# 工程结构

Gradle 工程包括：

* setting.gradle:设置当前工程包含的module
* build.gradle:包含两部分配置
  + 依赖的gradle版本
  + maven 库的路径，默认库jcenter，其他maven库
* gradlew、gradlew.bat 分别为调用gradle的sh和bat脚本
* 业务或者lib module
* local.properties,\*.iml, ./gradle/,./idea/ ,./build/由gradle自动生成,提交时忽略

# module 结构

* \*.jks: Java签名文件
* proguard-rules.pro: 混淆规则
* lib : 第三方jar包
* src: 源码
  + main
    - jniLibs: jar包依赖jnilib
    - java : java 源码
    - res: 资源文件
  + androidTest: 单元测试文件夹
* \*.jks: Java签名文件
* \*.iml, ./build/ 由gradle生成,提交时忽略
* build.gradle:
  + 编译配置
  + 使用的android sdk, buiildtools版本,当前以及最低的Android sdk版本,app/module版本号
  + 签名配置
  + 混淆配置
  + lint代码检查
  + 编译规则(aptOption)以及打包(packagingOption)
  + 依赖配置

# appbase module 说明

appbase包含三部分

1. 系统鉴权框架所必须的服务和鉴权器
2. UI框架
3. 工具类

## 依赖关系

* com.android.support:support-v4:21.0.3+
* com.android.support:appcompat-v7:21.0.3+
* com.android.support:support-v13:21.0.3+
* com.google.code.gson:gson:2.3.1+
* com.turbomanage.basic-http-client:http-client-android:0.89
* com.google.api-client:google-api-client:1.20.0
* com.belerweb:pinyin4j:2.5.0
* volley

## 源码说明

* \*.accounts.AccountAuthenticator: 鉴权器,在业务module的AndroidManifast中配置,被系统调用
* \*.accounts.AccountAuthenticatorService:鉴权服务,在业务module的AndroidManifast中配置,被系统调用
* \*.account.AccountUtils: 帐号相关工具类
* \*.app.BaseApp: 继承系统Application,在App启动时的初始化
* \*.base.model.BaseModel: model类的base,不推荐使用
* \*.base.ui.widget 一些可复用的UI组件
  + \*.CollectionView: 继承ListView,数据可分组
  + \*.CollectionViewCallback: callback,用于实现生成分组header和分组内容
  + \*.DrawShadowFrameLayout: 主界面侧滑菜单容器
  + \*.MultiSwipeRefreshLayout: 下拉刷新组件
  + \*.ObservableScroller: 滚动显示组件,可以 通过观察者监听滚动状态
  + \*.ScrimInsetsScrollView: 侧滑菜单控件, 定义菜单滑动时背景色变化的动作
* \*.net.HttpMethod: 这里只定义了GET,POST,不推荐使用
* \*.net.NetConnection:
* \*.util 工具类
  + \*.app.PrefUtils: App设置开关相关的工具类
  + \*.data: 集合类工具
    - \*.Lists: list 工具
    - \*.Maps: map工具
  + \*.database 数据库相关的工具
    - \*.JSONHandler: 业务module中io包中的类继承此类,实现process方法处理json,实现makeContentProviderOperations处理写数据库逻辑
    - \*.SelectionBuilder: ContentProvider中用来构造selection子句
    - \*.datatime 时间转换工具
    - \*.logcat.LogUtils 日志工具
    - \*.model.HashUtils: 生成hashcode
    - \*.network:.NetworkUtils: 检查网络连接是否可用
    - \*.observer.ThrottledContentObserver: 重写的ContentObserver的onChange方法,限制onChange被系统回调的频率
    - \*.string: 字符串处理工具
    - *.ui UI相关工具* .LUtils: Android 5.0 UI APIs Helper 类 *.TextViewUtils: TextView赋值类* .UIUtils: 其他UI相关的行为
    - \*.volley: 异步通信框架volley相关工具 -Config: appbase module 配置

## 资源说明

* color 使用xml生成UI组件配色
* drawable 使用xml生成ui组件
* drawable-\*dpi 图片,点九图资源文件
* layout: 布局文件
* values.colors: 定义颜色值常量,业务module可以复用这里的配置
* values.dimens: 定义UI尺寸
* values.integers: 定义数值型常量
* values.strings: 定义string常量
* values.style: 样式文件,业务module中的样式文件需要继承此文件中定义的样式
* value.attrs: 样式,布局文件中用到的自定义属性
* value.refs: 定义引用
* value.fonts: 布局,样式文件中用到的字体定义

# 在业务module 使用appbase定义的样式

代码参考 demoTheme model，过程如下：

1. 新建phone/table module作为业务 module,targetSdkVersion 选择22，minSdkVersion选择15，然后创建5个MainActivity
2. 修改业务 module的build.gradle，在android部分添加packageOptios，在dependencies部分添加依赖appbase的说明：

* android{  
   packagingOptions {  
   exclude 'META-INF/DEPENDENCIES'  
   exclude 'META-INF/NOTICE'  
   exclude 'META-INF/LICENSE'  
   exclude 'META-INF/LICENSE.txt'  
   exclude 'META-INF/NOTICE.txt'  
   }  
  }  
  dependencies {  
   compile fileTree(include: ['\*.jar'], dir: 'libs')  
   compile project(':appbase')  
  }

1. 引入appbase中定义样式
   1. 在src/main/res/values/styles.xml中添加:
   * <style name = "Theme.demo" parent="Theme.APPBase">  
      <item name="actionBarInsetStart">@dimen/keyline\_2</item>  
      </style>
   1. 修改AndroidManifast.xml的 application 元素的android:theme属性为 @style/Theme.demo
2. 在AndroidManifast.xml中授权调用系统Account鉴权功能

* <uses-permission android:name="android.permission.AUTHENTICATE\_ACCOUNTS" />  
   <uses-permission android:name="android.permission.GET\_ACCOUNTS" />  
   <uses-permission android:name="android.permission.READ\_PROFILE" />  
   <uses-permission android:name="android.permission.READ\_CONTACTS" />  
   <uses-permission android:name="android.permission.MANAGE\_ACCOUNTS" />

1. 复制base文件夹中的AbstractBaseActivity.java和RecentTasksStyler.java到 业务module中
2. 修改 RecentTasksStyler.java,引用当前module中的ic\_launcher

* if (sIcon == null) {  
   sIcon = BitmapFactory.decodeResource(resources, R.mipmap.ic\_launcher);  
   }

1. 修改Activity的布局文件为：

* <android.support.v4.widget.DrawerLayout  
   xmlns:android="http://schemas.android.com/apk/res/android"  
   xmlns:app="http://schemas.android.com/apk/res-auto"  
   xmlns:tools="http://schemas.android.com/tools"  
   android:id="@+id/drawer\_layout"  
   android:layout\_width="match\_parent"  
   android:layout\_height="match\_parent"  
   android:fitsSystemWindows="true"  
   tools:context="com.example.andrea.myapplication.MainActivity1">  
    
   <com.itic.mobile.base.ui.widget.DrawShadowFrameLayout  
   android:id="@+id/main\_content"  
   android:layout\_width="match\_parent"  
   android:layout\_height="match\_parent"  
   android:clipChildren="false"  
   app:shadowDrawable="@drawable/header\_shadow">  
    
   <com.itic.mobile.base.ui.widget.MultiSwipeRefreshLayout  
   android:id="@+id/swipe\_refresh\_layout"  
   android:layout\_width="match\_parent"  
   android:layout\_height="match\_parent">  
    
   <RelativeLayout  
   android:layout\_width="match\_parent"  
   android:layout\_height="match\_parent">  
    
   <TextView  
   android:layout\_width="wrap\_content"  
   android:layout\_height="wrap\_content"  
   android:textAppearance="?android:attr/textAppearanceMedium"  
   android:text="@string/hello\_world"  
   android:id="@+id/textView"  
   android:layout\_marginTop="69dp"  
   android:layout\_alignParentTop="true"  
   android:layout\_alignParentLeft="true"  
   android:layout\_alignParentStart="true" />  
   </RelativeLayout>  
   </com.itic.mobile.base.ui.widget.MultiSwipeRefreshLayout>  
    
   <LinearLayout  
   android:id="@+id/headerbar"  
   style="@style/HeaderBar"  
   android:layout\_width="match\_parent"  
   android:layout\_height="wrap\_content"  
   android:clickable="true"  
   android:orientation="vertical">  
    
   <include layout="@layout/toolbar\_actionbar" />  
    
   </LinearLayout>  
    
   </com.itic.mobile.base.ui.widget.DrawShadowFrameLayout>  
    
   <!-- Nav drawer -->  
   <include layout="@layout/navdrawer" />  
    
  </android.support.v4.widget.DrawerLayout>

1. 创建BaseActivityImpl，继承AbstractBaseActivity，实现
   * goToNavDrawerItem： 通过主菜单切换Activity
   * populateNavDrawer： 填充主菜单
   * getDrawerItemIconID： 获取Icon 的资源ID
   * getDrawerItemTitleID：获取Activity标题的资源ID
   * isSpecialItem： 设置特殊的Activity，具体参考AbstractBaseActivity中的注释

* 代码如下所示：
* public class BaseActivityImpl extends AbstractBaseActivity {  
    
   protected static final int NAVDRAWER\_ITEM\_1 = 0;  
   protected static final int NAVDRAWER\_ITEM\_2 = 1;  
   protected static final int NAVDRAWER\_ITEM\_3 = 2;  
   protected static final int NAVDRAWER\_ITEM\_4 = 3;  
   protected static final int NAVDRAWER\_ITEM\_5 = 4;  
    
   private static final int[] NAVDRAWER\_TITLE\_RES\_ID = new int[]{  
   R.string.title\_activity\_main1,  
   R.string.title\_activity\_main2,  
   R.string.title\_activity\_main3,  
   R.string.title\_activity\_main4,  
   R.string.title\_activity\_main5  
   };  
    
   private static final int[] NAVDRAWER\_ICON\_RES\_ID = new int[]{  
   R.mipmap.ic\_drawer\_item1,  
   R.mipmap.ic\_drawer\_item2,  
   R.mipmap.ic\_drawer\_item3,  
   R.mipmap.ic\_drawer\_item4,  
   R.mipmap.ic\_drawer\_item5  
   };  
    
   @Override  
   protected void requestDataRefresh() {  
    
   }  
    
   @Override  
   protected String getAccountType() {  
   return null;  
   }  
    
   @Override  
   protected void goToNavDrawerItem(int item) {  
   Intent intent;  
   switch(item){  
   case NAVDRAWER\_ITEM\_1:  
   intent = new Intent(getApplicationContext(),MainActivity1.class);  
   startActivity(intent);  
   finish();  
   break;  
   case NAVDRAWER\_ITEM\_2:  
   intent = new Intent(getApplicationContext(),MainActivity2.class);  
   startActivity(intent);  
   finish();  
   break;  
   case NAVDRAWER\_ITEM\_3:  
   intent = new Intent(getApplicationContext(),MainActivity3.class);  
   startActivity(intent);  
   finish();  
   break;  
   case NAVDRAWER\_ITEM\_4:  
   intent = new Intent(getApplicationContext(),MainActivity4.class);  
   startActivity(intent);  
   finish();  
   break;  
   case NAVDRAWER\_ITEM\_5:  
   intent = new Intent(getApplicationContext(),MainActivity5.class);  
   startActivity(intent);  
   break;  
   }  
   }  
    
   @Override  
   protected void watchSyncStateChange() {  
    
   }  
    
   @Override  
   protected void startLoginProcess() {  
    
   }  
    
   @Override  
   protected boolean isSpecialItem(int itemId) {  
   return itemId == NAVDRAWER\_ITEM\_5;  
   }  
    
   @Override  
   protected void populateActiveAccount(Account account) {  
    
   }  
    
   @Override  
   protected void populateNavDrawer() {  
   mNavDrawerItems.add(NAVDRAWER\_ITEM\_1);  
   mNavDrawerItems.add(NAVDRAWER\_ITEM\_2);  
   mNavDrawerItems.add(NAVDRAWER\_ITEM\_3);  
   mNavDrawerItems.add(NAVDRAWER\_ITEM\_4);  
   mNavDrawerItems.add(NAVDRAWER\_ITEM\_5);  
   createNavDrawerItems();  
   }  
    
   @Override  
   protected int getDrawerItemIconID(int itemID) {  
   return itemID >= 0 && itemID < NAVDRAWER\_ICON\_RES\_ID.length ? NAVDRAWER\_ICON\_RES\_ID[itemID] : 0;  
   }  
    
   @Override  
   protected int getDrawerItemTitleID(int itemID) {  
   return itemID >= 0 && itemID < NAVDRAWER\_TITLE\_RES\_ID.length ?NAVDRAWER\_TITLE\_RES\_ID[itemID] : 0;  
   }  
  }

1. 修改5个MainActivity，前4个修改为：

* public class MainActivity\* extends BaseActivityImpl {  
    
   @Override  
   protected void onCreate(Bundle savedInstanceState) {  
   super.onCreate(savedInstanceState);  
   setContentView(R.layout.activity\_main);  
   overridePendingTransition(0, 0);  
   }  
    
   //一级界面必须覆盖这个方法，返回值为BaseActivityImpl中定义的与该acivity对应的NAVDRAWER\_ITEM值  
   @Override  
   protected int getSelfNavDrawerItem() {  
   return NAVDRAWER\_ITEM\_\*;  
   }  
  }
* 特殊的MainActivity5修改为：
* public class MainActivity5 extends BaseActivityImpl {  
    
   @Override  
   protected void onCreate(Bundle savedInstanceState) {  
   super.onCreate(savedInstanceState);  
   setContentView(R.layout.activity\_main);  
   }  
    
   @Override  
   protected void onPostCreate(Bundle savedInstanceState) {  
   super.onPostCreate(savedInstanceState);  
   final Toolbar toolbar = getActionBarToolbar();  
   toolbar.setNavigationIcon(R.drawable.ic\_up);  
   toolbar.setNavigationOnClickListener(new View.OnClickListener() {  
   @Override  
   public void onClick(View v) {  
   finish();  
   }  
   });  
   }  
    
   //一级界面必须覆盖这个方法，返回值为BaseActivityImpl中定义的与该acivity对应的NAVDRAWER\_ITEM值  
   @Override  
   protected int getSelfNavDrawerItem() {  
   return NAVDRAWER\_ITEM\_5;  
   }  
  }

# 加入登录功能

代码参考 demologin ,过程如下:

1. 参考使用样式一节的 1 - 7 步，完成依赖appbase和创建MainActivity
2. 创建LoginActivity
3. 实现BaseActivityImpl的startLoginProcess方法,在其中判断当前是否有帐号,如果没有则finish当前activity,然后进入loginactivity实现getAccountType

* protected void startLoginProcess() {  
   if (!AccountUtils.hasActiveAccount(getApplicationContext())){  
   String defaultAccount = getDefaultAccount();  
   //如果系统中没有帐号，或者帐号的name属性为空，则需要进入LoginActivity发起登录  
   if (defaultAccount == null){  
   Intent intent = new Intent(getApplicationContext(),LoginActivity.class);  
   startActivity(intent);  
   finish();  
   }  
   AccountUtils.setActiveAccount(getApplicationContext(), defaultAccount);  
   }  
   if (!AccountUtils.hasActiveAccount(getApplicationContext())) {  
   Log.i(TAG, "Can't proceed with login -- no account chosen.");  
   return;  
   } else {  
   Log.i(TAG, "Chosen account: " + AccountUtils.getActiveAccountName(getApplicationContext()));  
   }  
  }

1. 创建authendicator.xml

* <?xml version="1.0" encoding="utf-8"?>  
   <account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"  
   android:accountType="@string/account\_type"  
   android:icon="@mipmap/ic\_launcher"  
   android:label="@string/app\_name"  
   android:smallIcon="@mipmap/ic\_launcher"/>

1. 在AndoridManifast.xml中配置鉴权服务,在其中引入上一步配置的authenicator

* <!-- 新增帐号是调用这个service -->  
   <service android:name="com.itic.mobile.accounts.AccountAuthenticatorService" >  
   <intent-filter>  
   <action android:name="android.accounts.AccountAuthenticator" />  
   </intent-filter>  
    
   <meta-data  
   android:name="android.accounts.AccountAuthenticator"  
   android:resource="@xml/authenicator" />  
   </service>

1. 实现LoginActivity中的登录逻辑，登录成功后将新账号添加进系统账号管理器,然后再打开MainActivity,最后finish LoginActivity

* if (success) {  
   createSyncAccount(getApplicationContext(),mEmail,mPassword);  
   startActivity(new Intent(getApplicationContext(),MainActivity.class));  
   finish();  
   }
* private void createSyncAccount(Context context, String username, String password){  
   Account account = AccountUtils.GetAccount(Config.ACCOUNT\_TYPE,username);  
   AccountManager accountManager = (AccountManager)context.getSystemService(Context.ACCOUNT\_SERVICE);  
   if (accountManager.addAccountExplicitly(account,password,null)){  
   AccountUtils.setActiveAccount(context,username);  
   }  
   }

# 使用AppDataHandler 和 JSONHandler 写数据

代码参考 demodata,过程如下: 1. 建立业务module,依赖appbase

1. 定义表结构,创建SQLiteOpenHelper,创建ContentProvider,实现建库/表以及CURD操作,参考Contract.java,Database.java,Provider.java
2. 创建\*Handler,并继承JSONHandler,实现process和makeContentProviderOperations方法.
   * process方法负责将JSONElement转换为bean,然后添加进集合或者Map
   * makeContentProviderOperations负责将 process处理好的数据封装进ContentProviderOption,并将ContentProviderOption封装进list. list 由AppDataHandler提供. 提供了对C,U,D数据的功能.

* public class ContactHandler extends JSONHandler {  
    
   private HashMap<String, Contact> mContacts = new HashMap<String, Contact>();  
    
   public ContactHandler(Context context) {  
   super(context);  
   }  
    
   @Override  
   public void makeContentProviderOperations(ArrayList<ContentProviderOperation> list) {  
   Uri uri = Contract.Contacts.CONTENT\_URI;  
   list.add(ContentProviderOperation.newDelete(uri).build());  
   for (Contact contact : mContacts.values()) {  
   ContentProviderOperation.Builder builder = ContentProviderOperation.newInsert(uri);  
   builder.withValue(Contract.Contacts.CONTACT\_ID, contact.contact\_id);  
   builder.withValue(Contract.Contacts.CONTACT\_COLOR, contact.contact\_color);  
   builder.withValue(Contract.Contacts.CONTACT\_TYPE, contact.contact\_type);  
   builder.withValue(Contract.Contacts.CONTACT\_NAME, contact.contact\_name);  
   builder.withValue(Contract.Contacts.ORG\_NAME, contact.org\_name);  
   builder.withValue(Contract.Contacts.POST, contact.post);  
   builder.withValue(Contract.Contacts.TEL\_OFFICE, contact.tel\_office);  
   builder.withValue(Contract.Contacts.TEL\_CELL, contact.tel\_cell);  
   builder.withValue(Contract.Contacts.SIM\_IMSI, contact.sim\_imsi);  
   builder.withValue(Contract.Contacts.TEL\_HOME, contact.tel\_home);  
   builder.withValue(Contract.Contacts.EMAIL, contact.email);  
   builder.withValue(Contract.Contacts.FAX, contact.fax);  
   builder.withValue(Contract.Contacts.PXH, contact.pxh);  
   builder.withValue(Contract.Contacts.TXLMLID, contact.txlmlid);  
   builder.withValue(Contract.Contacts.SORT\_KEY, contact.getSortKey());  
   builder.withValue(Contract.Contacts.CONTACT\_IMPORT\_HASHCODE, contact.getImportHashCode());  
   list.add(builder.build());  
   }  
   }  
    
   @Override  
   public void process(JsonElement element) {  
   for (Contact contact : new Gson().fromJson(element, Contact[].class)) {  
   mContacts.put(contact.contact\_id, contact);  
   }  
   }  
  }

1. 导入AppDataHandler,创建AppdatahandlerImpl,继承AppdataHandler,实现mappingJsonHandler方法.
   * mappingJsonHandler负责将各个子业务的JSONHandler导入AppDataHandler,然后通过AppDataHandler的applyData统一调用JSONHandler的process和makeContentProviderOperations来实现数据的CUD

* public class AppDataHandlerImpl extends AppDataHandler {  
    
   private ContactHandler mContactHandler;  
   private static final String DATA\_KEY\_CONTACTS = "contacts";  
    
    
   public AppDataHandlerImpl(Context ctx) {  
   super(ctx, Contract.BASE\_CONTENT\_URI);  
   setTopLevelPath(Contract.TOP\_LEVEL\_PATHS);  
   }  
    
   @Override  
   public void mappingJsonHandler() {  
   mHandlerForKey.put(DATA\_KEY\_CONTACTS,mContactHandler = new ContactHandler(mContext));  
   }  
  }

1. 在MainActivity中完成listview和adapter的绑定,在onStart方法中调用AppDataHandler.applyData,JSON数据在res/raw/bootstarp.json中定义,最后过LoaderManager完成数据 查询逻辑

* @Override  
  protected void onCreate(Bundle savedInstanceState) {  
   super.onCreate(savedInstanceState);  
   setContentView(R.layout.activity\_main);  
   lv = (ListView) findViewById(R.id.listView);  
   adapter = new ContactAdapter();  
   lv.setAdapter(adapter);  
   getLoaderManager().initLoader(ContactsQuery.NORMAL\_TOKEN,null,MainActivity.this);  
  }
* @Override  
  protected void onStart() {  
   super.onStart();  
   if (!PrefUtils.isDataBootstrapDone(this) && mDataBootstrapThread == null){  
   performDataBootstrap();  
   }  
  }  
    
  private void performDataBootstrap() {  
   mDataBootstrapThread = new Thread(  
   new Runnable() {  
   @Override  
   public void run() {  
   try {  
   String bootstrapJson = JSONHandler.parseResource(getApplicationContext(), R.raw.bootstrap\_data);  
   AppDataHandlerImpl dataHandler = new AppDataHandlerImpl(getApplicationContext());  
   dataHandler.applyData(new String[]{bootstrapJson}, Long.toString(DateTimeUtils.stringToDateTime("2015-02-09 00:00:00")), false);  
   Log.i(TAG, "End of bootstrap -- successful. Marking boostrap as done.");  
   PrefUtils.markDataBootstrapDone(getApplicationContext());  
   } catch (IOException e) {  
   PrefUtils.markDataBootstrapDone(getApplicationContext());  
   }  
   mDataBootstrapThread = null;  
   }  
   }  
   );  
   mDataBootstrapThread.start();  
  }
* public class MainActivity extends Activity implements LoaderManager.LoaderCallbacks<Cursor>{  
    
   private static final String TAG = "MainActivity";  
    
   private Cursor mCursor;  
   private ThrottledContentObserver mContactsObserver;  
    
   @Override  
   protected void onCreate(Bundle savedInstanceState) {  
   ....  
   }  
    
   @Override  
   protected void onStart() {  
   ....  
   }  
    
   @Override  
   protected void onResume() {  
   super.onResume();  
   mContactsObserver = new ThrottledContentObserver(new ThrottledContentObserver.Callbacks() {  
   @Override  
   public void onThrottledContentObserverFired() {  
   getLoaderManager().restartLoader(ContactsQuery.NORMAL\_TOKEN,null,MainActivity.this);  
   }  
   });  
   getContentResolver().registerContentObserver(Contract.Contacts.CONTENT\_URI, true, mContactsObserver);  
   }  
    
   @Override  
   protected void onPause() {  
   super.onPause();  
   getContentResolver().unregisterContentObserver(mContactsObserver);  
   }  
    
   @Override  
   public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
   Loader<Cursor> loader = null;  
   if (id == ContactsQuery.NORMAL\_TOKEN){  
   loader = new CursorLoader(this,Contract.Contacts.CONTENT\_URI,ContactsQuery.NORMAL\_PROJECTION,null,null,null);  
   }  
   return loader;  
   }  
    
   @Override  
   public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {  
   int token = loader.getId();  
   if (token == ContactsQuery.NORMAL\_TOKEN){  
   if (mCursor != null && mCursor != cursor){  
   mCursor.close();  
   }  
   mCursor = cursor;  
   mCursor.moveToPosition(-1);  
   int i = mCursor.getCount();  
   adapter.changeCursor(mCursor);  
   }  
   }  
    
   @Override  
   public void onLoaderReset(Loader<Cursor> loader) {  
    
   }  
    
   private class ContactAdapter extends CursorAdapter{  
    
   public ContactAdapter(){  
   super(getApplicationContext(),null,false);  
   }  
    
   @Override  
   public Contact getItem(int position) {  
   if (mCursor == null || !mCursor.moveToPosition(position)) {  
   return null;  
   }  
   final String xm = mCursor.getString(mCursor.getColumnIndex(Contract.Contacts.CONTACT\_NAME));  
   final String zw = mCursor.getString(mCursor.getColumnIndex(Contract.Contacts.POST));  
   final String szdw = mCursor.getString(mCursor.getColumnIndex(Contract.Contacts.ORG\_NAME));  
    
   Contact contact = new Contact();  
   contact.contact\_name = xm;  
   contact.org\_name = szdw;  
   contact.post = zw;  
    
   return contact;  
   }  
    
   @Override  
   public View newView(Context context, Cursor cursor, ViewGroup parent) {  
   LayoutInflater mInflater = (LayoutInflater) getApplicationContext().getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);  
   return mInflater.inflate(R.layout.list\_item\_contact, null);  
   }  
    
   @Override  
   public void bindView(View view, Context context, Cursor cursor) {  
   Holder holder = getHolder(view);  
   holder.xmView.setText(  
   TextViewUtils.setText(mCursor.getString(  
   mCursor.getColumnIndex(Contract.Contacts.CONTACT\_NAME))));  
   holder.szdwView.setText(  
   TextViewUtils.setText(mCursor.getString(  
   mCursor.getColumnIndex(Contract.Contacts.ORG\_NAME))));  
   holder.zwView.setText(  
   TextViewUtils.setText(mCursor.getString(  
   mCursor.getColumnIndex(Contract.Contacts.POST))));   
   }  
    
   private Holder getHolder(final View view){  
   Holder holder = (Holder)view.getTag();  
   if (holder == null){  
   holder = new Holder(view);  
   view.setTag(holder);  
   }  
   return holder;  
   }  
   }  
    
   private static class Holder {  
   public TextView xmView;  
   public TextView zwView;  
   public TextView szdwView;  
    
   public Holder(View view) {  
   xmView = (TextView) view.findViewById(R.id.xm);  
   zwView = (TextView) view.findViewById(R.id.zw);  
   szdwView = (TextView) view.findViewById(R.id.szdw);  
   }  
   }  
    
   private interface ContactsQuery {  
   int NORMAL\_TOKEN = 0x1;  
    
   String[] NORMAL\_PROJECTION = {  
   BaseColumns.\_ID,  
   Contract.Contacts.CONTACT\_NAME,  
   Contract.Contacts.POST,  
   Contract.Contacts.ORG\_NAME,  
   };  
   }  
  }