  最近在看一个bug，系统切换语言后，本来退到后台的音乐，会在通知栏上显示通知。为了解决这个bug，我学习了下android的语言切换流程，也参考了大量其他人的资料。（主要参考了http://blog.csdn.net/wqhjfree/article/details/8244520）在这里我将自己的探索记录下来，作为自己的学习记录，也希望能对有同样需要的人有个帮助。刚学android不久，如果中间有什么问题不对的请多多谅解，并指出错误，多交流，共同进步下。

       1.从setting入手，我们可以知道，在setting中，语言设置的入口在一个叫做LocalePicker的类里面，当点击里面的一个语言后会调用 onLocaleSelected(final Locale locale)方法。通过分析这个方法我们可以知道，最后切换语言并不是在这里完成，而是去调用了fromwork下的updateLocale（）方法（位于frameworks/base/core/java/com/android/internal/app/LocalePicker）。

     2.居然他是调用的fromwork下LocalePicker类中的updateLocale（）方法，我们继续往下看。通过分析，我们知道这个方法也不是一个最终的处理方法，而是去调用了ActivityManagerService.java中的updateConfiguration()函数。

    3.看看updateConfiguration（）方法做了什么事情，这里做了一些值得判断和初始工作。最重要的是他调用的updateConfigurationLocked(values, null, false, false)方法；

   4.updateConfigurationLocked中主要做了两件事：（1）改变现在的 configuration（这是一个系统配置的类，有兴趣的可以去了解下）；（2）确保所有正在运行的Activity都运行改变后的configuration。下面可以看看他到底是怎么完成这两件事的。首先，通过updateFrom(values)判断是不是真的语言发生了变化，如果改变了，从if条件走，在if里面，前面做一些判断之类的工作，到此也完成了第一步的工作。最重要的是for循环里面的操作，首先得到了所有运行过的app的集合，然后对每个app调用scheduleConfigurationChanged（）方法，进行语言的切换工作。

  5.scheduleConfigurationChanged是在ActivityThread中，这个方执行了 updatePendingConfiguration(config)和 queueOrSendMessage(H.CONFIGURATION\_CHANGED, config)两个方法。前面一个方法是更新Configuration；最主要的操作在queueOrSendMessage（）里面的handleConfigurationChanged((Configuration)msg.obj, null)方法中。

6.接着对handleConfigurationChanged进行分析，从中我们不难发现applyConfigurationToResourcesLocked（）这个是一个重新配置资源的函数，performConfigurationChanged(callbacks.get(i), config)这个方法是执行Configuration的改变。即最终完成语言的切换。

7.详细的分析下applyConfigurationToResourcesLocked做了哪些工作，updateFrom(config) 把config更新到Configuration中，后面 最主要的是在while () 中做了资源更新和删除就资源的操作。

8.performConfigurationChanged方法中，这是完成语言切换的最后一步了，首先判断当前activity的config和新的config是否一样，如果是一样什么都不做；如果不一样，则重启app，重新加载资源达到切换语言。

9.总结语言切换的大概流程是，判断configuration中的local即语言是不是有改变，如果有改变即为要切换语言。执行切换语言的时候，对那些已经运行过的程序，执行一个资源的清除和重新加载的过程，就完成了整个系统的语言切换。

最后，那个bug也就迎刃而解了，music运行之后，切换语言，他销毁了一次，又重新启动了，导致状态栏上有了通知提示。

android语言切换是在packages/apps/Settings/com/android/settings/LocalePicker.java的updateLocale()函数中调用.

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. /\*\*
2. \* Requests the system to update the system locale. Note that the system looks halted
3. \* for a while during the Locale migration, so the caller need to take care of it.
4. \*/
5. **public** **static** **void** updateLocale(Locale locale) {
6. **try** {
7. IActivityManager am = ActivityManagerNative.getDefault();
8. Configuration config = am.getConfiguration();
10. config.locale = locale;
12. // indicate this isn't some passing default - the user wants this remembered
13. config.userSetLocale = **true**;
15. am.updateConfiguration(config);
16. // Trigger the dirty bit for the Settings Provider.
17. BackupManager.dataChanged("com.android.providers.settings");
18. } **catch** (RemoteException e) {
19. // Intentionally left blank
20. }
21. }

 从注释可以看出, 只要本地local改变就会调用该函数. 查看ActivityManagerNative的getDefault()可以看到, 该函数返回的是远程服务对象ActivityManagerServices.java在本地的一个代理.  最终调用的是ActivityManagerService.java中的updateConfiguration()函数.

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. **public** **void** updateConfiguration(Configuration values) {
2. enforceCallingPermission(android.Manifest.permission.CHANGE\_CONFIGURATION,
3. "updateConfiguration()");
5. **synchronized**(**this**) {
6. **if** (values == **null** && mWindowManager != **null**) {
7. // sentinel: fetch the current configuration from the window manager
8. values = mWindowManager.computeNewConfiguration();
9. }
11. **if** (mWindowManager != **null**) {
12. mProcessList.applyDisplaySize(mWindowManager);
13. }
15. **final** **long** origId = Binder.clearCallingIdentity();
16. **if** (values != **null**) {
17. Settings.System.clearConfiguration(values);
18. }
19. updateConfigurationLocked(values, **null**, **false**, **false**);
20. Binder.restoreCallingIdentity(origId);
21. }
22. }

 该函数, 首先进行的是权限的校验. 然后调用updateConfigurationLocked()函数.

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. /\*\*
2. \* Do either or both things: (1) change the current configuration, and (2)
3. \* make sure the given activity is running with the (now) current
4. \* configuration.  Returns true if the activity has been left running, or
5. \* false if <var>starting</var> is being destroyed to match the new
6. \* configuration.
7. \* @param persistent TODO
8. \*/
9. **public** **boolean** updateConfigurationLocked(Configuration values,
10. ActivityRecord starting, **boolean** persistent, **boolean** initLocale) {
11. **int** changes = 0;
13. **boolean** kept = **true**;
15. **if** (values != **null**) {
16. Configuration newConfig = **new** Configuration(mConfiguration);
17. changes = newConfig.updateFrom(values);
18. **if** (changes != 0) {
19. **if** (DEBUG\_SWITCH || DEBUG\_CONFIGURATION) {
20. Slog.i(TAG, "Updating configuration to: " + values);
21. }
23. EventLog.writeEvent(EventLogTags.CONFIGURATION\_CHANGED, changes);
25. **if** (values.locale != **null** && !initLocale) {
26. saveLocaleLocked(values.locale,
27. !values.locale.equals(mConfiguration.locale),
28. values.userSetLocale, values.simSetLocale);
29. }

32. mConfigurationSeq++;
33. **if** (mConfigurationSeq <= 0) {
34. mConfigurationSeq = 1;
35. }
36. newConfig.seq = mConfigurationSeq;
37. mConfiguration = newConfig;
38. Slog.i(TAG, "Config changed: " + newConfig);
40. **final** Configuration configCopy = **new** Configuration(mConfiguration);
42. AttributeCache ac = AttributeCache.instance();
43. **if** (ac != **null**) {
44. ac.updateConfiguration(configCopy);
45. }
47. // Make sure all resources in our process are updated
48. // right now, so that anyone who is going to retrieve
49. // resource values after we return will be sure to get
50. // the new ones.  This is especially important during
51. // boot, where the first config change needs to guarantee
52. // all resources have that config before following boot
53. // code is executed.
54. mSystemThread.applyConfigurationToResources(configCopy);
56. **if** (persistent && Settings.System.hasInterestingConfigurationChanges(changes)) {
57. Message msg = mHandler.obtainMessage(UPDATE\_CONFIGURATION\_MSG);
58. msg.obj = **new** Configuration(configCopy);
59. mHandler.sendMessage(msg);
60. }
62. **for** (**int** i=mLruProcesses.size()-1; i>=0; i--) {
63. ProcessRecord app = mLruProcesses.get(i);
64. **try** {
65. **if** (app.thread != **null**) {
66. **if** (DEBUG\_CONFIGURATION) Slog.v(TAG, "Sending to proc "
67. + app.processName + " new config " + mConfiguration);
68. app.thread.scheduleConfigurationChanged(configCopy);
69. }
70. } **catch** (Exception e) {
71. }
72. }
73. Intent intent = **new** Intent(Intent.ACTION\_CONFIGURATION\_CHANGED);
74. intent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY
75. | Intent.FLAG\_RECEIVER\_REPLACE\_PENDING);
76. broadcastIntentLocked(**null**, **null**, intent, **null**, **null**, 0, **null**, **null**,
77. **null**, **false**, **false**, MY\_PID, Process.SYSTEM\_UID);
78. **if** ((changes&ActivityInfo.CONFIG\_LOCALE) != 0) {
79. broadcastIntentLocked(**null**, **null**,
80. **new** Intent(Intent.ACTION\_LOCALE\_CHANGED),
81. **null**, **null**, 0, **null**, **null**,
82. **null**, **false**, **false**, MY\_PID, Process.SYSTEM\_UID);
83. }
85. }
86. }
88. **if** (changes != 0 && starting == **null**) {
89. // If the configuration changed, and the caller is not already
90. // in the process of starting an activity, then find the top
91. // activity to check if its configuration needs to change.
92. starting = mMainStack.topRunningActivityLocked(**null**);
93. }
95. **if** (starting != **null**) {
96. kept = mMainStack.ensureActivityConfigurationLocked(starting, changes);
97. // And we need to make sure at this point that all other activities
98. // are made visible with the correct configuration.
99. mMainStack.ensureActivitiesVisibleLocked(starting, changes);
100. }
102. **if** (values != **null** && mWindowManager != **null**) {
103. mWindowManager.setNewConfiguration(mConfiguration);
104. }
106. **return** kept;
107. }

整个语言切换就在这个函数中完成. 咋一看似乎没感觉到该函数做了哪些事情. 我们首先来看注释: Do either or both things: (1) change the current configuration, and (2)  
 make sure the given activity is running with the (now) current. configuration大概意思是: 这个函数做了两件事情. (1). 改变当前的configuration. 意思就是让改变的configuration更新到当前configuration. (2) 确保所有正在运行的activity都能更新改变后的configuration.(这点是关键.) . 我们按照这个思路看看android是如何更新configuration. 查看代码 , 首先看到 这个函数首先判断values是否为空, 这里values肯定不为空的, 然后changes = newConfig.updateFrom(values); 我们看看updateFrom做了什么操作.

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. /\*\*
2. \* Copy the fields from delta into this Configuration object, keeping
3. \* track of which ones have changed.  Any undefined fields in
4. \* <var>delta</var> are ignored and not copied in to the current
5. \* Configuration.
6. \* @return Returns a bit mask of the changed fields, as per
7. \* {@link #diff}.
8. \*/
9. **public** **int** updateFrom(Configuration delta) {
10. **int** changed = 0;
11. ...
12. **if** (delta.locale != **null**
13. && (locale == **null** || !locale.equals(delta.locale))) {
14. changed |= ActivityInfo.CONFIG\_LOCALE;
15. locale = delta.locale != **null**
16. ? (Locale) delta.locale.clone() : **null**;
17. textLayoutDirection = LocaleUtil.getLayoutDirectionFromLocale(locale);
18. }
19. **if** (delta.userSetLocale && (!userSetLocale || ((changed & ActivityInfo.CONFIG\_LOCALE) != 0)))
20. {
21. userSetLocale = **true**;
22. changed |= ActivityInfo.CONFIG\_LOCALE;
23. }
24. ...
25. **return** changed;
26. }

因为语言改变了, 那么 (!locale.equals(delta.locale)) 是true. changed 大于0, 然后return changed. 回到ActivityManagerService.java的updateConfigurationLocked函数, 因为changed不为0 , 所以走if这个流程.  继续看代码

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. **for** (**int** i=mLruProcesses.size()-1; i>=0; i--) {
2. ProcessRecord app = mLruProcesses.get(i);
3. **try** {
4. **if** (app.thread != **null**) {
5. **if** (DEBUG\_CONFIGURATION) Slog.v(TAG, "Sending to proc "
6. + app.processName + " new config " + mConfiguration);
7. app.thread.scheduleConfigurationChanged(configCopy);
8. }
9. } **catch** (Exception e) {
10. }
11. }

首先看到的是mLurProcesses 是ArrayList<ProcessRecord>类型.  LRU : Least Recently Used保存所有运行过的进程.  ProcessRecord进程类, 一个apk文件运行时会对应一个进程. app.thread. 此处的thread代表的是ApplicationThreadNative.java类型.  然后调用其scheduleConfigurationChanged();  查看该函数

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. **public** **final** **void** scheduleConfigurationChanged(Configuration config)
2. **throws** RemoteException {
3. Parcel data = Parcel.obtain();
4. data.writeInterfaceToken(IApplicationThread.descriptor);
5. config.writeToParcel(data, 0);
6. mRemote.transact(SCHEDULE\_CONFIGURATION\_CHANGED\_TRANSACTION, data, **null**,
7. IBinder.FLAG\_ONEWAY);
8. data.recycle();
9. }

又是通过binder调用, 所以 , binder在android中是一个很重要的概念. 此处远程调用的是ActivityThread.java中的私有内部内ApplicationThread

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. **private** **class** ApplicationThread **extends** ApplicationThreadNative {
2. **private** **static** **final** String HEAP\_COLUMN = "%13s %8s %8s %8s %8s %8s %8s";
3. **private** **static** **final** String ONE\_COUNT\_COLUMN = "%21s %8d";
4. **private** **static** **final** String TWO\_COUNT\_COLUMNS = "%21s %8d %21s %8d";
5. **private** **static** **final** String TWO\_COUNT\_COLUMNS\_DB = "%21s %8d %21s %8d";
6. **private** **static** **final** String DB\_INFO\_FORMAT = "  %8s %8s %14s %14s  %s";

9. ...
10. **public** **void** scheduleConfigurationChanged(Configuration config) {
11. updatePendingConfiguration(config);
12. queueOrSendMessage(H.CONFIGURATION\_CHANGED, config);
13. }
14. ...
15. }

而ApplicationThread中的handler的CONFIGURATION\_CHANGED是调用handleConfigurationChanged()

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. **final** **void** handleConfigurationChanged(Configuration config, CompatibilityInfo compat) {
3. ArrayList<ComponentCallbacks2> callbacks = **null**;
5. ...         ...
6. applyConfigurationToResourcesLocked(config, compat);
8. ...
10. callbacks = collectComponentCallbacksLocked(**false**, config);
11. ...
13. **if** (callbacks != **null**) {
14. **final** **int** N = callbacks.size();
15. **for** (**int** i=0; i<N; i++) {
16. performConfigurationChanged(callbacks.get(i), config);
17. }
18. }

这个函数首先是调用applyConfigurationToResourcesLocked(). 看函数名大概可以推测: 将configuration应用到resources.这里configuration改变的是local 本地语言. 那而resources资源包含不就包含了语言, 图片这些资源吗.

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. **final** **boolean** applyConfigurationToResourcesLocked(Configuration config,
2. CompatibilityInfo compat) {
4. **int** changes = mResConfiguration.updateFrom(config);
5. DisplayMetrics dm = getDisplayMetricsLocked(**null**, **true**);

8. **if** (compat != **null** && (mResCompatibilityInfo == **null** ||
9. !mResCompatibilityInfo.equals(compat))) {
10. mResCompatibilityInfo = compat;
11. changes |= ActivityInfo.CONFIG\_SCREEN\_LAYOUT
12. | ActivityInfo.CONFIG\_SCREEN\_SIZE
13. | ActivityInfo.CONFIG\_SMALLEST\_SCREEN\_SIZE;
14. }
16. ...
18. Resources.updateSystemConfiguration(config, dm, compat);
20. ...
22. Iterator<WeakReference<Resources>> it =
23. mActiveResources.values().iterator();
24. **while** (it.hasNext()) {
25. WeakReference<Resources> v = it.next();
26. Resources r = v.get();
27. **if** (r != **null**) {
28. **if** (DEBUG\_CONFIGURATION) Slog.v(TAG, "Changing resources "
29. + r + " config to: " + config);
30. r.updateConfiguration(config, dm, compat);
31. //Slog.i(TAG, "Updated app resources " + v.getKey()
32. //        + " " + r + ": " + r.getConfiguration());
33. } **else** {
34. //Slog.i(TAG, "Removing old resources " + v.getKey());
35. it.remove();
36. }
37. }
39. **return** changes != 0;
40. }

Resources.updateSystemConfiguration()清除一部分系统资源, 并且将config更新到Resources, 而Resources包含了一个AssetManager对象, 该对象的核心实现是在AssetManager.cpp中完成的. 然后循环清空mActivityResources资源. 再回到handleConfigurationChanged()函数, 执行完updateSystemConfiguration后, 会循环该进程的所有activity:

if (callbacks != null) {

            final int N = callbacks.size();  
            for (int i=0; i<N; i++) {  
                performConfigurationChanged(callbacks.get(i), config);  
            }  
        }

再来看performConfigurationChanged的实现:

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. **private** **final** **void** performConfigurationChanged(
2. ComponentCallbacks2 cb, Configuration config) {
3. // Only for Activity objects, check that they actually call up to their
4. // superclass implementation.  ComponentCallbacks2 is an interface, so
5. // we check the runtime type and act accordingly.
6. Activity activity = (cb **instanceof** Activity) ? (Activity) cb : **null**;
7. **if** (activity != **null**) {
8. activity.mCalled = **false**;
9. }
11. **boolean** shouldChangeConfig = **false**;
12. **if** ((activity == **null**) || (activity.mCurrentConfig == **null**)) {
13. shouldChangeConfig = **true**;
14. } **else** {
16. // If the new config is the same as the config this Activity
17. // is already running with then don't bother calling
18. // onConfigurationChanged
19. **int** diff = activity.mCurrentConfig.diff(config);
20. **if** (diff != 0) {
21. // If this activity doesn't handle any of the config changes
22. // then don't bother calling onConfigurationChanged as we're
23. // going to destroy it.
24. **if** ((~activity.mActivityInfo.getRealConfigChanged() & diff) == 0) {
25. shouldChangeConfig = **true**;
26. }
27. }
28. }
30. **if** (DEBUG\_CONFIGURATION) Slog.v(TAG, "Config callback " + cb
31. + ": shouldChangeConfig=" + shouldChangeConfig);
32. **if** (shouldChangeConfig) {
33. cb.onConfigurationChanged(config);
35. **if** (activity != **null**) {
36. **if** (!activity.mCalled) {
37. **throw** **new** SuperNotCalledException(
38. "Activity " + activity.getLocalClassName() +
39. " did not call through to super.onConfigurationChanged()");
40. }
41. activity.mConfigChangeFlags = 0;
42. activity.mCurrentConfig = **new** Configuration(config);
43. }
44. }
45. }

该函数判断configuration是否改变, 如果改变那么shouldChangeConfig为true. 然后调用activity的onConfigurationChange(config);

**[java]** [view plaincopy](http://blog.csdn.net/wqhjfree/article/details/8244520)

1. /\*\*
2. \* Called by the system when the device configuration changes while your
3. \* activity is running.  Note that this will <em>only</em> be called if
4. \* you have selected configurations you would like to handle with the
5. \* {@link android.R.attr#configChanges} attribute in your manifest.  If
6. \* any configuration change occurs that is not selected to be reported
7. \* by that attribute, then instead of reporting it the system will stop
8. \* and restart the activity (to have it launched with the new
9. \* configuration).
10. \*
11. \* <p>At the time that this function has been called, your Resources
12. \* object will have been updated to return resource values matching the
13. \* new configuration.
14. \*
15. \* @param newConfig The new device configuration.
16. \*/
17. **public** **void** onConfigurationChanged(Configuration newConfig) {
18. mCalled = **true**;
20. mFragments.dispatchConfigurationChanged(newConfig);
22. **if** (mWindow != **null**) {
23. // Pass the configuration changed event to the window
24. mWindow.onConfigurationChanged(newConfig);
25. }
27. **if** (mActionBar != **null**) {
28. // Do this last; the action bar will need to access
29. // view changes from above.
30. mActionBar.onConfigurationChanged(newConfig);
31. }
32. }

查看注释, 大概意思是:  如果你的activity运行 , 设备信息有改变(即configuration改变)时由系统调用. 如果你在manifest.xml中配置了configChnages属性则表示有你自己来处理configuration change. 否则就重启当前这个activity.  而重启之前, 旧的resources已经被清空, 那么就会装载新的资源, 整个过程就完成了语言切换后 , 能够让所有app使用新的语言. 语言切换流程大概分为三步:

第一步:  判断configuration的local是否已经改变, 如果改变则将local更新到当前的configuration

第二步: 清空旧的资源.

第三步: 重启所有所有进程并加装新资源.

由于个人知识水平有限, 有些地方不免有些纰漏, 希望大牛多多指点.  也希望有共同兴趣爱好的人进行技术交流.