前言

本文基于Android N源码进行分析，其实Settings的数据库不仅能够搜索Settings下的项，它还可以搜索其它应用的一些信息，例如短信和通讯录，理论上都是可以实现的。。

涉及的类

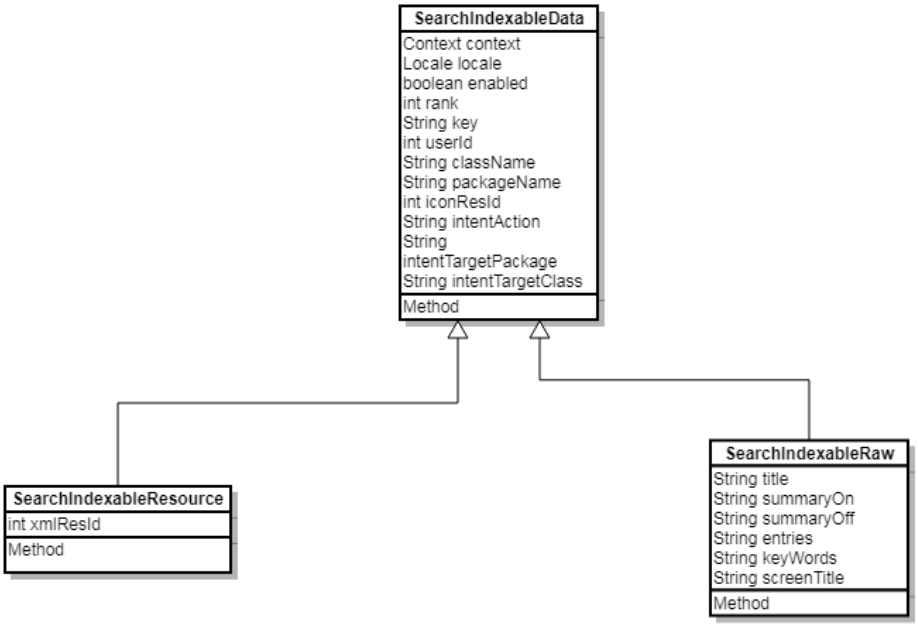
1. com.android.settings.search.Index.java
2. com.android.settings.DisplaySettings.java
3. com.android.settings.SettingsSearchIndexablesProvider.java
4. com.android.settings.search.SearchIndexableRaw.java
5. com.android.settings.search.SearchIndexableResources.java
6. android.provider.SearchIndexableResource.java
7. android.provider.SearchIndexableData.java
8. android.provider.SearchIndexablesProvider.java

搜索数据库

Settings里面的搜索框架是基于数据库来实现的，数据库的名称叫**search_index.db** ,Android M是在/data/data/com.android.settings/databases/下，Android N是在/data/user_de/0/com.android.settings/databases/下，如果不清楚的可按照下面的步骤进行查询

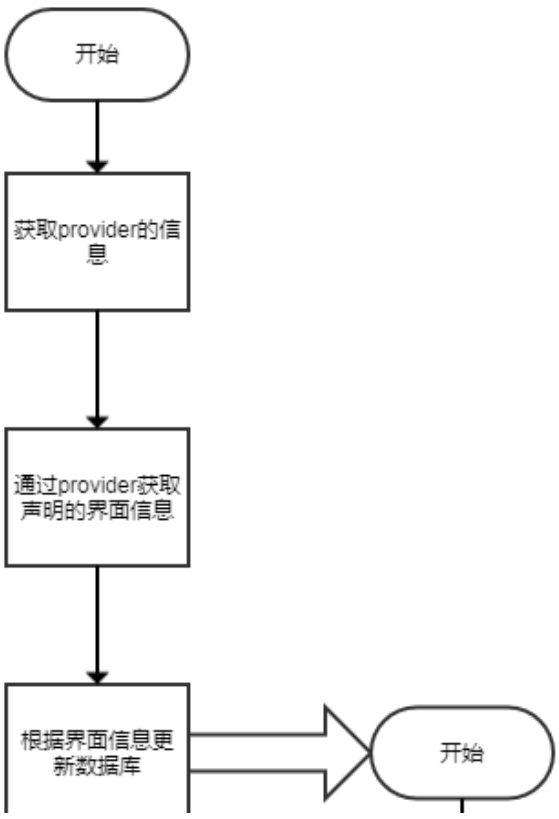
> 1. adb shell
> 2. find . "search_index.db"

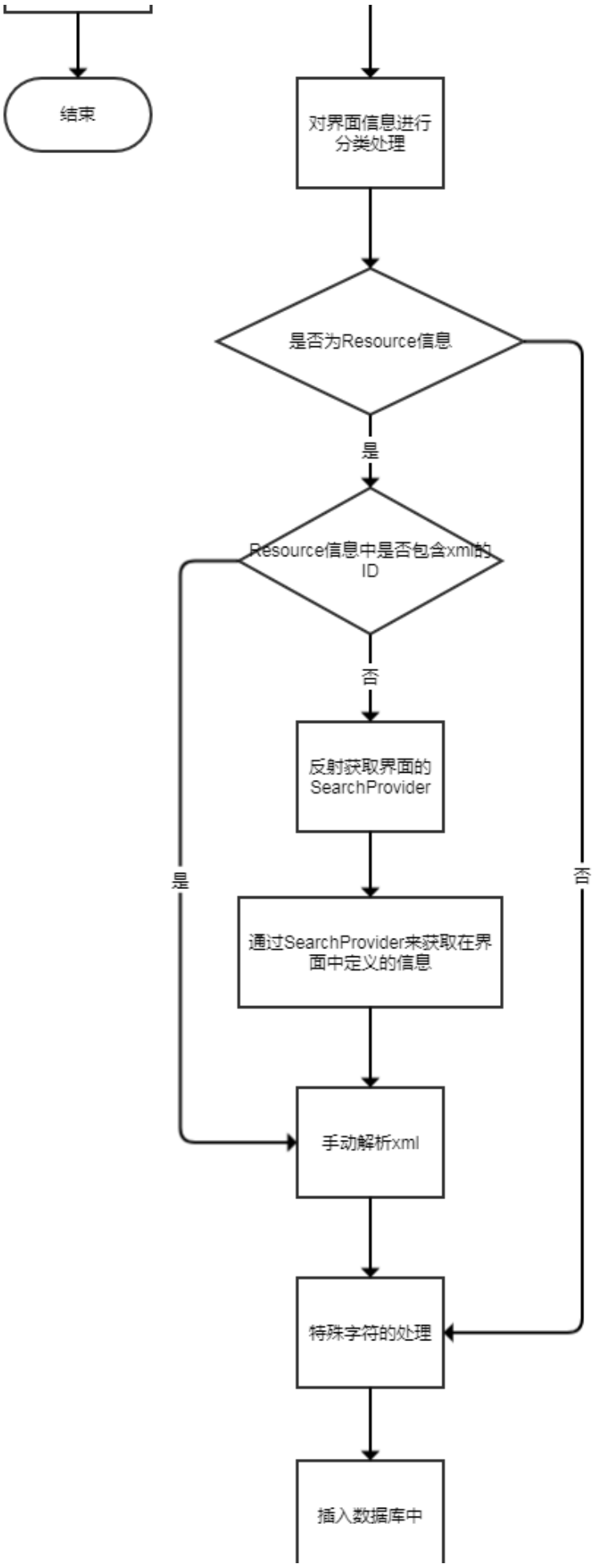
类图

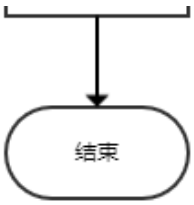

SearchIndexableData
Context context
Locale locale
boolean enabled
int rank
String key
int userId
String className
String packageName
int iconResId
String intentAction
String
intentTargetPackage
String intentTargetClass
Method
SearchIndexableResource
int xmlResId
Method
SearchIndexableRaw
String title
String summaryOn
String summaryOff
String entries
String keyWords
String screenTitle
Method

插入数据

流程图


开始
获取provider的信息
通过provider获取声明的界面信息
根据界面信息更新数据库
开始


结束
对界面信息进行分类处理
是否为Resource信息
是
Resource信息中是否包含xml的ID
否
反射获取界面的SearchProvider
通过SearchProvider来获取在界面中定义的信息
是
否
手动解析xml
特殊字符的处理
插入数据库中



1. 声明数据

声明数据的意思就是表明哪些界面的数据是可以被搜索的，例如在Settings中，SearchIndexableResources就声明了许多可以搜索的界面，Wifi、显示、应用管理等。。

```
//SearchIndexableResources.java
//声明显示是可以搜索的
sResMap.put(DisplaySettings.class.getName(),
            new SearchIndexableResource(
              Ranking.getRankForClassName(DisplaySettings.class.getName()),
              NO_DATA_RES_ID,
              DisplaySettings.class.getName(),
              R.drawable.ic_settings_display));
```

2. 创建Provider

Settings的数据库的来源是ContentProvider，所以创建一个应用内部的ContentProvider，继承自SearchIndexablesProvider，来进行提供数据，例如下面是Settings提供数据的

```java
//SettingsSearchIndexablesProvider.java
public class SettingsSearchIndexablesProvider extends SearchIndexablesProvider {
    private static final String TAG = "SettingsSearchIndexablesProvider";

    @Override
    public boolean onCreate() {
        return true;
    }

    //这个并不是ContentProvider的方法
    @Override
    public Cursor queryXmlResources(String[] projection) {
        //这里是创建了一个虚拟表，是用来提供数据的，因为我们的数据并不是存储在数据库中
        MatrixCursor cursor = new MatrixCursor(INDEXABLES_XML_RES_COLUMNS);
        //从SearchIndexableResources中获取类和包名等数据，即在SearchIndexableResources声明的数
据
        Collection<SearchIndexableResource> values = SearchIndexableResources.values();
        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; // intent target
package
            ref[COLUMN_INDEX_XML_RES_INTENT_TARGET_CLASS] = null; // intent target class
            cursor.addRow(ref);
        }
        return cursor;
    }

    @Override
    public Cursor queryRawData(String[] projection) {
        //这里没有RawData，都是空的
        MatrixCursor result = new MatrixCursor(INDEXABLES_RAW_COLUMNS);
        return result;
    }

    @Override
    public Cursor queryNonIndexableKeys(String[] projection) {
        //也没有不进入搜索的index
        MatrixCursor cursor = new MatrixCursor(NON_INDEXABLES_KEYS_COLUMNS);
        return cursor;
    }
}

//SearchIndexablesProvider.java
@SystemApi
public abstract class SearchIndexablesProvider extends ContentProvider {
    private static final String TAG = "IndexablesProvider";
```

```
    private String mAuthority;
    private UriMatcher mMatcher;

    private static final int MATCH_RES_CODE = 1;
    private static final int MATCH_RAW_CODE = 2;
    private static final int MATCH_NON_INDEXABLE_KEYS_CODE = 3;
    @Override
    public void attachInfo(Context context, ProviderInfo info) {
        mAuthority = info.authority;

        mMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mMatcher.addURI(mAuthority, SearchIndexablesContract.INDEXABLES_XML_RES_PATH,
                MATCH_RES_CODE);
        mMatcher.addURI(mAuthority, SearchIndexablesContract.INDEXABLES_RAW_PATH,
                MATCH_RAW_CODE);
        mMatcher.addURI(mAuthority, SearchIndexablesContract.NON_INDEXABLES_KEYS_PATH,
                MATCH_NON_INDEXABLE_KEYS_CODE);

        // Sanity check our setup
        if (!info.exported) {
            throw new SecurityException("Provider must be exported");
        }
        if (!info.grantUriPermissions) {
            throw new SecurityException("Provider must grantUriPermissions");
        }
        if
(!android.Manifest.permission.READ_SEARCH_INDEXABLES.equals(info.readPermission)) {
            throw new SecurityException("Provider must be protected by
READ_SEARCH_INDEXABLES");
        }

        super.attachInfo(context, info);
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs,
                        String sortOrder) {
        switch (mMatcher.match(uri)) {
            case MATCH_RES_CODE:
                return queryXmlResources(null);
            case MATCH_RAW_CODE:
                return queryRawData(null);
            case MATCH_NON_INDEXABLE_KEYS_CODE:
                return queryNonIndexableKeys(null);
            default:
                throw new UnsupportedOperationException("Unknown Uri " + uri);
        }
    }
    public abstract Cursor queryXmlResources(String[] projection);
    public abstract Cursor queryRawData(String[] projection);
    public abstract Cursor queryNonIndexableKeys(String[] projection);
```

```
    @Override
    public String getType(Uri uri) {
        switch (mMatcher.match(uri)) {
            case MATCH_RES_CODE:
                return SearchIndexablesContract.XmlResource.MIME_TYPE;
            case MATCH_RAW_CODE:
                return SearchIndexablesContract.RawData.MIME_TYPE;
            case MATCH_NON_INDEXABLE_KEYS_CODE:
                return SearchIndexablesContract.NonIndexableKey.MIME_TYPE;
            default:
                throw new IllegalArgumentException("Unknown URI " + uri);
        }
    }

    /**
     * Implementation is provided by the parent class. Throws by default, and cannot be
overriden.
     */
    @Override
    public final Uri insert(Uri uri, ContentValues values) {
        throw new UnsupportedOperationException("Insert not supported");
    }

    /**
     * Implementation is provided by the parent class. Throws by default, and cannot be
overriden.
     */
    @Override
    public final int delete(Uri uri, String selection, String[] selectionArgs) {
        throw new UnsupportedOperationException("Delete not supported");
    }

    /**
     * Implementation is provided by the parent class. Throws by default, and cannot be
overriden.
     */
    @Override
    public final int update(
            Uri uri, ContentValues values, String selection, String[] selectionArgs) {
        throw new UnsupportedOperationException("Update not supported");
    }
}
```

3. 获取真实的数据

之前我们获取的并不是真实的数据，真实的数据则是动态获取的，首先，提供数据的是各个具体的页面

```java
//DisplaySettings.java
//书写的时候SEARCH_INDEX_DATA_PROVIDER是固定的，不能进行修改，否则不生效
//3.1
public static final Indexable.SearchIndexProvider SEARCH_INDEX_DATA_PROVIDER =
            new BaseSearchIndexProvider() {
                //从xml中获取数据，到时会解析xml
                @Override
                public List<SearchIndexableResource> getXmlResourcesToIndex(Context
context,
                        boolean enabled) {
                    ArrayList<SearchIndexableResource> result =
                            new ArrayList<SearchIndexableResource>();

                    SearchIndexableResource sir = new SearchIndexableResource(context);
                    sir.xmlResId = R.xml.display_settings;
                    result.add(sir);

                    return result;
                }
                //获取单个Preference的数据
                @Override
                public List<SearchIndexableRaw> getRawDataToIndex(Context context,
boolean enabled) {
                    return null;
                }

                //获取不加入搜索的数据
                @Override
                public List<String> getNonIndexableKeys(Context context) {
                    ArrayList<String> result = new ArrayList<String>();
                    if (!context.getResources().getBoolean(
                            com.android.internal.R.bool.config_dreamsSupported)) {
                        result.add(KEY_SCREEN_SAVER);
                    }
                    if (!isAutomaticBrightnessAvailable(context.getResources())) {
                        result.add(KEY_AUTO_BRIGHTNESS);
                    }
                    if (!NightDisplayController.isAvailable(context)) {
                        result.add(KEY_NIGHT_DISPLAY);
                    }
                    if (!isLiftToWakeAvailable(context)) {
                        result.add(KEY_LIFT_TO_WAKE);
                    }
                    if (!isDozeAvailable(context)) {
                        result.add(KEY_DOZE);
                    }
                    if (!RotationPolicy.isRotationLockToggleVisible(context)) {
                        result.add(KEY_AUTO_ROTATE);
                    }
                    if (!isTapToWakeAvailable(context.getResources())) {
                        result.add(KEY_TAP_TO_WAKE);
                    }

                    if (!isCameraGestureAvailable(context.getResources())) {
```

```
                    result.add(KEY_CAMERA_GESTURE);
                }
                if (!isVrDisplayModeAvailable(context)) {
                    result.add(KEY_VR_DISPLAY_PREF);
                }
                return result;
            }
        };
```

然后是动态的

```
//Index.java
//3.2
public class Index {
  //通过反射获取SearchProvider
  //具体使用见
  private static final String FIELD_NAME_SEARCH_INDEX_DATA_PROVIDER =
            "SEARCH_INDEX_DATA_PROVIDER";

  //更新数据库
  //3.2.1
  public void update() {
        AsyncTask.execute(new Runnable() {
            @Override
            public void run() {
                final Intent intent = new
Intent(SearchIndexablesContract.PROVIDER_INTERFACE);
                List<ResolveInfo> list =
mContext.getPackageManager().queryIntentContentProviders(intent, 0);

                final int size = list.size();
                for (int n = 0; n < size; n++) {
                    final ResolveInfo info = list.get(n);
                    if (!isWellKnownProvider(info)) {
                        continue;
                    }
                    final String authority = info.providerInfo.authority;
                    final String packageName = info.providerInfo.packageName;
                     //获取的是数据的声明，最主要的还是获取声明的类名，为下面的反射做准备
                    //见3.2.2
                    addIndexablesFromRemoteProvider(packageName, authority);
                    //类似不再做分析
                    addNonIndexablesKeysFromRemoteProvider(packageName, authority);
                }

                mDataToProcess.fullIndex = true;
                //更新Settings
                //见3.2.4
                updateInternal();
            }
        });
    }
  //3.2.2
  private boolean addIndexablesFromRemoteProvider(String packageName, String authority)
{
        try {
            final int baseRank = Ranking.getBaseRankForAuthority(authority);

            final Context context = mBaseAuthority.equals(authority) ?
                    mContext : mContext.createPackageContext(packageName, 0);
            //查询provider的声明
            final Uri uriForResources = buildUriForXmlResources(authority);
            //见3.2.3

            addIndexablesForXmlResourceUri(context, packageName, uriForResources,
```

```
                    SearchIndexablesContract.INDEXABLES_XML_RES_COLUMNS, baseRank);
            //查询provider
            final Uri uriForRawData = buildUriForRawData(authority);
            //几乎一样，不做分析
            addIndexablesForRawDataUri(context, packageName, uriForRawData,
                    SearchIndexablesContract.INDEXABLES_RAW_COLUMNS, baseRank);
            return true;
        } catch (PackageManager.NameNotFoundException e) {
            Log.w(LOG_TAG, "Could not create context for " + packageName + ": "
                    + Log.getStackTraceString(e));
            return false;
        }
    }
  //3.2.3
  private void addIndexablesForXmlResourceUri(Context packageContext, String
packageName,
            Uri uri, String[] projection, int baseRank) {

        final ContentResolver resolver = packageContext.getContentResolver();
        //最终会调用queryXmlResources方法
        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;
        }

        try {
            final int count = cursor.getCount();
            Log.d("JonnyPeng", "addIndexablesForXmlResourceUri() : resource count: " +
count);
            if (count > 0) {
                while (cursor.moveToNext()) {
                    final int providerRank = cursor.getInt(COLUMN_INDEX_XML_RES_RANK);
                    final int rank = (providerRank > 0) ? baseRank + providerRank :
baseRank;

                    final int xmlResId = cursor.getInt(COLUMN_INDEX_XML_RES_RESID);

                    final String className =
cursor.getString(COLUMN_INDEX_XML_RES_CLASS_NAME);
                    final int iconResId =
cursor.getInt(COLUMN_INDEX_XML_RES_ICON_RESID);

                    final String action =
cursor.getString(COLUMN_INDEX_XML_RES_INTENT_ACTION);
                    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(packageContext);
                    sir.rank = rank;
                    sir.xmlResId = xmlResId;
                    sir.className = className;
                    sir.packageName = packageName;
                    sir.iconResId = iconResId;
                    sir.intentAction = action;
                    sir.intentTargetPackage = targetPackage;
                    sir.intentTargetClass = targetClass;
                     //添加sir进updateData中
                    addIndexableData(sir);
                }
            }
        } finally {
            cursor.close();
        }
    }
  public void addIndexableData(SearchIndexableData data) {
        synchronized (mDataToProcess) {
            mDataToProcess.dataToUpdate.add(data);
        }
  }
  //3.2.4
  private void updateInternal() {
        synchronized (mDataToProcess) {
            //更新数据的后台线程
            //见3.2.5
            final UpdateIndexTask task = new UpdateIndexTask();
            UpdateData copy = mDataToProcess.copy();
            task.execute(copy);
            mDataToProcess.clear();
        }
    }
  //3.2.5
  private class UpdateIndexTask extends AsyncTask<UpdateData, Integer, Void> {
    @Override
        protected Void doInBackground(UpdateData... params) {
            try {
                final List<SearchIndexableData> dataToUpdate = params[0].dataToUpdate;
                final List<SearchIndexableData> dataToDelete = params[0].dataToDelete;
                final Map<String, List<String>> nonIndexableKeys =
params[0].nonIndexableKeys;

                final boolean forceUpdate = params[0].forceUpdate;
                final boolean fullIndex = params[0].fullIndex;

                final SQLiteDatabase database = getWritableDatabase();
                if (database == null) {
                    Log.e(LOG_TAG, "Cannot update Index as I cannot get a writable
database");
                    return null;
                }

                final String localeStr = Locale.getDefault().toString();
```

```
                try {
                    database.beginTransaction();
                    if (dataToDelete.size() > 0) {
                        processDataToDelete(database, localeStr, dataToDelete);
                    }
                    //如果没有声明数据，则不会更新数据
                    if (dataToUpdate.size() > 0) {
                        //更新数据
                        //见3.2.6
                        processDataToUpdate(database, localeStr, dataToUpdate,
nonIndexableKeys,
                                forceUpdate);
                    }
                    database.setTransactionSuccessful();
                } finally {
                    database.endTransaction();
                }
                if (fullIndex) {
                    IndexDatabaseHelper.setLocaleIndexed(mContext, localeStr);
                }
            } catch (SQLiteFullException e) {
                Log.e(LOG_TAG, "Unable to index search, out of space", e);
            }

            return null;
        }
        //处理数据更新
        //3.2.6
        private boolean processDataToUpdate(SQLiteDatabase database, String localeStr,
                List<SearchIndexableData> dataToUpdate, Map<String, List<String>>
nonIndexableKeys,
                boolean forceUpdate) {
            ...
            final int count = dataToUpdate.size();
            for (int n = 0; n < count; n++) {
                final SearchIndexableData data = dataToUpdate.get(n);
                try {
                    //见3.2.7
                    indexOneSearchIndexableData(database, localeStr, data,
nonIndexableKeys);
                } catch (Exception e) {
                    Log.e(LOG_TAG, "Cannot index: " + (data != null ? data.className :
data)
                                    + " for locale: " + localeStr, e);
                }
            }
            ...
            return result;
        }



  }
```

```
  //行插入和资源插入，行插入的情况还没遇到过，
  //3.2.7
  private void indexOneSearchIndexableData(SQLiteDatabase database, String localeStr,
            SearchIndexableData data, Map<String, List<String>> nonIndexableKeys) {
        if (data instanceof SearchIndexableResource) {
            Log.d("JonnyPeng", "indexOneSearchIndexableData() :
SearchIndexableResource");
            //见3.2.8
            indexOneResource(database, localeStr, (SearchIndexableResource) data,
nonIndexableKeys);
        } else if (data instanceof SearchIndexableRaw) {
            Log.d("JonnyPeng", "indexOneSearchIndexableData() : SearchIndexableRaw");
            //直接会插入数据库
            indexOneRaw(database, localeStr, (SearchIndexableRaw) data);
        }
    }
  //获取一个资源的索引
  //3.2.8
  private void indexOneResource(SQLiteDatabase database, String localeStr,
            SearchIndexableResource sir, Map<String, List<String>>
nonIndexableKeysFromResource) {

        if (sir == null) {
            Log.e(LOG_TAG, "Cannot index a null resource!");
            return;
        }

        final List<String> nonIndexableKeys = new ArrayList<String>();
        //如果在SearchIndexResource指定了xmlResId，则直接使用这个xml，在界面里面的provider就不会
生效
        if (sir.xmlResId > SearchIndexableResources.NO_DATA_RES_ID) {
            List<String> resNonIndxableKeys =
nonIndexableKeysFromResource.get(sir.packageName);
            if (resNonIndxableKeys != null && resNonIndxableKeys.size() > 0) {
                nonIndexableKeys.addAll(resNonIndxableKeys);
            }
            //手动解析xml。。并且调用updateOneRawFilterData()
            indexFromResource(sir.context, database, localeStr,
                    sir.xmlResId, sir.className, sir.iconResId, sir.rank,
                    sir.intentAction, sir.intentTargetPackage, sir.intentTargetClass,
                    nonIndexableKeys);
        } else {
            if (TextUtils.isEmpty(sir.className)) {
                Log.w(LOG_TAG, "Cannot index an empty Search Provider name!");
                return;
            }
            //反射获取clazz
            final Class<?> clazz = getIndexableClass(sir.className);
            if (clazz == null) {
                //之前插入不完整的情况，我非常怀疑是这个地方出了问题。。
                Log.d(LOG_TAG, "SearchIndexableResource '" + sir.className +


                        "' should implement the " + Indexable.class.getName() + "
```

```
interface!");
                return;
            }

            // Will be non null only for a Local provider implementing a
            // SEARCH_INDEX_DATA_PROVIDER field
            //获取provider
            //见3.2.9
            final Indexable.SearchIndexProvider provider =
getSearchIndexProvider(clazz);
            if (provider != null) {
                List<String> providerNonIndexableKeys =
provider.getNonIndexableKeys(sir.context);
                if (providerNonIndexableKeys != null && providerNonIndexableKeys.size()
> 0) {
                    nonIndexableKeys.addAll(providerNonIndexableKeys);
                }
                 //从provider里面获取索引
                //见3.2.10
                indexFromProvider(mContext, database, localeStr, provider,
sir.className,
                        sir.iconResId, sir.rank, sir.enabled, nonIndexableKeys);
            }
        }
    }
  //获取对应界面的provider
  //3.2.9
  private Indexable.SearchIndexProvider getSearchIndexProvider(final Class<?> clazz) {
        try {
            final Field f = clazz.getField(FIELD_NAME_SEARCH_INDEX_DATA_PROVIDER);
            return (Indexable.SearchIndexProvider) f.get(null);
        } catch (NoSuchFieldException e) {
            Log.d(LOG_TAG, "Cannot find field '" + FIELD_NAME_SEARCH_INDEX_DATA_PROVIDER
+ "'");
        } catch (SecurityException se) {
            Log.d(LOG_TAG,
                    "Security exception for field '" +
FIELD_NAME_SEARCH_INDEX_DATA_PROVIDER + "'");
        } catch (IllegalAccessException e) {
            Log.d(LOG_TAG,
                    "Illegal access to field '" + FIELD_NAME_SEARCH_INDEX_DATA_PROVIDER
+ "'");
        } catch (IllegalArgumentException e) {
            Log.d(LOG_TAG,
                    "Illegal argument when accessing field '" +
                            FIELD_NAME_SEARCH_INDEX_DATA_PROVIDER + "'");
        }
        return null;
    }
    //3.2.10
    private void indexFromProvider(Context context, SQLiteDatabase database, String
localeStr,

            Indexable.SearchIndexProvider provider, String className, int iconResId, int
```

```
rank,
            boolean enabled, List<String> nonIndexableKeys) {

        if (provider == null) {
            Log.w(LOG_TAG, "Cannot find provider: " + className);
            return;
        }

        final List<SearchIndexableRaw> rawList = provider.getRawDataToIndex(context,
enabled);
        //获取行的数据，并插入
        if (rawList != null) {
            final int rawSize = rawList.size();
            for (int i = 0; i < rawSize; i++) {
                SearchIndexableRaw raw = rawList.get(i);

                // Should be the same locale as the one we are processing
                if (!raw.locale.toString().equalsIgnoreCase(localeStr)) {
                    continue;
                }

                if (nonIndexableKeys.contains(raw.key)) {
                    continue;
                }
                 //插入数据到数据库
                updateOneRowWithFilteredData(database, localeStr,
                        raw.title,
                        raw.summaryOn,
                        raw.summaryOff,
                        raw.entries,
                        className,
                        raw.screenTitle,
                        iconResId,
                        rank,
                        raw.keywords,
                        raw.intentAction,
                        raw.intentTargetPackage,
                        raw.intentTargetClass,
                        raw.enabled,
                        raw.key,
                        raw.userId);
            }
        }

        //获取xml的id
        final List<SearchIndexableResource> resList =
                provider.getXmlResourcesToIndex(context, enabled);
        if (resList != null) {
            final int resSize = resList.size();
            for (int i = 0; i < resSize; i++) {
                SearchIndexableResource item = resList.get(i);


                // Should be the same locale as the one we are processing
```

```
                if (!item.locale.toString().equalsIgnoreCase(localeStr)) {
                    continue;
                }

                final int itemIconResId = (item.iconResId == 0) ? iconResId :
item.iconResId;
                final int itemRank = (item.rank == 0) ? rank : item.rank;
                String itemClassName = (TextUtils.isEmpty(item.className))
                        ? className : item.className;
                //进行解析xml,并将数据进行插入数据库
                indexFromResource(context, database, localeStr,
                        item.xmlResId, itemClassName, itemIconResId, itemRank,
                        item.intentAction, item.intentTargetPackage,
                        item.intentTargetClass, nonIndexableKeys);
            }
        }
    }


    //最终调用
    private void updateOneRow(SQLiteDatabase database, String locale, String
updatedTitle,
            String normalizedTitle, String updatedSummaryOn, String normalizedSummaryOn,
            String updatedSummaryOff, String normalizedSummaryOff, String entries,
String className,
            String screenTitle, int iconResId, int rank, String spaceDelimitedKeywords,
            String intentAction, String intentTargetPackage, String intentTargetClass,
            boolean enabled, String key, int userId) {

        if (TextUtils.isEmpty(updatedTitle)) {
            return;
        }

        // The DocID should contains more than the title string itself (you may have two
settings
        // with the same title). So we need to use a combination of the title and the
screenTitle.
        StringBuilder sb = new StringBuilder(updatedTitle);
        sb.append(screenTitle);
        int docId = sb.toString().hashCode();

        // TINN BEGIN
        // HCABN-745, kui.xie, 20161206
        // avoid putting the cellbroadcastreceiver to the search db
        if("com.android.cellbroadcastreceiver".equals(intentTargetPackage)
                &&
"com.android.cellbroadcastreceiver.CellBroadcastSettings".equals(intentTargetClass))  {
            return;
        }
        //TINNO END

        ContentValues values = new ContentValues();

        values.put(IndexColumns.DOCID, docId);
```

```
        values.put(IndexColumns.LOCALE, locale);
        values.put(IndexColumns.DATA_RANK, rank);
        values.put(IndexColumns.DATA_TITLE, updatedTitle);
        values.put(IndexColumns.DATA_TITLE_NORMALIZED, normalizedTitle);
        values.put(IndexColumns.DATA_SUMMARY_ON, updatedSummaryOn);
        values.put(IndexColumns.DATA_SUMMARY_ON_NORMALIZED, normalizedSummaryOn);
        values.put(IndexColumns.DATA_SUMMARY_OFF, updatedSummaryOff);
        values.put(IndexColumns.DATA_SUMMARY_OFF_NORMALIZED, normalizedSummaryOff);
        values.put(IndexColumns.DATA_ENTRIES, entries);
        values.put(IndexColumns.DATA_KEYWORDS, spaceDelimitedKeywords);
        values.put(IndexColumns.CLASS_NAME, className);
        values.put(IndexColumns.SCREEN_TITLE, screenTitle);
        values.put(IndexColumns.INTENT_ACTION, intentAction);
        values.put(IndexColumns.INTENT_TARGET_PACKAGE, intentTargetPackage);
        values.put(IndexColumns.INTENT_TARGET_CLASS, intentTargetClass);
        values.put(IndexColumns.ICON, iconResId);
        values.put(IndexColumns.ENABLED, enabled);
        values.put(IndexColumns.DATA_KEY_REF, key);
        values.put(IndexColumns.USER_ID, userId);
        //JonnyPeng, DATE: 20170426, Add some important log for insert datas
        android.util.Log.d("JonnyPeng", "----------------DIVIDER----------------");
        android.util.Log.d("JonnyPeng", "updatedTitle: " + updatedTitle);
        android.util.Log.d("JonnyPeng", "className: " + className);
        android.util.Log.d("JonnyPeng", "screenTitle: " + screenTitle);
        android.util.Log.d("JonnyPeng", "----------------DIVIDER----------------");
        //JonnyPeng, DATE: 20170426, Add some important log for insert datas

        database.replaceOrThrow(Tables.TABLE_PREFS_INDEX, null, values);
    }
}
```

搜索数据

SELECT data_rank, data_title, data_summary_on, data_summary_off, data_entries, data_keywords, class_name, screen_title, icon, intent_action, intent_target_package, intent_target_class, enabled, data_key_reference FROM prefs_index WHERE prefs_index MATCH 'data_title:w* OR data_title_normalized:w* OR data_keywords:w*' AND locale = 'zh_CN_#Hans' AND enabled = 1 ORDER BY data_rank

Search secondary query: SELECT data_rank, data_title, data_summary_on, data_summary_off, data_entries, data_keywords, class_name, screen_title, icon, intent_action, intent_target_package, intent_target_class, enabled, data_key_reference FROM prefs_index WHERE prefs_index MATCH 'data_summary_on:w* OR data_summary_on_normalized:w* OR data_summary_off:w* OR data_summary_off_normalized:w* OR data_entries:w' *AND locale = 'zh_CN_#Hans' AND enabled = 1 EXCEPT SELECT data_rank, data_title, data_summary_on, data_summary_off, data_entries, data_keywords, class_name, screen_title, icon, intent_action, intent_target_package, intent_target_class, enabled, data_key_reference FROM prefs_index WHERE prefs_index MATCH 'data_title:w* OR data_title_normalized:w* OR data_keywords:w*' AND locale = 'zh_CN_#Hans' AND enabled = 1 ORDER BY data_rank*

将两个结果进行合并