系统实现

# 环境搭建

正式开始记事本的实现之前，还必须做一些基础的前期准备工作。良好的开始是成功的一半，只有奠定了良好的基础，后期才能方便快速地完成具体的程序实现。

## 安装JDK和eclipse

1. 安装JDK

访问 <http://www.oracle.com/technetwork/java/javase/downloads/index.html> 下载JDK1.7，放在一个E:\Program Files目录下，配置一下环境变量如下：

PATH：

E:\Program Files\Java\jdk1.7.0\_02\bin

E:\Program Files\Java\jre7\bin

CLASSPATH：

E:\Program Files\Java\jdk1.7.0\_02\lib\dt.jar

E:\Program Files\Java\jdk1.7.0\_02\lib\tools.jar

1. 安装eclipse

访问 <http://www.eclipse.org/downloads> 页面下载Eclipse安装包即可。

## 安装android SDK

访问<http://developer.android.com/sdk/index.html>页面下载Android SDK starter package压缩包，解压后，点击里面的SDK Manager.exe，SDK Manager会自动去网站上查看有什么SDK可供下载，选择需要的版本下载即可。

## 安装Eclipse插件ADT

1. 打开Eclipse，点击Help-->Install New Software
2. 点击"Add"按钮，在弹窗中，Name随便写，Location写入 http://dl-ssl.google.com/android/eclipse/
3. Eclipse自动侦测地址中的插件，全选之后下一步就可以了，之后安装完成即可。
4. Eclipse中点击Windows --> Preference，在Android标签中的SDK Location中，选中SDK目录，使ADT指向SDK。

至此，基本的android开发环境搭建完成。

# 功能实现

## 总述

### Android基本结构

一般android的应用程序主要有四种构成部分：Activity，Intent，Service，Content Provider。在本次的记事本应用程序中，用到了其中的Activity，Intent，Content Provider三个部分。

另外一个比较重要的文件是androidManifest.xml，这个文件用来声明应用程序组件和它们的功能需求是什么，当应用程序中需要某些组件时，必须在这个xml文件中声明才可用。

1. **Activity**

Activity是四个构建模块中最普通的一个，一个activity在应用程序中通常是一个单独的屏幕，每一个activity是实现作为一个继承于activity基类的单独的类。例如在本次的记事本应用中，首页的记事列表是一个屏幕，编辑查看记事的页面是另一个屏幕，它们都是作为单独的activity存在的，当新建或者选择某一个记事时，就会进入编辑查看记事的页面，这一过程是通过启动一个activity来实现的。并且当一个新的屏幕开启，先前的屏幕被暂停压入历史堆栈，用户可以向后导航返回到先前打开的屏幕。

1. **Intent和IntentFilter**

Android使用一个名为intent的特殊类让应用程序从一个屏幕移动到另一个屏幕，intent描述了一个应用程序想要干什么。Intent数据结构的两个重要部分是：要响应的action和data。典型的action值MAIN，VIEW，PICK，EDIT。Data表示成一个统一资源定界符。

Intent是一个有效的请求来做什么事，而IntentFilter是描述一个activity能够处理什么样的Intent。

从一个屏幕到另一个屏幕的导航通过解决Intent来完成。要向前导航，一个activity调用startActivity(myIntent)。系统接下来查找所以应用程序的IntentFilter，并选择一个IntentFilter最适合myIntent的activity。

1. **Content Provider**

应用程序可以把它们的数据存储到文件中，一个SQLite数据库中，设置中，或者其他任何有意义的结构中。Content Provider可以让应用程序之间共享数据，一个content Provider是一个类，它实现了方法的一个标准设置来让其他应用程序存储和恢复被Content Provider处理的数据类型。

### 系统架构设计

本次的记事本应用中共分为四个包：activity，model，tools，view。

activity包里面的类继承自基类activity，主要用来处理两个activity之间的跳转和数据传递，包括的类有：ClockAlertActivity.java，ClockReceiver.java，NoteEditActivity.java，ShareReceiver.java。

model包里面的类用来从数据中读取数据或者存储数据，包括的类有：ClockModel.java，NoteItemModel.java，分别处理闹钟数据和记事数据。

view包里面的类继承自基类view，用来控制界面的显示和事件的侦听，包括的类比较多，有MainView.java，NoteItemView.java，NoteListView.java，ClockCalendar.java等。

Tools包里面只有两个类：Lunar.java，DBHelper.java。Lunar类用来处理农历的显示，DBHelper类封装了各种数据库操作。

## 记事模块

### 总述

记事模块的实现主要分为NoteEditView，NoteItemView，NotesListView，NoteItemModel，NoteEditActivity五个类，每个类控制着不同的部分。

NotesListView类控制的是首页记事列表的展现，包括加载每个记事item，更新列表以及每个item的点击事件处理等。

NoteItemView类控制的是记事列表中每个item的展现，包括两行记事内容，修改时间，是否有闹钟以及删除事件的处理。

NoteEditView类控制的是记事编辑页面，包括导航栏上各个按钮的点击事件处理以及编辑记事内容，添加图片、音频、闹钟等操作都是以这个页面为入口。

NoteItemModel类主要是用来获取每个记事的数据的。

NoteEditActivity类是一个activity，用来加载EditView视图以及添加图片的一些处理。

### 记事MODEL

在model包中，有NoteItemModel类，这个类的作用是创建一个记事的实例，这个实例包含了记事的所有信息，包括内容、创建时间、闹钟等。它有四个构造函数，可以创建空的实例或者根据参数从数据库中读取相应记事的所有数据，供其他类使用。

当我们新建一个记事时，需要创建一个空的NoteItemModel实例。构造函数如下：

**public** NoteItemModel(Context context) {

**this**.mContext = context;

**this**.mId = -1;

**this**.mClockId = -1;

**this**.mCreateDate = **new** Date().getTime();

**this**.mModifyDate = **new** Date().getTime();

**this**.mContent = "";

**this**.mAudio = "";

**this**.mSequence = 0;

**this**.mClockModel = **null**;

**this**.findImages(**this**.mContent);

}

当编辑记事或者需要获取记事的某一项数据如记事内容时，可以调用其他的构造函数，例如如果我们知道记事的id，就可以调用以id为参数的构造函数：

**public** NoteItemModel(Context context, **int** id) {

**this**.mContext = context;

DBHelper dbHelper = DBHelper.*getInstance*(context);

Cursor cursor = dbHelper.query(DBHelper.TABLE.*NOTES*, **null**,

NoteItemModel.*ID* + " = " + id, **null**, **null**);

cursor.moveToFirst();

**this**.mId = cursor.getInt(*ID\_COLUMN*);

**this**.mClockId = cursor.getInt(*CLOCK\_ID\_COLUMN*);

**this**.mCreateDate = cursor.getLong(*CREATE\_DATE\_COLUMN*);

**this**.mModifyDate = cursor.getLong(*MODIFY\_DATE\_COLUMN*);

**this**.mContent = cursor.getString(*CONTENT\_COLUMN*);

**this**.mAudio = cursor.getString(*AUDIO\_COLUMN*);

**this**.mSequence = cursor.getInt(*SEQUENCE\_COLUMN*);

**this**.mClockModel = **this**.getClockModel(**this**.mClockId);

**this**.findImages(**this**.mContent);

}

该构造函数先根据id从数据库中查询记事，再把记事的每一项数据赋值给实例属性。

另外两个构造函数分别以Cursor和ContentValues为参数，原理基本相同，在此不列举。

在这个类中，还包括获取和设置实例属性的方法，如getId和setId分别用来获取和设置id属性，还有getModifyDate和setModifyDate分别用来获取和设置修改时间属性。其他属性的获取和设置类似，在此不列举。

### 添加图片

在图片记事中除了添加文字之外，最重要的功能是添加图片，添加图片的功能主要在NoteEditActivity.java和NoteEditView.java中实现，其中NoteEditView类主要是为按钮注册事件，而对图片的数据处理都是在NoteEditActivity类中完成的。

在NoteEditView.java中，为导航栏上的图片按钮注册点击事件，一旦侦听到点击事件，即调用addPicture函数。addPicture函数的主要作用是启动“相机”或者“图库”程序，它首先弹出一个dialog，让用户选择是“拍照”还是“从相册中选择”，然后根据用户的选择去启动相应的程序。启用其他的应用程序或者activity需要用到android的Intent类，这里我们需要创建一个Intent实例，利用不同的参数来告诉Intent是启动“相机”还是启动“图库”，如果是启动“相机”，则参数为*android.media.action.IMAGE\_CAPTURE*，如果启动“图库”，则参数为*Intent.ACTION\_GET\_CONTENT，*最后调用startActivityForResult来启动应用程序。startActivityForResult有两个参数，第一个参数就是我们创建的Intent实例，第二个参数作为requestCode，标识是“拍照”还是“从图库中选择”。

代码如下：

**protected** **void** addPicture() {

((InputMethodManager) **this**.getContext().getSystemService(

Context.*INPUT\_METHOD\_SERVICE*)).hideSoftInputFromWindow(

**this**.getWindowToken(), InputMethodManager.*HIDE\_NOT\_ALWAYS*);

**final** CharSequence[] items = { "从相册中选择", "拍照" };

AlertDialog dlg = **new** AlertDialog.Builder(**this**.getContext())

.setTitle("选择应用程序")

.setItems(items, **new** DialogInterface.OnClickListener() {

**public** **void** onClick(DialogInterface dialog, **int** item) {

**if** (item == 1) {

Intent getImageByCamera = **new** Intent(

"android.media.action.IMAGE\_CAPTURE");

((Activity) NoteEditView.**this**.getContext())

.startActivityForResult(getImageByCamera, 2);

} **else** {

Intent getImage = **new** Intent(

Intent.*ACTION\_GET\_CONTENT*);

getImage.addCategory(Intent.*CATEGORY\_OPENABLE*);

getImage.setType("image/\*");

((Activity) NoteEditView.**this**.getContext())

.startActivityForResult(getImage, 1);

}

}

}).create();

dlg.show();

}

在NoteEditActivity.java中，接收“相机”或者“图库”返回的数据的是onActivityResult函数，这个函数有两个参数requestCode和intent，intent承载着返回的图片数据，而requestCode则标识是相机返回的数据还是图库返回的数据，从而进行不同的处理。

如果是相机返回的数据，则利用intent.getData()获取数据，获取的数据是Uri实例，需要用BitmapFactory类把数据转换成bitmap（位图）；如果是图库返回的数据，则利用intent.getExtras()获取数据，获取的数据是Bundle实例，需要用get("data")获取bitmap（位图）数据。

不管是那种格式的数据，最终都将转换成bitmap，然后对bitmap进行处理。这里主要有三种处理：

首先，对bitmap进行缩小。为了保证图片不变形，这里采用的是等比缩小，并且图片的最终width不超过300px，height不超过200px，如果图片原始高度height大于原始宽度width，则根据width来缩小，即如果width小于300，则保持不变，如果大于300，则固定成300，height随之缩小；反之，则根据height来缩小，即如果height小于200，则保持不变，如果大于200，则height固定成200，width随之缩小。最终得到图片高度和宽度的缩放比scaleWidth和scaleHeight，再利用matrix.postScale(scaleWidth, scaleHeight)实施缩放动作，最后用Bitmap.createBitmap(originalBitmap, 0, 0, width, height,matrix, true)生成新的bitmap图片。

代码如下：

**private** Bitmap resizeImage(Bitmap originalBitmap, **int** newWidth,

**int** newHeight) {

**int** width = originalBitmap.getWidth();

**int** height = originalBitmap.getHeight();

**float** scaleWidth = 1;

**float** scaleHeight = (**float**) newHeight / height;

**if**(width <= height){

**if**(width <= newWidth){

scaleWidth = 1;

}

**else**{

scaleWidth = (**float**) newWidth / width;

}

scaleHeight = scaleWidth;

}

**else**{

**if**(height <= newHeight){

scaleHeight = 1;

}

**else**{

scaleHeight = (**float**) newHeight / height;

}

scaleWidth = scaleHeight;

}

Matrix matrix = **new** Matrix();

matrix.postScale(scaleWidth, scaleHeight);

Bitmap resizedBitmap = Bitmap.*createBitmap*(originalBitmap, 0, 0, width,

height, matrix, **true**);

**return** resizedBitmap;

}

第二，保存图片。这里将图片保存到sdcard里面，格式为jpeg格式，并且为了保证图片命名不冲突，以当前的时间戳作为图片的名称，调用System.currentTimeMillis()即可获取当前的时间戳。先利用图片路径创建File文件实例，再利用FileOutputStream类创建输出流outStream，最后将outStream作为参数传递给bitmap.compress函数，将输出流压缩成jpeg格式保存。

代码如下：

**private** String savePicture(Bitmap bitmap) {

File root = **this**.getExternalFilesDir(Environment.*DIRECTORY\_PICTURES*);

String name = "/pic\_" + System.*currentTimeMillis*() + ".jpeg";

**try** {

File f = **new** File(root.getAbsolutePath() + name);

f.createNewFile();

FileOutputStream outStream = **new** FileOutputStream(f.getPath());

bitmap.compress(Bitmap.CompressFormat.*JPEG*, 100, outStream);

outStream.flush();

outStream.close();

**return** name;

} **catch** (IOException e) {

e.printStackTrace();

}

**return** **null**;

}

第三，显示图片，将图片显示在EditView里面。这种显示图片的方式比较困难，因为EditView本身是不能显示图片的，所以这里采用了变通的方式，即将图片当成文本一样来处理，这样的一个好处是我们可以添加多张图片，并且图片和文字的位置可以随意，我们可以在文本的任意地方添加图片，用户体验更好。把图片当成文本来处理，需要用到的是Image Span和SpannableString这两个类，这两个类的作用是可以对文本进行各种特别的设置，比如颜色、大小、首行缩进，或者是在一段文本中加入图片。先利用bitmap创建Image Span实例，然后创建一个SpannableString实例，实例的内容为[ox64+图片路径]，以便插入用ImageSpan对象封装的图像，将 ImageSpan实例插入SpannableString中，插入的位置是从，SpannableString的开始位置0到末尾位置，这样就能将SpannableString的字符串内容覆盖，最后将图片追加到EditText光标所在的位置。

代码如下：

**public** **void** showBitmapImg(Bitmap bitmap) {

bitmap = resizeImage(bitmap, 300, 200);

String filePath = **this**.savePicture(bitmap);

ImageSpan imageSpan = **new** ImageSpan(NoteEditActivity.**this**, bitmap);

SpannableString spannableString = **new** SpannableString("[0x64"

+ filePath + "]");

spannableString.setSpan(imageSpan, 0, spannableString.length(),

Spannable.*SPAN\_EXCLUSIVE\_EXCLUSIVE*);

**int** index = mNoteEditView.mContent.getSelectionStart(); Editable edit\_text = mNoteEditView.mContent.getEditableText();

**if** (index < 0 || index >= edit\_text.length()) {

edit\_text.append(spannableString);

} **else** {

edit\_text.insert(index, spannableString);

}

**this**.mNoteEditView.findImages();

}

第四，将图片路径连同记事内容保存在数据库中。在显示图片时，我们创建了一个SpannableString实例，它的内容是 [ox64+图片路径]字符串，我们先显示图片时，实际上只是用图片覆盖了这些字符串，但字符串还是存在的，它作为内容的一部分，所以选择导航栏上的保存按钮，即可将记事的内容保存在数据库中。代码如下：

**if** (NoteEditView.**this**.getContentLength() > 0) {

NoteEditView.**this**.finishEdit();

Bundle bundle = **new** Bundle();

bundle.putInt("NOTE\_ID", NoteEditView.**this**.getModelId());

NoteEditActivity mActvity = (NoteEditActivity) NoteEditView.**this**

.getContext();

mActvity.setResult(Activity.*RESULT\_OK*, mActvity.getIntent()

.putExtras(bundle));

mActvity.finish();

}

### 添加音频

添加音频与添加图片稍微有点类似，在记事编辑页面的导航栏中，有一个audio图标，我们为这个图标绑定点击事件，当发生点击事件时，就会弹出音频的录制框，里面有三个button：“start”“ok”“cancel”，选择“start”button即可开始音频录制，并且这个button的内容会随着音频的不同状态而改变，点击后触发的操作也会随之不同。例如当音频开始录制时，这个button内容变成“stop”，点击即可停止录制，之后button变成“play”，点击即可播放刚才录制的音频。为了实现这个功能，需要为音频设置四个状态，分别是unstart，recording，recorded，playing，通过这四个状态来判断button点击后应该执行哪种操作。

在本次的开发中，对音频的操作涉及到“录制”、“播放”，“暂停”，“保存”，“删除”，这些操作都封装在NoteEditAudioView.java这个类中。

音频的录制主要利用的是android的MediaRecorder类，这个类是专门用来录制音频的，它包含了录制音频基本的方法，如prepare()，start()，stop()，release()，利用这些方法可以实现音频的录制和停止功能。

音频的播放利用的是android的MediaPlayer类，这个类包含的方法跟MediaRecorder类似，它可以用来实现音频的播放和暂停功能。

音频录制的代码如下：

**public** **void** startRecord() {

// **TODO** Auto-generated method stub

**this**.mRecorder = **new** MediaRecorder();

**this**.mRecorder.setAudioSource(MediaRecorder.AudioSource.*MIC*); **this**.mRecorder.setOutputFormat(MediaRecorder.OutputFormat.*THREE\_GPP*);

**this**.mRecorder.setOutputFile(mRecordFile);

**this**.mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.*AMR\_NB*);

**try** {

mRecorder.prepare();

} **catch** (IOException e) {

e.printStackTrace();

}

**this**.mRecorder.start();

}

音频播放的代码如下：

**private** **void** startPlay() {

// **TODO** Auto-generated method stub

**this**.mPlayer = **new** MediaPlayer();

**this**.mPlayer.setOnCompletionListener(**new** OnCompletionListener() {

@Override

**public** **void** onCompletion(MediaPlayer mp) {

NoteEditAudioView.**this**.mAudioBtn

.setText(R.string.*note\_edit\_audio\_play*);

NoteEditAudioView.**this**.stopPlay();

NoteEditAudioView.**this**.status = *STATUS\_RECORDED*;

}

});

**try** {

mPlayer.setDataSource(mRecordFile);

mPlayer.prepare();

mPlayer.start();

} **catch** (IOException e) {

e.printStackTrace();

}

}

音频的保存与图片的保存方式类似，以时间戳为名称保存成3gp格式。与图片不同是在数据库中，音频的路径保存在单独的字段audio中，而图片的路径保存在与文本相同的字段中。

音频的路径为：

**this**.mRecordFile = **this**.getContext()

.getExternalFilesDir(Environment.*DIRECTORY\_MUSIC*)

.getAbsolutePath();

**this**.mRecordFile += "/audio\_" + System.*currentTimeMillis*() + ".3gp";

音频的保存代码如下：

**public** **void** saveAudio(**int** noteId) {

ContentValues cv = **new** ContentValues();

cv.put(NoteItemModel.*AUDIO*, **this**.mRecordFile);

DBHelper dbHelper = DBHelper.*getInstance*(**this**.getContext());

dbHelper.update(DBHelper.TABLE.*NOTES*, cv, NoteItemModel.*ID* + " = "

+ noteId, **null**);

}

### 解析内容

由于记事中可能会含有图片，甚至是多张图片，而图片路径和文字都保存在数据库同一个字段中，所以在显示内容时，需要对内容进行解析，将其中的图片路径替换成图片显示。

对于含有图片的记事内容，我们保存时都是将图片的路径连同记事的文本内容一起保存的，其中图片的路径前面含有”[ox64”字符，后面为”]”字符，解析时就说以此为标识解析的。解析内容的方法为setContent(String content)。

首先，需要先将content显示在记事中，this.mContent.setText(content)。

然后，解析出图片的路径，这里需要一个位置的标识变量index，初值为0。获取”[ox64”字符串在content中从index开始第一次出现的位置index，index+5即为图片路径的开始位置start，然后获取”]”在content中从start开始第一次出现的位置end，为图片路径的结束位置，content.substring(start, end)即可获取图片的路径。

最后，将图片显示在内容中，方法与添加图片的时相同，先把图片编码成bitmap，再利用ImageSpan和SpannableString将图片显示在记事内容中。因为会有多张图片，所以需要一个while循环遍历出所有的图片，每次循环后，index都要加1进入下次循环。代码如下：

**private** **void** setContent(String content) {

**this**.mContent.setText(content);

**int** index = 0;

**while** (content.indexOf("[0x64", index) >= 0) {

index = content.indexOf("[0x64", index);

**int** start = index + 5;

**int** end = content.indexOf("]", start);

File root = **this**.getContext().getExternalFilesDir(

Environment.*DIRECTORY\_PICTURES*);

String filePath = root.getAbsolutePath()

+ content.substring(start, end);

Bitmap bitmap = BitmapFactory.*decodeFile*(filePath);

ImageSpan imageSpan = **new** ImageSpan(**this**.getContext(), bitmap);

SpannableString spannableString = **new** SpannableString(

content.subSequence(index, end + 1));

spannableString.setSpan(imageSpan, 0, spannableString.length(),

Spannable.*SPAN\_EXCLUSIVE\_EXCLUSIVE*);

Editable edit\_text = **this**.mContent.getEditableText();

edit\_text.delete(index, end + 1);

**if** (index < 0 || index >= content.length()) {

edit\_text.append(spannableString);

} **else** {

edit\_text.insert(index, spannableString);

}

index += 1;

}

}

## 闹钟模块

### 设置闹钟

设置闹钟，首先设置闹钟的参数，包括弹出日期，弹出时间，弹出间隔，弹出次数，设置好这些参数后，选择“ok”保存到数据库，这些比较简单，这里就不多说了。

然后需要利用intent调用ClockReceiver.class（将在后面详细介绍），将此记事的id传递给ClockReceiver，以便在闹钟时间到时接收闹钟广播：

Intent intent = **new** Intent(**this**.getContext(), ClockReceiver.**class**); intent.setData(ContentUris.*withAppendedId*(Uri.*parse*(

"content://xue\_yyno te/note"), **this**.mNoteItemModel.getId()));

之后启动系统的AlarmManager进程，为此闹钟添加一个进程：

PendingIntent pendingIntent = PendingIntent.*getBroadcast*(

**this**.getContext(), 0, intent, 0);

AlarmManager am = (AlarmManager) **this**.getContext().getSystemService(

Context.*ALARM\_SERVICE*);

am.set(AlarmManager.*RTC\_WAKEUP*, mClockModel.getTimeInMillis(),

pendingIntent);

最后判断是否为重复闹钟，如果是则利用setRepeating方法设置重复闹钟。

**if** (mClockModel.getAlertTimes() > 1

&& mClockModel.getAlertInterval() > 0) {

am.setRepeating(AlarmManager.*RTC\_WAKEUP*,

mClockModel.getTimeInMillis(),

mClockModel.getAlertIntervalInMillis(), pendingIntent);

}

### 闹钟MODEL

对闹钟参数的设置和获取以及数据库的操作都封装在ClockModel.java类中，这个类在model包中，它的主要作用的从数据库中读取设置闹钟的各种参数，供其他类使用，同时也负责新建闹钟，把用户设置的各项参数保存到数据库。它有两个构造函数：ClockModel()和ClockModel(Cursor cursor)，前者用来新建一个空的clock实例，后者则从数据库中读取闹钟的信息，建一个带有各项数据的clock，供其他类使用。ClockModel()和ClockModel(Cursor cursor)代码如下：

**public** ClockModel() {

**this**.mId = -1;

**this**.mTime = System.*currentTimeMillis*();

**this**.mAlertInterval = 0;

**this**.mAlertTimes = 1;

**this**.mAlertCount = 1;

}

**public** ClockModel(Cursor cursor) {

**this**.mId = cursor.getInt(*ID\_COLUMN*);

**this**.mTime = cursor.getLong(*TIME\_COLUMN*);

**this**.mAlertInterval = cursor.getInt(*ALERT\_INTERVAL\_COLUMN*);

**this**.mAlertTimes = cursor.getInt(*ALERT\_TIMES\_COLUMN*);

**this**.mAlertCount = cursor.getInt(*ALERT\_COUNT\_COLUMN*);

}

更新闹钟数据库利用的是ContentValues类实现的，具体实现方法如下：

**public** ContentValues formatContentValues() {

ContentValues cv = **new** ContentValues();

cv.put(*TIME*, **this**.mTime);

cv.put(*ALERT\_INTERVAL*, **this**.mAlertInterval);

cv.put(*ALERT\_TIMES*, **this**.mAlertTimes);

cv.put(*ALERT\_COUNT*, **this**.mAlertCount);

**return** cv;

}

ClockModel.java中还封装了设置和获取闹钟各项数据信息的方法，例如：

getAlertInterval ()：用来获取闹钟的弹出间隔；

setAlertInterval (int alertInterval)：用来设置闹钟的弹出间隔；

addAlertCount()：用来更新闹钟的已经弹出的次数，闹钟每弹出一次都会调用这个方法来更新数据库，ALERT\_COUNT字段加1；

isClockFinished()：用来判断闹钟是否结束，根据闹钟的弹出次数ALERT\_TIMES以及已经弹出的次数ALERT\_COUNT来判断；

另外还有设置和获取闹钟时间、闹钟弹出次数等信息的方法，由于比较多，在这不一一列举了。

### 日历实现

日历部分的实现主要分为两部分，一部分是农历的实现，一部分是日历视图的实现。

1. **农历的实现**

农历的实现封装在lunar.java类中，主要是利用一份1900--2049的阴历月份的数据实现的，数据如下：

**final** **static** **long**[] *lunarInfo* = **new** **long**[] { 0x04bd8, 0x04ae0, 0x0a570,

0x054d5, 0x0d260, 0x0d950, 0x16554, 0x056a0, 0x09ad0, 0x055d2,

0x04ae0, 0x0a5b6, 0x0a4d0, 0x0d250, 0x1d255, 0x0b540, 0x0d6a0,

...... };

每一项以二进制的格式表示，共有20位，代表某一年的农历信息。1-4位表示当年有无闰年，有的话，为闰月的月份，没有的话，为0。5-16位表示除了闰月外的正常月份是大月还是小月，1为30天，0为29天。17-20位表示表示闰月是大月还是小月，仅当存在闰月的情况下有意义。

例如：1982年的数据是0x0a974(0000 1010 1010 0111 0100)

表示1982年的4月为闰月，即有第二个4月，且是闰小月。从1月到13月的天数依次为：30、29、30、29、29(闰月)、30、29、30、29、29、30、30、30。

利用这份数据，我们可以获取某一年的总天数，某个月的天数以及闰月的天数，具体实现如下：

获取y年的总天数，先按小月29天算，得出日期348，加上大月多出的1天，再加上闰月天数

**final** **private** **static** **int** yearDays(**int** y) {

**int** i, sum = 348;

**for** (i = 0x8000; i > 0x8; i >>= 1) {

**if** ((*lunarInfo*[y - 1900] & i) != 0)

sum += 1;

}

**return** (sum + *leapDays*(y));

}

获取y年闰月的天数

**final** **private** **static** **int** leapDays(**int** y) {

**if** (*leapMonth*(y) != 0) {

**if** ((*lunarInfo*[y - 1900] & 0x10000) != 0)

**return** 30;

**else**

**return** 29;

} **else**

**return** 0;

}

获取y年润哪个月

**final** **private** **static** **int** leapMonth(**int** y) {

**return** (**int**) (*lunarInfo*[y - 1900] & 0xf);

}

获取y年m月的天数

**final** **private** **static** **int** monthDays(**int** y, **int** m) {

**if** ((*lunarInfo*[y - 1900] & (0x10000 >> m)) == 0)

**return 29;**

**else**

**return 30;**

}

Lunar类有一个构造函数lunar(Date cal)，它的作用是求出cal是当月的第几天。具体实现如下：

首先，求出与1900年1月31日相差的天数offset ：

offset = (int) ((cal.getTime() - baseDate.getTime()) / 86400000L)。

然后，用offset减去每农历年的天数，计算出offset是当年的第几天：

**for** (iYear = 1900; iYear < 2050 && offset > 0; iYear++) {

daysOfYear = *yearDays*(iYear);

offset -= daysOfYear;

}

**if** (offset < 0) {

offset += daysOfYear;

iYear--;

}

year = iYear;

然后，调用leapMonth获取润哪个月，用offset逐个减去每月（农历）的天数，求出当天是本月的第几天：

leapMonth = leapMonth(iYear);

**for** (iMonth = 1; iMonth < 13 && offset > 0; iMonth++) {

**if** (leapMonth > 0 && iMonth == (leapMonth + 1) && !leap) {

--iMonth;

leap = **true**;

daysOfMonth = *leapDays*(year);

} **else**

daysOfMonth = *monthDays*(year, iMonth);

offset -= daysOfMonth;

// 解除闰月

**if** (leap && iMonth == (leapMonth + 1))

leap = **false**;

}

最后，offset为0时，并且刚才计算的月份是闰月或者offset小于0时，校正offset：

**if** (offset == 0 && leapMonth > 0 && iMonth == leapMonth + 1) {

**if** (leap) { leap = **false**;

} **else** {leap = **true**;

--iMonth;

}

}

**if** (offset < 0) {offset += daysOfMonth;

--iMonth;

}

1. **视图的实现**

日历视图

### 闹钟提醒

闹钟弹出提醒的实现封装在ClockAlertActivity.java类中，这个类继承activity，是一个activity子类，所以需要在AndroidManifest.xml里面声明：

<activity

android:name=*".activity.ClockAlertActivity"*

android:launchMode=*"singleInstance"*

android:theme=*"@style/Theme.clock"*>

</activity>

另外，闹钟的启动需要用到android的Broadcast机制，它是一种广泛运用的在应用程序之间传输信息的机制，其中BroadcastReceiver是对发送出来的 Broadcast进行过滤接受并响应的一类组件。当闹钟的时间到时，系统就会向应用程序发出广播，所以我们需要建一个继承自BroadcastReceiver的广播接收器，来接收系统发出的广播。广播接收器的代码如下：

**public** **class** ClockReceiver **extends** BroadcastReceiver {

**public** **void** onReceive(Context context, Intent intent) {

intent.setClass(context, ClockAlertActivity.**class**);

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

context.startActivity(intent);

}

}

并且需要在AndroidManifest.xml里面注册声明：

<receiver android:name=*".activity.ClockReceiver"*

android:process=*":remote"*/>

当广播接收器接收到广播时，就会调用ClockAlertActivity。

首先，获取该闹钟对应的记事的id，然后利用该id创建一个note model实例mNoteModel，该实例包含了本记事的所有信息，这样我们就可以获取记事的内容以及与该记事相关的闹钟model，记事的内容主要用于显示在弹框中，闹钟model主要是在闹钟提醒后，调用addAlertCount修改闹钟已经弹出的次数。代码如下：

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

mNoteModel = **new** NoteItemModel(**this**, mNoteId);

mClockModel = **this**.mNoteModel.getClockModel();

mSnippet = **this**.mNoteModel.getContentForItem();

mSnippet = mSnippet.length() > *SNIPPET\_PREW\_MAX\_LEN* ? mSnippet

.substring(0, *SNIPPET\_PREW\_MAX\_LEN*)

+ getResources().getString(R.string.*note\_snippet\_info*)

: mSnippet;

然后，弹出提醒框，同时响铃。响铃的实现方法是playAlarmSound，它利用的是MediaPlayer类，比较简单，在这里不多说。

弹出的提醒框中，包括三个button以及记事的内容，这三个button分别是“go it”“delete”和“I known”，其中“go it”和“delete”绑定了点击事件，选择“go it”会调用NoteEditActivity，进入相应的记事页面，显示完整的记事内容。代码如下：

Intent intent = **new** Intent(**this**, NoteEditActivity.**class**);

intent.setAction(Intent.*ACTION\_VIEW*);

intent.putExtra("ID", mNoteId);

startActivity(intent);

选择“delete”，则调用deleteClock方法，删除该闹钟。删除时首先获取系统的AlarmManager进程，将该进程取消，然后再从数据库中将闹钟删除。代码如下：

AlarmManager am =

(AlarmManager)**this**.getSystemService(Context.*ALARM\_SERVICE*);

am.cancel(pendingIntent);

DBHelper dbHelper = DBHelper.*getInstance*(**this**);

dbHelper.delete(DBHelper.TABLE.*ALERTS*, ClockModel.*ID* + " = "

+ **this**.mNoteModel.getClockId(), **null**);

**this**.mNoteModel.setClockId(-1);

dbHelper.update(DBHelper.TABLE.*NOTES*,

**this**.mNoteModel.formatContentValuesWithoutId(),

NoteItemModel.*ID* + " = " + **this**.mNoteModel.getId(), **null**);

最后，当提醒结束后，弹框消失，响铃停止，判断闹钟是否已经完全结束，如果是，则调用deleteClock删除闹钟，反之，则调用mClockModel.addAlertCount()，将闹钟已经弹出的次数加1。

## 分享模块

分享模块分为分享出去和接收分享，它们的实现利用的都是Intent类，通过设置或者获取intent的action和type，可以区分是何种分享。其中分享出去实现了图文分享和音频分享，而接收分享，现在暂不支持接收音频类的分享。

### 分享出去

1. **图文分享**

图文分享包括纯文本分享、单图文分享、多图文分享，通过设置intent的type和action来区分。

当用户选择了分享时，首先创建一个intent实例，action为ACTION\_SEND；然后判断分享的内容是否含有图片，如果图片为空，则说明是纯文字分享，如果含有一张图片，则说明此分享是单图文分享，如果含有多张图片，则为多图文分享。

对于几种不同分享类型的action和type值的设置如下：

纯文字分享：action为ACTION\_SEND，type为text/plain

单图文分享：action为ACTION\_SEND，type为image/\*

多图文分享：action为ACTION\_SEND\_MULTIPLE，type为images/\*

设置了intent的action和type后，需要为intent添加数据，数据主要分为两种：文本数据和图片数据，而添加数据用的是intent的putExtra方法。this.mNoteItemModel.getContent()获取文本数据，赋值给Intent.EXTRA\_TEXT；this.mNoteItemModel.getImageUri(images.get(0));获取单图片数据，如果是多个图片，则循环调用getImageUri函数获取图片数据，将图片数据赋值给Intent.EXTRA\_STREAM。这里的getContent()和getImageUri()均为noteItemModel类所封装的方法。

最后利用startActivity启动分享，startActivity(Intent.createChooser(intent, "Share"))。

图文分享的代码如下：

**protected** **void** addShare() {

// **TODO** Auto-generated method stub

((InputMethodManager) **this**.getContext().getSystemService(

Context.*INPUT\_METHOD\_SERVICE*)).hideSoftInputFromWindow(

**this**.getWindowToken(), InputMethodManager.*HIDE\_NOT\_ALWAYS*);

Intent intent = **new** Intent(Intent.*ACTION\_SEND*);

ArrayList<String> images = **this**.mNoteItemModel.getImages();

Log.*i*(*TAG*, images.toString());

**if** (images.isEmpty())

intent.setType("text/plain");

**else** **if** (images.size() == 1) {

intent.setType("image/\*");

} **else** **if** (images.size() > 1) {

intent.setAction(Intent.*ACTION\_SEND\_MULTIPLE*);

intent.setType("images/\*");

}

intent.putExtra(Intent.*EXTRA\_TITLE*, "Share my note...");

intent.putExtra(Intent.*EXTRA\_TEXT*, **this**.mNoteItemModel.getContent());

**if** (images.size() == 1) {

Uri uri = **this**.mNoteItemModel.getImageUri(images.get(0));

intent.putExtra(Intent.*EXTRA\_STREAM*, uri);

} **else** **if** (images.size() > 1) {

ArrayList<Uri> uris = **new** ArrayList<Uri>();

**for** (String image : images) {

uris.add(**this**.mNoteItemModel.getImageUri(image));

}

intent.putParcelableArrayListExtra(Intent.*EXTRA\_STREAM*, uris);

}

**this**.getContext().startActivity(Intent.*createChooser*(intent, "Share"));

}

1. **音频分享**

音频分享跟图文分享的原理是一样的，都是利用intent，音频分享的intent action为ACTION\_SEND，type为audio/\*。当一个记事中添加了音频后，就会有一个音频的图片，如果用户长按这个音频时，就可以分享此音频和记事的文本内容。文本数据赋值给Intent.EXTRA\_TEXT,音频数据赋值给Intent.EXTRA\_STREAM。

音频分享的代码如下：

**public** **boolean** onLongClick(View v) {

((InputMethodManager) getContext().getSystemService(

Context.*INPUT\_METHOD\_SERVICE*)).hideSoftInputFromWindow(

getWindowToken(), InputMethodManager.*HIDE\_NOT\_ALWAYS*);

Intent intent = **new** Intent(Intent.*ACTION\_SEND*);

intent.setType("audio/\*");

intent.putExtra(Intent.*EXTRA\_TITLE*, "Share my note...");

intent.putExtra(Intent.*EXTRA\_TEXT*, mNoteItemModel.getContent());

Uri uri = mNoteItemModel.getAudioUri();

intent.putExtra(Intent.*EXTRA\_STREAM*, uri);

getContext().startActivity(

Intent.*createChooser*(intent, "Share audio"));

**return** **false**;

}

### 接收分享

接收分享部分的实现单独作为一个类ShareReceiver，这个类继承自activity，只有在AndroidManifest.xml里面声明后才可以用，除了声明activity外，还需要声明intent-filter，用来对分享的类型进行过滤。其中intent-filter声明需要包含action，data，category。其声明如下：

<activity android:name=*".activity.ShareReceiver"* >

<intent-filter>

<action android:name=*"android.intent.action.SEND"* />

<action android:name=*"android.intent.action.SEND\_MULTIPLE"*/>

<category android:name=*"android.intent.category.DEFAULT"* />

<data android:mimeType=*"text/plain"* />

<data android:mimeType=*"image/\*"* />

</intent-filter>

</activity>

Action和data都有两种类型，分别用来接收文字和图片的分享，其他类型的分享都会被过滤掉。当有分享进来时，就会启动ShareReceiver这个activity，调用onCreate函数，onCreate函数获取传递进来的Intent以及Intent的action和type属性，然后判断Intent的action和type类型，确定是文字分享还是图片分享，再分别调用相应的函数进行处理。onCreate函数如下：

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

Intent intent = getIntent();

String action = intent.getAction();

String type = intent.getType();

**if** (Intent.*ACTION\_SEND*.equals(action) && type != **null**) {

**if** ("text/plain".equals(type)) {

handleSendText(intent);

} **else** **if** (type.startsWith("image/")) {

handleSendImage(intent);

}

} **else** **if** (Intent.*ACTION\_SEND\_MULTIPLE*.equals(action) && type != **null**)

{

**if** (type.startsWith("image/")) {

handleSendMultipleImages(intent);

}

}

}

纯文字分享的处理函数是handleSendText，只含有单张图片的分享处理函数时handleSendImage，含有多个图片的分享处理函数是handleSendMultipleImages，它们的参数都为Intent。每一种分享，都会新建一个记事，调用NoteEditActivity，数据绑定在Bundle实例中传递。由于新建记事有很多不同的情况，所以用ID来区别是哪一种情况，比如ID=-1是用户自己新建的记事，ID=-2是分享进来的文字和单图片新建的记事，ID=-3是分享进来的多图片新建的记事。

处理文字图片分享的函数如下：

**private** **void** handleSendText(Intent intent) {

String sharedText = intent.getStringExtra(Intent.*EXTRA\_TEXT*);

**if** (sharedText != **null**) {

Intent newEditIntent = **new** Intent(**this**, NoteEditActivity.**class**);

Bundle bundle = **new** Bundle();

bundle.putInt("ID", -2);

bundle.putString("CONTENT", sharedText);

newEditIntent.putExtras(bundle);

startActivity(newEditIntent);

}

}

**private** **void** handleSendImage(Intent intent) {

Uri imageUri = (Uri) intent.getParcelableExtra(Intent.*EXTRA\_STREAM*);

**if** (imageUri != **null**) {

intent.setClass(**this**, NoteEditActivity.**class**);

intent.putExtra("ID", -2);

intent.putExtra("CONTENT", intent.getStringExtra(Intent.*EXTRA\_TEXT*));

startActivity(intent);

}

}

**private** **void** handleSendMultipleImages(Intent intent) {

ArrayList<Uri> imageUris = intent

.getParcelableArrayListExtra(Intent.*EXTRA\_STREAM*);

**if** (imageUris != **null**) {

intent.setClass(**this**, NoteEditActivity.**class**);

intent.putExtra("ID", -3);

intent.putExtra("CONTENT", intent.getStringExtra(Intent.*EXTRA\_TEXT*));

startActivity(intent);

}

}

新建记事时，对于分享中的图片处理与图片记事中对图片的处理方式基本相同，都是利用bitmap，调用showBitmapImg函数。

# 数据库实现

在此次的开发中，数据存储利用的是SQLite数据库，为了方便管理以及代码的简洁规范，所有的数据库操作都封装在DBHelper.java类中，包括数据表的创建以及对数据的增删查改等主要操作。另外由于对数据库的操作需要用到SQLite的一些方法，所以在DBHelper类的开头需要引入SQLiteDatabase，SQLiteOpenHelper两个包。

## 数据表创建

此次的记事本应用共有两个数据表：

notes（ID,CLOCK\_ID,CREATE\_DATE,MODIFY\_DATE,CONTENT,AUDIO,SEQUENCE）

alerts（ID,TIME,ALERT\_INTERVAL,ALERT\_TIMES,ALERT\_COUNT）

它们分别用来存储单个记事的信息和闹钟的信息，其创建字符串如下：

notes表的创建字符串：

**private** **static** **final** String *CREATE\_NOTES\_TABLE\_SQL* = "CREATE TABLE "

+ TABLE.*NOTES*

+ " ("

+ NoteItemModel.*ID*

+ " INTEGER PRIMARY KEY,"

+ NoteItemModel.*CLOCK\_ID* + " INTEGER NOT NULL DEFAULT 0,"

+ NoteItemModel.*CREATE\_DATE* + " INTEGER NOT NULL DEFAULT 0,"

+ NoteItemModel.*MODIFY\_DATE* + " INTEGER NOT NULL DEFAULT 0,"

+ NoteItemModel.*CONTENT* + " TEXT　NOT NULL DEFAULT '',"

+ NoteItemModel.*AUDIO* + " TEXT　NOT NULL DEFAULT '',"

+ NoteItemModel.*SEQUENCE* + " INTEGER NOT NULL DEFAULT 0" + ")";

Alerts表的创建字符串：

**private** **static** **final** String *CREATE\_ALERTS\_TABLE\_SQL* = "CREATE TABLE "

+ TABLE.*ALERTS* + " (" + ClockModel.*ID* + " INTEGER PRIMARY KEY,"

+ ClockModel.*TIME* + " INTEGER NOT NULL DEFAULT 0,"

+ ClockModel.*ALERT\_INTERVAL* + " INTEGER　NOT NULL DEFAULT 0,"

+ ClockModel.*ALERT\_TIMES* + " INTEGER　NOT NULL DEFAULT 0,"

+ ClockModel.*ALERT\_COUNT* + " INTEGER　NOT NULL DEFAULT 0" + ")";

## 数据库操作

DAHelper类中封装的基本操作主要有“插入”“删除”“查找”“更新”，各自的封装函数如下：

插入函数：

**public** **long** insert(String tableName, ContentValues values) {

**if** (**this**.db.isOpen() && **this**.db.isReadOnly()) {

db.close();

db = **this**.getWritableDatabase();

}

**long** id = db.insert(tableName, **null**, values);

**return** id;

}

查找函数：

**public** Cursor query(String tableName, String[] columns, String selection,

String[] selectionArgs, String orderBy) {

**if** (**this**.db.isOpen() && !**this**.db.isReadOnly()) {

db.close();

db = **this**.getReadableDatabase();

}

Cursor cursor = db.query(tableName, columns, selection, selectionArgs,

**null**, **null**, orderBy);

**return** cursor;

}

更新函数：

**public** **boolean** update(String tableName, ContentValues values,

String whereClause, String[] whereArgs) {

**if** (**this**.db.isOpen() && **this**.db.isReadOnly())

db = **this**.getWritableDatabase();

**int** rows = db.update(tableName, values, whereClause, whereArgs);

**if** (rows < 0)

**return** **false**;

**return** **true**;

}

删除函数：

**public** **boolean** delete(String tableName, String whereClause,

String[] whereArgs) {

**if** (**this**.db.isOpen() && **this**.db.isReadOnly())

db = **this**.getWritableDatabase();

**int** rows = db.delete(tableName, whereClause, whereArgs);

**if** (rows < 1)

**return** **false**;

**return** **true**;

}