="397" name="boot\_count" value="43" package="android" />

<setting id="4" name="airplane\_mode\_toggleable\_radios" value="bluetooth,wifi,nfc" package="android" />

<setting id="53" name="network\_scoring\_provisioned" value="1" package="android" />

<setting id="45" name="device\_name" value="rm500" package="android" />

### /data/system/users/0 # cat settings\_system.xml

<setting id="30" name="dual\_screen\_icon\_used" value="0" package="android" />

<setting id="60" name="kji\_fn\_tip" value="1" package="com.android.settings" />

<setting id="12" name="screen\_off\_timeout" value="60000" package="android" />

<setting id="47" name="sre\_state" value="0" package="kji.system.launcher" />

<setting id="46" name="screen\_brightness" value="255" package="com.android.systemui" />

### /data/system/users/0 # cat settings\_secure.xml

<setting id="56" name="user\_setup\_complete" value="1" package="com.kpad.launcher" />

<setting id="11" name="long\_press\_timeout" value="400" package="android" />

<setting id="37" name="sysui\_tuner\_version" value="1" package="com.android.systemui" />

<setting id="19" name="screensaver\_components" value="com.google.android.deskclock/com.android.deskclock.Screensaver" package="android" />

<setting id="44" name="show\_note\_about\_notification\_hiding" value="0" package="com.android.systemui" />

<setting id="26" name="install\_non\_market\_apps" value="0" package="android" />

<setting id="33" name="show\_ime\_with\_hard\_keyboard" value="0" package="android" />

<setting id="35" name="multi\_press\_timeout" value="300" package="android" />

<setting id="51" name="enabled\_input\_methods" value="com.android.inputmethod.latin/.LatinIME" package="android" />

### Input

/data/system/users/0 # cat settings\_secure.xml | grep input

<setting id="55" name="default\_input\_method" value="com.android.inputmethod.latin/.LatinIME" package="android" />

<setting id="64" name="input\_methods\_subtype\_history" value="com.android.inputmethod.latin/.LatinIME;-921088104" package="android" />

<setting id="72" name="selected\_input\_method\_subtype" value="-1" package="android" />

<setting id="46" name="selected\_spell\_checker" value="com.android.inputmethod.latin/.spellcheck.AndroidSpellCheckerService" package="android" />

<setting id="51" name="enabled\_input\_methods" value="com.android.inputmethod.latin/.LatinIME" package="android" />

## media数据库

*media相关的数据路径：   
data/data/com.android.providers.meida   
external.db是外置文件路径(SD卡存储的文件)   
internal.db是内存文件路径(手机存储的文件)*

## Ref

[**Android SettingProvider详解**](http://blog.csdn.net/peng_cao/article/details/50887509)

[Settings.Global](https://developer.android.com/reference/android/provider/Settings.Global.html)

# SettingsLib概述

### 流程图



### Dashboard category的整体布局



看上面这幅图，从上而下分别是Condition，Suggestion和各个显示的Item项。

接下来我们来看这部分在代码中是如何构建的？

### Dashboard category数据的加载

<https://blog.csdn.net/h_gao/article/details/58201121>

dashboard的中文意思指的是仪表板

在Settings N中，则将dashboard这部分的逻辑抽取了出来，放在/frameworks/base/packages/SettingsLib/目录下。N中不再使用dashboard\_categories.xml这个文件来描述Settings各选项的架构，而且将Dashboard的初始化放在SettingsLib中来处理，首先看下面的图片：

7.0的Settings的选项不再从dashboard\_categories.xml中加载选项列表，而是通过在Androidmanifest.xml中，配置intent-filter的Action，否。

#### 标记Activity属于哪一个Dashboard category

TileUtils.Java中通过几个Action进行获取系统中对应的activity，如Settings中的几个

private static final String SETTINGS\_ACTION ="com.android.settings.action.SETTINGS";

private static final String OPERATOR\_SETTINGS ="com.android.settings.OPERATOR\_APPLICATION\_SETTING";

private static final String OPERATOR\_DEFAULT\_CATEGORY ="com.android.settings.category.wireless";

private static final String MANUFACTURER\_SETTINGS ="com.android.settings.MANUFACTURER\_APPLICATION\_SETTING";

private static final String MANUFACTURER\_DEFAULT\_CATEGORY ="com.android.settings.category.device";

***Categories定义在Settings/res/values/donottranslate.xml中，分为四个大的Category，如下代码***

Settings/res/values/donottranslate.xml

<string name="category\_key\_wireless">com.android.settings.category.wireless</string>

<string name="category\_key\_device">com.android.settings.category.device</string>

<string name="category\_key\_personal">com.android.settings.category.personal</string>

<string name="category\_key\_system">com.android.settings.category.system</string>

#### TileUtils.java中定义的Meta Data

F

public static final String META\_DATA\_PREFERENCE\_ICON = "com.android.settings.icon";

public static final String META\_DATA\_PREFERENCE\_TITLE = "com.android.settings.title";

public static final String META\_DATA\_PREFERENCE\_SUMMARY = "com.android.settings.summary";

通过PackageManager进行搜索，获取到这一系列的activity信息，

PackageManager pm = context.getPackageManager();

List<ResolveInfo> results = pm.queryIntentActivitiesAsUser(intent,PackageManager.GET\_META\_DATA, user.getIdentifier());

#### AndroidManifest.xml通过meta-data配置

icon、title、summary，那这就有Settings中显示的图标、标题和说明。还有在Settings中显示的分类、目标Fragment。

Settings 的AndroidManifest.xml 下面那代码是Settings->about phone的Activity配置。

<activity android:name="Settings$DeviceInfoSettingsActivity"

android:theme="@style/Theme.SubSettingsDialogWhenLarge"

android:label="@string/device\_info\_settings"

android:icon="@drawable/ic\_settings\_about"

android:taskAffinity="com.android.settings"

android:parentActivityName="Settings">

<intent-filter android:priority="1">

<action android:name="android.settings.DEVICE\_INFO\_SETTINGS" />

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

</intent-filter>

<intent-filter>

<action android:name="android.intent.action.MAIN" />

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

<category android:name="android.intent.category.VOICE\_LAUNCH" />

</intent-filter>

<intent-filter android:priority="-1">

<action android:name="com.android.settings.action.SETTINGS" />

</intent-filter>

<meta-data android:name="com.android.settings.category"

android:value="com.android.settings.category.system" />

<meta-data android:name="com.android.settings.title"

android:resource="@string/about\_settings" />

<meta-data android:name="com.android.settings.FRAGMENT\_CLASS"

android:value="com.android.settings.DeviceInfoSettings" />

<meta-data android:name="com.android.settings.PRIMARY\_PROFILE\_CONTROLLED"

android:value="true" />

</activity>

Settings显示各选项的思路比较简单，所以直接在Settings里面添加功能选项还是比较简单的。

#### 在通过PackageManager进行指定的Action进行搜索

，那么就可以获取到需要显示的选项列表了，并且也需要在代码中进行判断，判断哪些功能需要显示与

**对于这些Dashboard的icon，title和Summary有的在AndroidManifest.xml中有配置meta-data有的却没有，我感觉这里应该用的是Activity节点下的icon，title（lablel**

**）**

上面的getCategories方法主要分为两个部分来看，首先通过PackageManager获得各个Category的信息保存到ArrayList中，接着对ArrayList中的数据按照优先级进行排序，这样主界面拿到这些数据就可以显示了。

*//* ***TODO: Needs much optimization, too many PM queries going on here.***

如果需要定制白名单，需要在这里定制：白名单添加原理

***OPERATOR\_SETTINGS***

***MANUFACTURER\_SETTINGS***

**if**(!isSupportBluetooth && tileIntent != **null** && tileIntent.toString().contains(**"com.android.settings/.Settings$BluetoothSettingsActivity"**))  
 **continue**;

#### Collections.*sort*

Hjk

ArrayList<DashboardCategory> categories = **new** ArrayList<>(categoryMap.values());  
**for** (DashboardCategory category : categories) {  
 Collections.*sort*(category.**tiles**, ***TILE\_COMPARATOR***);  
}  
Collections.*sort*(categories, ***CATEGORY\_COMPARATOR***);

**public static final** Comparator<Tile> ***TILE\_COMPARATOR*** =  
 **new** Comparator<Tile>() {  
 @Override  
 **public int** compare(Tile lhs, Tile rhs) {  
 **return** rhs.**priority** - lhs.**priority**;  
 }  
};  
  
**private static final** Comparator<DashboardCategory> ***CATEGORY\_COMPARATOR*** =  
 **new** Comparator<DashboardCategory>() {  
 @Override  
 **public int** compare(DashboardCategory lhs, DashboardCategory rhs) {  
 **return** rhs.**priority** - lhs.**priority**;  
 }  
};

#### 分析

主题还是要回归到分析代码上面来

# Settings概述

## 版本迭代

### Android 8.0 Settings

在7.0的基础上，去掉了7.0新加的侧滑菜单（可能是觉得有点鸡肋吧）。多加了一级页面，把原来类别标题变成的第一级菜单的子项。在代码架构也稍加变动，并引入架构组件之LifeCycle（生命周期感知，本文不作介绍）



<https://blog.csdn.net/qq_31012033/article/details/79522813>

Android8.0 在settings中添加蓝牙耳机的电池电量信息：https://blog.csdn.net/Aaron121314/article/details/78236104

### Android 7.0 Settings

多了一个建议选项，多了个侧边栏，操作更加便捷了。

7.0的Settings的选项不再从dashboard\_categories.xml中加载选项列表，而是通过在Androidmanifest.xml中，配置intent-filter的Action，在通过PackageManager进行指定的Action进行搜索，那么就可以获取到需要显示的选项列表了，并且也需要在代码中进行判断，判断哪些功能需要显示与否。



作了一些调整，如上面的截图。   
- 增加了侧滑菜单，采用v4下的DrawerLayout来实现；   
- 在Settings主界面增加了Condition，能够在设置列表中显示状态；   
- 在Settings主界面增加了Suggestion

### Android 6.0

之前做Android 6.0开发的，都会了解到6.0的Settings加载选项是通过加载dashboard\_categories.xml，获取需要显示的选项，并且在SettingsActivity中也进行判断是否要显示，所以在6.0上添加一个选项是比较简单的，直接在dashboard\_categories.xml添加icon、title、summary，也可以添加目标fragment和Intent，这样就可以顺利跳转到对应的界面了。但是在7.0上，google对Settings进行了重构。

系统设置是Android系统中非常重要的系统应用，也是整个android系统的控制中枢，关于设备的硬件，状态，软件，安全等都需要通过系统设置进行控制和查看。比如wifi状态，网络连接状。具体的功能是SettingProvider造作数据库实现的。

特别的，系统设置并不同于大多数的其他系统应用，系统设置不仅拥有platform签名（即系统签名），而且属于内核应用，所以系统设置要比非内核应用的系统应用有更大的权限。系统签名方法见第1章。

|  |
| --- |
| <activity android:name="Settings"  android:taskAffinity="com.android.settings"  android:label="@string/settings\_label\_launcher"  android:launchMode="singleTask">  <intent-filter>  <action android:name="android.settings.SETTINGS" />  <category android:name="android.intent.category.DEFAULT" />  </intent-filter>  <meta-data android:name="com.android.settings.PRIMARY\_PROFILE\_CONTROLLED"  android:value="true" /> </activity> |

在5.1.1的Settings源码中,Setting extends SettingActivity.查看SettingActivity发现，在SettingActivity中加载了设置的主界面布局，名为dashboard\_categories，文件目录为项目目录/res/xml/dashboard\_categories文件，布局文件中使用的布局标签为以及列表项,SettingsActivity中加载该布局文件的方法为loadCategoriesFromResource(),该方法存在于buildDashboardCategories(List categories)方法中,其中的行参就是所包含的设置列表项,加载该xml布局的方法就是解析xml解析器，对布局文件进行解析，现将该方法贴在此处，顺便回顾一下解析xml文件的方法：

如上的加载和解析的方法其实是在onResume方法中调用的,在onResume方法中调用了invalidateCategories方法，在invalidateCategories方法中我们会发现其实是采用了回发消息然后处理的消息处理机制，在此方法中发送了MSG\_BUILD\_CATEGORIES消息，在类声明处定义的名为mHandler的Handler中进行处理，随即调用buildDashboardCategories方法.

下面我们重新说回设置界面，上面说到解析和加载xml文件及相关数据展示，下面设计到的就是我们点击每一项时

## 总体架构

Settings主要的功能一句话描述：保存用户设置的值到Settings数据库，其他应用或Framework层通过监听SettingsProvider数据库的变化，来做一些相应的处理操作。

### 涉及的目录路径(只列出部分)：

代码相关目录：

*android\frameworks\base\packages\SettingsLib 7.0重构Settings后多出来的部分   
android\frameworks\base\packages\SettingsProvider settings数据库相关   
android\packages\apps\Settings Settings相关   
android\frameworks\base\core\java\android\provider settings数据库字段定义相关   
android\packages\providers\MediaProvider铃声相关   
android\packages\screensavers屏保相关   
android\packages\inputmethods输入法相关*

apk相关目录

>源码编译后apk生成目录:

android\out\target\product\msm8940\_64\system\priv-app\Settings\Settings.apk

android\out\target\product\msm8940\_64\system\priv-app\MediaProvider\MediaProvider.apk

android\out\target\product\msm8940\_64\system\priv-app\SettingsProvider\SettingsProvider.ap

### Settings目录结构

android\packages\apps\Settings   
- src   
- - com.android.settings Settings应用整体所用的java文件   
- - com.android.settings.accessibility 辅助功能相关   
- - com.android.settings.accounts 账户相关   
- - com.android.settings.application 应用管理相关   
- - com.android.settings.backup 备份相关   
- - com.android.settings.bluetooth 蓝牙相关   
- - com.android.settings.dashboard Settings主界面相关   
- - com.android.settings.datausage 数据流量相关   
- - com.android.settings.deletionhelper   
- - com.android.settings.deviceinfo 关于手机相关   
- - com.android.settings.display 显示相关   
- - com.android.settings.drawable 菜单图片相关   
- - com.android.settings.fingerprint 指纹相关   
- - com.android.settings.fuelgauge   
- - com.android.settings.gestures 手势相关   
- - com.android.settings.inputmethod 输入法相关   
- - com.android.settings.localepicker 语言相关   
- - com.android.settings.location 位置相关   
- - com.android.settings.nfc nfc相关   
- - com.android.settings.notification 通知相关相关   
- - com.android.settings.overlay   
- - com.android.settings.password 密码相关   
- - com.android.settings.print 打印相关   
- - com.android.settings.qstile   
- - com.android.settings.search 快速搜索相关   
- - com.android.settings.sim SIM卡相关   
- - com.android.settings.support   
- - com.android.settings.tts tts播报相关   
- - com.android.settings.users 用户相关   
- - com.android.settings.utils 工具类   
- - com.android.settings.voice   
- - com.android.settings.vpn2   
- - com.android.settings.wfd   
- - com.android.settings.widget 小部件相关   
- - com.android.settings.wifi wifi相关   
- AndroidManifest.xml 清单文件

### SettingsLib目录结构

android\frameworks\base\packages\SettingsLib\src\com\android\settingslib   
- src   
- - com.android.settingslib.accessibility 辅助功能相关   
- - com.android.settingslib.accounts 账户相关   
- - com.android.settingslib.animation 动画相关   
- - com.android.settingslib.application 应用管理相关   
- - com.android.settingslib.bluetooth 蓝牙相关   
- - com.android.settingslib.datetime 日期相关   
- - com.android.settingslib.deviceinfo 关于手机相关   
- - com.android.settingslib.display 显示相关   
- - com.android.settingslib.drawable 菜单图片相关   
- - com.android.settingslib.drawer 侧滑菜单相关   
- - com.android.settings.dream 休眠相关   
- - com.android.settings.graph   
- - com.android.settings.location 位置相关   
- - com.android.settings.net 网络相关   
- - com.android.settings.users 用户相关   
- - com.android.settings.widget 小部件相关   
- - com.android.settings.wifi wifi相关   
- AndroidManifest.xml 清单文件

### 暗码

手机暗码(字母和数字组合)：在拨号界面输入暗码即可进入相应的选项或打开相关的功能，各个类型的手机暗码不一样，这里说的是我司用的暗码，另外暗码主要为了方便研发开发使用，有的手机厂商在手机量产前会去掉暗码功能(即暗码无效)

后台log暗码：##564##

版本号暗码：##123321##

root暗码：##55555##

## Android 7 menifest

android.uid.system **uses-permission**

为啥还要权限申明呢？

### permission

## 情景分析

### Dashboard category数据的加载

<https://blog.csdn.net/h_gao/article/details/58201121>

dashboard的中文意思指的是仪表板

在Settings N中，则将dashboard这部分的逻辑抽取了出来，放在/frameworks/base/packages/SettingsLib/目录下。N中不再使用dashboard\_categories.xml这个文件来描述Settings各选项的架构，而且将Dashboard的初始化放在SettingsLib中来处理，首先看下面的图片：

### 添加第三方apk到Settings选项中

Step1：添加action：可以被PackageManager搜索到的activity，搜索到后添加到Settings的选项列表中，private static final String EXTRA\_SETTINGS\_ACTION ="com.android.settings.action.EXTRA\_SETTINGS";

Step2：添加显示的选项信息：在该apk的AndroidManifest.xml对应的启动activity中添加以下meta-data。注意混淆代码问题。

Step3：TileUtils.java白名单

这个也是Android考虑到的一些安全上的问题，没有添加到白名单上面的包名，就不会显示出来。

/// Extra package white list for add item to Settings @{

private static final String[] EXTRA\_PACKAGE\_WHITE\_LIST = {};

/// @}

最后就是全编，刷机验证了。

<https://blog.csdn.net/qq_25804863/article/details/50229461>

<https://blog.csdn.net/H_Gao/article/details/58201121>

https://www.demgs.com/105558.html

### Summary加载原理

Summary的显示流程



#### ianSummaryLoader

1、创建一个Handler对象；   
2、创建异步线程；   
3、利用双重循环遍历每个title，然后通过异步线程处理各个tile

可以发现最后把 SummaryProvider和 Component已键值对的形式放到了一个ArrayMap中。   
这里关键看下getSummaryProvider的实现

#### getSummaryProvider

这个方法主要做了以下几件事：   
1、获取MetaData；   
2、获取MetaData的的META\_DATA\_KEY\_FRAGMENT\_CLASS值的值   
3、根据上一步获取的字符串 通过反射来获取这个类（上一步获取的字符串形式就是“com.android.settings.fuelgauge.PowerUsageSummary” 这样的）   
4、通过上步获取的类获取类中的SUMMARY\_PROVIDER\_FACTORY这个属性   
5、返回类中的SummaryProvider对象。

[Android 7.0 Settings Summary 小记](https://blog.csdn.net/asd1031/article/details/76165085)

#### Set

通过搜索可以定位到这个配置的地方是在 下面代码加粗的地方配置

public void setListening(boolean listening) {

if (listening) {

// TODO: Listen.

BatteryInfo.getBatteryInfo(mContext, new BatteryInfo.Callback() {

@Override

public void onBatteryInfoLoaded(BatteryInfo info) {

\*mLoader.setSummary(SummaryProvider.this, "("+(int)BatteryCapcityValue+"mAh)"+info.mChargeLabelString);\*

}

});

}

}

知其然要知其所以然 所以今天分析下为什么一个Summary 需要在这个地方配置。   
进入setSummary 方法

public void setSummary(SummaryProvider provider, final CharSequence summary) {

final ComponentName component= mSummaryMap.get(provider);

mHandler.post(new Runnable() {

@Override

public void run() {

// Since tiles are not always cached (like on locale change for instance),

// we need to always get the latest one.

Tile tile = mAdapter.getTile(component);

if (tile == null) return;

if (DEBUG) Log.d(TAG, "setSummary " + tile.title + " - " + summary);

tile.summary = summary;

mAdapter.notifyChanged(tile);

}

});

}

* 1可以发现主要就是设置titile的summary 然后通知Adapter更新，而Adapter 是DashboardAdapter的实例，是设置给DashboardSummary的，通过上面的时序图其实可以Settings的布局其实就是在DashboardSummary中进行处理的。

而mLoader.setSummary(SummaryProvider.this, “(“+(int)BatteryCapcityValue+”mAh)”+info.mChargeLabelString)

### 跳转到各个系统设置的方法

adb shell am start com.android.settings

// 第一个参为包名，第二个各个设置的类名(可以参考下面，包名不用改变)

ComponentName cm = new ComponentName("com.android.settings",

"com.android.settings.RadioInfo");

intent.setComponent(cm);

intent.setAction("android.intent.action.VIEW");

startActivity(intent);

跳转到各个系统设置的列表

com.android.settings.AccessibilitySettings 辅助功能设置

　　com.android.settings.ActivityPicker 选择活动

　　com.android.settings.ApnSettings APN设置

　　com.android.settings.ApplicationSettings 应用程序设置

　　com.android.settings.BandMode 设置GSM/UMTS波段

　　com.android.settings.BatteryInfo 电池信息

　　com.android.settings.DateTimeSettings 日期和坝上旅游网时间设置

　　com.android.settings.DateTimeSettingsSetupWizard 日期和时间设置

　　com.android.settings.DevelopmentSettings 应用程序设置=》开发设置

　　com.android.settings.DeviceAdminSettings 设备管理器

　　com.android.settings.DeviceInfoSettings 关于手机

　　com.android.settings.Display 显示——设置显示字体大小及预览

　　com.android.settings.DisplaySettings 显示设置

　　com.android.settings.DockSettings 底座设置

　　com.android.settings.IccLockSettings SIM卡锁定设置

　　com.android.settings.InstalledAppDetails 语言和键盘设置

　　com.android.settings.LanguageSettings 语言和键盘设置

　　com.android.settings.LocalePicker 选择手机语言

　　com.android.settings.LocalePickerInSetupWizard 选择手机语言

　　com.android.settings.ManageApplications 已下载（安装）软件列表

　　com.android.settings.MasterClear 恢复出厂设置

　　com.android.settings.MediaFormat 格式化手机闪存

　　com.android.settings.PhysicalKeyboardSettings 设置键盘

　　com.android.settings.PrivacySettings 隐私设置

　　com.android.settings.ProxySelector 代理设置

　　com.android.settings.RadioInfo 手机信息

　　com.android.settings.RunningServices 正在运行的程序（服务）

　　com.android.settings.SecuritySettings 位置和安全设置

　　com.android.settings.Settings 系统设置

　　com.android.settings.SettingsSafetyLegalActivity 安全信息

　　com.android.settings.SoundSettings 声音设置

　　com.android.settings.TestingSettings 测试——显示手机信息、电池信息、使用情况统计、Wifi information、服务信息

　　com.android.settings.TetherSettings 绑定与便携式热点

　　com.android.settings.TextToSpeechSettings 文字转语音设置

　　com.android.settings.UsageStats 使用情况统计

　　com.android.settings.UserDictionarySettings 用户词典

　　com.android.settings.VoiceInputOutputSettings 语音输入与输出设置

　　com.android.settings.WirelessSettings 无线和网络设置

参考自：<http://blog.csdn.net/aikongmeng/article/details/44748841>

### 分屏

此类为了分屏

MetricsEvent相关信息

@Override  
    protected int getMetricsCategory() {  
        return MetricsEvent.DATE\_TIME;  
    }

字段定义

J:\xxx1\android\frameworks\base\proto\src\metrics\_constants.proto

类文件

J:\xxx\android\frameworks\base\core\java\com\android\internal\logging\MetricsLogger.java

out目录生成的文件

J:\xxx\android\out\target\common\obj\JAVA\_LIBRARIES\framework-protos\_intermediates\src\proto\src\com\android\internal\logging\MetricsProto.java

Tile文件

J:\YL50B71\android\frameworks\base\packages\SettingsLib\src\com\android\settingslib\drawer\Tile.java

J:\YL50B71\android\frameworks\base\packages\SettingsLib\src\com\android\settingslib\drawer\TileUtils.java

## 黑+蓝修改表格

Hdrawer\_layout

SettingsDrawerActivity

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 图标 | 1号字 | 2号字 |
| 首页： | 背景色  黑 | 蓝色 |  |  |
| 条件面板  打开飞行模式会出现：  有根白色的粗线 |  |  |  |  |
| 建议面板 |  |  |  |  |
| Wlan->设置 | **Preference**？？？？  ***wifi\_configure\_settings*** |  |  |  |
| Wlan-更多 | ***MENU？？？*** |  |  |  |
| Wlan-开关 | ？？？？ |  |  |  |
| Wlan-更多-高级 | **Preference** |  |  |  |
| Wlan对话框 | **DialogWhenLarge**  背景和字体。。 |  |  |  |
| 蓝牙-更多 | **同wifi** |  |  |  |
| 流量使用-**菜单** | **菜单** |  |  |  |
| 流量使用- | **Preference** |  |  |  |
| 流量使用-柱状图 | **代码写的。。** |  |  |  |
| 流量使用-程序 | **Preference** |  |  |  |
| 流量使用-流量图 | **Wlan/以太网/** |  |  |  |
| 网络流量限制 | **Preference** |  |  |  |
| 网络与便携式热点 |  |  |  |  |
| 网络与便携式热点-设置热点 |  |  |  |  |
| VPN | **Dlg** |  |  |  |
| 移动网络 |  |  |  |  |
| 重置网络设置 |  |  |  |  |
| 以太网 |  |  |  |  |
|  |  |  |  |  |
| 显示-亮度 |  |  |  |  |
| 壁纸-动态壁纸 |  |  |  |  |
| 休眠 | **有蓝色了？** |  |  |  |
| 屏保 | **菜单**  **开关**  **文字** |  |  |  |
| 字体大小 | **右侧文字** |  |  |  |
| 显示大小 |  |  |  |  |
| 设备旋转时 |  |  |  |  |
| 投射 | **菜单**  **空文本** |  |  |  |
| Hdmi | **Preference** |  |  |  |
| 通知 | **菜单弹窗**  **Tab过滤弹窗**  **分割线** |  |  |  |
| 通知：配置通知 | **锁屏配置，弹窗** |  |  |  |
| 音量 | **分割线** |  |  |  |
| 勿扰 | **很多，需要屏蔽掉吧。。** |  |  |  |
| 默认通知铃声 | **弹窗** |  |  |  |
| 默认闹钟铃声 | **弹窗** |  |  |  |
| 其他提示音 | **3个开关** |  |  |  |
| 投射 | **空文本，菜单** |  |  |  |
| 应用 | **分割线，设置菜单** |  |  |  |
| 应用-配置 | **应用权限** |  |  |  |
|  | **应用权限列表：开关，列表** |  |  |  |
|  | **打开连接，文字颜色列表** |  |  |  |
|  | **辅助应用和语音输入** |  |  |  |
|  | **注屏幕应用，弹窗。。** |  |  |  |
|  | **浏览器应用，弹窗** |  |  |  |
|  | **警报应用** |  |  |  |
|  | **特殊访问权限，屏蔽掉吧** |  |  |  |
|  | **重置偏好设置，** |  |  |  |
| 存储 | **百分比线，分隔线** |  |  |  |
| 电池 | **分隔线** |  |  |  |
|  | **菜单：电池优化** | Tab列表 |  |  |
|  |  | **重置偏好** |  |  |
| **内存** | **Popupdlg** |  |  |  |
|  | **各个应用使用的内存** |  |  |  |
|  |  | **排序菜单** |  |  |
|  |  | **列表，分隔线，百分比**  **文字颜色** |  |  |
| **个人** |  |  |  |  |
| **位置信息** | **菜单，分隔线** |  |  |  |
|  | **位置信息模式** | **分割线，文字，单选按钮** |  |  |
|  | **最近信息位置请求** | **文字，按钮** |  |  |
|  |  | **存储** |  |  |
|  |  | **流量使用** |  |  |
|  |  | **权限** |  |  |
|  |  | **通知** |  |  |
|  |  | **通知** |  |  |

导航栏

Actionbar

侧滑页面：

蓝色图标，

白字

分割线

# WLAN

## WIFI选项

Wifi是系统设置中的第一项功能，通过该功能可以进行“关闭/打开WIFI”、扫描可用网络，连接网络等操作，setWifiApEnabled需要系统权限@SystemApi。核心代码如下：

|  |
| --- |
| @Override **public void** onCheckedChanged(CompoundButton switchView, **boolean** isChecked) {  KPADLogUtil.*E*(**TAG**, **"onCheckedChanged,isChecked="** + isChecked);  *//Do nothing if called as a result of a state machine event* setSwitchBarEnabled(**false**);  **if** (**mStateMachineEvent**) {//避免循环调用，造成死循环  **return**;  }  *// Disable tethering if enabling Wifi。关闭热点* **int** wifiApState = **mWifiManager**.getWifiApState();  **if** (isChecked && ((wifiApState == WifiManager.WIFI\_AP\_STATE\_ENABLING) ||  (wifiApState == WifiManager.WIFI\_AP\_STATE\_ENABLED))) {  **mWifiManager**.setWifiApEnabled(**null**, **false**);  }   **if** (!**mWifiManager**.setWifiEnabled(isChecked)) {//打开wifi  *// Error* **mSwitchBar**.setEnabled(**true**);  Toast.*makeText*(**mContext**, **"wifi error"**, Toast.***LENGTH\_SHORT***).show();  } } |

**飞行模式**的初衷是为了在飞机飞行状态下一次性关闭所有可以发出无线信号的设备而加入的功能。这些//设备主要包括3G/2G, Wi-Fi、蓝牙、NFC等设备。

热点(Access Point)和Wi-Fi不能共存

### SoftAp

市面上大多数手机支持SoftAp功能, 系统的Setting程序TetherSettings.java将响应此动作。tether这个词意思是拴绳，拴住的意思，理解起来有点晦涩，这里我的理解是分享的意思，比如 WIFI\_TETHERING(用WIFI分享网路),  USB\_TETHERING（用USB分享网络）……

那么这些网络分享如何满足什么条件才开启呢？疑问

### 从WirelessSettings开始说

在packages/apps/Settings/src/com/android/settings/WirelessSettings.java 的onCreate方法中会去检测系统是否Tether

1. ConnectivityManager cm =
2. (ConnectivityManager) activity.getSystemService(Context.CONNECTIVITY\_SERVICE);
3. **if** (isSecondaryUser || !cm.isTetheringSupported()) {
4. getPreferenceScreen().removePreference(findPreference(KEY\_TETHER\_SETTINGS));
5. } **else** {
6. Preference p = findPreference(KEY\_TETHER\_SETTINGS);
7. p.setTitle(Utils.getTetheringLabel(cm));
8. }
9. protectByRestrictions(KEY\_TETHER\_SETTINGS);

1）调用获取ConnectivityManager实例对象，并调用isTetheringSupported方法来检测是否支持Tether

2）调用getTetheringLabel查看支持哪些网络共享方式，并显示到界面

先看isTetheringSupported方法

1. **public** boolean isTetheringSupported() {
2. **try** {
3. **return** mService.isTetheringSupported();
4. } **catch** (RemoteException e) {
5. **return** **false**;
6. ...

它调用了frameworks/base/services/core/java/com/android/server/ConnectivityService.java 的isTetheringSupported 方法

**[cpp]** [view plain](http://blog.csdn.net/jshazk1989/article/details/17094415) [copy](http://blog.csdn.net/jshazk1989/article/details/17094415)

1. **public** boolean isTetheringSupported() {
2. enforceTetherAccessPermission();
3. **int** defaultVal = (SystemProperties.get("ro.tether.denied").equals("true") ? 0 : 1);
4. boolean tetherEnabledInSettings = (Settings.Global.getInt(mContext.getContentResolver(),
5. Settings.Global.TETHER\_SUPPORTED, defaultVal) != 0);
6. **return** tetherEnabledInSettings && ((mTethering.getTetherableUsbRegexs().length != 0 ||
7. mTethering.getTetherableWifiRegexs().length != 0 ||
8. mTethering.getTetherableBluetoothRegexs().length != 0) &&
9. mTethering.getUpstreamIfaceTypes().length != 0);
10. }

 读取系统ro.tether.denied的属性，返回true则表示关闭tether功能，返回false表示打开tether功能。（有些设备不能显示portable wifi 就是因为此属性返回true,这里需要修改设置为false）

之后会检测是否有配置文件配置了tether，有则使用设置的值，没有查询到配置这使用查询ro.tether.denied属性的默认值。

这里我们假设得到的tetherEnableInSettings为ture，也不意味着就能正确打开Portable Wi-Fi hostpot 界面。因为还需要检测是否支持WIFI网络共享配置

mTethering.getTetherableWifiRegexs().length!=0 检测是否支持网络共享配置，这里Android4.4 与之前的版本有一定区别，但总体不变，大家可以参考其他代码做一些变通

### getTetherableWifiRegexs()方法

frameworks/base/services/java/com/android/server/connectivity/Tethering.java

**[cpp]** [view plain](http://blog.csdn.net/jshazk1989/article/details/17094415) [copy](http://blog.csdn.net/jshazk1989/article/details/17094415)

1. **return** mTetherableWifiRegexs;

直接返回了一个变量。 此变量在frameworks/base/services/java/com/android/server/connectivity/Tethering.java

updateConfiguration()方法中赋值

**[cpp]** [view plain](http://blog.csdn.net/jshazk1989/article/details/17094415) [copy](http://blog.csdn.net/jshazk1989/article/details/17094415)

1. mTetherableWifiRegexs = tetherableWifiRegexs;

而 tetherableWifiRegexs变量赋值也在此方法中

**[cpp]** [view plain](http://blog.csdn.net/jshazk1989/article/details/17094415) [copy](http://blog.csdn.net/jshazk1989/article/details/17094415)

1. String[] tetherableWifiRegexs = mContext.getResources().getStringArray(
2. com.android.internal.R.array.config\_tether\_wifi\_regexs);

这里的config\_tether\_wifi\_regexs数组由配置文件生成

### 配置文件config.xml

frameworks/base/core/res/res/values/config.xml

**[html]** [view plain](http://blog.csdn.net/jshazk1989/article/details/17094415) [copy](http://blog.csdn.net/jshazk1989/article/details/17094415)

1. **<span** style="font-size:14px;"**>**    <!-- List of regexpressions describing the interface (if any) that represent tetherable
2. Wifi interfaces.  If the device doesn't want to support tethering over Wifi this
3. should be empty.  An example would be "softap.\*" --**>**
4. **<string-array** translatable="false" name="config\_tether\_wifi\_regexs"**>**
5. **</string-array>**
6. **</span>**

这里看到需要在此默认的config.xml中的   name="config\_tether\_wifi\_regexs" 子标签下添加用作softap的接口(一般为wlan0 或者eth0 视你的WIFI驱动程序而定)

修改后的标签：

**[html]** [view plain](http://blog.csdn.net/jshazk1989/article/details/17094415) [copy](http://blog.csdn.net/jshazk1989/article/details/17094415)

1. **<span** style="font-size:14px;"**>**    <!-- List of regexpressions describing the interface (if any) that represent tetherable
2. Wifi interfaces.  If the device doesn't want to support tethering over Wifi this
3. should be empty.  An example would be "softap.\*" --**>**
4. **<string-array** translatable="false" name="config\_tether\_wifi\_regexs"**>**
5. **<item>**"wlan0"**</item>**
6. **</string-array>**
7. **</span>**

OK 此处mTethering.getTetherableWifiRegexs().length!=0 返回true， 整个isTetheringSupported 返回真

## 连接窗口

连接对话框：WifiDialog

WifiDialog **extends** AlertDialog

Title的颜色

**super**(context,R.style.***Theme\_AlertDialog***);

### EAP方法

### 阶段2身份验证

### CA证书

### 身份

### 匿名身份

### 密码

### 显示密码

### 高级选项

#### 代理

#### IP设置

## 配置WLAN：ConfigureWifiSettings

F

((SettingsActivity) getActivity()).startPreferencePanel(  
 ConfigureWifiSettings.**class**.getCanonicalName(), **null**,  
 R.string.wifi\_configure\_titlebar, **null**, **this**, 0);

### 网络通知

### 在休眠状态下保持WLAN网络连接

### MAC地址；IP地址

## 更多

### 刷新

### 高级选项

# 蓝牙

## 设备列表

## 设置

### 刷新

### 重命名此设备

### 显示收到的文件

# 流量使用情况

### 数据流量

入口类:DataUsageSummary

Kw: DataUsageSummary 源码

## 流量消耗情况

## 流量节省程序DataSaverSummary

DataSaverSummary **extends** SettingsPreferenceFragment

## WLAN网络流量消耗

## 网络流量限制

## 以太网流量消耗

# 飞行模式

# 网络共享和便携式热点

## USB网络共享

## 便携式WLAN热点

## 设置wlan热点 网络名称

### 安全性

### 密码

### 显示密码

### 选择AP频段

## 蓝牙网络共享

# VPN

## VPN列表

## 编辑VPN配置文件

### 名称

### 类型

### 服务器地址

### PPP加密

### 显示高级选项

#### DNS搜索网域名

#### DNS服务器

#### 转发路线

### 用户名

### 密码

### 始终开始的VPN

# 移动网络

## 移动数据网络漫游

## 首选网络类型

## 接入点名称（APN）

## 网络运营商

# 重置网络设置

## 重置设置

# Ethernet

## ip address

## netmask

## gateway

## dns1

## dns2

## Ethernet ip mode

# 显示设置/DisplaySettings/ SettingsPreferenceFragment

DisplaySettings **extends** SettingsPreferenceFragment **implements** Preference.OnPreferenceChangeListener, Indexable

addPreferencesFromResource(R.xml.***display\_settings***);

## 亮度

## 自动调节亮度

<**PreferenceScreen  
 android:key="brightness"  
 android:title="@string/brightness"  
 settings:keywords="@string/keywords\_display\_brightness\_level"**>  
 <**intent android:action="android.intent.action.SHOW\_BRIGHTNESS\_DIALOG"** />  
</**PreferenceScreen**>

最终调起

com.android.systemui/.settings.BrightnessDialog

## 壁纸WallpaperTypeSettings

ACTION\_SET\_WALLPAPER = "android.intent.action.SET\_WALLPAPER";

**final** List<ResolveInfo> rList = pm.queryIntentActivities(intent,  
 PackageManager.***MATCH\_DEFAULT\_ONLY***);

addPreferencesFromResource(R.xml.***wallpaper\_settings***);

该 Intent 常量是一个 String，表示启用设置壁纸的 Activity，也就是说只要我们的系统中有这样的 Activity（action 为 android.intent.action.SET\_WALLPAPER）就可以出现在选择器中。

原生的 android 系统中有三个这样的 Activity:

1）. WallpaperChooser.java

这是 Launcher 中的一个类，主要是选择壁纸的操作，和 Launcher.java 在一个包下面。通过 Launcher 的 Manifest.xml 文件就可以看到答案：

<activity

android:name="com.android.launcher2.WallpaperChooser"

android:label="@string/pick\_wallpaper"

android:icon="@drawable/ic\_launcher\_wallpaper"

android:screenOrientation="nosensor"

android:finishOnCloseSystemDialogs="true">

<intent-filter>

<action android:name="android.intent.action.SET\_WALLPAPER"/>

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

</intent-filter>

</activity>

2）. LiveWallpaperListActivity.java

位于 /packages/wallpapers/LivePicker/src/com/android/wallpaper/livepicker 下面，主要是选择动态壁纸。其 Manifest.xml 文件：

<activity android:name="LiveWallpaperListActivity"

android:icon="@drawable/ic\_launcher\_live\_wallpaper"

android:label="@string/live\_wallpaper\_picker\_title"

android:theme="@android:style/Theme.NoTitleBar"

android:screenOrientation="nosensor">

<intent-filter>

<action android:name="android.service.wallpaper.LIVE\_WALLPAPER\_CHOOSER" />

<action android:name="android.intent.action.SET\_WALLPAPER" />

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

</intent-filter>

</activity>

3）. Photographs.java

在以前的版本中，android 使用的是Gallery，现在改变为 Gallery3D，

位于 /packages/apps/Gallery3D/src/com/cooliris/media，对应的 Manifest.xml 文件可自行查阅。

### 图库

### 动态壁纸

### 壁纸

dumpsys activity | grep mFoc

com.android.wallpaperpicker/.WallpaperPickerActivity t458}

## 休眠TimeoutListPreference/RestrictedListPreference

Gf

**com.android.settings.TimeoutListPreference**

TimeoutListPreference **extends** RestrictedListPreference

## 屏保

### 列表选项

#### 万花筒

#### 照片桌面

#### 相框

### 更多

#### 立即启动

#### 启动屏保的时间

## 字体大小

## 显示大小

## 投射

### 投射列表

### 更多

#### 开启无线显示

## HDMI （HdmiSettings/ SettingsPreferenceFragment）

分辨率设置

屏幕缩放

屏幕旋转

开关->蓝色风格

### HDMI输出模式

ListPreference

Dlg被重载掉了。。

开关，dlg都不对，为啥呢

# 通知设置

## 应用列表

### 显示时不发出提示音

### 屏幕锁定时

#### 显示所有通知内容

#### 隐藏敏感通知内容

#### 完全不显示通知

### 覆盖勿扰设置

## 设置

### 屏幕锁定时

#### 显示所有通知内容

#### 隐藏敏感通知内容

#### 完全不显示通知

## 更多

### 显示系统进程

### 重置应用偏好设置

# 声音 SoundSettings

VolumeSeekBarPreference

R.raw.***media\_volume***

首先列出一个代码路线：

SoundSettings.java --->

MediaVolumePreferenceController.java --->

VolumeSeekBarPreferenceController.java --->

VolumeSeekBarPreference.java --->

SeekBarVolumizer.java

详细分析：

SoundSettings.java： 该类文件就是设置-声音设置页面的Fragment代码。首先请看buildPreferenceControllers()方法，该方法将所有的PreferenceController用一个List装起来，我们找到MediaVolumePreferenceController；

MediaVolumePreferenceController.java ： 这个类没什么好说的，就是一些属性获取的接口，我们可以直接看父类VolumeSeekBarPreferenceController；

VolumeSeekBarPreferenceController.java ： 这个类也非常简单，主要操作就是在方法displayPreference()中，该方法对VolumeSeekBarPreference进行初始化，并设置一些属性。

## 设计原理

### VolumeSeekBarPreference.java

该类需要关注init()方法，该方法获取播放铃声的uri。

final Uri sampleUri = mStream == AudioManager.STREAM\_MUSIC ? getMediaVolumeUri() : null;

另外，初始化了一个SeekBarVolumizer，具体代码如下：

if (mVolumizer == null) {

mVolumizer = new SeekBarVolumizer(getContext(), mStream, sampleUri, sbvc);

}

mVolumizer.setSeekBar(mSeekBar); // 此方法将界面中的seekbar赋给mVolumizer

### SeekBarVolumizer.java

SeekBarVolumizer implements OnSeekBarChangeListener, Handler.Callback

SeekBarVolumizer .setSeekBar方法给seekbar设置了一个监听OnSeekBarChangeListener。

seekBar监听器有三个需要实现的方法：

onProgressChanged() //滑动过程中调用

onStartTrackingTouch() //刚开始滑动时调用

onStopTrackingTouch() //结束滑动时调用

#### SeekBarVolumizer.SeekBarVolumizer

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromTouch) {

if (fromTouch) {

postSetVolume(progress);

}

if (mCallback != null) {

mCallback.onProgressChanged(seekBar, progress, fromTouch);

}

}

在onProgressChanged()中调用postSetVolume()，postSetVolume()中采用handler机制发送消息：MSG\_SET\_STREAM\_VOLUME，消息处理代码：

#### handleMessage

@Override

public boolean handleMessage(Message msg) {

switch (msg.what) {

case MSG\_SET\_STREAM\_VOLUME:

if (mMuted && mLastProgress > 0) {

mAudioManager.adjustStreamVolume(mStreamType, AudioManager.ADJUST\_UNMUTE, 0);

} else if (!mMuted && mLastProgress == 0) {

**mAudioManager.adjustStreamVolume(mStreamType, AudioManager.ADJUST\_MUTE, 0);**

}

mAudioManager.setStreamVolume(mStreamType, mLastProgress,

AudioManager.FLAG\_SHOW\_UI\_WARNINGS);

break;

case MSG\_START\_SAMPLE:

onStartSample();

break;

case MSG\_STOP\_SAMPLE:

onStopSample();

break;

case MSG\_INIT\_SAMPLE:

onInitSample();

break;

default:

Log.e(TAG, "invalid SeekBarVolumizer message: "+msg.what);

}

return true;

}

mAudioManager.setStreamVolume(mStreamType, mLastProgress,

AudioManager.FLAG\_SHOW\_UI\_WARNINGS);

此处就是音量调节的主要代码。

#### postStartSample

其次，在结束滑动时会有声音发出供用户识别音量，在onStopTrackingTouch()中调postStartSample()方法，postStartSample()方法也是通过handler发送消息进行处理，调用到onStartSample()，该方法会播放铃声mRingtone.play()；

以上就是比较浅层的设置中调节音量的代码逻辑。

其中的两个主要方法还需要继续细细品味mAudioManager.setStreamVolume和mRingtone.play

## 媒体音量

## 闹钟音量

## 通知音量

VolumeSeekBarPreference

**public void** onMuted(**boolean** muted, **boolean** zenMuted) {

静音原理

## 勿扰ZenModeSettings

Android N去除了情景模式，取而代之的是勿扰模式。勿扰模式的入口有两处，下拉栏和设置声音里面。下面我们就从设置声音入口，看看勿扰模式的设置流程。

首先，勿扰模式的首页有三种选项，分别是仅允许优先打扰内容、自动规则和屏蔽视觉打扰。各页面分别对应的类文件如下：

首页 ：ZenModeSettings.java 用于设置入口，更新一些summary；

仅允许优先打扰内容：ZenModePrioritySettings.java 设置可以打扰的内容；

自动规则：ZenModeAutomationSettings.java 有时间规则和时间规则；

屏蔽视觉打扰：ZenModeVisualInterruptionSettings.java 通知是否显示屏幕

NotificationManager notificationManager = **mManager**.getContext().getSystemService(  
 NotificationManager.**class**);  
notificationManager.setZenMode(Settings.Global.ZEN\_MODE\_OFF, **null**, ***TAG***);

<https://www.jianshu.com/p/b7665cfe95c8>

### ZenModeSettingsBase

在ZenModeSettingsBase中有一个mRules的集合，存储着用户自己定义的AutomaticZenRule，而mRules的值是NotificationManager中获取的的。也就是这里规则的维护存储也还是在NotificationManager中，如下代码。

abstract public class ZenModeSettingsBase extends RestrictedSettingsFragment {

...

protected Context mContext;

protected Set<Map.Entry<String, AutomaticZenRule>> mRules;

...

private void updateZenMode(boolean fireChanged) {

final int zenMode = Settings.Global.getInt(getContentResolver(), Global.ZEN\_MODE, mZenMode);

if (zenMode == mZenMode) return;

mZenMode = zenMode;

if (DEBUG) Log.d(TAG, "updateZenMode mZenMode=" + mZenMode);

if (fireChanged) {

onZenModeChanged();

}

}

protected String addZenRule(AutomaticZenRule rule) {

try {

String id = NotificationManager.from(mContext).addAutomaticZenRule(rule);

final AutomaticZenRule savedRule =

NotificationManager.from(mContext).getAutomaticZenRule(id);

maybeRefreshRules(savedRule != null, true);

return id;

} catch (Exception e) {

return null;

}

}

protected boolean setZenRule(String id, AutomaticZenRule rule) {

final boolean success =

NotificationManager.from(mContext).updateAutomaticZenRule(id, rule);

maybeRefreshRules(success, true);

return success;

}

protected boolean removeZenRule(String id) {

final boolean success =

NotificationManager.from(mContext).removeAutomaticZenRule(id);

maybeRefreshRules(success, true);

return success;

}

protected void maybeRefreshRules(boolean success, boolean fireChanged) {

if (success) {

mRules = getZenModeRules();

if (DEBUG) Log.d(TAG, "Refreshed mRules=" + mRules);

if (fireChanged) {

onZenModeConfigChanged();

}

}

}

protected void setZenMode(int zenMode, Uri conditionId) {

NotificationManager.from(mContext).setZenMode(zenMode, conditionId, TAG);

}

private Set<Map.Entry<String, AutomaticZenRule>> getZenModeRules() {

Map<String, AutomaticZenRule> ruleMap

= NotificationManager.from(mContext).getAutomaticZenRules();

return ruleMap.entrySet();

}

...

知道上面的东西，ZenModeAutomationSettings就简单多了，就是mRules列表的展示，以及添加删除等操作。只是这里面添加可以选择两种模式，分别是活动规则和时间规则。分别对应的类是ZenModeEventRuleSettings和ZenModeScheduleRuleSettings，他们都继承自ZenModeRuleSettingsBase。这里ZenModeAutomationSettings虽然代码很多，但都是各种弹出框的操作，就不分析了。主要看看活动规则和时间规则这两个里面分别又有哪些操作吧。

时间规则的列表，分别有规则名称、星期几、开始时间、结束时间、勿扰、闹钟响铃时间可覆盖结束时间（在所设结束时间或下一次闹钟响铃时（两者选其先）停止）。由private ScheduleInfo mSchedule;中来存储着时间规则的一些数据。

活动规则的页面的列表，分别是名字、在以下日历活动期间、回复内容如下的活动、勿扰。由private EventInfo mEvent;中来存储着时间规则的一些数据。以上两个规则里面其余都只是进行一些修改操作。这些操作最后的修改的数据都会通过调用ZenModeRuleSettingsBase里面的updateRule()方法保存到NotificationManager里。代码如下

#### ZenModeScheduleRuleSettings

Fsg

mStart = new TimePickerPreference(getPrefContext(), mgr);

mStart.setKey(KEY\_START\_TIME);

mStart.setTitle(R.string.zen\_mode\_start\_time);

mStart.setCallback(new TimePickerPreference.Callback() {

@Override

public boolean onSetTime(final int hour, final int minute) {

if (mDisableListeners) return true;

if (!ZenModeConfig.isValidHour(hour)) return false;

if (!ZenModeConfig.isValidMinute(minute)) return false;

if (hour == mSchedule.startHour && minute == mSchedule.startMinute) {

return true;

}

if (DEBUG) Log.d(TAG, "onPrefChange start h=" + hour + " m=" + minute);

mSchedule.startHour = hour;

mSchedule.startMinute = minute;

updateRule(ZenModeConfig.toScheduleConditionId(mSchedule));

return true;

}

Settings/src/com/android/settings/notification/ZenModeRuleSettingsBase.java

protected void updateRule(Uri newConditionId) {

mRule.setConditionId(newConditionId);

setZenRule(mId, mRule);

}

protected boolean setZenRule(String id, AutomaticZenRule rule) {

final boolean success =

NotificationManager.from(mContext).updateAutomaticZenRule(id, rule);

maybeRefreshRules(success, true);

return success;

}

自此所有在Setting里面的勿扰模式的代码分析结束了，综上可知，Settings模块中勿扰只是一个界面的展示，其数据都放在NotificationManager进行管理操作。

### 仅允许优先打扰内容ZenModePrioritySettings

设置在勿扰处于仅限优先打扰的时候，这种情况下，哪些联系人或者提醒等可以通知您。

首先在OnCreate方法中获得到了一个Policy（notificationManager的内部类）的对象

mPolicy = NotificationManager.from(mContext).getNotificationPolicy();

然后对各个Preference进行监听，获得的值通过savePolicy（...）来保存。savePolicy（...）方法内部，其实是调用了NotificationManager的setNotificationPolicy方法。

首先它负责自定义优先打扰的内容，其中包括闹钟、提醒、活动、消息、通话（仅限来自联系人）、重复来电者（如果同一个人在15分钟内第二次来电，则允许显示通知）。这面也仅是一些开关，将最后的更改用NotificationManager.from(mContext).setNotificationPolicy(mPolicy);进行保存，其中也只有消息和通话需要选择，我们来看一下通话的设置以及保存，其余的类似：

#### 闹钟

#### 提醒

#### 活动

#### 消息

#### 通话

#### 重复来电者

### 自动规则ZenModeAutomationSettings

可以自定义多个自动规则，这些规则指哪些活动或者时间内自动开启勿扰模式。同时可以开启闹钟响铃时结束勿扰模式

自动规则：ZenModeAutomationSettings.java 有时间规则和时间规则；

#### 列表

#### 添加规则

### 屏蔽视觉打扰 ZenModeVisualInterruptionSettings

屏蔽视觉打扰：ZenModeVisualInterruptionSettings.java 通知是否显示屏幕

处于勿扰模式时的两种更优化方案的开关

#### 屏幕开启时屏蔽

禁止在勿扰模式下被静音的通知在屏幕上短暂显示或弹出

#### 屏幕关闭时屏蔽

禁止在勿扰模式下被静音的通知开启屏幕

### REF

[Android7.1勿扰功能简析](https://blog.csdn.net/tubby_ting/article/details/54924392)

## 默认通知铃声

## 默认闹钟铃声

## 其他提示音

### 屏幕锁定提示音

### 充电提示音

### 触摸提示音

## 投射

### 设备列表

### 更多

#### 开启无线显示

# 应用管理

## 应用列表

### 软件名

### 版本

### 停用/强制停止

### 存储

### 流量使用情况

#### 时间

#### 总计

#### 后台数据

#### 不限制数据流量

### 权限

#### 权限列表

#### 更多/显示全部权限

### 通知

#### 显示时不发出提示音

#### 屏幕锁定时

#### 覆盖勿扰设置

### 默认打开

### 电池

### 内存

## 设置

### 应用权限

#### 位置信息

#### 存储空间

#### 日历

#### 电话

#### 相机

#### 短信

#### 身体传感器

#### 通讯录

#### 麦克风

### 打开连接

#### 在应用中打开连接

### 辅助应用和语音输入

#### 辅助应用

### 主屏幕应用

### 浏览器应用

### 紧急警报应用

### 特殊访问权限

#### 电池优化

#### 设备管理器

#### 勿扰权限

#### 出现在其他应用上

#### VR助手服务

#### 修改系统设置

#### 通知使用权

#### 付费短信权限

#### 不受流量控制

#### 使用情况访问权限

## 更多

### 显示系统进程

### 重置应用偏好设置

# 存储管理

## 内部存储使用情况

## 内部共享存储空间

### 应用

### 图片

### 视频

### 音频

### 系统

### 其他

### 缓存数据

### 浏览

## SD卡

### 根目录列表

### 搜索

### 改变显示方式

### 更改排序方式

### 更多

#### 新建窗口

#### 新建文件夹

#### 显示内部存储设备

#### 存储设置

# 内存管理

## 平均内存使用量

## 各个应用使用的内存

# 用户

## 用户和个人资料

## 添加用户或个人资料

### 用户

### 受限个人资料

## 添加用户（设备锁定时）

# 设备-通知

## Zen

zen\_mode

**public static final** String ***ZEN\_MODE*** = **"zen\_mode"**;  
  
*/\*\** ***@hide*** *\*/* **public static final int *ZEN\_MODE\_OFF*** = 0;  
*/\*\** ***@hide*** *\*/* **public static final int *ZEN\_MODE\_IMPORTANT\_INTERRUPTIONS*** = 1;  
*/\*\** ***@hide*** *\*/* **public static final int *ZEN\_MODE\_NO\_INTERRUPTIONS*** = 2;  
*/\*\** ***@hide*** *\*/* **public static final int *ZEN\_MODE\_ALARMS*** = 3;  
  
*/\*\** ***@hide*** *\*/* **public static** String zenModeToString(**int** mode) {  
 **if** (mode == ***ZEN\_MODE\_IMPORTANT\_INTERRUPTIONS***) **return "ZEN\_MODE\_IMPORTANT\_INTERRUPTIONS"**;  
 **if** (mode == ***ZEN\_MODE\_ALARMS***) **return "ZEN\_MODE\_ALARMS"**;  
 **if** (mode == ***ZEN\_MODE\_NO\_INTERRUPTIONS***) **return "ZEN\_MODE\_NO\_INTERRUPTIONS"**;  
 **return "ZEN\_MODE\_OFF"**;  
}

adb shell settings get global zen\_mode

adb shell settings put global zen\_mode 2

settings put global zen\_mode 2

# 位置信息LocationSettings

## 位置模式

### 开关逻辑

Sdfg

**if** (isChecked) {  
 setLocationMode(android.provider.Settings.Secure.LOCATION\_MODE\_PREVIOUS);  
} **else** {  
 setLocationMode(android.provider.Settings.Secure.***LOCATION\_MODE\_OFF***);  
}

**public void** setLocationMode(**int** mode) {  
 **if** (isRestricted()) {  
 *// Location toggling disabled by user restriction. Read the current location mode to  
 // update the location master switch.* **if** (Log.*isLoggable*(***TAG***, Log.***INFO***)) {  
 Log.*i*(***TAG***, **"Restricted user, not setting location mode"**);  
 }  
 mode = Settings.Secure.*getInt*(getContentResolver(), Settings.Secure.***LOCATION\_MODE***,  
 Settings.Secure.***LOCATION\_MODE\_OFF***);  
 **if** (**mActive**) {  
 onModeChanged(mode, **true**);  
 }  
 **return**;  
 }  
 Intent intent = **new** Intent(***MODE\_CHANGING\_ACTION***);  
 intent.putExtra(***CURRENT\_MODE\_KEY***, **mCurrentMode**);  
 intent.putExtra(***NEW\_MODE\_KEY***, mode);  
 getActivity().sendBroadcast(intent, android.Manifest.permission.***WRITE\_SECURE\_SETTINGS***);  
 Settings.Secure.*putInt*(getContentResolver(), Settings.Secure.***LOCATION\_MODE***, mode);  
 refreshLocationMode();  
}

### adb 命令

Fda

**mLocationMode**.setOnPreferenceClickListener(  
 **new** Preference.OnPreferenceClickListener() {  
 @Override  
 **public boolean** onPreferenceClick(Preference preference) {  
 activity.startPreferencePanel(  
 LocationMode.**class**.getName(), **null**,  
 R.string.***location\_mode\_screen\_title***, **null**, LocationSettings.**this**,  
 0);  
 **return true**;  
 }  
 });

**com.android.settings.lo**c**ation.MODE\_CHANGING**

am broadcast -a com.android.settings.location.MODE\_CHANGING --ei CURRENT\_MODE 3 --ei NEW\_MODE 0

settings put secure location\_providers\_allowed “”

settings put secure location\_providers\_allowed “network,gps”

### 高精确度

### 低耗电量

### 仅限设备

## 扫描

# 安全

## 屏幕锁定

### 当前屏幕锁定设置

### 活动

### 图案

### PIN码

### 密码

## 加密平板电脑

## 显示密码

## 设备管理器

## 未知来源

# 账号

## 账号列表

## 添加账号

Settings$ChooseAccountActivity

## 自动同步数据

# 个人-语言和输入法

白色分割线

## 语言com.android.settings.localepicker.LocaleListEditor

### 语言列表

列表白色->黑色

### 添加语言

**this**.mSearchView = (SearchView)searchMenuItem.getActionView();

### 移除

菜单黑色

## 拼写检查工具

### 开关

### 语言

弹窗

文字颜色

### 默认拼写检查工具

## 虚拟键盘

分隔线，字体颜色

### 键盘

黑色主题吧，白色主题？

青色title

知己

com.android.inputmethod.latin调用的是这个，UI定制有坑啊。。必须搞主题引擎了。

#### 语言

分隔线，开关

禁用颜色

#### 偏好设置

背景，字体颜色，开关颜色

#### 外观和布局

##### 主题背景

相关类：VirtualKeyboardFragment

InputMethodPreference：输入法列表

主题引擎来了

com.android.inputmethod.latin/.LatinIME

Intent { act=android.intent.action.MAIN cmp=com.android.inputmethod.latin/.settings.SettingsActivity }

##### 自定义输入样式

#### 华航输入

#### 文字更正

个人字典奔溃？？？

附加字典

### 管理键盘

背景，文字颜色，开关风格

## 实体键盘

文字，开关，分割线

## 文字转语音（TTS）输出

## 指针速度

奔溃了，配置有问题啊

# 备份和重置

## 恢复出厂设置

最近看恢复出厂的一个问题，以前也查过这方面的流程，所以这里整理一些AP+framework层的流程；

入口：setting-->备份与重置--->恢复出厂设置--->重置手机--->清除全部内容：**自动执行-->手机关机--->开机--->进行恢复出厂的操作--->开机流程**；

需要root权限

App层: sendBroadcast(new Intent("Android.intent.action.MASTER\_CLEAR"));

Adb层:

恢复出厂设置，删除虚拟机资源文件缓存

adb root

adb shell am broadcast -a android.intent.action.MASTER\_CLEAR

发出了这个广播，后台server会完成后续操作。主意应用程序要有android.permission.MASTER\_CLEAR 这个权限

## 恢复出厂设置

### 回复平板电脑出厂设置

文字颜色！！

# 日期和时间设置

分割线

开关风格

白色背景！字体黑色！

## 日期和时间

com.android.settings. DateTimeSettings

((AlarmManager) context.getSystemService(Context.ALARM\_SERVICE)).setTime(when);

IAlarmManager mService.setTime(millis);

## 设置日期

白色

## 设置时间

## 选择时区

### 时区列表

### 排序

## 使用24小时制

# 无障碍

## 服务

## 字母

## 放大手势

## 字体大小

## 显示大小

## 指针停止移动后点击

## 高对比文字

## 输出密码

## 大号鼠标指针

## 单声道音频

## 无障碍快捷方式

## 文字转语音（TTS）输出

## 触摸和按住延迟

## 颜色反转

## 色彩校正

# 开发者选项

## And7 开发者选项DevelopmentSettings

# 关于遥控器

## 状态信息

### 电池状态

### 电池电量

### sim卡状态

### imei信息

### IP地址

### WLANMAC地址

### 蓝牙地址

### 序列号

### 已开机时间

## 法律信息

### 开源代码许可

### 系统webview许可

### 壁纸

## 监管标签

## 型号

## Android版本

## andriod安全补丁程序级别

## 内核版本

## 系统版本

## 状态信息

# 其他

## CryptKeeper

## FallbackHome

<https://blog.csdn.net/fu_kevin0606/article/details/65437594>

<!-- Triggered when user-selected home app isn't encryption aware -->

adb shell am start -n com.android.settings/com.android.settings.FallbackHome