## 开源代码标注报告

撰写人员：林杜铭 洪志榜 郑宇星 钟圣杰

### 开源软件的代码标注概况

小组对net.micode.notes.ui、net.micode.notes.gtask.remote、net.micode.notes.gtask.data包中共25个类的代码进行了标注，标注的代码行数共有7772行。

具体的，这些代码注释的分部如表2所示。

表1. 注释概况

（表中序号根据实际情况调整，包中类或方法较多则按类编号，否则按包编号）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **包名称** | **类名称** | **功能简述** | **标注的代码行数** | **标注人** |
|  | Ui | AlarmAlertActivity | 闹钟铃声 | 192 | 林杜铭 |
|  | Ui | AlarmInitReceiver | 创建闹钟 | 56 | 林杜铭 |
|  | Ui | AlarmReceiver | 实现alarm功能 | 19 | 林杜铭 |
|  | Ui | [DateTimePicker](https://so.csdn.net/so/search?q=DateTimePicker&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/edgarrr/article/details/_blank) | 时间改变监听器 | 491 | 林杜铭 |
|  | Ui | DateTimePickerDialog | 显示监听时间 | 87 | 林杜铭 |
|  | Ui | DropdownMenu | 下拉菜单 | 49 | 林杜铭 |
|  | Ui | FoldersListAdapter | 便签数据库和用户的交互 | 71 | 林杜铭 |
|  | Ui | NoteEditActivity | 标签的编辑 | 1067 | 洪志榜 |
|  | Ui | NoteEditText | 设置便签文本框 | 270 | 洪志榜 |
|  | Ui | NoteItemData | 电话号码数据 | 214 | 洪志榜 |
|  | Ui | NotesListActivity | 菜单列表主函数 | 1002 | 洪志榜 |
|  | Ui | NotesListAdapter | 便签表连接器 | 257 | 洪志榜 |
|  | Ui | NotesListItem | 列表各项目属性 | 116 | 洪志榜 |
|  | Ui | NotesPreferenceActivity | 对背景颜色和字体大小的数据储存 | 514 | 洪志榜 |
|  | remote | GTaskASyncTask.java | 向用户提示当前同步的状态，是一个用于交互的方法 | 102 | 郑宇星 |
|  | remote | GTaskClient.java | 实现GTASK的登录操作，进行GTASK任务的创建，创建任务列表，从网络上获取任务和任务列表的内容 | 612 | 郑宇星 |
|  | remote | GTaskManager.java | 刷新本地，给sync的ID对应上最后更改过的对象 | 846 | 郑宇星 |
|  | remote | GTaskSyncService.java | Service是在一段不定的时间运行在后台，不和用户交互的应用组件 | 124 | 郑宇星 |
|  | remote | GTaskASyncTask.java | 向用户提示当前同步的状态，是一个用于交互的方法 | 102 | 郑宇星 |
|  | remote | GTaskClient.java | 实现GTASK的登录操作，进行GTASK任务的创建，创建任务列表，从网络上获取任务和任务列表的内容 | 612 | 钟圣杰 |
|  | gtask.data | MetaData | 得到类的简写名称存入字符串TAG中 | 110 | 钟圣杰 |
|  | gtask.data | Node | 同步操作的基础数据类型，定义了相关指示同步操作的常量 | 79 | 钟圣杰 |
|  | gtask.data | SqlData | 用于支持小米便签最底层的数据库相关操作，和sqlnote的关系上是子集关系，即data是note的子集（节点）。  SqlData其实就是也就是所谓数据中的数据 | 210 | 钟圣杰 |
|  | gtask.data | SqlNote.java | 用于支持小米便签最底层的数据库相关操作，和sqldata的关系上是父集关系，即note是data的子父集。 | 543 | 钟圣杰 |
|  | gtask.data | ActionFailureException | 支持小米便签运行过程中的运行异常处理 | 27 | 钟圣杰 |
| 共计 | 3个包 | 25类 |  | 7772行代码标注 |  |
|  |  |  |  |  |  |

部分详细代码如下：

AlarmAlertActivity.java

public class AlarmAlertActivity extends Activity implements OnClickListener, OnDismissListener {

private long mNoteId; //文本在数据库存储中的ID号

private String mSnippet; //闹钟提示时出现的文本片段

private static final int SNIPPET\_PREW\_MAX\_LEN = 60;

MediaPlayer mPlayer;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

//Bundle类型的数据与Map类型的数据相似，都是以key-value的形式存储数据的

//onsaveInstanceState方法是用来保存Activity的状态的

//能从onCreate的参数savedInsanceState中获得状态数据

requestWindowFeature(Window.FEATURE\_NO\_TITLE);

//界面显示——无标题

final Window win = getWindow();

win.addFlags(WindowManager.LayoutParams.FLAG\_SHOW\_WHEN\_LOCKED);

if (!isScreenOn()) {

win.addFlags(WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON

//保持窗体点亮

| WindowManager.LayoutParams.FLAG\_TURN\_SCREEN\_ON

//将窗体点亮

| WindowManager.LayoutParams.FLAG\_ALLOW\_LOCK\_WHILE\_SCREEN\_ON

//允许窗体点亮时锁屏

| WindowManager.LayoutParams.FLAG\_LAYOUT\_INSET\_DECOR);

}//在手机锁屏后如果到了闹钟提示时间，点亮屏幕

Intent intent = getIntent();

try {

mNoteId = Long.valueOf(intent.getData().getPathSegments().get(1));

mSnippet = DataUtils.getSnippetById(this.getContentResolver(), mNoteId);

//根据ID从数据库中获取标签的内容；

//getContentResolver（）是实现数据共享，实例存储。

mSnippet = mSnippet.length() > SNIPPET\_PREW\_MAX\_LEN ? mSnippet.substring(0,

SNIPPET\_PREW\_MAX\_LEN) + getResources().getString(R.string.notelist\_string\_info)

: mSnippet;

//判断标签片段是否达到符合长度

} catch (IllegalArgumentException e) {

e.printStackTrace();

return;

}

/\*

try

{

// 代码区

}

catch(Exception e)

{

// 异常处理

}

代码区如果有错误，就会返回所写异常的处理。\*/

mPlayer = new MediaPlayer();

if (DataUtils.visibleInNoteDatabase(getContentResolver(), mNoteId, Notes.TYPE\_NOTE)) {

showActionDialog();

//弹出对话框

playAlarmSound();

//闹钟提示音激发

} else {

finish();

//完成闹钟动作

}

}

private boolean isScreenOn() {

//判断屏幕是否锁屏，调用系统函数判断，最后返回值是布尔类型

PowerManager pm = (PowerManager) getSystemService(Context.POWER\_SERVICE);

return pm.isScreenOn();

}

private void playAlarmSound() {

//闹钟提示音激发

Uri url = RingtoneManager.getActualDefaultRingtoneUri(this, RingtoneManager.TYPE\_ALARM);

//调用系统的铃声管理URI，得到闹钟提示音

int silentModeStreams = Settings.System.getInt(getContentResolver(),

Settings.System.MODE\_RINGER\_STREAMS\_AFFECTED, 0);

if ((silentModeStreams & (1 << AudioManager.STREAM\_ALARM)) != 0) {

mPlayer.setAudioStreamType(silentModeStreams);

} else {

mPlayer.setAudioStreamType(AudioManager.STREAM\_ALARM);

}

try {

mPlayer.setDataSource(this, url);

//方法：setDataSource(Context context, Uri uri)

//解释：无返回值，设置多媒体数据来源【根据 Uri】

mPlayer.prepare();

//准备同步

mPlayer.setLooping(true);

//设置是否循环播放

mPlayer.start();

//开始播放

} catch (IllegalArgumentException e) {

// TODO Auto-generated catch block

e.printStackTrace();

//e.printStackTrace()函数功能是抛出异常， 还将显示出更深的调用信息

//System.out.println(e)，这个方法打印出异常，并且输出在哪里出现的异常

} catch (SecurityException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IllegalStateException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

AlarmInitReceiver.java

public class AlarmInitReceiver extends BroadcastReceiver {

private static final String [] PROJECTION = new String [] {

NoteColumns.ID,

NoteColumns.ALERTED\_DATE

};

//对数据库的操作，调用标签ID和闹钟时间

private static final int COLUMN\_ID = 0;

private static final int COLUMN\_ALERTED\_DATE = 1;

@Override

public void onReceive(Context context, Intent intent) {

long currentDate = System.currentTimeMillis();

//System.currentTimeMillis()产生一个当前的毫秒

//这个毫秒其实就是自1970年1月1日0时起的毫秒数

Cursor c = context.getContentResolver().query(Notes.CONTENT\_NOTE\_URI,

PROJECTION,

NoteColumns.ALERTED\_DATE + ">? AND " + NoteColumns.TYPE + "=" + Notes.TYPE\_NOTE,

new String[] { String.valueOf(currentDate) },

//将long变量currentDate转化为字符串

null);

//Cursor在这里的作用是通过查找数据库中的标签内容，找到和当前系统时间相等的标签

if (c != null) {

if (c.moveToFirst()) {

do {

long alertDate = c.getLong(COLUMN\_ALERTED\_DATE);

Intent sender = new Intent(context, AlarmReceiver.class);

sender.setData(ContentUris.withAppendedId(Notes.CONTENT\_NOTE\_URI, c.getLong(COLUMN\_ID)));

PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, sender, 0);

AlarmManager alermManager = (AlarmManager) context

.getSystemService(Context.ALARM\_SERVICE);

alermManager.set(AlarmManager.RTC\_WAKEUP, alertDate, pendingIntent);

} while (c.moveToNext());

}

c.close();

}

//然而通过网上查找资料发现，对于闹钟机制的启动，通常需要上面的几个步骤

//如新建Intent、PendingIntent以及AlarmManager等

//这里就是根据数据库里的闹钟时间创建一个闹钟机制

}

}

AlarmReceiver.java

public class AlarmReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

intent.setClass(context, AlarmAlertActivity.class);

//启动AlarmAlertActivity

intent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

//activity要存在于activity的栈中，而非activity的途径启动activity时必然不存在一个activity的栈

//所以要新起一个栈装入启动的activity

context.startActivity(intent);

}

}

//这是实现alarm这个功能最接近用户层的包，基于上面的两个包，

//作用还需要深究但是对于setClass和addFlags的

GTaskSyncService.java

package net.micode.notes.gtask.remote;

/\*

\* Service是在一段不定的时间运行在后台，不和用户交互的应用组件

\* 主要方法：

\* private void startSync() 启动一个同步工作

\* private void cancelSync() 取消同步

\* public void onCreate()

\* public int onStartCommand(Intent intent, int flags, int startId) service生命周期的组成部分，相当于重启service（比如在被暂停之后），而不是创建一个新的service

\* public void onLowMemory() 在没有内存的情况下如果存在service则结束掉这的service

\* public IBinder onBind()

\* public void sendBroadcast(String msg) 发送同步的相关通知

\* public static void startSync(Activity activity)

\* public static void cancelSync(Context context)

\* public static boolean isSyncing() 判读是否在进行同步

\* public static String getProgressString() 获取当前进度的信息

\*/

public class GTaskSyncService extends Service {

public final static String ACTION\_STRING\_NAME = "sync\_action\_type";

public final static int ACTION\_START\_SYNC = 0;

public final static int ACTION\_CANCEL\_SYNC = 1;

public final static int ACTION\_INVALID = 2;

public final static String GTASK\_SERVICE\_BROADCAST\_NAME = "net.micode.notes.gtask.remote.gtask\_sync\_service";

public final static String GTASK\_SERVICE\_BROADCAST\_IS\_SYNCING = "isSyncing";

public final static String GTASK\_SERVICE\_BROADCAST\_PROGRESS\_MSG = "progressMsg";

private static GTaskASyncTask mSyncTask = null;

private static String mSyncProgress = "";

//开始一个同步的工作

private void startSync() {

if (mSyncTask == null) {

mSyncTask = new GTaskASyncTask(this, new GTaskASyncTask.OnCompleteListener() {

public void onComplete() {

mSyncTask = null;

sendBroadcast("");

stopSelf();

}

});

sendBroadcast("");

mSyncTask.execute(); //这个函数让任务是以单线程队列方式或线程池队列方式运行

}

}

private void cancelSync() {

if (mSyncTask != null) {

mSyncTask.cancelSync();

}

}

@Override

public void onCreate() { //初始化一个service

mSyncTask = null;

}

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

Bundle bundle = intent.getExtras();

if (bundle != null && bundle.containsKey(ACTION\_STRING\_NAME)) {

switch (bundle.getInt(ACTION\_STRING\_NAME, ACTION\_INVALID)) {

//两种情况，开始同步或者取消同步

case ACTION\_START\_SYNC:

startSync();

break;

case ACTION\_CANCEL\_SYNC:

cancelSync();

break;

default:

break;

}

return START\_STICKY; //等待新的intent来是这个service继续运行

}

return super.onStartCommand(intent, flags, startId);

}

@Override

public void onLowMemory() {

if (mSyncTask != null) {

mSyncTask.cancelSync();

}

}

public IBinder onBind(Intent intent) { //不知道干吗用的

return null;

}

public void sendBroadcast(String msg) {

mSyncProgress = msg;

Intent intent = new Intent(GTASK\_SERVICE\_BROADCAST\_NAME); //创建一个新的Intent

intent.putExtra(GTASK\_SERVICE\_BROADCAST\_IS\_SYNCING, mSyncTask != null); //附加INTENT中的相应参数的值

intent.putExtra(GTASK\_SERVICE\_BROADCAST\_PROGRESS\_MSG, msg);

sendBroadcast(intent); //发送这个通知

}

public static void startSync(Activity activity) {//执行一个service，service的内容里的同步动作就是开始同步

GTaskManager.getInstance().setActivityContext(activity);

Intent intent = new Intent(activity, GTaskSyncService.class);

intent.putExtra(GTaskSyncService.ACTION\_STRING\_NAME, GTaskSyncService.ACTION\_START\_SYNC);

activity.startService(intent);

}

public static void cancelSync(Context context) {//执行一个service，service的内容里的同步动作就是取消同步

Intent intent = new Intent(context, GTaskSyncService.class);

intent.putExtra(GTaskSyncService.ACTION\_STRING\_NAME, GTaskSyncService.ACTION\_CANCEL\_SYNC);

context.startService(intent);

}

public static boolean isSyncing() {

return mSyncTask != null;

}

public static String getProgressString() {

return mSyncProgress;

}

}

MetaData.java

package net.micode.notes.gtask.data;

public class MetaData extends Task {

/\*

\* 功能描述：得到类的简写名称存入字符串TAG中

\* 实现过程：调用getSimpleName ()函数

\*/

private final static String TAG = MetaData.class.getSimpleName();

private String mRelatedGid = null;

/\*

\* 功能描述：设置数据，即生成元数据库

\* 实现过程：调用JSONObject库函数put ()，Task类中的setNotes ()和setName ()函数

\* 参数注解：

\*/

public void setMeta(String gid, JSONObject metaInfo)

{

//对函数块进行注释

try {

metaInfo.put(GTaskStringUtils.META\_HEAD\_GTASK\_ID, gid);

/\*

\* 将这对键值放入metaInfo这个jsonobject对象中

\*/

} catch (JSONException e) {

Log.e(TAG, "failed to put related gid");

/\*

\* 输出错误信息

\*/

}

setNotes(metaInfo.toString());

setName(GTaskStringUtils.META\_NOTE\_NAME);

}

/\*

\* 功能描述：获取相关联的Gid

\*/

public String getRelatedGid() {

return mRelatedGid;

}

/\*

\* 功能描述：判断当前数据是否为空，若为空则返回真即值得保存

\* Made By CuiCan

\*/

@Override

public boolean isWorthSaving() {

return getNotes() != null;

}

/\*

\* 功能描述：使用远程json数据对象设置元数据内容

\* 实现过程：调用父类Task中的setContentByRemoteJSON ()函数，并

\* 参数注解：

\*/

@Override

public void setContentByRemoteJSON(JSONObject js) {

super.setContentByRemoteJSON(js);

if (getNotes() != null) {

try {

JSONObject metaInfo = new JSONObject(getNotes().trim());

mRelatedGid = metaInfo.getString(GTaskStringUtils.META\_HEAD\_GTASK\_ID);

} catch (JSONException e) {

Log.w(TAG, "failed to get related gid");

/\*

\* 输出警告信息

\*/

mRelatedGid = null;

}

}

}

/\*

\* 功能描述：使用本地json数据对象设置元数据内容，一般不会用到，若用到，则抛出异常

\* Made By CuiCan

\*/

@Override

public void setContentByLocalJSON(JSONObject js) {

// this function should not be called

throw new IllegalAccessError("MetaData:setContentByLocalJSON should not be called");

/\*

\* 传递非法参数异常

\*/

}

/\*

\* 功能描述：从元数据内容中获取本地json对象，一般不会用到，若用到，则抛出异常

\* Made By CuiCan

\*/

@Override

public JSONObject getLocalJSONFromContent() {

throw new IllegalAccessError("MetaData:getLocalJSONFromContent should not be called");

/\*

\* 传递非法参数异常

\* Made By Cui Can

\*/

}

/\*

\* 功能描述：获取同步动作状态，一般不会用到，若用到，则抛出异常

\* Made By CuiCan

\*/

@Override

public int getSyncAction(Cursor c) {

throw new IllegalAccessError("MetaData:getSyncAction should not be called");

/\*

\* 传递非法参数异常

\* Made By Cui Can

\*/

}

}

### 实践收获和体会

//<用文字描述注释开源软件实践收获和体会>

收获1：在编写小米便签开源代码注释的过程中，我深刻体会到了代码注释的重要性。注释不仅能帮助其他开发者更好地理解代码逻辑，还能在日后自己回顾代码时提供重要参考。在编写注释时，我注重保持简洁明了，避免冗余和模糊的表达，力求让每一条注释都能准确反映代码的功能和意图。

收获2：在深入剖析小米便签的开源代码时，我感受到了开源社区的力量。每一行代码背后都是开发者们的智慧和汗水，他们通过开源的方式，让更多的人能够学习和使用他们的成果。在编写注释的过程中，我尽量保持与原有代码风格的一致性，同时也加入了一些自己的理解和思考。

收获3：通过编写小米便签开源代码的注释，我更加深入地理解了软件开发的规范和流程。在注释中，我注重描述函数或模块的功能、输入参数、返回值以及可能的异常情况，这有助于构建一个清晰、易读的代码结构。同时，我也学到了如何在编写注释时保持客观性和准确性，避免主观臆断和误导性的描述。这次经历不仅提升了我的编程技能，也让我对软件开发有了更加全面的认识。

问题1：代码理解能力尚有待加强

问题2：团队的协作能力还有待加强