# 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**

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

### 还是用navicat看吧

### sqlite3

# 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：



Ctrl+Z推出命令模式

rm500:/data/system\_de/0 # sqlite3 accounts\_de.db

sqlite> .databases

seq name file

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

0 main /data/system\_de/0/accounts\_de.db

sqlite> .dump shared\_accounts

PRAGMA foreign\_keys=OFF;

BEGIN TRANSACTION;

CREATE TABLE shared\_accounts ( \_id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, type TEXT NOT NULL, UNIQUE(name,type));

COMMIT;

sqlite> .show

echo: off

eqp: off

explain: off

headers: off

mode: list

nullvalue: ""

output: stdout

colseparator: "|"

rowseparator: "\n"

stats: off

width:

**sqlite> select \* from accounts;**

[7|key.guan@kji.com|com.kji.oo||1535041382744](mailto:7|key.guan@kji.com|com.kji.oo||1535041382744)

就一个账户名字，password呢，收藏这些数据怎么来的？app2再登陆一次？？

## 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)

<https://www.jianshu.com/p/59302a400944>

Android 系统提供媒体库 URI 与 数据库的对应关系

