**1.本文说明**

    本文主要针对L平台上Settings模块正常启动流程做一个简要分析，并试着分析一下Settings下面Storage选项的实现过程。

**2.Settings概览**

    在之前的KK平台上Settings模块的第一个Activity名字为Settings，其继承的是PreferenceActivity，设置的每一个选项都是对应的一个Header对象，并且Header对象允许显示switch控件，button控件，checkbox控件等。如下图2.1，WLAN和蓝牙上使用到了switch开关。但在L上面，WLAN和蓝牙的这两个开关已经去掉了，如图2.2，在Settings模块的首个页面似乎就只是一个普通的Listview，那它用的还是不是Header呢？或者说取而代之的是什么呢？下一节详细说明。



图2.1 kkSettings首界面截图



图2.2 L Settings首界面

**3 .L Settings 模块首界面初始化流程**

L Settings模块首界面为Settings，继承自SettingsActivity，SettingsActivity继承自Activity。

首先看一下Settings.java代码可以发现它没有重写任何SettingsActiviy的方法，也没有增加任何自己的方法，唯独增加了许多静态内部类，如：

    /\*

    \* Settings subclasses for launching independently.

    \*/

**public static class** BluetoothSettingsActivity **extends** SettingsActivity { /\* empty \*/ }

**public static class** WirelessSettingsActivity **extends** SettingsActivity { /\* empty \*/ }

**public static class** SimSettingsActivity **extends** SettingsActivity { /\* empty \*/ }

**public static class** TetherSettingsActivity **extends** SettingsActivity { /\* empty \*/ }

**public static class** VpnSettingsActivity **extends** SettingsActivity { /\* empty \*/ }

**public static class** DateTimeSettingsActivity **extends** SettingsActivity { /\* empty \*/ }

**public static class** StorageSettingsActivity **extends** SettingsActivity { /\* empty \*/ }

**public static class** WifiSettingsActivity **extends** SettingsActivity { /\* empty \*/ }

    . . .

看注释可以知道，这些子类是为了启动特定独立的Settings选项而创建的，例如在某个应用里需要设置无线那么只需要启动 WirelessSettingsActivity 就可以了。

所以Settings模块的启动流程直接看SettingsActiviy就行了。

    3.1 SettingsActivity.onCreate方法

onCreate方法是Activity的生命周期第一步，看看 SettingsActivity在这里都做了些什么？

*// Should happen before any call to getIntent()*

*getMetaData();*

这个方法用来获得Activity的额外数据mFragmentClass，如果可以获得这个数据，那么下面会去显示mFragmentClass对应的Activity。直接启动Settings模块不会获得这个数据。

*mIsShowingDashboard = className.equals(Settings.****class****.getName());*

这一步很重要，因为我们是从Settings这个Activity过来的，所以这里的 mIsShowingDashboard 为 true 。

*// This is a "Sub Settings" when:*

*// - this is a real SubSettings*

*// - or :settings:show\_fragment\_as\_subsetting is passed to the Intent*

***final******boolean****isSubSettings = className.equals(SubSettings.****class****.getName()) ||*

*intent.getBooleanExtra(EXTRA\_SHOW\_FRAGMENT\_AS\_SUBSETTING,****false****);*

这个判断很重要但很明显这时isSubSettings的值是fasle，暂时忽略。

*setContentView(mIsShowingDashboard ?*

*R.layout.settings\_main\_dashboard : R.layout.settings\_main\_prefs);*

前面知道这里的 mIsShowingDashboard为true，所以这里使用的布局文件为R.layout.settings\_main\_dashboard。settings\_main\_dashboard.xml文件如下：

*<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"*

*android:id="@+id/main\_content"*

*android:layout\_height="match\_parent"*

*android:layout\_width="match\_parent"*

*android:background="@color/dashboard\_background\_color"*

*/>*

由于mIsShowingDashboard为true，直接走到下面这段

***else****{*

*// No UP affordance if we are displaying the main Dashboard*

*mDisplayHomeAsUpEnabled =****false****;*

*// Show Search affordance*

*mDisplaySearch =****true****;*

*mInitialTitleResId = R.string.dashboard\_title;*

*switchToFragment(DashboardSummary.****class****.getName(),****null****,****false****,****false****,*

*mInitialTitleResId, mInitialTitle,****false****);*

*}*

这里看到switchToFragment这个方法，可以知道这里是要切换DashboardSummary这个Fragment.

接下来就看看DashboardSummary是个什么玩意？

dashboard中文意思是仪表盘，这里是指DashboardSummary就是用来显示Settings所有选项的。

在DashboardSummary的onCreateView里加载了这个布局文件R.layout.dashboard：

*<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"*

*android:id="@+id/dashboard"*

*android:layout\_width="match\_parent"*

*android:layout\_height="match\_parent"*

*android:scrollbarStyle="outsideOverlay"*

*android:clipToPadding="false">*

*<LinearLayout*

*android:id="@+id/dashboard\_container"*

*android:layout\_width="match\_parent"*

*android:layout\_height="match\_parent"*

*android:layout\_gravity="center\_horizontal"*

*android:paddingStart="@dimen/dashboard\_padding\_start"*

*android:paddingEnd="@dimen/dashboard\_padding\_end"*

*android:paddingTop="@dimen/dashboard\_padding\_top"*

*android:paddingBottom="@dimen/dashboard\_padding\_bottom"*

*android:orientation="vertical"*

*/>*

*</ScrollView>*

看了上面的布局文件可以知道Settings的选项视图应该就是显示在dashboard\_container中了。

我们来看下面这张时序图：

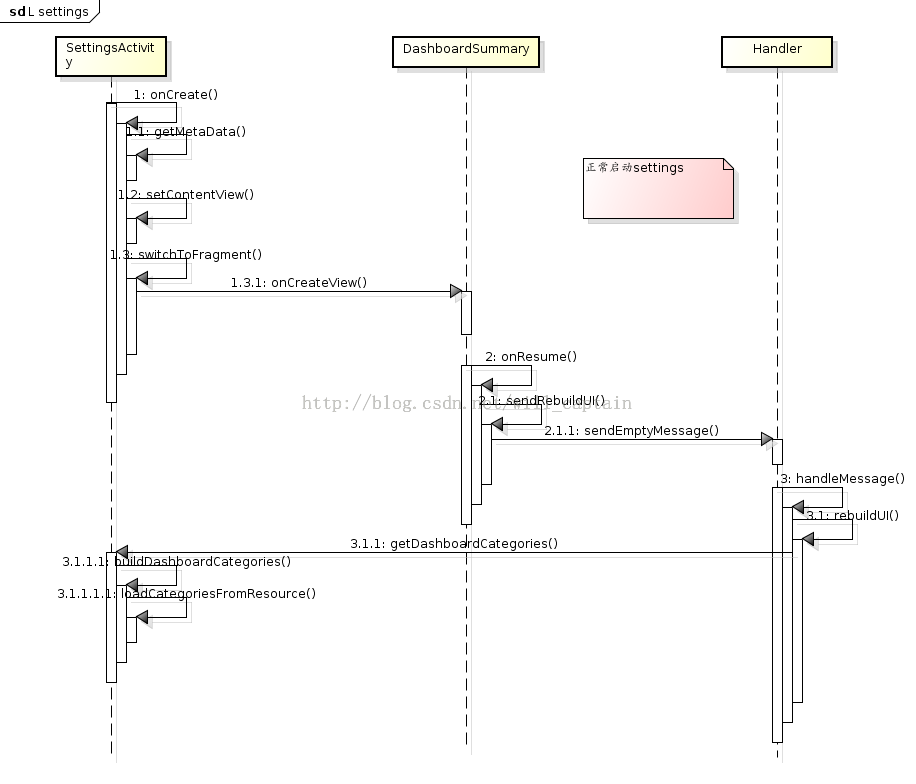


图3.1 L Settings 初始化时序图

DashboardSummary走完onCreateView方法后会走onResume,然后一路下来又会调到SettingsActivity的

*loadCategoriesFromResource(R.xml.dashboard\_categories, categories);*

这一步是通过 R.xml.dashboard\_categories来加载categories，这里的categorys为ArrayList<DashboardCategory>mCategories。接着来看看dashboard\_categories.xml这个文件吧

：

*<dashboard-categories*

*xmlns:android="http://schemas.android.com/apk/res/android">*

*<!-- WIRELESS and NETWORKS -->*

*<dashboard-category*

*android:id="@+id/wireless\_section"*

*android:title="@string/header\_category\_wireless\_networks" >*

*<!-- Wifi -->*

*<dashboard-tile*

*android:id="@+id/wifi\_settings"*

*android:title="@string/wifi\_settings\_title"*

*android:fragment="com.android.settings.wifi.WifiSettings"*

*android:icon="@drawable/ic\_settings\_wireless"*

*/>*

*<!-- Bluetooth -->*

*<dashboard-tile*

*android:id="@+id/bluetooth\_settings"*

*android:title="@string/bluetooth\_settings\_title"*

*android:fragment="com.android.settings.bluetooth.BluetoothSettings"*

*android:icon="@drawable/ic\_settings\_bluetooth2"*

*/>*

*<!-- SIM Cards -->*

*<dashboard-tile*

*android:id="@+id/sim\_settings"*

*android:title="@string/sim\_settings\_title"*

*android:fragment="com.android.settings.sim.SimSettings"*

*android:icon="@drawable/ic\_sim\_sd"*

*/>*

*<!-- Data Usage -->*

*<dashboard-tile*

*android:id="@+id/data\_usage\_settings"*

*android:title="@string/data\_usage\_summary\_title"*

*android:fragment="com.android.settings.DataUsageSummary"*

*android:icon="@drawable/ic\_settings\_data\_usage"*

*/>*

*<!-- Operator hook -->*

*<dashboard-tile*

*android:id="@+id/operator\_settings"*

*android:fragment="com.android.settings.WirelessSettings" >*

*<intent android:action="com.android.settings.OPERATOR\_APPLICATION\_SETTING" />*

*</dashboard-tile>*

*<!-- Other wireless and network controls -->*

*<dashboard-tile*

*android:id="@+id/wireless\_settings"*

*android:title="@string/radio\_controls\_title"*

*android:fragment="com.android.settings.WirelessSettings"*

*android:icon="@drawable/ic\_settings\_more"*

*/>*

*</dashboard-category>*

*<!-- DEVICE -->*

*<dashboard-category*

*android:id="@+id/device\_section"*

*android:title="@string/header\_category\_device" >*

*<!-- Home -->*

*<dashboard-tile*

*android:id="@+id/home\_settings"*

*android:title="@string/home\_settings"*

*android:fragment="com.android.settings.HomeSettings"*

*android:icon="@drawable/ic\_settings\_home"*

*/>*

*<!-- Display -->*

*<dashboard-tile*

*android:id="@+id/display\_settings"*

*android:title="@string/display\_settings"*

*android:fragment="com.android.settings.DisplaySettings"*

*android:icon="@drawable/ic\_settings\_display"*

*/>*

*<!-- Notifications -->*

*<dashboard-tile*

*android:id="@+id/notification\_settings"*

*android:title="@string/notification\_settings"*

*android:fragment="com.android.settings.notification.NotificationSettings"*

*android:icon="@drawable/ic\_settings\_notifications"*

*/>*

*<!-- Storage -->*

*<dashboard-tile*

*android:id="@+id/storage\_settings"*

*android:title="@string/storage\_settings"*

*android:fragment="com.android.settings.deviceinfo.Memory"*

*android:icon="@drawable/ic\_settings\_storage"*

*/>*

*<!-- Battery -->*

*<dashboard-tile*

*android:id="@+id/battery\_settings"*

*android:title="@string/power\_usage\_summary\_title"*

*android:fragment="com.android.settings.fuelgauge.PowerUsageSummary"*

*android:icon="@drawable/ic\_settings\_battery"*

*/>*

*<!-- Application Settings -->*

*<dashboard-tile*

*android:id="@+id/application\_settings"*

*android:title="@string/applications\_settings"*

*android:fragment="com.android.settings.applications.ManageApplications"*

*android:icon="@drawable/ic\_settings\_applications"*

*/>*

*<!-- Manage users -->*

*<dashboard-tile*

*android:id="@+id/user\_settings"*

*android:title="@string/user\_settings\_title"*

*android:fragment="com.android.settings.users.UserSettings"*

*android:icon="@drawable/ic\_settings\_multiuser"*

*/>*

*<!-- Manage NFC payment apps -->*

*<dashboard-tile*

*android:id="@+id/nfc\_payment\_settings"*

*android:title="@string/nfc\_payment\_settings\_title"*

*android:fragment="com.android.settings.nfc.PaymentSettings"*

*android:icon="@drawable/ic\_settings\_nfc\_payment"*

*/>*

*<!-- Manufacturer hook -->*

*<dashboard-tile*

*android:id="@+id/manufacturer\_settings"*

*android:fragment="com.android.settings.WirelessSettings">*

*<intent android:action="com.android.settings.MANUFACTURER\_APPLICATION\_SETTING" />*

*</dashboard-tile>*

*</dashboard-category>*

*<!-- PERSONAL -->*

*<dashboard-category*

*android:id="@+id/personal\_section"*

*android:title="@string/header\_category\_personal" >*

*<!-- Location -->*

*<dashboard-tile*

*android:id="@+id/location\_settings"*

*android:title="@string/location\_settings\_title"*

*android:fragment="com.android.settings.location.LocationSettings"*

*android:icon="@drawable/ic\_settings\_location"*

*/>*

*<!-- Security -->*

*<dashboard-tile*

*android:id="@+id/security\_settings"*

*android:title="@string/security\_settings\_title"*

*android:fragment="com.android.settings.SecuritySettings"*

*android:icon="@drawable/ic\_settings\_security"*

*/>*

*<!-- Account -->*

*<dashboard-tile*

*android:id="@+id/account\_settings"*

*android:title="@string/account\_settings\_title"*

*android:fragment="com.android.settings.accounts.AccountSettings"*

*android:icon="@drawable/ic\_settings\_accounts"*

*/>*

*<!-- Language -->*

*<dashboard-tile*

*android:id="@+id/language\_settings"*

*android:title="@string/language\_settings"*

*android:fragment="com.android.settings.inputmethod.InputMethodAndLanguageSettings"*

*android:icon="@drawable/ic\_settings\_language"*

*/>*

*<!-- Backup and reset -->*

*<dashboard-tile*

*android:id="@+id/privacy\_settings"*

*android:title="@string/privacy\_settings"*

*android:fragment="com.android.settings.PrivacySettings"*

*android:icon="@drawable/ic\_settings\_backup"*

*/>*

*</dashboard-category>*

*<!-- SYSTEM -->*

*<dashboard-category*

*android:id="@+id/system\_section"*

*android:title="@string/header\_category\_system" >*

*<!-- Date & Time -->*

*<dashboard-tile*

*android:id="@+id/date\_time\_settings"*

*android:title="@string/date\_and\_time\_settings\_title"*

*android:fragment="com.android.settings.DateTimeSettings"*

*android:icon="@drawable/ic\_settings\_date\_time"*

*/>*

*<!-- Accessibility feedback -->*

*<dashboard-tile*

*android:id="@+id/accessibility\_settings"*

*android:title="@string/accessibility\_settings"*

*android:fragment="com.android.settings.accessibility.AccessibilitySettings"*

*android:icon="@drawable/ic\_settings\_accessibility"*

*/>*

*<!-- Print -->*

*<dashboard-tile*

*android:id="@+id/print\_settings"*

*android:title="@string/print\_settings"*

*android:fragment="com.android.settings.print.PrintSettingsFragment"*

*android:icon="@drawable/ic\_settings\_print"*

*/>*

*<!-- Development -->*

*<dashboard-tile*

*android:id="@+id/development\_settings"*

*android:title="@string/development\_settings\_title"*

*android:fragment="com.android.settings.DevelopmentSettings"*

*android:icon="@drawable/ic\_settings\_development"*

*/>*

*<!-- About Device -->*

*<dashboard-tile*

*android:id="@+id/about\_settings"*

*android:title="@string/about\_settings"*

*android:fragment="com.android.settings.DeviceInfoSettings"*

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

*/>*

*</dashboard-category>*

*</dashboard-categories>*

根据这个文件我们可以知道了，所谓的dashboard就是Settings模块首界面的一个抽象。而dashboard-categorys则是设置分类集合的抽象，而dashboard-category是分类的抽象，dashboard-tile就是分类下每个选项的抽象了。代码中的List<DashboardCategory>对应dashboard-categorys， DashboardCategory对应dashboard-category，而dashboard-tile则对因代码中的DashboardTile。

当加载完这些对象后SettingsActivity会将得到的 mCategories 返回给DashboardSummary来初始化Settings的设置选项。

下面这段代码就是DashboardSummary.rebuildUI()中完成界面的初始化

***long****start = System.currentTimeMillis();*

***final****Resources res = getResources();*

*mDashboard.removeAllViews();*

*List<DashboardCategory> categories =*

*((SettingsActivity) context).getDashboardCategories(****true****);*

***final******int****count = categories.size();*

***for****(****int****n = 0; n < count; n++) {*

*DashboardCategory category = categories.get(n);*

*View categoryView = mLayoutInflater.inflate(R.layout.dashboard\_category, mDashboard,*

***false****);*

*TextView categoryLabel = (TextView) categoryView.findViewById(R.id.category\_title);*

*categoryLabel.setText(category.getTitle(res));*

*ViewGroup categoryContent =*

*(ViewGroup) categoryView.findViewById(R.id.category\_content);*

***final******int****tilesCount = category.getTilesCount();*

***for****(****int****i = 0; i < tilesCount; i++) {*

*DashboardTile tile = category.getTile(i);*

*DashboardTileView tileView =****new****DashboardTileView(context);*

*updateTileView(context, res, tile, tileView.getImageView(),*

*tileView.getTitleTextView(), tileView.getStatusTextView());*

*tileView.setTile(tile);*

*categoryContent.addView(tileView);*

*}*

*// Add the category*

*mDashboard.addView(categoryView);*

*}*

这段代码我就不具体分析了，逻辑很简单，遍历categories这个列表来获取DashboardCategory对象，将所有DashboardCategory对象和DashboardCategory对象中的DashboardTile对象转化为视图对象并添加到主视图对象mDashboard中。

到这里SettingsActivity的onCreate方法就算结束了。总结一下，

    1.onCreate完成的任务是切换DashboardSmmary这个Fragment，然后从dashboard\_categories.xml中读取预先配置好的文件来初始化Settings的首界面视图。

    2.L中舍弃了Header类，取而代之的是DashboardCategory和DashboardTile类。

**4.0 L Settings模块storage选项分析**

当用户点击L settings模块中的storage选项，程序的流程又是什么样的，下面就具体看一下。

前面知道了配置文件中的dashboard-tile节点对应的是DashboardTile类，而DashboardTile对应的又是DashboardTileView这个视图类，一个DashboardTileView的对象就是屏幕上的一个设置选项，当我们点击storage选项时就会激发DashboardTileView的onClick响应：

*@Override*

***public******void****onClick(View v) {*

***if****(mTile.fragment !=****null****) {*

*Utils.startWithFragment(getContext(), mTile.fragment,         mTile.fragmentArguments,****null****, 0,*

*mTile.titleRes, mTile.getTitle(getResources()));*

*}****else******if****(mTile.intent !=****null****) {*

*getContext().startActivity(mTile.intent);*

*}*

*}*

这里看到 mTile.fragment 这个变量，它所代表的就是上面dashboard\_categories.xml文件下的

*<!-- Storage -->*

*<dashboard-tile*

*android:id="@+id/storage\_settings"*

*android:title="@string/storage\_settings"*

*android:fragment="c**om.android.settings.deviceinfo.Memory"*

*android:icon="@drawable/ic\_settings\_storage"*

*/>*

这个节点中的*android:fragment="com.android.settings.deviceinfo.Memory"*属性。

接着一路跟踪Utils.*startWithFragment*方法，走到下面这段代码：

***public******static******void****startWithFragment(Context context, String fragmentName, Bundle args,*

*Fragment resultTo,****int****resultRequestCode,****int****titleResId, CharSequence title,*

***boolean****isShortcut) {*

*Intent intent = onBuildStartFragmentIntent(context, fragmentName, args, titleResId,*

*title, isShortcut);*

***if****(resultTo ==****null****) {*

*context.startActivity(intent);*

*}****else****{*

*resultTo.startActivityForResult(intent, resultRequestCode);*

*}*

*}*

我们先看*onBuildStartFragmentIntent这个方法：*

***public******static****Intent onBuildStartFragmentIntent(Context context, String fragmentName,*

*Bundle args,****int****titleResId, CharSequence title,****boolean****isShortcut) {*

*Intent intent =****new****Intent(Intent.ACTION\_MAIN);*

*intent.setClass(context, SubSettings.****class****);*

*intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT, fragmentName);*

*intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_ARGUMENTS, args);*

*intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_TITLE\_RESID, titleResId);*

*intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_TITLE, title);*

*intent.putExtra(SettingsActivity.EXTRA\_SHOW\_FRAGMENT\_AS\_SHORTCUT, isShortcut);*

***return****intent;*

*}*

这里就是生成一个Intent嘛，这个Intent使用来启动一个叫SubSettings的Activity的，而细心的同学也能发现SubSettings也是继承自SettingsActivity的。然后给Intent加了一些额外的信息。

接着就是context.startActivity(intent);来启动这个Activity了。

4.1storage 之 SubSettings 启动流程

由于SubSettings也是继承的SettingsActivity而且也几乎没做任何额外的扩充，这里分析SubSettings的启动流程也可已直接分析SettingsActivity。

在SettingsActivity的onCreate方法中有这么一行：

*mIsShowingDashboard = className.equals(Settings.****class****.getName());*

因为这次我们启动的是SubSettings，所以mIsShowingDashboard为false。所以代码会走下面这段：

***if****(!mIsShowingDashboard) {*

*// Search is shown we are launched thru a Settings "shortcut". UP will be shown*

*// only if it is a sub settings*

***if****(mIsShortcut) {*

*mDisplayHomeAsUpEnabled = isSubSettings;*

*mDisplaySearch =****false****;*

*}****else******if****(isSubSettings) {*

*mDisplayHomeAsUpEnabled =****true****;*

*mDisplaySearch =****true****;*

*}****else****{*

*mDisplayHomeAsUpEnabled =****false****;*

*mDisplaySearch =****false****;*

*}*

*setTitleFromIntent(intent);*

*Bundle initialArguments = intent.getBundleExtra(EXTRA\_SHOW\_FRAGMENT\_ARGUMENTS);*

*switchToFragment(initialFragmentName, initialArguments,****true****,****false****,*

*mInitialTitleResId, mInitialTitle,****false****);*

*}*

重点看switchToFragment这个方法：

***private****Fragment switchToFragment(String fragmentName, Bundle args,****boolean****validate,*

***boolean****addToBackStack,****int****titleResId, CharSequence title,****boolean****withTransition) {*

***if****(validate && !isValidFragment(fragmentName)) {*

***throw******new****IllegalArgumentException("Invalid fragment for this activity: "*

*+ fragmentName);*

*}*

*Fragment f = Fragment.instantiate(****this****, fragmentName, args);*

*FragmentTransaction transaction = getFragmentManager().beginTransaction();*

*transaction.replace(R.id.main\_content, f);*

***if****(withTransition) {*

*TransitionManager.beginDelayedTransition(mContent);*

*}*

***if****(addToBackStack) {*

*transaction.addToBackStack(SettingsActivity.BACK\_STACK\_PREFS);*

*}*

***if****(titleResId > 0) {*

*transaction.setBreadCrumbTitle(titleResId);*

*}****else******if****(title !=****null****) {*

*transaction.setBreadCrumbTitle(title);*

*}*

*transaction.commitAllowingStateLoss();*

*getFragmentManager().executePendingTransactions();*

***return****f;*

*}*

很明显，这个方法就是用来填充一个名字为fragmentName的Fragment的，这里的fragmentName是从最前面一路传过来的，值为com.android.settings.deviceinfo.Memory 。我们可以转移阵地了，从这一刻开始，界面会被Memory这个Fragment填充，而这个Fragment因该就是用来显示storage使用情况的。

4.2              Storage 之 Memory 流程分析

还是从Memory.java 的onCreate看起，这里第一个关键的方法是

*addPreferencesFromResource(R.xml.device\_info\_memory);*

由于此方法涉及到的代码比较多，这里可以先看一下关于这个方法的时序图：

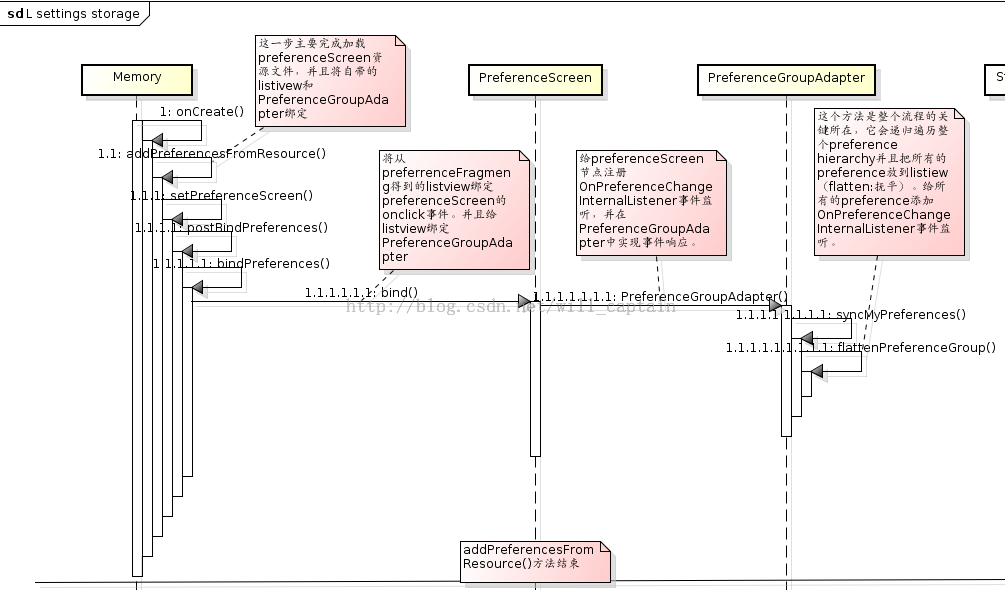


图4.1 addPreferencesFromResource方法相关时序图

首先addPreferencesFromResource会从文件R.xml.*device\_info\_memory* 来加载Memory下的节点构成：

*<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android"*

*xmlns:settings="http://schemas.android.com/apk/res/com.android.settings"*

*android:title="@string/storage\_settings"*

*settings:keywords="@string/keywords\_storage">*

*<!-- Preference categories are dynamically created based on the list of available storage volumes -->*

*</PreferenceScreen>*

这个文件很简单，只是单单指出了Memory下的根节点PregerenceScreen，看到上面的这句注释也指出Preference Category会根据获得的storage volumes(类似与分区或磁盘)被动态的添加PreferenceScreen下面。

设置完Memory的根节点PregerenceScreen后会有一个bindPreference的操作，这里会给Memory自带的ListView设置OnItemClickListener和绑定PreferenceGroupAdapter适配器。在PreferenceGroupAdapter中获得了PregerenceScreen对象的引用。

由于当前PreferenceScreen没有任何子节点，暂时不对PreferenceGroupAdapte的实现做具体分析。

到此为止addPreferencesFromResource方法就算结束了，紧接着又来了一个有意思的方法：

*addCategory(StorageVolumePreferenceCategory.buildForInternal(context));*

这个方法首先是获得一个表示内部存储的StorageVolumePreferenceCategory，接着调用 addCategory 将其添加到PreferenceScreen节点下：

***private******void****addCategory(StorageVolumePreferenceCategory category) {*

*mCategories.add(category);*

*getPreferenceScreen().addPreference(category);*

*category.init();*

*}*

然后调用自身的init方法来初始化与Internal storage有关的信息。init方法里面的代码就不贴了，比较长，主要就是获得代表不同存储信息的Preference，如总容量，可用空间，应用，图片视频，音频，下载内容等，然后把这些Preference添加到StorageVolumePreferenceCategory节点下。

添加完内部存储后，接下来的代码回去遍历其他存储，这里默认只有内部存储，添加其他存储忽略。

看完了onCreate接下来看onResume,在这个方法中会调用StorageVolumePreferenceCategory的onResume方法来初始化里面每个preference的属性值。时序图如下：

图4.2 StorageVolumePreferenceCategory的onResume方法时序图

这里具体实现就不多讲，StorageVolumePreferenceCategory是调用StorageMeasurement类来完成计算，再通过回调StorageVolumePreferenceCategory内部实现的接口MeasurementReceiver中的updateApproximate和updateDetails方法来分别更新总使用量和分类的具体占用量，完成这已系列计算后Memory的界面如下图所示：

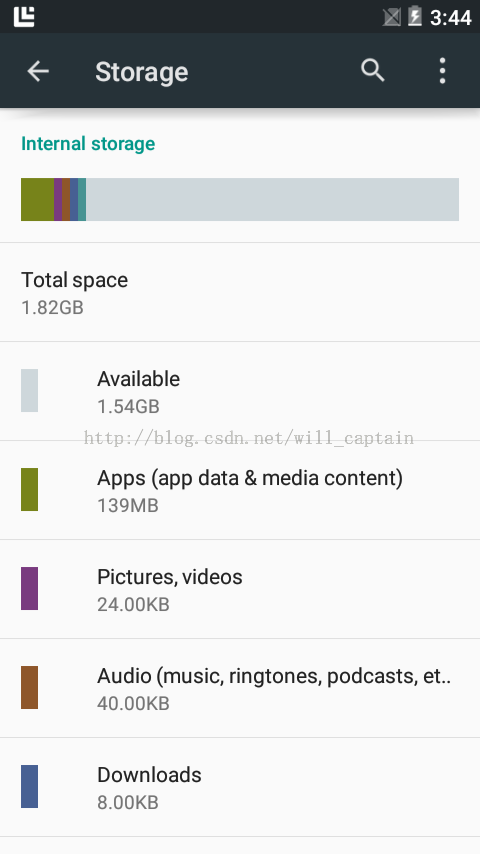


图4.3 L Settings模块storage界面

到此为止，storage的初始化就结束了，来总结一下storage的一个初始化流程：

     1.遍历本地的storage volume并将其转换为StorageVolumePreferenceCategory添加到PreferenceScreen。

     2.添加StorageVolumePreferenceCategory内部的各种类型的使用量StorageItemPreference。

     3.初始化StorageVolumePreferenceCategory内部的各个StorageItemPreference的属性值。