

leagmain


[博客园](#) [首页](#) [新随笔](#) [联系](#) [管理](#) [订阅](#) [XML](#)

随笔- 9 文章- 0 评论- 0

Android源码解析--Quick Search in Settings

欢迎大家加入群里交流：429664282

基于：android-6.0.1_r17 f4b8ad6

Android Settings中存在一个SearchIndexablesProvider，它提供了可供快速检索的设置项。通过它，我们可以决定哪些系统设置可以被快速检索，那些可以不被检索。

SearchIndexablesProvider简介

SearchIndexablesProvider是Android标准API，在SDK中可以找到，路径是"android.provider.SearchIndexablesProvider"。它是一个虚基类，提供了一些抽象方法和常用方法。

抽象方法如下：

```
Cursor queryXmlResources(String[])
```

```
Cursor queryRawData(String[])
```

```
Cursor queryNonIndexableKeys(String[])
```

昵称：[leagmain](#)园龄：[5年9个月](#)粉丝：[0](#)关注：[13](#)[+加关注](#)

<	2019年2月						>
日	一	二	三	四	五	六	
27	28	29	30	31	1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	1	2	
3	4	5	6	7	8	9	

搜索

常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)

queryXmlResources方法返回一个Cursor，这个Cursor中包含了所有可以被索引的XmlResource，方法的参数是个String数组，用来表示传入查询的列。

queryRawData方法同queryRawData很类似，唯一区别是返回所有可以被索引的RawData。

queryNonIndexableKeys返回所有可以不被所有的NonIndexableKey。

检索数据来源：SearchIndexableResources

在Settings中，所有可供检索的数据资源，均被定义在SearchIndexableResources类中。

例如，Wifi设置页面的所有可供检索的数据，都通过WifiSettings.java提供。

```
1 static {  
2  
3 sResMap.put(WifiSettings.class.getName(),  
4  
5 new SearchIndexableResource(  
6  
7 Ranking.getRankForClassName(WifiSettings.class.getName()),  
8  
9 NO_DATA_RES_ID,  
10  
11 WifiSettings.class.getName(),  
12  
13 R.drawable.ic_settings_wireless));
```

特殊检索数据：Indexable

在Settings中，提供了一类特殊的检索数据，那就是实现了Indexable接口的类。

Indexable是定义在Settings内部的一种检索资源，通过代码的方式，返回可供检索的数据，包括SearchIndexableResource和SearchIndexRaw。

所有实现了Indexable的类，必须提供一个public static的SEARCH_INDEX_DATA_PROVIDER变量，用来供Settings利用反射获取到相应的检索数据。

[最新评论](#)[我的标签](#)

我的标签

[ADB\(1\)](#)[Rhythmbox\(1\)](#)

随笔档案

[2016年5月 \(1\)](#)[2014年3月 \(1\)](#)[2014年2月 \(1\)](#)[2013年12月 \(1\)](#)[2013年10月 \(2\)](#)[2013年9月 \(1\)](#)[2013年8月 \(1\)](#)[2013年5月 \(1\)](#)

阅读排行榜

[1. Android ListView实现仿iPhone实现左滑删除按钮\(2290\)](#)[2. Ubuntu 13.10 Rhythmbox 播放器不能播放MP3。安装插件\(1728\)](#)[3. Android源码解析--Quick Search in Settings\(1498\)](#)[4. 最新腾讯QQ \(2013 Beta \) 与Android ADB服务冲突解决办法\(663\)](#)[5. 我的 ubuntu 12.04.2修复Grub\(447\)](#)

推荐排行榜

[1. Android源码解析--Quick Search in Settings\(1\)](#)

```
/**
 * Interface for classes whose instances can provide data for indexing.
 *
 * Classes implementing the Indexable interface must have a static field c
 * <code>SEARCH_INDEX_DATA_PROVIDER</code>, which is an object implementir
 * {@link Indexable.SearchIndexProvider} interface.
 *
 * See {@link android.provider.SearchIndexableResource} and {@link SearchI
 */
public interface Indexable {
```

检索数据提供者：SettingsSearchIndexablesProvider

在Settings中，SettingsSearchIndexablesProvider实现了SearchIndexablesProvider，并且提供了可供检索的数据资源。这个数据资源以静态map的方式定义在SearchIndexableResources中，并且在queryXmlResources方法中返回给数据请求者。

```
@Override
public Cursor queryXmlResources(String[] projection) {
    MatrixCursor cursor = new MatrixCursor(INDEXABLES_XML_RES_COLUMNS);
    Collection<SearchIndexableResource> values = SearchIndexableResource
    for (SearchIndexableResource val : values) {
        Object[] ref = new Object[7];
        ref[COLUMN_INDEX_XML_RES_RANK] = val.rank;
        ref[COLUMN_INDEX_XML_RES_RESID] = val.xmlResId;
        ref[COLUMN_INDEX_XML_RES_CLASS_NAME] = val.className;
        ref[COLUMN_INDEX_XML_RES_ICON_RESID] = val.iconResId;
        ref[COLUMN_INDEX_XML_RES_INTENT_ACTION] = null; // intent action
        ref[COLUMN_INDEX_XML_RES_INTENT_TARGET_PACKAGE] = null; // inter
        ref[COLUMN_INDEX_XML_RES_INTENT_TARGET_CLASS] = null; // intent
        cursor.addRow(ref);
    }
    return cursor;
}
```

Settings并未提供任何的RawData和NonIndexableKey，所以它的queryRawData返回了一个空的Cursor。

```
@Override
public Cursor queryRawData(String[] projection) {
    MatrixCursor result = new MatrixCursor(INDEXABLES_RAW_COLUMNS);
    return result;
}

@Override
public Cursor queryNonIndexableKeys(String[] projection) {
    MatrixCursor cursor = new MatrixCursor(NON_INDEXABLES_KEYS_COLUMNS);
    return cursor;
}
```

更新检索数据库

在每次打开Settings导航页面的时候，Index.update()方法都会被调用，用来更新检索数据。

```
if (mIsShowingDashboard) {
    // Run the Index update only if we have some space
    if (!Utils.isLowStorage(this)) {
        Index.getInstance(getApplicationContext()).update();
    } else {
        Log.w(LOG_TAG, "Cannot update the Indexer as we are running low c
    }
}
```

Step1. 获取所有SearchIndexablesProvider.

```
public void update() {
    final Intent intent = new Intent(SearchIndexablesContract.PROVIDER
    List<ResolveInfo> list =
        mContext.getPackageManager().queryIntentContentProviders(i
```

Step2. 添加可供检索的数据到缓存：addIndexablesFromRemoteProvider

对于每一个SearchIndexablesProvider，都需要进行这个操作。

Step2.1. 从XmlResource中添加检索数据：addIndexablesForXmlResourceUri

所有定义在SearchIndexableResources中的检索数据，均会在这一步中被添加进来。接下来，让我们看看具体实现：

Step2.1.1 获取检索数据的Cursor

```
private void addIndexablesForXmlResourceUri(Context packageContext, String
    Uri uri, String[] projection, int baseRank) {

    final ContentResolver resolver = packageContext.getContentResolver();
    final Cursor cursor = resolver.query(uri, projection, null, null, null);

    if (cursor == null) {
        Log.w(LOG_TAG, "Cannot add index data for Uri: " + uri.toString());
        return;
    }
}
```

Step2.1.2 创建SearchIndexableResource，并加入到更新数据的缓存列表：

```

final int count = cursor.getCount();
if (count > 0) {
    while (cursor.moveToNext()) {
        final int providerRank = cursor.getInt(COLUMN_INDEX_XML_RES_RANK)
        final int rank = (providerRank > 0) ? baseRank + providerRank : b

        final int xmlResId = cursor.getInt(COLUMN_INDEX_XML_RES_RESID);

        final String className = cursor.getString(COLUMN_INDEX_XML_RES_CL
        final int iconResId = cursor.getInt(COLUMN_INDEX_XML_RES_ICON_RES

        final String action = cursor.getString(COLUMN_INDEX_XML_RES_INTEN
        final String targetPackage = cursor.getString(
            COLUMN_INDEX_XML_RES_INTENT_TARGET_PACKAGE);
        final String targetClass = cursor.getString(
            COLUMN_INDEX_XML_RES_INTENT_TARGET_CLASS);

        SearchIndexableResource sir = new SearchIndexableResource(package
        sir.rank = rank;
        sir.xmlResId = xmlResId;
        sir.className = className;
        sir.packageName = packageName;
        sir.iconResId = iconResId;
        sir.intentAction = action;
        sir.intentTargetPackage = targetPackage;
        sir.intentTargetClass = targetClass;

        addIndexableData(sir);
    }
}

```

```

public void addIndexableData(SearchIndexableData data) {
    synchronized (mDataToProcess) {
        mDataToProcess.dataToUpdate.add(data);
    }
}

```

Step2.2. 从RawData中添加检索数据 : addIndexablesForRawDataUri

Settings并未提供任何RawData，所以我们可以忽略这个步骤。

Step3. 添加不可供检索的数据到缓存

Settings并未提供任何NonIndexableKey，所以我们可以忽略这个步骤。

Step4. 更新缓存的检索数据

更新缓存的检索数据在updateInternal()方法中实现，下面我们来逐步分析下。

缓存数据结构

```
private static class UpdateData {  
    public List<SearchIndexableData> dataToUpdate;  
    public List<SearchIndexableData> dataToDelete;  
    public Map<String, List<String>> nonIndexableKeys;
```

到现在为止，Settings提供的检索数据均被添加到"dataToUpdate"列表中。其他两个未包含任何Settings提供的检索数据。

Step4.1 创建一个UpdateIndexTask的后台任务来更新检索数据库

```
private void updateInternal() {  
    synchronized (mDataToProcess) {  
        final UpdateIndexTask task = new UpdateIndexTask();  
        UpdateData copy = mDataToProcess.copy();  
        task.execute(copy);  
        mDataToProcess.clear();  
    }  
}
```

我们下面只关心dataToUpdate的相关操作。

Step4.2 更新dataToUpdate缓存 : processDataToUpdate

在这个方法中，我们对每一个SearchIndexableData都使用indexOneSearchIndexableData方法进行更新。


```

final int count = dataToUpdate.size();
for (int n = 0; n < count; n++) {
    final SearchIndexableData data = dataToUpdate.get(n);
    try {
        indexOneSearchIndexableData(database, localeStr, data, nonIndexableKeys);
    } catch (Exception e) {
        Log.e(LOG_TAG,
            "Cannot index: " + data.className + " for locale: " + localeStr);
    }
}
}

```

Step4.3 对检索数据进行索引 : `indexOneSearchIndexableData`

这里的检索数据可以包括RawData和XmlResource，又或者Settings返回的SearchIndexableResource。

这里为什么是SearchIndexableResource？因为SettingsSearchIndexablesProvider把所有可以检索的数据都已SearchIndexableResource插入到Cursor中，并且返回给数据查询者，这里也就是Settings本身。

```

private void indexOneSearchIndexableData(SQLiteDatabase database, String localeStr,
    SearchIndexableData data, Map<String, List<String>> nonIndexableKeys) {
    if (data instanceof SearchIndexableResource) {
        indexOneResource(database, localeStr, (SearchIndexableResource) data);
    } else if (data instanceof SearchIndexableRaw) {
        indexOneRaw(database, localeStr, (SearchIndexableRaw) data);
    }
}

```

Step4.4 对每一条检索数据资源进行数据检索 : `indexOneResource`

前面我们提到过，Settings里面有一种特殊的检索数据资源：Indexable。

在这里，如果我们发现这条检索数据资源的xmlResId是NO_DATA_RES_ID，也就是说，当前这个检索数据资源是一条特殊的检索数据资源。我们就需要利用反射机制得到它的public static的SEARCH_INDEX_DATA_PROVIDER。

Step4.4.1. 处理普通的检索数据资源 : `xmlResId != NO_DATA_RES_ID`


```
if (sir.xmlResId > SearchIndexableResources.NO_DATA_RES_ID) {  
    List<String> resNonIndexableKeys = nonIndexableKeysFromResource.get(sir.xmlResId);  
    if (resNonIndexableKeys != null && resNonIndexableKeys.size() > 0) {  
        nonIndexableKeys.addAll(resNonIndexableKeys);  
    }  
  
    indexFromResource(sir.context, database, localeStr,  
        sir.xmlResId, sir.className, sir.iconResId, sir.rank,  
        sir.intentAction, sir.intentTargetPackage, sir.intentTargetClass, sir.intentTargetUri,  
        nonIndexableKeys);  
}
```

Step4.4.1.1. 从XML中解析检索数据 : indexFromResource

在这里就到了整个实现流程的最末端了，解析整个PreferenceScreen的XML文件，并且把每一个符合条件的项加入到检索数据库。

Step4.4.2. 处理特殊的检索数据资源 : Indexable

```

    } else {
        if (TextUtils.isEmpty(sir.className)) {
            Log.w(LOG_TAG, "Cannot index an empty Search Provider name!");
            return;
        }

        final Class<?> clazz = getIndexableClass(sir.className);
        if (clazz == null) {
            Log.d(LOG_TAG, "SearchIndexableResource '" + sir.className +
                "' should implement the " + Indexable.class.getName() + "
            return;
        }

        // Will be non null only for a Local provider implementing a
        // SEARCH_INDEX_DATA_PROVIDER field
        final Indexable.SearchIndexProvider provider = getSearchIndexProvider(sir);
        if (provider != null) {
            List<String> providerNonIndexableKeys = provider.getNonIndexableKeys();
            if (providerNonIndexableKeys != null && providerNonIndexableKeys.size() > 0) {
                nonIndexableKeys.addAll(providerNonIndexableKeys);
            }

            indexFromProvider(mContext, database, localeStr, provider, sir.className,
                sir.iconResId, sir.rank, sir.enabled, nonIndexableKeys);
        }
    }
}

```

Step4.4.2.1. 从特殊的检索数据源获取检索数据 : indexFromProvider

当拿到SEARCH_INDEX_DATA_PROVIDER变量后，我们得到它的两个方法返回值，根据返回值的具体内容，我们将其更新到数据库。

[好文要顶](#)
[关注我](#)
[收藏该文](#)

[leagmain](#)
[关注 - 13](#)
[粉丝 - 0](#)
[+加关注](#)

1

[推荐](#)


0

[反对](#)

« 上一篇 : [Android 解决调用系统相册打不开图片 DecodeServices报解码错误](#)

posted @ 2016-05-17 14:14 [leagmain](#) 阅读(1497) 评论(0) [编辑](#) [收藏](#)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】超50万C++/C#源码: 大型实时仿真HMI组态CAD\GIS图形源码！

【推荐】专业便捷的企业级代码托管服务 - Gitee 码云

相关博文：

- [android Settings 解析](#)
- [Android7.0 添加快速设定Quick Settings Tile](#)
- [Android fragment源码全解析](#)
- [android Settings入门](#)
- [使用Powershell更新Search Settings](#)

最新新闻：

- [明星微博转发破1亿次 微博官方回应：调整上限为100万](#)
 - [暴风集团发布澄清公告：冯鑫依然是公司的控股股东和实际控制人](#)
 - [阿里云上VMware云解决方案开始邀请测试](#)
 - [高瓴资本张磊：警惕各方面都No.1的人](#)
 - [互联网裁员阴影下，如何避免成为那只“火鸡？”](#)
- » [更多新闻...](#)

Copyright ©2019 leagmain