**详解Android 7.0 Settings 加载选项**

 更新时间：2017年02月14日 14:13:03   作者：nearbyYoung   [[https://www.jb51.net/skin/2018/images/text-message.png](https://www.jb51.net/article/105558.htm#comments) 我要评论](https://www.jb51.net/article/105558.htm#comments)

本篇文章主要介绍了Android 7.0 Settings 加载选项，Android 7.0 Settings顶部多了一个建议选项，多了个侧边栏，操作更加便捷了，有兴趣的可以了解一下。

先写在前面，这说的Settings加载选项是指Settings这个应用显示在主界面的选项，这个修改需要对系统源码进行修改。

Android 7.0 Settings顶部多了一个建议选项，多了个侧边栏，操作更加便捷了。

                  原生7.0主界面                                                          原生7.0侧边栏

**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。

**Android 7.0**

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

如果需要Settings中添加打包好的apk，需要三个步骤。

1. 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

复制代码代码如下:

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

1、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。最好是有该apk提供公司的技术支持，不然有混淆的代码就比较难进行更改、回编译了。

/\*\*

\* The key used to get the category from metadata of activities of action

\* {@link #EXTRA\_SETTINGS\_ACTION}

\* The value must be one of:

\* <li>com.android.settings.category.wireless</li>

\* <li>com.android.settings.category.device</li>

\* <li>com.android.settings.category.personal</li>

\* <li>com.android.settings.category.system</li>

\*/

private static final String EXTRA\_CATEGORY\_KEY = "com.android.settings.category";

/\*\*

\* Name of the meta-data item that should be set in the AndroidManifest.xml

\* to specify the icon that should be displayed for the preference.

\*/

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

/\*\*

\* Name of the meta-data item that should be set in the AndroidManifest.xml

\* to specify the title that should be displayed for the preference.

\*/

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

/\*\*

\* Name of the meta-data item that should be set in the AndroidManifest.xml

\* to specify the summary text that should be displayed for the preference.

\*/

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

在TileUtils.java的白名单中添加对应的包名。

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。

相比android6.0，在7.0上添加功能选项显得更简单了，只需要写好了功能，在Settings的AndroidManifest.xml中添加必要的参数，或者是在独立的apk中AndroidManifest.xml中配置必要的参数，再在TileUtils.java中添加包名就好了。需要修改的地方更少了，并且Google将很方方法放到了com.android.settinglibs里，精简了一部分代码。

### [Android 4.4 Settings 应用初步分析](https://862123204-qq-com.iteye.com/blog/2144790)

**博客分类：**

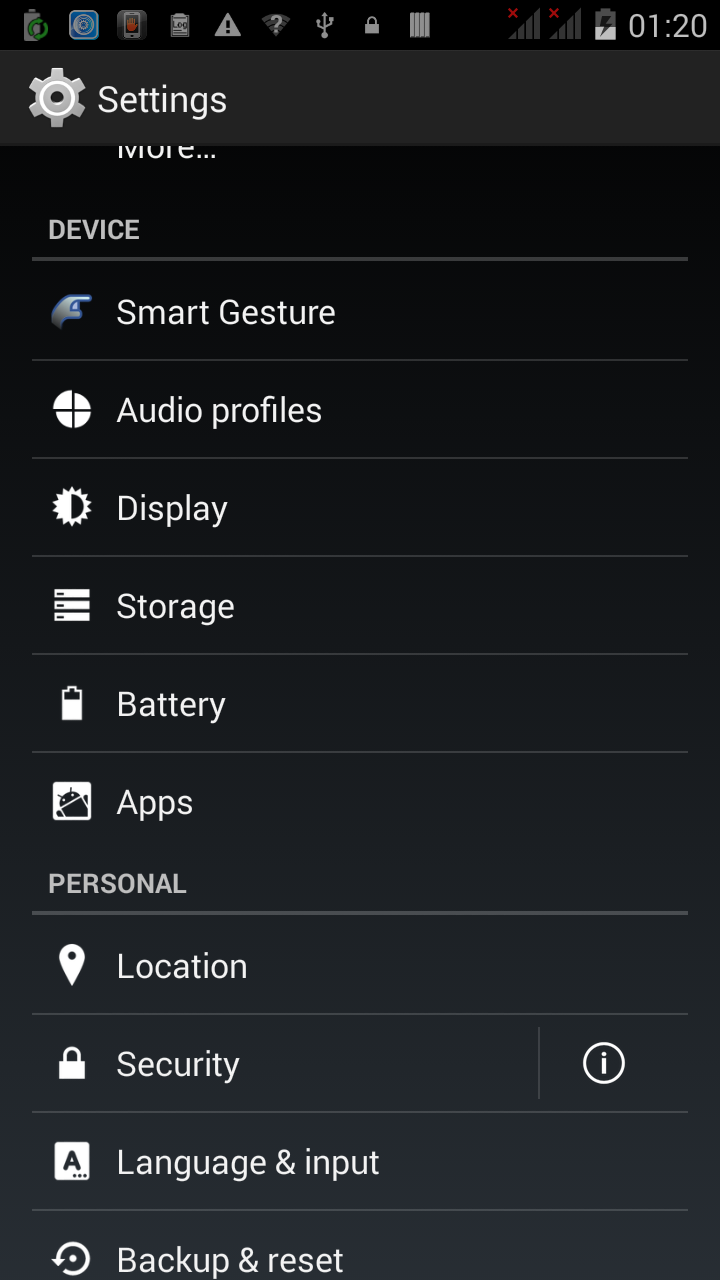
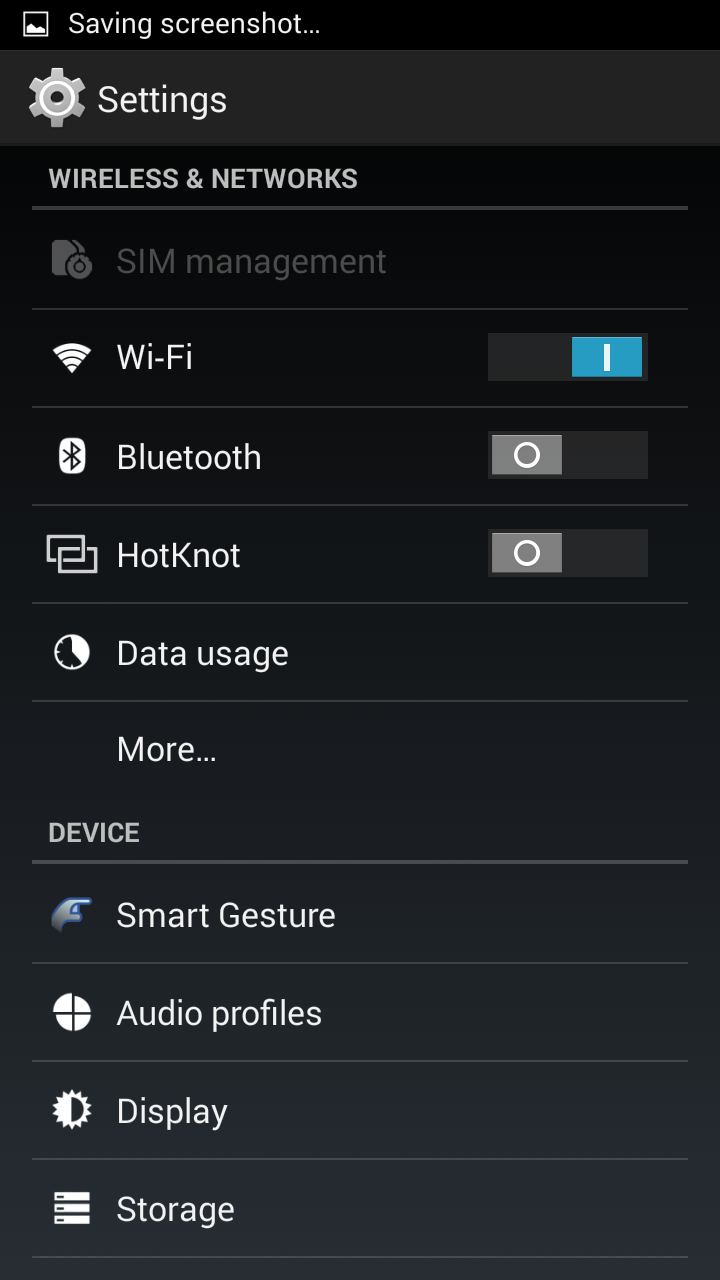
* [Android](https://862123204-qq-com.iteye.com/category/232019)

[Android Setting](http://www.iteye.com/blogs/tag/Android%20Setting)

一次偶然要在设置里面增加一个菜单,需要修改到settings\_headers.xml 文件(res/layout/xml) 文件,所以就觉得要看一下这个流程.就做一下笔记,语言组织能力不行啊.

分析Android 源码的时候导入单个应用的时候一般是会有很多错误的,因为需要导入系统编译之后生成的jar包才能消除eclipse 里面的哪些红色xx.

**1.Settings的UI**



**2.流程分析**

从AndroidManifest.xml 中查看

<category android:name="android.intent.category.LAUNCHER" /> 知道Settings.java 是这个应用入口activity.

Settings 继承了PreferenceActivity .他的布局文件是settings\_headers.xml

这个文件里面都是这些header,效果可以参考上面的效果图1和图2.

**Xml代码  [收藏代码](javascript:void())**

1. <!-- WIRELESS and NETWORKS  分类-->
2. **<header** android:id="@+id/wireless\_section"
3. android:title="@string/header\_category\_wireless\_networks" **/>**
5. <!-- Sim management 普通项-->
6. **<header**
7. android:id="@+id/sim\_settings"
8. android:icon="@drawable/ic\_settings\_dualsim"
9. android:fragment="com.mediatek.gemini.SimManagement"
10. android:title="@string/gemini\_sim\_management\_title" **/>**

com.android.settings.Settings.java 这个activity 是通过回调onBuildHeaders方法来加载进入应用之后的第一个布局文件的,然后调用 loadHeadersFromResource(R.xml.settings\_headers, headers);

来解析 文件.

onBuildHeaders 和loadHeadersFromResource 方法都是父类PreferenceActivity 的方法.

Settings.java 重写onBuildHeaders 方法的实现的源码如下:

**Java代码  [收藏代码](javascript:void())**

1. /\*\*
2. \* Populate the activity with the top-level headers.
3. \*/
4. @Override
5. **public** **void** onBuildHeaders(List<Header> headers) {
6. **if** (!onIsHidingHeaders()) {
7. PDebug.Start("loadHeadersFromResource");
8. loadHeadersFromResource(R.xml.settings\_headers, headers);
9. PDebug.End("loadHeadersFromResource");
10. updateHeaderList(headers);
11. }
12. }

loadHeadersFromResource 方法就是解析settings\_headers.xml 文件并保持相关的数据到List<Header> headers 里面.

Header 定义很多变量来和settings\_headers.xml 里面节点一一对应,

public long id = HEADER\_ID\_UNDEFINED;

public int titleRes;

public CharSequence title;

public String fragment;

public Bundle fragmentArguments;

public Intent intent;

public Bundle extras;

………

通过跟踪Setting.java 的父类(PreferenceActivity)的继承关系知道他其实也是一个ListActivity.java ,全部的设置项也是使用ListView来显示的.

HeaderAdapter这个适配类是Setting.java 的内部类,它会判断之后来加载对应的view和数据来显示UI.

HeaderAdapter已经定义了4中类型的View 类型

**Java代码  [收藏代码](javascript:void())**

1. **static** **final** **int** HEADER\_TYPE\_CATEGORY = 0;//用来分类的
2. **static** **final** **int** HEADER\_TYPE\_NORMAL = 1;//常规项
3. **static** **final** **int** HEADER\_TYPE\_SWITCH = 2;//开关项
4. **static** **final** **int** HEADER\_TYPE\_BUTTON = 3;//按钮项

前3种应该都见过,为了让大家看到第4项,我把稍微修改了一下我的HeaderAdapter源码(见getview方法的中的有//add的部分),也就是上面图2中的security 选项.

HeaderAdapter 的getHeaderType 方法决定了配置在settings\_headers.xml 里面的header的类型.

HeaderAdapter 的getView 方法根据header的类型 来加载对应的布局文件.

**Java代码  [收藏代码](javascript:void())**

1. **static** **int** getHeaderType(Header header) {
2. **if** (header.fragment == **null** && header.intent == **null**) {
3. **return** HEADER\_TYPE\_CATEGORY;
4. } **else** **if** (header.id == R.id.wifi\_settings || header.id == R.id.bluetooth\_settings
5. || header.id == R.id.hotknot\_settings) {
6. **return** HEADER\_TYPE\_SWITCH;
7. } **else** **if** (header.id == R.id.security\_settings) {
8. **return** HEADER\_TYPE\_BUTTON;
9. } **else** {
10. **return** HEADER\_TYPE\_NORMAL;
11. }
12. }

但要注意的是在getView方法里面,当发现一个header 的类型是button的时候也会给header 的button增加一个onclick事件的,这个事件和header本事的onHeaderClick 是没有冲突的,因为2者不受同一个控件.

**Java代码  [收藏代码](javascript:void())**

1. @Override
2. **public** View getView(**int** position, View convertView, ViewGroup parent) {
3. HeaderViewHolder holder;
4. Header header = getItem(position);
5. **int** headerType = getHeaderType(header);
6. Log.d("zhangle","getHeaderType" + header.title  + " headerType=" + headerType);
7. View view = **null**;
9. **if** (convertView == **null**) {
10. holder = **new** HeaderViewHolder();
11. **switch** (headerType) {
12. **case** HEADER\_TYPE\_CATEGORY:
13. view = **new** TextView(getContext(), **null**,
14. android.R.attr.listSeparatorTextViewStyle);
15. holder.title = (TextView) view;
16. **break**;
18. **case** HEADER\_TYPE\_SWITCH:
19. view = mInflater.inflate(R.layout.preference\_header\_switch\_item, parent,
20. **false**);
21. holder.icon = (ImageView) view.findViewById(R.id.icon);
22. holder.title = (TextView)
23. view.findViewById(com.android.internal.R.id.title);
24. holder.summary = (TextView)
25. view.findViewById(com.android.internal.R.id.summary);
26. holder.switch\_ = (Switch) view.findViewById(R.id.switchWidget);
27. **break**;
29. **case** HEADER\_TYPE\_BUTTON:
30. view = mInflater.inflate(R.layout.preference\_header\_button\_item, parent,
31. **false**);
32. holder.icon = (ImageView) view.findViewById(R.id.icon);
33. holder.title = (TextView)
34. view.findViewById(com.android.internal.R.id.title);
35. holder.summary = (TextView)
36. view.findViewById(com.android.internal.R.id.summary);
37. holder.button\_ = (ImageButton) view.findViewById(R.id.buttonWidget);
38. holder.divider\_ = view.findViewById(R.id.divider);
39. **break**;
41. **case** HEADER\_TYPE\_NORMAL:
42. view = mInflater.inflate(
43. R.layout.preference\_header\_item, parent,
44. **false**);
45. holder.icon = (ImageView) view.findViewById(R.id.icon);
46. holder.title = (TextView)
47. view.findViewById(com.android.internal.R.id.title);
48. holder.summary = (TextView)
49. view.findViewById(com.android.internal.R.id.summary);
50. **break**;
51. }
52. view.setTag(holder);
53. } **else** {
54. view = convertView;
55. holder = (HeaderViewHolder) view.getTag();
56. }
58. // All view fields must be updated every time, because the view may be recycled
59. **switch** (headerType) {
60. **case** HEADER\_TYPE\_CATEGORY:
61. holder.title.setText(header.getTitle(getContext().getResources()));
62. **break**;
64. **case** HEADER\_TYPE\_SWITCH:
65. // Would need a different treatment if the main menu had more switches
66. **if** (header.id == R.id.wifi\_settings) {
67. mWifiEnabler.setSwitch(holder.switch\_);
68. } **else** **if** (header.id == R.id.bluetooth\_settings){
69. mBluetoothEnabler.setSwitch(holder.switch\_);
70. } **else** **if** (header.id == R.id.hotknot\_settings){
71. mHotKnotEnabler.setSwitch(holder.switch\_);
72. }
73. updateCommonHeaderView(header, holder);
74. **break**;
76. **case** HEADER\_TYPE\_BUTTON:
77. **if** (header.id == R.id.security\_settings) {
78. **boolean** hasCert = DevicePolicyManager.hasAnyCaCertsInstalled();
79. hasCert = **true**;//add
80. **if** (hasCert) {
81. holder.button\_.setVisibility(View.VISIBLE);
82. holder.divider\_.setVisibility(View.VISIBLE);
83. **boolean** isManaged = mDevicePolicyManager.getDeviceOwner() != **null**;
84. isManaged = **true**; //add
85. **if** (isManaged) {
86. holder.button\_.setImageResource(R.drawable.ic\_settings\_about);
87. } **else** {
88. holder.button\_.setImageResource(
89. android.R.drawable.stat\_notify\_error);
90. }
91. holder.button\_.setOnClickListener(**new** OnClickListener() {
92. @Override
93. **public** **void** onClick(View v) {
94. Intent intent = **new** Intent(
95. android.provider.Settings.ACTION\_MONITORING\_CERT\_INFO);
96. getContext().startActivity(intent);
97. }
98. });
99. } **else** {
100. holder.button\_.setVisibility(View.GONE);
101. holder.divider\_.setVisibility(View.GONE);
102. }
103. }
104. updateCommonHeaderView(header, holder);
105. **break**;
107. **case** HEADER\_TYPE\_NORMAL:
108. updateCommonHeaderView(header, holder);
109. **break**;
110. }
111. // /M: add for sim management feature
112. **if** (header.id == R.id.sim\_settings) {
113. /// M: Customize SIM string
114. holder.title.setText(mExt.customizeSimDisplayString(
115. getContext().getString(R.string.gemini\_sim\_management\_title), SLOT\_ALL));
116. handleDisableHolder(holder, view);
117. } **else** {
118. handleEnableHolder(holder, view);
119. }
120. **return** view;
121. }

那么每一个header 是如果响应点击操作的呢.这个就要看Setting.java的onHeaderClick 方法了, onHeaderClick 方法会调用父类的onHeaderClick方法来打开相关的应用,其父类是根据我们配置在settings\_headers.xml里面的fragment和intent 来打开相对应的activity的.

Setting.java -- onHeaderClick

**Java代码  [收藏代码](javascript:void())**

1. **public** **void** onHeaderClick(Header header, **int** position) {
2. **boolean** revert = **false**;
3. **if** (header.id == R.id.account\_add) {
4. revert = **true**;
5. }
7. **super**.onHeaderClick(header, position);
9. **if** (revert && mLastHeader != **null**) {
10. highlightHeader((**int**) mLastHeader.id);
11. } **else** {
12. mLastHeader = header;
13. }
14. }

PreferenceActivity -- onHeaderClick

**Java代码  [收藏代码](javascript:void())**

1. **public** **void** onHeaderClick(Header header, **int** position) {
2. **if** (header.fragment != **null**) {
3. **if** (mSinglePane) {
4. Log.d(TAG, "onHeaderClick, single pane and startWithFragment.");
5. **int** titleRes = header.breadCrumbTitleRes;
6. **int** shortTitleRes = header.breadCrumbShortTitleRes;
7. **if** (titleRes == 0) {
8. titleRes = header.titleRes;
9. shortTitleRes = 0;
10. }
11. startWithFragment(header.fragment, header.fragmentArguments, **null**, 0,
12. titleRes, shortTitleRes);
13. } **else** {
14. Log.d(TAG, "onHeaderClick, multiple pane and switchToHeader.");
15. switchToHeader(header);
16. }
17. } **else** **if** (header.intent != **null**) {
18. Log.d(TAG, "onHeaderClick, start activity with header intent.");
19. startActivity(header.intent);
20. }
21. }

**Android Settings开发之修改**

2014年03月25日 17:23:06 [WPJY](https://me.csdn.net/wangjinyu501) 阅读数：20040

 版权声明： https://blog.csdn.net/wangjinyu501/article/details/22077803

  版本：1.0

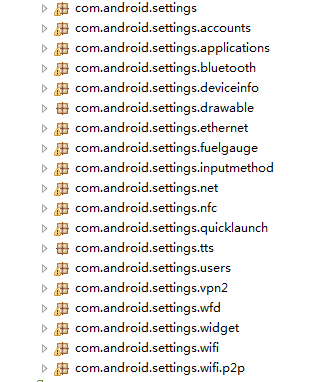
  日期：2014.3.20  2014.3.25

  版权：© 2014 kince 转载注明出处

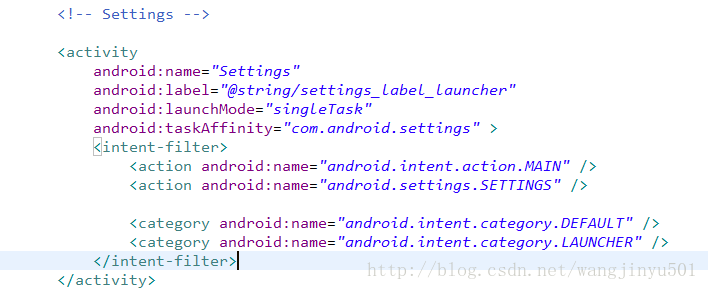
  下面是Seeings应用的截图：



  可以看出这是很典型的使用了Fragment后的界面，设置里面有WIFI、蓝牙、显示、存储、应用等众多功能。左边的每一项，对应着右边的一个设置界面，Fragment有四个子类：DialogFragment, ListFragment, PreferenceFragment, WebViewFragment。很明显，Settings用的是PreferenceFragment。接着看一下Settings源码package结构：



   主题部分的实现主要在com.android.settings下面，其他包主要是用于各自功能实现，所以重点说这个包下面的类。在AndroidManifest.xml文字中，看到程序入口是Settings类：



   打开Settings类，是继承于PreferenceActivity：

https://img-blog.csdn.net/20140325171039218?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd2FuZ2ppbnl1NTAx/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

   其他的继承关系如下：PreferenceActivity --> ListActivity --> Activity。PreferenceActivity主要用于Settings，关于如何使用可以参考API（http://developer.android.com/reference/android/preference/PreferenceActivity.html）以及guide（http://developer.android.com/guide/topics/ui/settings.html）。和它相关联的类有header、fragment、preference。每一个header就是左边的一个选项条目，像蓝牙、应用等，选择之后右边就会显示对应的fragment（平板），然后fragment和preference联系在一起，组成了一个个设置项。一般在activity中设置布局，用的是setContentView()，在PreferenceActivity中，是需要继承onBuildHeaders(List)这个方法，

1. @Override
2. public void onBuildHeaders(List<Header> target) {
3. loadHeadersFromResource(R.xml.preference\_headers, target);
4. }

  去生成选项表，点击选项表的一个条目，右边显示对应的Fragment，这就是很典型的header+fragment组合，所以如果想在Settings基础之上添加条目的话，在这个方法里面的xml文件中添加即可，然后对应上fragment。下面分析一下执行流程：

  首先进入onCreate(Bundle savedInstanceState)方法里面，代码如下：

1. @Override
2. protected void onCreate(Bundle savedInstanceState) {
3. if (getIntent().getBooleanExtra(EXTRA\_CLEAR\_UI\_OPTIONS, false)) {
4. getWindow().setUiOptions(0);
5. }
6. mAuthenticatorHelper = new AuthenticatorHelper();
7. mAuthenticatorHelper.updateAuthDescriptions(this);
8. mAuthenticatorHelper.onAccountsUpdated(this, null);
9. mDevelopmentPreferences = getSharedPreferences(DevelopmentSettings.PREF\_FILE,
10. Context.MODE\_PRIVATE);
11. getMetaData();
12. mInLocalHeaderSwitch = true;
13. super.onCreate(savedInstanceState);
14. mInLocalHeaderSwitch = false;
15. if (!onIsHidingHeaders() && onIsMultiPane()) {
16. highlightHeader(mTopLevelHeaderId);
17. // Force the title so that it doesn't get overridden by a direct launch of
18. // a specific settings screen.
19. setTitle(R.string.settings\_label);
20. }
21. // Retrieve any saved state
22. if (savedInstanceState != null) {
23. mCurrentHeader = savedInstanceState.getParcelable(SAVE\_KEY\_CURRENT\_HEADER);
24. mParentHeader = savedInstanceState.getParcelable(SAVE\_KEY\_PARENT\_HEADER);
25. }
26. // If the current header was saved, switch to it
27. if (savedInstanceState != null && mCurrentHeader != null) {
28. //switchToHeaderLocal(mCurrentHeader);
29. showBreadCrumbs(mCurrentHeader.title, null);
30. }
31. if (mParentHeader != null) {
32. setParentTitle(mParentHeader.title, null, new OnClickListener() {
33. public void onClick(View v) {
34. switchToParent(mParentHeader.fragment);
35. }
36. });
37. }
38. // Override up navigation for multi-pane, since we handle it in the fragment breadcrumbs
39. if (onIsMultiPane()) {
40. getActionBar().setDisplayHomeAsUpEnabled(false);
41. getActionBar().setHomeButtonEnabled(false);
42. }
43. }

  第一个if用于设置window ui的对修改来说不用考虑了，意义不大，

1. mAuthenticatorHelper = new AuthenticatorHelper();
2. mAuthenticatorHelper.updateAuthDescriptions(this);
3. mAuthenticatorHelper.onAccountsUpdated(this, null);

  这个段代码用于认证以及更新账户信息，接着往下看：

mDevelopmentPreferences = getSharedPreferences(DevelopmentSettings.PREF\_FILE,Context.MODE\_PRIVATE);

  用于之后保存数据，然后是getMetaData();这个方法，代码如下：

1. private void getMetaData() {
2. try {
3. ActivityInfo ai = getPackageManager().getActivityInfo(getComponentName(),
4. PackageManager.GET\_META\_DATA);
5. if (ai == null || ai.metaData == null) return;
6. mTopLevelHeaderId = ai.metaData.getInt(META\_DATA\_KEY\_HEADER\_ID);
7. mFragmentClass = ai.metaData.getString(META\_DATA\_KEY\_FRAGMENT\_CLASS);
8. // Check if it has a parent specified and create a Header object
9. final int parentHeaderTitleRes = ai.metaData.getInt(META\_DATA\_KEY\_PARENT\_TITLE);
10. String parentFragmentClass = ai.metaData.getString(META\_DATA\_KEY\_PARENT\_FRAGMENT\_CLASS);
11. if (parentFragmentClass != null) {
12. mParentHeader = new Header();
13. mParentHeader.fragment = parentFragmentClass;
14. if (parentHeaderTitleRes != 0) {
15. mParentHeader.title = getResources().getString(parentHeaderTitleRes);
16. }
17. }
18. } catch (NameNotFoundException nnfe) {
19. // No recovery
20. }
21. }

  这个方法用于设置mParentHeader的Fragment以及title。下面具体举两个例子，关于如何修改Settings。

一、添加headers

      header即是左边的菜单，如下图左侧。它的布局文件在res下的xml文件夹中，名字是settings\_headers.xml。打开如下：

1. <?xml version="1.0" encoding="utf-8"?>
2. *<!--*
3. *Copyright (C) 2010 The Android Open Source Project*
4. *Licensed under the Apache License, Version 2.0 (the "License");*
5. *you may not use this file except in compliance with the License.*
6. *You may obtain a copy of the License at*
7. *http://www.apache.org/licenses/LICENSE-2.0*
8. *Unless required by applicable law or agreed to in writing, software*
9. *distributed under the License is distributed on an "AS IS" BASIS,*
10. *WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.*
11. *See the License for the specific language governing permissions and*
12. *limitations under the License.*
13. *-->*
15. <preference-headers xmlns:android="http://schemas.android.com/apk/res/android" >
17. *<!-- WIRELESS and NETWORKS -->*
18. <header
19. android:id="@+id/wireless\_section"
20. android:title="@string/header\_category\_wireless\_networks" />
22. *<!-- Wifi -->*
23. <header
24. android:id="@+id/wifi\_settings"
25. android:fragment="com.android.settings.wifi.WifiSettings"
26. android:icon="@drawable/ic\_settings\_wireless"
27. android:title="@string/wifi\_settings\_title" />
29. *<!-- Bluetooth -->*
30. <header
31. android:id="@+id/bluetooth\_settings"
32. android:fragment="com.android.settings.bluetooth.BluetoothSettings"
33. android:icon="@drawable/ic\_settings\_bluetooth2"
34. android:title="@string/bluetooth\_settings\_title" />
36. *<!-- Data Usage -->*
37. <header
38. android:id="@+id/data\_usage\_settings"
39. android:fragment="com.android.settings.DataUsageSummary"
40. android:icon="@drawable/ic\_settings\_data\_usage"
41. android:title="@string/data\_usage\_summary\_title" />
43. *<!-- Operator hook -->*
44. <header
45. android:id="@+id/operator\_settings"
46. android:fragment="com.android.settings.WirelessSettings" >
47. <intent android:action="com.android.settings.OPERATOR\_APPLICATION\_SETTING" />
48. </header>
50. *<!-- Other wireless and network controls -->*
51. <header
52. android:id="@+id/wireless\_settings"
53. android:breadCrumbTitle="@string/wireless\_networks\_settings\_title"
54. android:fragment="com.android.settings.WirelessSettings"
55. android:icon="@drawable/empty\_icon"
56. android:title="@string/radio\_controls\_title" />
58. *<!-- DEVICE -->*
59. <header
60. android:id="@+id/device\_section"
61. android:title="@string/header\_category\_device" />
63. *<!-- Sound -->*
64. <header
65. android:id="@+id/sound\_settings"
66. android:fragment="com.android.settings.SoundSettings"
67. android:icon="@drawable/ic\_settings\_sound"
68. android:title="@string/sound\_settings" />
70. *<!-- Display -->*
71. <header
72. android:id="@+id/display\_settings"
73. android:fragment="com.android.settings.DisplaySettings"
74. android:icon="@drawable/ic\_settings\_display"
75. android:title="@string/display\_settings" />
77. *<!-- Storage -->*
78. <header
79. android:id="@+id/storage\_settings"
80. android:fragment="com.android.settings.deviceinfo.Memory"
81. android:icon="@drawable/ic\_settings\_storage"
82. android:title="@string/storage\_settings" />
84. *<!-- Battery -->*
85. <header
86. android:id="@+id/battery\_settings"
87. android:fragment="com.android.settings.fuelgauge.PowerUsageSummary"
88. android:icon="@drawable/ic\_settings\_battery"
89. android:title="@string/power\_usage\_summary\_title" />
91. *<!-- Application Settings -->*
92. <header
93. android:id="@+id/application\_settings"
94. android:fragment="com.android.settings.applications.ManageApplications"
95. android:icon="@drawable/ic\_settings\_applications"
96. android:title="@string/applications\_settings" />
98. *<!-- Manage users -->*
99. <header
100. android:id="@+id/user\_settings"
101. android:fragment="com.android.settings.users.UserSettings"
102. android:icon="@drawable/ic\_settings\_multiuser"
103. android:title="@string/user\_settings\_title" />
105. *<!-- Manufacturer hook -->*
106. <header
107. android:id="@+id/manufacturer\_settings"
108. android:fragment="com.android.settings.WirelessSettings" >
109. <intent android:action="com.android.settings.MANUFACTURER\_APPLICATION\_SETTING" />
110. </header>
112. *<!-- PERSONAL -->*
113. <header
114. android:id="@+id/personal\_section"
115. android:title="@string/header\_category\_personal" />
117. *<!-- Location -->*
118. <header
119. android:id="@+id/location\_settings"
120. android:fragment="com.android.settings.LocationSettings"
121. android:icon="@drawable/ic\_settings\_location"
122. android:title="@string/location\_settings\_title" />
124. *<!-- Security -->*
125. <header
126. android:id="@+id/security\_settings"
127. android:fragment="com.android.settings.SecuritySettings"
128. android:icon="@drawable/ic\_settings\_security"
129. android:title="@string/security\_settings\_title" />
131. *<!-- Language -->*
132. <header
133. android:id="@+id/language\_settings"
134. android:fragment="com.android.settings.inputmethod.InputMethodAndLanguageSettings"
135. android:icon="@drawable/ic\_settings\_language"
136. android:title="@string/language\_settings" />
138. *<!-- Backup and reset -->*
139. <header
140. android:id="@+id/privacy\_settings"
141. android:fragment="com.android.settings.PrivacySettings"
142. android:icon="@drawable/ic\_settings\_backup"
143. android:title="@string/privacy\_settings" />
144. <header
145. android:id="@+id/kytusers\_settings"
146. android:fragment="com.android.settings.KytUsersSettings"
147. android:icon="@drawable/ic\_settings\_backup"
148. android:title="考易通账户" />
150. *<!-- ACCOUNTS section -->*
151. <header
152. android:id="@+id/account\_settings"
153. android:title="@string/account\_settings" />
154. <header
155. android:id="@+id/account\_add"
156. android:icon="@drawable/ic\_menu\_add"
157. android:title="@string/add\_account\_label" >
158. <intent android:action="android.settings.ADD\_ACCOUNT\_SETTINGS" />
159. </header>
161. *<!-- SYSTEM -->*
162. <header
163. android:id="@+id/system\_section"
164. android:title="@string/header\_category\_system" />
166. *<!-- Date & Time -->*
167. <header
168. android:id="@+id/date\_time\_settings"
169. android:fragment="com.android.settings.DateTimeSettings"
170. android:icon="@drawable/ic\_settings\_date\_time"
171. android:title="@string/date\_and\_time\_settings\_title" />
173. *<!-- Accessibility feedback -->*
174. <header
175. android:id="@+id/accessibility\_settings"
176. android:fragment="com.android.settings.AccessibilitySettings"
177. android:icon="@drawable/ic\_settings\_accessibility"
178. android:title="@string/accessibility\_settings" />
180. *<!-- Development -->*
181. <header
182. android:id="@+id/development\_settings"
183. android:fragment="com.android.settings.DevelopmentSettings"
184. android:icon="@drawable/ic\_settings\_development"
185. android:title="@string/development\_settings\_title" />
187. *<!-- About Device -->*
188. <header
189. android:id="@+id/about\_settings"
190. android:fragment="com.android.settings.DeviceInfoSettings"
191. android:icon="@drawable/ic\_settings\_about"
192. android:title="@string/about\_settings" />
194. </preference-headers>

  这些header分别对应着各自的菜单，如果想要添加还是删除就在这里修改即可。比如我们不想要蓝牙模块了，那就直接把下面这个header删除即可，添加的话类似。

1. *<!-- Bluetooth -->*
2. <header
3. android:id="@+id/bluetooth\_settings"
4. android:fragment="com.android.settings.bluetooth.BluetoothSettings"
5. android:icon="@drawable/ic\_settings\_bluetooth2"
6. android:title="@string/bluetooth\_settings\_title" />

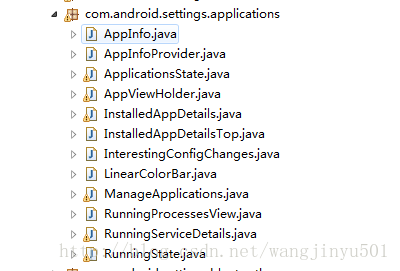
   如果是做添加操作的话，不要忘了创建你的PreferenceFragment，然后在header里面添加id、fragment、icon、title等，如上面那样。

二、修改显示的应用

      先看一下应用显示的部分：



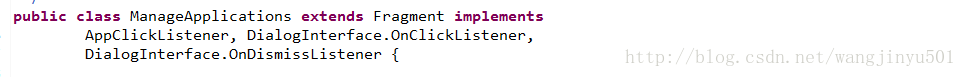
  看一下在源码中对应的包：



   应用显示是一个滑动的界面，猜测是用ViewPager实现的，下面开始寻找实现。首先进入xml文件夹找到settings\_headers，找到这一段代码：

1. *<!-- Application Settings -->*
2. <header
3. android:id="@+id/application\_settings"
4. android:fragment="com.android.settings.applications.ManageApplications"
5. android:icon="@drawable/ic\_settings\_applications"
6. android:title="@string/applications\_settings" />

  然后打开对应的fragment，ManageApplications。发现这个它是继承于Fragment



  既然继承于Fragment，那就直接定位到onCreate()方法，

1. @Override
2. public void onCreate(Bundle savedInstanceState) {
3. super.onCreate(savedInstanceState);
5. setHasOptionsMenu(true);
7. mApplicationsState = ApplicationsState.getInstance(getActivity().getApplication());
8. Intent intent = getActivity().getIntent();
9. String action = intent.getAction();
10. int defaultListType = LIST\_TYPE\_DOWNLOADED;
11. String className = getArguments() != null
12. ? getArguments().getString("classname") : null;
13. if (className == null) {
14. className = intent.getComponent().getClassName();
15. }
16. if (className.equals(RunningServicesActivity.class.getName())
17. || className.endsWith(".RunningServices")) {
18. defaultListType = LIST\_TYPE\_RUNNING;
19. } else if (className.equals(StorageUseActivity.class.getName())
20. || Intent.ACTION\_MANAGE\_PACKAGE\_STORAGE.equals(action)
21. || className.endsWith(".StorageUse")) {
22. mSortOrder = SORT\_ORDER\_SIZE;
23. defaultListType = LIST\_TYPE\_ALL;
24. } else if (Settings.ACTION\_MANAGE\_ALL\_APPLICATIONS\_SETTINGS.equals(action)) {
25. // Select the all-apps list, with the default sorting
26. defaultListType = LIST\_TYPE\_ALL;
27. }
29. if (savedInstanceState != null) {
30. mSortOrder = savedInstanceState.getInt(EXTRA\_SORT\_ORDER, mSortOrder);
31. int tmp = savedInstanceState.getInt(EXTRA\_DEFAULT\_LIST\_TYPE, -1);
32. if (tmp != -1) defaultListType = tmp;
33. mShowBackground = savedInstanceState.getBoolean(EXTRA\_SHOW\_BACKGROUND, false);
34. }
36. mDefaultListType = defaultListType;
38. final Intent containerIntent = new Intent().setComponent(
39. StorageMeasurement.DEFAULT\_CONTAINER\_COMPONENT);
40. getActivity().bindService(containerIntent, mContainerConnection, Context.BIND\_AUTO\_CREATE);
42. mInvalidSizeStr = getActivity().getText(R.string.invalid\_size\_value);
43. mComputingSizeStr = getActivity().getText(R.string.computing\_size);
45. TabInfo tab = new TabInfo(this, mApplicationsState,
46. getActivity().getString(R.string.filter\_apps\_third\_party),
47. LIST\_TYPE\_DOWNLOADED, this, savedInstanceState);
48. mTabs.add(tab);
50. if (!Environment.isExternalStorageEmulated()) {
51. tab = new TabInfo(this, mApplicationsState,
52. getActivity().getString(R.string.filter\_apps\_onsdcard),
53. LIST\_TYPE\_SDCARD, this, savedInstanceState);
54. mTabs.add(tab);
55. }
57. tab = new TabInfo(this, mApplicationsState,
58. getActivity().getString(R.string.filter\_apps\_running),
59. LIST\_TYPE\_RUNNING, this, savedInstanceState);
60. mTabs.add(tab);
62. tab = new TabInfo(this, mApplicationsState,
63. getActivity().getString(R.string.filter\_apps\_all),
64. LIST\_TYPE\_ALL, this, savedInstanceState);
65. mTabs.add(tab);
66. }

  这里主要是初始化TabInfo的数据，之后显示程序的时候会用到。接下来定位到onCreateView()方法，这个方法主要是初始化界面，

1. @Override
2. public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
3. // initialize the inflater
4. mInflater = inflater;
6. View rootView = mInflater.inflate(R.layout.manage\_applications\_content,
7. container, false);
8. mContentContainer = container;
9. mRootView = rootView;
11. mViewPager = (ViewPager) rootView.findViewById(R.id.pager);
12. MyPagerAdapter adapter = new MyPagerAdapter();
13. mViewPager.setAdapter(adapter);
14. mViewPager.setOnPageChangeListener(adapter);
15. PagerTabStrip tabs = (PagerTabStrip) rootView.findViewById(R.id.tabs);
16. tabs.setTabIndicatorColorResource(android.R.color.holo\_blue\_light);
18. // We have to do this now because PreferenceFrameLayout looks at it
19. // only when the view is added.
20. if (container instanceof PreferenceFrameLayout) {
21. ((PreferenceFrameLayout.LayoutParams) rootView.getLayoutParams()).removeBorders = true;
22. }
24. if (savedInstanceState != null && savedInstanceState.getBoolean(EXTRA\_RESET\_DIALOG)) {
25. buildResetDialog();
26. }
28. if (savedInstanceState == null) {
29. // First time init: make sure view pager is showing the correct tab.
30. for (int i = 0; i < mTabs.size(); i++) {
31. TabInfo tab = mTabs.get(i);
32. if (tab.mListType == mDefaultListType) {
33. mViewPager.setCurrentItem(i);
34. break;
35. }
36. }
37. }
38. return rootView;
39. }

  可以看到，正是使用了ViewPager，另外还有PagerTabStrip。先看一下最下面的：

1. if (savedInstanceState == null) {
2. // First time init: make sure view pager is showing the correct tab.
3. for (int i = 0; i < mTabs.size(); i++) {
4. TabInfo tab = mTabs.get(i);
5. if (tab.mListType == mDefaultListType) {
6. mViewPager.setCurrentItem(i);
7. break;
8. }
9. }
10. }

  这段代码的作用就是用于设置默认显示哪个选项卡的程序，所以就是“已下载”这个界面。然后回过头看一下上面的代码，

1. MyPagerAdapter adapter = new MyPagerAdapter();
2. mViewPager.setAdapter(adapter);

  这个就是熟悉的适配器了，所以显示程序的数据在这里面。开打这个类，

1. class MyPagerAdapter extends PagerAdapter
2. implements ViewPager.OnPageChangeListener {
3. int mCurPos = 0;
5. @Override
6. public int getCount() {
7. return mTabs.size();
8. }
10. @Override
11. public Object instantiateItem(ViewGroup container, int position) {
12. TabInfo tab = mTabs.get(position);
13. View root = tab.build(mInflater, mContentContainer, mRootView);
14. container.addView(root);
15. return root;
16. }
18. @Override
19. public void destroyItem(ViewGroup container, int position, Object object) {
20. container.removeView((View)object);
21. }
23. @Override
24. public boolean isViewFromObject(View view, Object object) {
25. return view == object;
26. }
28. @Override
29. public CharSequence getPageTitle(int position) {
30. return mTabs.get(position).mLabel;
31. }
33. @Override
34. public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels) {
35. }
37. @Override
38. public void onPageSelected(int position) {
39. mCurPos = position;
40. }
42. @Override
43. public void onPageScrollStateChanged(int state) {
44. if (state == ViewPager.SCROLL\_STATE\_IDLE) {
45. updateCurrentTab(mCurPos);
46. }
47. }
48. }

  可以看到，有几个滑动的页卡，是通过mTabs这个类来控制的，而它就是TabInfo类型。接着看一下绘制视图的方法：

1. MyPagerAdapter @Override
2. public Object instantiateItem(ViewGroup container, int position) {
3. TabInfo tab = mTabs.get(position);
4. View root = tab.build(mInflater, mContentContainer, mRootView);
5. container.addView(root);
6. return root;
7. }

   首先是获取到一个tab，然后这个tab用它的build()方法去生成一个视图，最后放到ViewPager显示。所以问题的重点分析就是TabInfo这个类了。先说buid()方法，

1. public View build(LayoutInflater inflater, ViewGroup contentParent,
2. View contentChild) {
3. if (mRootView != null) {
4. return mRootView;
5. }
7. mInflater = inflater;
8. mRootView = inflater
9. .inflate(
10. mListType == LIST\_TYPE\_RUNNING ? R.layout.manage\_applications\_running
11. : R.layout.manage\_applications\_apps, null);
12. mLoadingContainer = mRootView.findViewById(R.id.loading\_container);
13. mLoadingContainer.setVisibility(View.VISIBLE);
14. mListContainer = mRootView.findViewById(R.id.list\_container);
15. if (mListContainer != null) {
16. // Create adapter and list view here
17. View emptyView = mListContainer
18. .findViewById(com.android.internal.R.id.empty);
19. ListView lv = (ListView) mListContainer
20. .findViewById(android.R.id.list);
21. if (emptyView != null) {
22. lv.setEmptyView(emptyView);
23. }
24. lv.setOnItemClickListener(this);
25. lv.setSaveEnabled(true);
26. lv.setItemsCanFocus(true);
27. lv.setTextFilterEnabled(true);
28. mListView = lv;
29. mApplications = new ApplicationsAdapter(mApplicationsState,
30. this, mFilter);
31. mListView.setAdapter(mApplications);
32. mListView.setRecyclerListener(mApplications);
33. mColorBar = (LinearColorBar) mListContainer
34. .findViewById(R.id.storage\_color\_bar);
35. mStorageChartLabel = (TextView) mListContainer
36. .findViewById(R.id.storageChartLabel);
37. mUsedStorageText = (TextView) mListContainer
38. .findViewById(R.id.usedStorageText);
39. mFreeStorageText = (TextView) mListContainer
40. .findViewById(R.id.freeStorageText);
41. Utils.prepareCustomPreferencesList(contentParent, contentChild,
42. mListView, false);
43. if (mFilter == FILTER\_APPS\_SDCARD) {
44. mStorageChartLabel.setText(mOwner.getActivity().getText(
45. R.string.sd\_card\_storage));
46. } else {
47. mStorageChartLabel.setText(mOwner.getActivity().getText(
48. R.string.internal\_storage));
49. }
50. applyCurrentStorage();
51. }
52. mRunningProcessesView = (RunningProcessesView) mRootView
53. .findViewById(R.id.running\_processes);
54. if (mRunningProcessesView != null) {
55. mRunningProcessesView.doCreate(mSavedInstanceState);
56. }
58. return mRootView;
59. }

  第四行是生成了一view对象，

mRootView = inflater.inflate(mListType == LIST\_TYPE\_RUNNING ? R.layout.manage\_applications\_running: R.layout.manage\_applications\_apps, null);

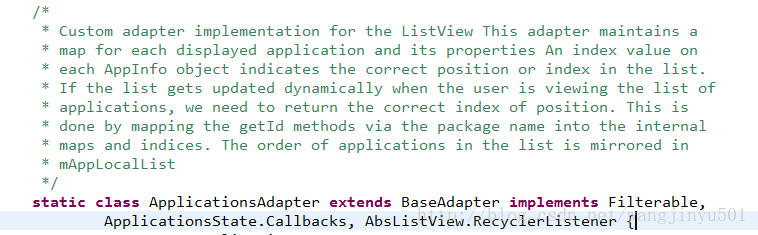
  这行代码用来判断加载哪个布局文件，确实viewpager虽然动态显示四个页面，但是其中三个的布局是一样的，唯一不一样的就是显示正在运行的界面。

1. ListView lv = (ListView) mListContainer
2. .findViewById(android.R.id.list);
3. if (emptyView != null) {
4. lv.setEmptyView(emptyView);
5. }
6. lv.setOnItemClickListener(this);
7. lv.setSaveEnabled(true);
8. lv.setItemsCanFocus(true);
9. lv.setTextFilterEnabled(true);
10. mListView = lv;
11. mApplications = new ApplicationsAdapter(mApplicationsState,
12. this, mFilter);
13. mListView.setAdapter(mApplications);

  这段代码就是用于显示app程序了，然后定位到ApplicationsAdapter这个类，在它的构造方法里面，传入三个参数。一个是mApplicationsState对象，以后用于对ApplicationsState类进行操作；一个是TabInfo，用来显示不同的界面；一个是过滤器，是标识显示哪个界面。进入这个构造方法，

1. public ApplicationsAdapter(ApplicationsState state, TabInfo tab,
2. int filterMode) {
3. mState = state;
4. mSession = state.newSession(this);
5. mTab = tab;
6. mContext = tab.mOwner.getActivity();
7. mFilterMode = filterMode;
8. }

   发现没有在这里传入什么数据，然后看一下ApplicationsAdapter这个类，发现它继承了三个接口，第一个是过滤用的，第三个是系统SDK接口用于ListView循环处理。而第二个接口正是负责处理数据的，



  它有六个回调方法：

1. public static interface Callbacks {
2. public void onRunningStateChanged(boolean running);
4. public void onPackageListChanged();
6. public void onRebuildComplete(ArrayList<AppEntry> apps);
8. public void onPackageIconChanged();
10. public void onPackageSizeChanged(String packageName);
12. public void onAllSizesComputed();
13. }

  第三个回调方法onRebuildComplete(ArrayList<AppEntry> apps)正是用于返回app Entities的，但是到这里如果我们还是按这条线是分析不下去了，找不到这个数据是从哪里来的。所以不能按照这个思路往下走了，也就是说app程序数据不是在这里获取的，那会是什么地方呢？一般情况下，我们是在onStart或者onCreat方法里面，但是Setting里面都没有这样做，那往下看一下onResume吧。果然Settings是在这个方法里面加载数据的，

1. @Override
2. public void onResume() {
3. super.onResume();
4. mActivityResumed = true;
5. updateCurrentTab(mViewPager.getCurrentItem());
6. updateOptionsMenu();
7. }

  首先是调用了updateCurrentTab(mViewPager.getCurrentItem())方法，然后updateCurrentTab方法里面又调用了TabInfo的resume方法，在TabInfo的resume方法里面接着调用了ApplicationsAdapter 的resume方法，又在ApplicationsAdapter 的resume方法里面调用Session的resume方法，最后又在Session的resume方法里面调用doResumeIfNeededLocked()方法，这个方法就是从系统读取程序信息的，代码如下：

1. void doResumeIfNeededLocked() {
2. if (mResumed) {
3. return;
4. }
5. mResumed = true;
6. if (mPackageIntentReceiver == null) {
7. mPackageIntentReceiver = new PackageIntentReceiver();
8. mPackageIntentReceiver.registerReceiver();
9. }
10. //这个mApplications就是所有程序数据，如果你想过滤哪些程序的信息，对这个集合进行修改即可。比如你在做定制机的时候，不想自己的程序显示在Settings里面，那就在这里修改。
11. mApplications = mPm.getInstalledApplications(mRetrieveFlags);
12. if (mApplications == null) {
13. mApplications = new ArrayList<ApplicationInfo>();
14. }
16. if (mInterestingConfigChanges.applyNewConfig(mContext.getResources())) {
17. // If an interesting part of the configuration has changed, we
18. // should completely reload the app entries.
19. mEntriesMap.clear();
20. mAppEntries.clear();
21. } else {
22. for (int i = 0; i < mAppEntries.size(); i++) {
23. mAppEntries.get(i).sizeStale = true;
24. }
25. }
26. for (int i = 0; i < mApplications.size(); i++) {
27. final ApplicationInfo info = mApplications.get(i);
28. // Need to trim out any applications that are disabled by
29. // something different than the user.
30. if (!info.enabled
31. && info.enabledSetting != PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED\_USER) {
32. mApplications.remove(i);
33. i--;
34. continue;
35. }
36. final AppEntry entry = mEntriesMap.get(info.packageName);
37. if (entry != null) {
38. entry.info = info;
39. }
40. }
41. mCurComputingSizePkg = null;
42. if (!mBackgroundHandler.hasMessages(BackgroundHandler.MSG\_LOAD\_ENTRIES)) {
43. mBackgroundHandler
44. .sendEmptyMessage(BackgroundHandler.MSG\_LOAD\_ENTRIES);
45. }
46. };

   至此，完成了显示程序的剖析，其他模块类似，只要按照流程走就行了。

相似博文推荐：

  1、[Android PreferenceActivity点击Header是如何处理的？](http://blog.csdn.net/droyon/article/details/18009373)

  2、[android-setting](http://www.360doc.com/userhome.aspx?userid=9171956&cid=267)

  3、[Android4.0设置界面修改总结](http://my.oschina.net/cjkall/blog/195789)

  4、[Android源码之“应用程序界面“分析一( 从settings开始)](http://blog.csdn.net/liranke/article/details/17715455)

  5、[Android 应用程序分析之Settings](http://blog.csdn.net/caszhao/article/details/5986900)

  6、[菜鸟Android4.0 Settings分析（一）](http://blog.csdn.net/wu_shu_jun/article/details/8514465)

  7、[Android Settings源码结构分析与自实现](http://blog.csdn.net/luohaowang320/article/details/18232819)

  8、[Settings源码分析](http://blog.csdn.net/androiddeveloper_lee/article/details/10189971)

  9、[Settings修改记录](http://blog.csdn.net/androiddeveloper_lee/article/details/9793113)

# Android Settings模块架构浅析

2016年03月30日 10:51:12 [默默9518](https://me.csdn.net/sgzy001) 阅读数：2967

## ****概述****

Android Settings模块说简单也简单，说难也难，里面涉及到的知识点也挺多的。

我们知道Settings主要是用于配置一些系统选项或属性值，通过修改设置项就能达到修改系统配置的作用。

那么问题来了，Settings是如何实现修改后能改变系统配置的呢？Settings又是采用怎样的架构实现的呢？里面又涉及到哪些知识点呢？

让我们一起来揭开她的[神秘面纱](https://www.baidu.com/s?wd=%E7%A5%9E%E7%A7%98%E9%9D%A2%E7%BA%B1&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)吧！

## ****原理分析****

Settings的主要功能就是改变系统配置，那么他是如何做到的呢？

* **Settings的战友SettingsProvider**

通过跟踪Settings源码发现，Settings并不是[孤军作战](https://www.baidu.com/s?wd=%E5%AD%A4%E5%86%9B%E4%BD%9C%E6%88%98&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，它还有一个战友 - SettingsProvider。

SettingsProvider又是做什么的呢？其实看到这个名字我们不难猜到他扮演着什么样的角色。

SettingsProvider继承ContentProvider，ContentProvider在android中主要扮演着数据共享的角色。

SettingsProvider中有一个数据库，并且这个数据库是对外公开的。

Settings与SettingsProvider之间又是什么关系呢？且看下面分析。

* **Settings和SettingsProvider之间的关系**

Settings源码位置：

packages/apps/Settings/

SettingsProvider源码位置：

frameworks/base/packages/SettingsProvider/

frameworks/base/core/java/android/provider/Settings.java

db在数据库中存在的位置：

/data/data/com.android.providers.settings/databases/settings.db

他们之间存在什么联系呢？

其实，Settings会对SettingsProvider中的数据库进行操作和监听。

Settings中大部分选项都会涉及到对SettingsProvider的操作。

* **原理分析**

通过跟踪代码发现，Settings大部分操作的就是SettingsProvider中的数据，也有一些直接操作系统属性的等等。

当用户在修改系统设置时，大部分实际上是在修改SettingsProvider中的值。

当SettingsProvider数据库中的值被改变时，一些系统服务什么的就会监听到，这时候就会通过jni等当时操作底层，从而达到系统属性或配置改变的效果。

## ****架构分析****

Settings处在安卓的应用层，不同于市场上的app，Settings属于系统app，也是一个比较特别的app。

* **Settings特点**

1.Settings页面很多，但是Activity却很少，基本上都是使用PreferenceFragment

2.Settings中包含大量对provider的操作与监听

3.Settings UI基本上都是采用Preference来实现

* **Settings架构**

1.Settings主界面Activity使用的是Settings

2.Settings子界面Activity基本上都是使用SubSettings

3.Settings与SubSettings中都是空Activity，这里的空Activity指的是没有重写7大生命周期方法

4.Settings与SubSettings都是继承于SettingsActivity

5.主界面使用的layout是：settings\_main\_dashboard，子界面使用的layout是：settings\_main\_prefs

6.主界面settings\_main\_dashboard中是使用DashboardSummary(Fragment)进行填充，子界面都是使用各自的Fragment进行填充

7.子界面fragment基本上都是直接或间接继承SettingsPreferenceFragment

8.主界面选项列表是定义在dashboard\_categories.xml中，此文件是在SettingsActivity的buildDashboardCategories方法中进行解析的

9.在Settings类中定义了很多static class，这些类都是继承SettingsActivity，但都是空的，如BluetoothSettingsActivity

这些类主要用于对外提供跳转页面，比如从SystemUI跳转至Settings中的某个界面

10.Settings类中定义了的static class被定义在AndroidManifest中，通过meta-data参数将对应的Fragment绑定在一起

11.在Activity中填充Fragment主要使用的是SettingsActivity中的switchToFragment方法

* settings\_main\_dashboard中只有一个FrameLayout，后面会将其替换为DashboardSummary

1. <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2. android:id="@+id/main\_content"
3. android:layout\_width="match\_parent"
4. android:layout\_height="match\_parent"
5. android:background="@color/dashboard\_background\_color" />

* settings\_main\_prefs中也存在一个叫main\_content的FrameLayout，后面会将其替换为各自的Fragment，switch\_bar与button\_bar只有在某些页面才会显示

1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2. android:layout\_width="match\_parent"
3. android:layout\_height="match\_parent"
4. android:orientation="vertical" >
6. <LinearLayout
7. android:layout\_width="match\_parent"
8. android:layout\_height="0px"
9. android:layout\_weight="1"
10. android:orientation="vertical" >
12. <com.android.settings.widget.SwitchBar
13. android:id="@+id/switch\_bar"
14. android:layout\_width="match\_parent"
15. android:layout\_height="?android:attr/actionBarSize"
16. android:background="@drawable/switchbar\_background"
17. android:theme="?attr/switchBarTheme" />
19. <FrameLayout
20. android:id="@+id/main\_content"
21. android:layout\_width="match\_parent"
22. android:layout\_height="match\_parent"
23. android:background="?attr/preferenceBackgroundColor" />
24. </LinearLayout>
26. <RelativeLayout
27. android:id="@+id/button\_bar"
28. android:layout\_width="match\_parent"
29. android:layout\_height="wrap\_content"
30. android:layout\_weight="0"
31. android:visibility="gone" >
33. <Button
34. android:id="@+id/back\_button"
35. android:layout\_width="150dip"
36. android:layout\_height="wrap\_content"
37. android:layout\_alignParentStart="true"
38. android:layout\_margin="5dip"
39. android:text="@\*android:string/back\_button\_label" />
41. <LinearLayout
42. android:layout\_width="wrap\_content"
43. android:layout\_height="wrap\_content"
44. android:layout\_alignParentEnd="true"
45. android:orientation="horizontal" >
47. <Button
48. android:id="@+id/skip\_button"
49. android:layout\_width="150dip"
50. android:layout\_height="wrap\_content"
51. android:layout\_margin="5dip"
52. android:text="@\*android:string/skip\_button\_label"
53. android:visibility="gone" />
55. <Button
56. android:id="@+id/next\_button"
57. android:layout\_width="150dip"
58. android:layout\_height="wrap\_content"
59. android:layout\_margin="5dip"
60. android:text="@\*android:string/next\_button\_label" />
61. </LinearLayout>
62. </RelativeLayout>
64. </LinearLayout>

* **Settings主界面结构**

1.从图中可以看到，红色框中的属于一个DashboardCategory，蓝色框中的属于DashboardTileView

2.在DashboardSummary中有多个DashboardCategory，DashboardCategory中包含一个title和多个DashboardTileView

3.DashboardTileView具有onClick方法，点击后启动子界面，使用的是Utils.startWithFragment进行跳转

4.startWithFragment方法中将子界面的Fragment传递给activity，这里会绑定对应的activity，也就是SubSettings

Utils.startWithFragment关键方法

1. public static void startWithFragment(Context context, String fragmentName, Bundle args,
2. Fragment resultTo, int resultRequestCode, int titleResId,
3. CharSequence title) {
4. startWithFragment(context, fragmentName, args, resultTo, resultRequestCode,
5. null */\* titleResPackageName \*/*, titleResId, title, false */\* not a shortcut \*/*);
6. }

9. public static void startWithFragment(Context context, String fragmentName, Bundle args,
10. Fragment resultTo, int resultRequestCode, String titleResPackageName, int titleResId,
11. CharSequence title, boolean isShortcut) {
12. Intent intent = onBuildStartFragmentIntent(context, fragmentName, args,
13. titleResPackageName,
14. titleResId, title, isShortcut);
15. if (resultTo == null) {
16. context.startActivity(intent);
17. } else {
18. resultTo.startActivityForResult(intent, resultRequestCode);
19. }
20. }
22. public static Intent onBuildStartFragmentIntent(Context context, String fragmentName,
23. Bundle args, String titleResPackageName, int titleResId, CharSequence title,
24. boolean isShortcut) {
25. Intent intent = new Intent(Intent.ACTION\_MAIN);
26. if (BluetoothSettings.class.getName().equals(fragmentName)) {
27. intent.setClass(context, SubSettings.BluetoothSubSettings.class);
28. intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_AS\_SUBSETTING, true);
29. } else if (WifiSettings.class.getName().equals(fragmentName)) {
30. intent.setClass(context, SubSettings.WifiSubSettings.class);
31. intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_AS\_SUBSETTING, true);
32. } else {
33. intent.setClass(context, SubSettings.class);
34. }
35. intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT, fragmentName);
36. intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_ARGUMENTS, args);
37. intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_TITLE\_RES\_PACKAGE\_NAME,
38. titleResPackageName);
39. intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_TITLE\_RESID, titleResId);
40. intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_TITLE, title);
41. intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_AS\_SHORTCUT, isShortcut);
42. return intent;
43. }

SettingsActivity.onCreate方法中的关键代码

1. @Override
2. protected void onCreate(Bundle savedState) {
3. super.onCreate(savedState);
5. *// Should happen before any call to getIntent()*
6. getMetaData();
8. final Intent intent = getIntent();
10. *// Getting Intent properties can only be done after the super.onCreate(...)*
11. final String initialFragmentName = intent.getStringExtra(EXTRA\_SHOW\_FRAGMENT);
13. mIsShortcut = isShortCutIntent(intent) || isLikeShortCutIntent(intent) ||
14. intent.getBooleanExtra(EXTRA\_SHOW\_FRAGMENT\_AS\_SHORTCUT, false);
16. final ComponentName cn = intent.getComponent();
17. final String className = cn.getClassName();
19. mIsShowingDashboard = className.equals(Settings.class.getName());
21. *// This is a "Sub Settings" when:*
22. *// - this is a real SubSettings*
23. *// - or :settings:show\_fragment\_as\_subsetting is passed to the Intent*
24. final boolean isSubSettings = className.equals(SubSettings.class.getName()) ||
25. intent.getBooleanExtra(EXTRA\_SHOW\_FRAGMENT\_AS\_SUBSETTING, false);
27. setContentView(mIsShowingDashboard ?
28. R.layout.settings\_main\_dashboard : R.layout.settings\_main\_prefs);
30. mContent = (ViewGroup) findViewById(R.id.main\_content);
32. getFragmentManager().addOnBackStackChangedListener(this);
34. if (savedState != null) {
35. ......
36. } else {
37. if (!mIsShowingDashboard) {
38. ......
39. setTitleFromIntent(intent);
41. Bundle initialArguments = intent.getBundleExtra(EXTRA\_SHOW\_FRAGMENT\_ARGUMENTS);
42. switchToFragment(initialFragmentName, initialArguments, true, false,
43. mInitialTitleResId, mInitialTitle, false);
44. } else {
45. ......
46. mInitialTitleResId = R.string.dashboard\_title;
47. switchToFragment(DashboardSummary.class.getName(), null, false, false,
48. mInitialTitleResId, mInitialTitle, false);
49. }
50. }
51. ......
52. }

1.当点击主界面上的item时会调用Utils.startWithFragment方法

2.在Utils.startWithFragment会跳转至SubSettings,对应的fragment也作为参数传递给了SubSettings

3.SubSettings是一个空的activity，但SubSettings继承于SettingsActivity，因此会调用父类SettingsActivity的onCreate方法

4.在onCreate方法中，className为SubSettings，isSubSettings为true，mIsShowingDashboard为false

5.因此会执行switchToFragment(initialFragmentName, initialArguments, true, false, mInitialTitleResId, mInitialTitle, false);

6.通过switchToFragment将settings\_main\_prefs的main\_content替换为了子界面对应的fragment

## ****简单类图****

从下面类图中可以看出

1.Settings中主要的Activity为SettingsActivity，其他基本上都是继承该activity，并且其他基本上都是空的

2.Settings中fragment基本上都是继承至SettingsPreferenceFragment

## ****时序图****

下面的时序图为点击Settings图标启动Settings，在点击item启动子界面的时序图

从图中可以看出启动的一个流程，按照这个流程，几乎所有的界面都会执行SettingsActivity

**Android N Settings模块与Android M Settings模块差异**

2016年11月03日 21:16:15 [天木楠](https://me.csdn.net/niexu19900104) 阅读数：4107

 版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/niexu19900104/article/details/53026132

Android N Settings 与之前的Android版本Settings模块在UI上右较大的改变，最直观的差异在于Android N 设置[界面](https://www.baidu.com/s?wd=%E7%95%8C%E9%9D%A2&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)增加了向右滑动的抽屉效果。

*先回想下android M设置的主界面时怎么显示和交互的？*

*android M设置的主界面是定义在res/xml/dashboard\_categories.xml这个文件中，其SettingsActivity.java继承与Activity,在其onCreate方法中，通过mIsShowingDashboard变量，控制加载不同的布局R.layout.settings\_main\_dashboard或者 R.layout.settings\_main\_prefs（Settings的主界面或者二级三级菜单），mIsShowingDashboard Boolear类型的变量是直接判断当前要启动的Activity的类是不是Settings.class，因为Settings这个类才是Settings的入口类。若是加载Settings的主界面，则是加载R.layout.settings\_main\_dashboard布局，这个布局里面仅仅只是一个FrameLayout的ViewContainer。*

*SettingsActivity.java中buildDashboardCategories（）->loadCategoriesFromResource(R.xml.dashboard\_categories, categories, this) ->updateTilesList（）去解析这个布局文件，动态添加或删除一级菜单的item，然后将主界面加载到FrameLayout的ViewContainer中，这样一级菜单就显示出来了。*

对于二级菜单，在设置模块中有一个Settings.java的类，该类中定义了许多直接继承与SettingsActivity的空实现Activity,如果细心点还会发现，这些空实现的Activity在AndroidMainifest.xml中都有<meta-data/>属性，这是为什么呢？原因很简单，上面说过，既然这些Activity都是继承与SettingsActivity，那么要启动这些Activity时必然会走SettingsActivity中onCreate()方法，刚在还提到SettingsActivity中onCreate()方法是需要加载一级菜单还是加载二级菜单，取决于*mIsShowingDashboard变量，而现在要确定加载的是二级菜单的具体哪个菜单？则需要用到AndroidMainifest.xml中声明该Activity时定义的<meta-data/>属性。*

*当然，并不是所有的设置项都是这样，例如若要在Settings中添加一个选项，跳转到其他应用中，那么也可以直接在dashboard\_categories.xml中添加相应的item，直接用intent跳转过去也时可以的。*

*那么，Android N中的设置主界面又时怎么显示和交互的呢？*

*Android N 在代码架构上和Android M最大的区别是Android N设置代码分为了两部分，一部分是在packages/apps/Settings,另一部分是在frameworks/base/packages/SettingsLib。*

*Android N Settings增加了向右滑动的抽屉效果，即无论在一级二级还是三级菜单界面下，只要向右滑动就可以调出Settings的主界面(事实上是无论哪一级菜单，只要改级菜单继承了SettingsActivity，那么就可以向右滑动调出设置的主界面)。接下来分析Android N Settings的启动流程。*

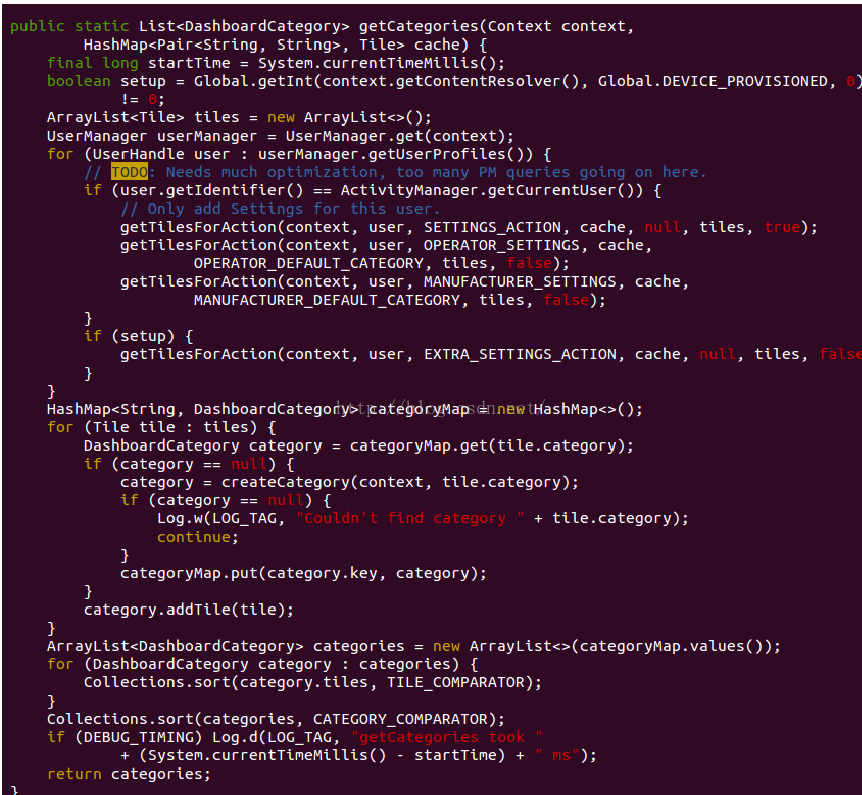
*先看设置的入口类，Settings.java,该类和之前一样，没什么变化，一样继承与SettingsActivity.java，内部也定义了很多空实现的内部类，空Activity继承与SettingsActivity。那么应该和android M一样，真正的入口还是在SettingsActivity中。直接看SettingsActivity类，发现SettingsActivity不再是继承与Activity类，而是继承与rameworks/base/packages/SettingsLib下面的SettingsDrawerActivity，看一下SettingsDrawerActivity是什么鬼？？*

*SettingsDrawerActivity从名字来看，好像和实现滑动效果有关，到该类一看究竟，发现它加载的布局时settings\_with\_drawer.xml，这个布局中用到android.support.v4.widget.DrawerLayout，DrawerLayout 布局就具有左右滑动的抽屉效果。在看settings\_with\_drawer.xml布局中有一个listView控件，猜想该控件就是用来显示抽屉布局中的设置的主界面（后面在分析）。*

*继续看SettingsActivity的onCreate方法，其加载布局的地方和之前没什么区别，可是却发现有区别的地方在从DashboardSummary.class中的获取要加载item的数据部分，DashboardSummary.java的rebuildUI()中，调用了SettingsDrawerActivity的getDashboardCategories()获取数据。也就是说在设置中主界面上要显示那些选项也是rameworks/base/packages/SettingsLib中的SettingsDrawerActivity的getDashboardCategories()获取来的，而不是像Android M一样将布局定义在res/xml/dashboard\_categories.xml，现在在Settings模块下面已经没有了dashboard\_categories.xml文件，那么数据到底是怎么获取出来的呢？*

*现在来分析SettingsDrawerActivity类，在SettingsDrawerActivity的onCreate方法中，如上面提到，它加载的布局是settings\_with\_drawer.xml，它是外层是DrawerLayout，所以能左右滑动（可自行*[*百度*](https://www.baidu.com/s?wd=%E7%99%BE%E5%BA%A6&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)*DrawerLayout用法），该布局中有listView，这个listView就是我们左右滑动看到的DrawerLayout的主界面，listView的Adapter是SettingsDrawerAdapter的实例，SettingsDrawerAdapter继承于BaseAdapter,SettingsDrawerActivity中通过getDashboardCategories()   >TileUtils.getCategories(****this****, sTileCache),获取的数据，它将数据也是封装成****DashboardCategory类型。***

***再来看 TileUtils类的getCategories()是如何返回DashboardCategory对象的List的?先看代码：***

******

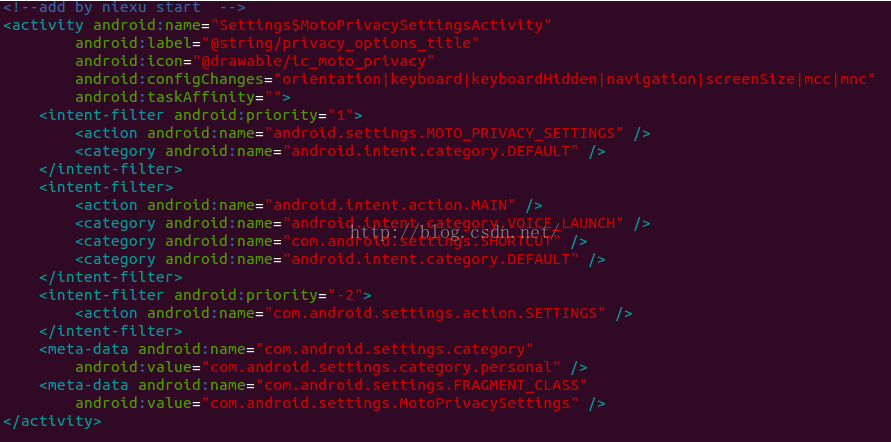
***getCategories()方法差不多可分为三步：***

***1.先调用****getTilesForAction() ->getTilesForInten（）从AndroidMainifast.xml中声明的activity信息中解析，并将解析的数据以Tile类型数据结构封装。*

*2，遍历存储所有Tile的集合，将通过tile.category属性，将其分类，并构建成category，并将category存储在对应的categoryMap中。*

*3.构建categories，并且排序，排序是按照DashboardCategory的priority排序的。*

*先看一下Title数据类型，它里面封装的每一个变量都是和AndroidMainifast.xml中声明Activity时<meta-data/>标签一一对应的。*



*Title  : 对应每个item显示的标题*

*icon   :对应每个Item显示的图标*

*summary  : 对应每个item显示的详细信息，summary*

*intent  :item 对应的intent*

*cetagory :item 属于哪一个catagory*

*priority  : item 的位置有关，排序会用到*

*extras  ：Optional additional data for use by subclasses of the activity，从启动它的类携带过来的信息*

*meta-data ：The metaData from the activity that defines this tile，定义的meta-data*

*通过这样的方法获取到了Settings主菜单和左后滑动抽屉效果需要显示的数据，然后将数据抛给SettingsDrawerAdapter，在它的getView方法中去创建对应的view，就将界面展示出来了。*

*总结一下：到了Android N设置中没有主界面显示的布局文件，而是直接解析AndroidManifast.xml文件，从该文件中获得并处理要显示的信息。*