# 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。

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);

## 小结：Get/set方法

### 脚本命令settings

settings get system ringtone

settings get global ringtone

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

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

ethernet

## 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)

# Settings

## 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）

。



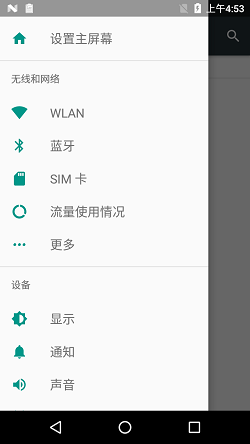
<https://blog.csdn.net/qq_31012033/article/details/79522813>

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。



### 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

## Android7定制

### 加载选项

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。

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";

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());

并且在AndroidManifest.xml通过meta-data配置了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。

#### 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

## 网络

### 数据流量

入口类:DataUsageSummary

Kw: DataUsageSummary 源码

## 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

## 蓝牙

## 流量使用情况

## 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: 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

## 日期和时间

com.android.settings. DateTimeSettings

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

IAlarmManager mService.setTime(millis);

## 跳转到各个系统设置的方法

// 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>

## And7 开发者选项DevelopmentSettings