SuperDiary

软件设计文档

The SuperDiary team

赖成锴 廖溢机 刘中喆 杨照功 杨梓阳 朱达辉

https://github.com/SDPCoder/SuperDiary/

目录

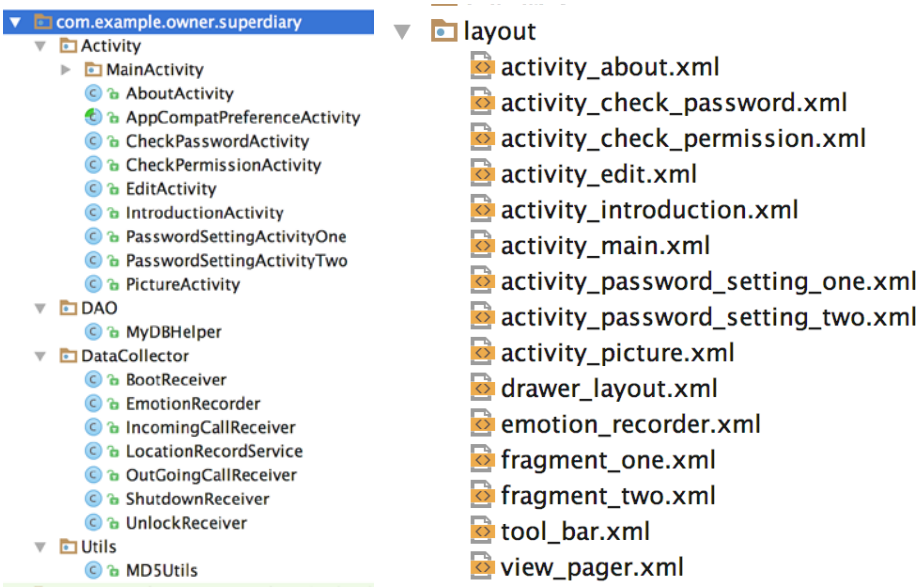
技术选型理由

本项目采用 **Android 应用开发**。Android 系统是世界上覆盖率最高的移动操作系统。比起 iOS 系统，Android 系统更为开放，提供更多的 API。如开机广播、锁屏广播、持续运行的后台服务等，这些 API 是 iOS 所不能提供的，也是我们的 APP 所必须的。

此外由于 Android 系统是开放的，在 Android 系统上开发并不需要提交高额的费用。

模块划分

我们的模块划分为数据库访问模块、页面模块（XML）、页面与后台交互模块（Activity）、数据收集模块、小工具模块。小工具模块的主要内容是 MD5 加密模块。



在开发初期，这个模块的划分并不清晰。主要的分工模式是按 Activity 划分，每个人实现相应的功能。经过一段时间的开发之后，我们发现，Activity 代码中有大量重复实现的功能。我们经过研究得出结论，可以将数据库的访问抽象出来，然后专门由一个人维护。此外，数据的收集也应该由专门的模块进行，其他人只需要调用接口即可。于是我们对软件进行重构，改进了设计方案。如此可以有效降低开发成本，提高代码的复用性与可维护性。

设计技术

1. 结构化编程

首先，如上所述，我们的软件进行了模块划分，这是结构化编程的体现之一，不再赘述。

此外，我们的代码中也体现了结构化编程的思想。结构化编程强调，代码应该自顶向下，我们的代码体现了该思想。

具体如下：

1.1 所有 Activity 中获得 View 对象的代码，都写在 initView()函数中，由 Activity 的 OnCreate 函数调用。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_check_password);

    initView();
    initEvent(getIntent().getExtras());
}
```

上图是 checkPasswordActivity.java 的 20~26 行。

1.2 所有 Activity 中设置监听器的代码，都写在 initEvent(…)函数中，由 Activity 的 OnCreate 函数调用。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_edit);

    initView();
    initEvent(getIntent().getExtras());
}
```

上图是 EditActivity.java 的 24~30 行。

```
private void initEvent(final Bundle bundle) {
    String today = bundle.getString("today");
    FILE_NAME = today + "_diary.txt";
    saveBtn.setOnClickListener((view) → {
            try (FileOutputStream fileOutputStream = openFileOutput(FILE_NAME, MODE_PRIVATE)) {
                String str = editText.getText().toString();
                fileOutputStream.write(str.getBytes());
                Toast.makeText(EditActivity.this, "保存成功", Toast.LENGTH_SHORT).show();
            } catch (IOException ex) {
                Log.e("TAG", "Fail to save file.");
            }
    });

    clearBtn.setOnClickListener((view) → { editText.setText(""); });

    loadText();
}
```

上图是 EditActivity.java 的 32~56 行。

我们还规定了良好的代码规范，这满足了结构化编程中清晰第一的原则。

所有函数以首字母小写的驼峰命名法命名。

```
public void loadText() {
```

（EditActivity.java, Line 64）

面向对象编程

面向对象编程的最基本要求是将由数据和操作数据的方法组成的对象作为编程的基本元素。在本次项目中，我们将数据库抽象为一个数据库对象，然后通过代理类 MyDBHelper 来提供对其的增删查。

```
public class MyDBHelper extends SQLiteOpenHelper {
    private String TABLE_NAME;
    private Map<String, String>  table_head;
    private int columnNum;
    private String[] colNameArray;

    public MyDBHelper(Context context, String name, String table_name, Map<S

    @Override
    public void onCreate(SQLiteDatabase db) {...}

    @Override
    public void onOpen(SQLiteDatabase db) {...}

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)

    public boolean insert(Map<String, Object> record) {...}

    public boolean insertNow(Map<String, Object> record) {...}

    public int deleteAll() {...}

    public List queryAll() {...}

    public List query(Calendar date) {...}

    public List queryToday() { return query(Calendar.getInstance()); }

    private List converToEntry(Cursor cursor) {...}

    public int getSize() {...}
}
```

（MyDBHelper.java）

此外，我们也将 MD5 加密工具抽象成一个对象，对外提供加密函数。输入一个字符串，则可以输出加密后的字符串。

```
// MD5加密
public class MD5Utils {
    public static String encode(String password) {
        try {
            MessageDigest digest = MessageDigest.getInstance("MD5");
            byte[] result = digest.digest(password.getBytes());
            StringBuffer sb = new StringBuffer();
            for (byte b : result) {
                int number = b & 0xff;
                String str = Integer.toHexString(number);
                if (str.length() == 1) {
                    sb.append("0");
                }
                sb.append(str);
            }
            return sb.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

设计模式

1. 数据访问对象模式

数据访问对象模式（Data Access Object Pattern）或 DAO 模式用于把低级的数据访问 API 或操作从高级的业务服务中分离出来。我们的代码中的 MyDBHelper 实现了低级数据访问 API 的抽象。

```
public class MyDBHelper extends SQLiteOpenHelper {
    private String TABLE_NAME;
    private Map<String, String>  table_head;
    private int columnNum;
    private String[] colNameArray;

    public MyDBHelper(Context context, String name, String table_name, Map<String, S

    @Override
    public void onCreate(SQLiteDatabase db) {...}

    @Override
    public void onOpen(SQLiteDatabase db) {...}

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {...}

    public boolean insert(Map<String, Object> record) {...}

    public boolean insertNow(Map<String, Object> record) {...}

    public int deleteAll() {...}

    public List queryAll() {...}

    public List query(Calendar date) {...}

    public List queryToday() {
        return query(Calendar.getInstance());
    }

    private List converToEntry(Cursor cursor) {...}

    public int getSize() {...}
}
```

2. 单例模式

单例模式可避免对个对象造成的语义不明确和代码冲突。Android 框架中有许多单例的类，我们使用了其中一些。

如在 LocationRecordServices 中，我们使用了 GPS 传感器的 API。该 API 返回一个单例的对象，表示手机传感器。

```
private void initStepCountService() {
    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    SensorEventListener sensorEventListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            Toast.makeText(getApplicationContext(), "Walking", Toast.LENGTH_SHORT)
                    .show();
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {

        }
    };

    Sensor stepSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
    sensorManager.registerListener(sensorEventListener, stepSensor, Sensor.TYPE_STEP_COUNTER, SensorManage
```

通过对这些单例 API 的调用，我们对单例模式有了更深入的认识。

技术方法和工具

1. 界面设计：使用了 ViewPager，形成左右滑动切换页面的效果。另外首页还添加了一个 Drawerlayout，实现侧滑菜单功能。

ViewPager 的使用方法：

1）ViewPager 类直接继承了 ViewGroup 类，所有它是一个容器类，可以在其中添加其他的 view 类。

2）ViewPager 类需要一个 PagerAdapter 适配器类给它提供数据。

3）ViewPager 经常和 Fragment 一起使用，并且提供了专门的 FragmentPagerAdapter 和 FragmentStatePagerAdapter 类供 Fragment 中的 ViewPager 使用。

ViewPager 的 Adapter 设置：

```
private class MyAdapter extends FragmentStatePagerAdapter {

    public MyAdapter(FragmentManager fm) { super(fm); }
    fragment1 one = new fragment1();
    fragment2 two = new fragment2();

    @Override
    public Fragment getItem(int position) {
        two.setFragmentOne(one);

        if(0 == position){
            return one;
        }else{
            return two;
        }
    }

    @Override
    public int getCount() { return tabNames.length; }

    @Override
    public CharSequence getPageTitle(int position) { return tabNames[position]; }
}

viewPager.setAdapter(new MyAdapter(getSupportFragmentManager()));
((mData)getApplication()).viewPager = viewPager;
```

Drawerlayout 的使用方法：

DrawerLayout 是 Android support.v4 中的一个抽屉视图控件。使用这个控件，可以生成通过在屏幕上水平滑动打开或者关闭菜单，能给用户一个不错的体验效果。

```
//创建返回键 并实现打开关/闭监听
mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout, toolbar, "打开了", "关闭了") {
    @Override
    public void onDrawerOpened(View drawerView) { super.onDrawerOpened(drawerView); }

    @Override
    public void onDrawerClosed(View drawerView) { super.onDrawerClosed(drawerView); }
};
mDrawerToggle.syncState();
mDrawerLayout.addDrawerListener(mDrawerToggle);
```

2. 自动记录功能实现

记录每天起床时间：

通过**开机广播接收器**记录开机时间，通过之前实现的MyDBHelper写到**数据库**中

前台**服务**通过查数据库获得今日起床时间，显示到**CardView**

```
<receiver
    android:name=".Receivers.BootReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

```
public void onReceive(Context context, Intent intent) {
    Intent intent2 = new Intent(context, LocationRecordService.class);
    Log.i("BootReceiver", "Receive boot broadcast");
    intent2.putExtra("time", System.currentTimeMillis());
    context.startService(intent2);

    Map<String, String> table_header = new HashMap<>();
    MyDBHelper mydb = new MyDBHelper(context, "superdiary_test001.db", "bootTime", table_header,null, 1);
    Map<String, Object> record = new HashMap<>();
    mydb.insertNow(record);
}
```

记录每天睡觉时间：

通过**关机广播接收器**记录关机时间，通过之前实现的MyDBHelper写到**数据库**中

前台查数据库获得昨晚睡觉时间，显示到CardView

（注：起床和睡觉时间采用的是手机开关机时间，可能与实际的起床和睡觉时间略有不符，这里是为了自动获取，故采用此方法）

记录手机使用次数：

通过**解锁广播接收器**记录解锁时间，每次解锁将解锁时间通过之前实现的MyDBHelper写到数据库中

前台查数据库获得今日手机使用次数，显示到CardView

记录联系最多的人：

通过**接收来电广播和去电广播**，将来电去电号码记录到数据库中

前台通过查数据库获得今日联系最多的人，显示到CardView

记录心情：

布局好记录心情的**Widget**，点击Widget就调用**后台服务**的相关接口记录当前时间和心情（量化为数），同时弹出**Notification**提示。下图1是设定OnClickPendingIntent的方法，相当于在Activity中设置OnClickListener。下图2是响应相应的Intent的方法。

```
private static void setOnClickListeners(Context context, RemoteViews views) {
    views.setOnClickPendingIntent(R.id.widget_happy,
            getPendingSelfIntent(context, HAPPY_CLICKED));
    views.setOnClickPendingIntent(R.id.widget_normal,
            getPendingSelfIntent(context, NORMAL_CLICKED));
    views.setOnClickPendingIntent(R.id.widget_sad,
            getPendingSelfIntent(context, SAD_CLICKED));
}
```

```
@Override
public void onReceive(Context context, Intent intent) {
    super.onReceive(context, intent);
    Log.i("EmotionRecorder", intent.getAction());
    if (HAPPY_CLICKED.equals(intent.getAction())) {
        Log.i("EmotionRecorder", "Happy");
        showNotification(context, "心情：快乐记录成功");
        Toast.makeText(context, "心情：快乐\n记录成功", Toast.LENGTH_SHORT).show();


        Map<String, String> table_header = new HashMap<>();
        table_header.put("emotion", "INTEGER");
        MyDBHelper mydb = new MyDBHelper(context, "superdiary_test001.db", "emotion", table_header,null, 1);
        Map<String, Object> record = new HashMap<>();
        record.put("emotion", 100);
        mydb.insertNow(record);
        mydb.close();
    }

    if (NORMAL_CLICKED.equals(intent.getAction())) {
        Log.i("EmotionRecorder", "NORMAL");
        showNotification(context, "心情：普通记录成功");
        Toast.makeText(context, "心情：普通\n记录成功", Toast.LENGTH_SHORT).show();


        Map<String, String> table_header = new HashMap<>();
        table_header.put("emotion", "INTEGER");
        MyDBHelper mydb = new MyDBHelper(context, "superdiary_test001.db", "emotion", table_header,null, 1);
        Map<String, Object> record = new HashMap<>();
        record.put("emotion", 0);
        mydb.insertNow(record);
```

前台查数据库获得今日心情值，即今日所有心情记录的加权均值，显示到CardView

Notification设置：

```
private void showNotification(Context context, String text) {
    Notification.Builder builder = new Notification.Builder(context);
    builder.setContentTitle("超级日记")
            .setContentText(text)
            .setTicker(text)
            .setPriority(Notification.PRIORITY_DEFAULT)
            .setSmallIcon(R.drawable.appicon)
            .setLargeIcon(BitmapFactory.decodeResource(context.getResources(), R.drawable.appicon))
            .setAutoCancel(true)
            .setContentIntent(PendingIntent.getActivity(context, 0, new Intent(), 0))
            .setWhen(System.currentTimeMillis());
    Notification notify = builder.build();
    NotificationManager manager =
            (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
    manager.notify(0, notify);
}
```

记录逗留地点：

后台服务维护一个今日曾逗留地点的数据表，表头为（经纬度，逗留分钟数）。位置变化在100m范围内算同一个位置。每1min从**GPS传感器**获取一次位置，若当前位置是新位置，则插入新纪录。则若位置未改变，则当前位置的逗留时间增加1min。然后将逗留地点显示在地图中。使用了百度地图时用到了**网络访问**相关知识。

```
private void initLocService() {
    mLocationmanger = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);
    if (mLocationmanger.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
        provider = mLocationmanger.getProvider(LocationManager.GPS_PROVIDER);
    } else if (mLocationmanger.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
        provider = mLocationmanger.getProvider(LocationManager.NETWORK_PROVIDER);
    } else {
        Toast.makeText(LocationRecordService.this, "Error: No location provider enabled!", Toast.LENGTH_S
    }
```

```
private double distance(Location loc1, Location loc2) {
    double long1, lat1, long2, lat2; // t:经度 x:维度
    long1 = loc1.getLongitude();
    lat1 = loc1.getLatitude();
    long2 = loc2.getLongitude();
    lat2 = loc2.getLatitude();

    double a, b, R;
    R = 6378137; // 地球半径
    lat1 = lat1 * Math.PI / 180.0;
    lat2 = lat2 * Math.PI / 180.0;
    a = lat1 - lat2;
    b = (long1 - long2) * Math.PI / 180.0;
    double d;
    double sa2, sb2;
    sa2 = Math.sin(a / 2.0);
    sb2 = Math.sin(b / 2.0);
    d = 2
            * R
            * Math.asin(Math.sqrt(sa2 * sa2 + Math.cos(lat1)
            * Math.cos(lat2) * sb2 * sb2));
    return d;
}
```

3. 数据库实现

MyDBHelper类

数据库使用**SQLite**，并使用了**ContentProvider**的知识。该类与数据库的某个表绑定。与之前实验中的实现不同的是，这里的实现支持自定义表头。这是通过在初始化实例的时候传入一个存储键值对(列名，列类型)的Map容器来实现的。插入的记录也相应地根据自定义的表头来确定格式，同样通过Map容器以(列名，值)的形式保存记录，传入该类的insert方法。

下图是为记录来电和去电的数据表绑定MyDBHelper.

```
Map<String, String> table_header = new HashMap<>();
table_header.put("type", "TEXT");
table_header.put("phoneNum", "TEXT");
MyDBHelper mydb = new MyDBHelper(context, "superdiary_test001.db", "calls", table_header,null, 1);
Map<String, Object> record = new HashMap<>();
record.put("phoneNum", number);
record.put("type", "out");
mydb.insertNow(record);
mydb.close();
```

查询数据库时，该类会将所得Cursor变量转化为元素为以Map容器的形式保存的record的List容器再返回，而之前lab中的实现则是用固定的类的对象装。这里的实现更加灵活。下图是具体实现。

```
private List converToEntry(Cursor cursor) {
    List<Map<String, Object>> list = new ArrayList<>();
    if (cursor.moveToFirst()) {
        do {
            Map<String, Object> record = new HashMap<>();
            for (int i = 0; i < columnNum; i++) {
                String colName = colNameArray[i];
                switch (table_head.get(colName)) {
                    case "TEXT":
                        record.put(colName, cursor.getString(cursor.getColumnIndex(colName)));
                        break;
                    case "INTEGER":
                        record.put(colName, cursor.getInt(cursor.getColumnIndex(colName)));
                        break;
                    case "REAL":
                        record.put(colName, cursor.getDouble(cursor.getColumnIndex(colName)));
                        break;
                }
            }

            list.add(record);
        } while (cursor.moveToNext());
    }
    return list;
}
```

4. 历史日记功能实现

采用了**CalendarView**组件。从CalendarView中点击某一天会跳转到当天的日记界面，并且直接从数据库中相关的表中调出当天的信息，显示在CardView中。日历点击事件监听：

```
calendarView.setOnDateChangedListener((widget, selectedDate, selected) -> {
        Calendar date = selectedDate.getCalendar();
        one.setDate(date);
        one.loadData(date);

        mData sharedData = ((mData)getActivity().getApplication());
        ViewPager vp = sharedData.viewPager;
        vp.setCurrentItem(vp.getCurrentItem() - 1);
});
```

5. 动画按钮实现

有两处使用了按钮的动画效果：滑动菜单左上角的按钮和右下角的按钮。右下角的按钮使用 FloatingActionButton 组件实现。动画效果的实现原理是使用**Animation**。

Android 系统提供了很多丰富的 API 去实现 UI 的 2D 与 3D 动画，最主要的划分可以分为如下几类：

- **View Animation：**视图动画在古老的 Android 版本系统中就已经提供了，只能被用来设置 View 的动画。
- **Drawable Animation：** 这种动画（也叫 Frame 动画、帧动画）其实可以划分到视图动画的类别，专门用来一个一个的显示 Drawable 的 resources，就像放幻灯片一样。
- **Property Animation：** 属性动画只对 Android 3.0（API 11）以上版本的 Android 系统才有效，这种动画可以设置给任何 Object，包括那些还没有渲染到屏幕上的对象。这种动画是可扩展的，可以让你自定义任何类型和属性的动画。

6. 文字记录功能实现

使用到了**文件读写**的知识。将每天记录的文字使用 **Internal Storage** 保存在文件中。保存在 Internal Storage 上的文件，只有这个应用自己可以看到和使用，其他应用和用户本身无法访问这些文件。

写文件：

```
saveBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        try (FileOutputStream fileOutputStream = openFileOutput(FILE_NAME, MODE_PRIVATE)) {
            String str = editText.getText().toString();
            fileOutputStream.write(str.getBytes());
            Toast.makeText(EditActivity.this, "保存成功", Toast.LENGTH_SHORT).show();
        } catch (IOException ex) {
            Log.e("TAG", "Fail to save file.");
        }
    }
});
```

读文件：

```
public void load() {
    try (FileInputStream fileInputStream = openFileInput(FILE_NAME)) {
        byte[] contents = new byte[fileInputStream.available()];
        fileInputStream.read(contents);
        editText.setText(new String(contents));
        editText.setSelection(editText.getText().length());
    } catch (IOException ex) {
        Log.e("TAG", "Fail to load file.");
    }
}
```

7. 图片记录功能实现

拍照记录功能：

首先要开启拍照功能权限，然后调用系统摄像头进行**拍照（使用 MediaStore.ACTION_IMAGE_CAPTURE）**，并编写图片裁剪函数对图片进行**裁剪**，最后使用 setImageBitmap 把图片显示到 ImgeView 中。另外，使用 **External Storage** 方式将拍照并裁剪后的图片**保存到本地**。

拍照：

```
//将File对象转换为Uri并启动拍照程序
imageUri = Uri.fromFile(outputImage);
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  //跳转至系统拍照界面
intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);           //指定图片输出地址
startActivityForResult(intent,REQUEST_TAKE_PHOTO);            //启动照相
```

裁剪图片：

```
// 裁剪照片
public void startPhotoZoom(Uri uri, double width, double height) {
    Intent intent = new Intent("com.android.camera.action.CROP");
    intent.setDataAndType(uri, "image/*");
    // 设置裁剪
    intent.putExtra("crop", "true");
    intent.putExtra("scale", true);// 去黑边
    // aspectX aspectY 是宽高的比例
    intent.putExtra("aspectX", width/height);
    intent.putExtra("aspectY", 1);
    // outputX outputY 是裁剪图片宽高
    intent.putExtra("outputX", width);
    intent.putExtra("outputY", height);
    // 图片格式
    intent.putExtra("outputFormat", Bitmap.CompressFormat.JPEG.toString());
    intent.putExtra("noFaceDetection", true); // 取消人脸识别
    intent.putExtra("return-data", true); // true:返回uri, false: 不返回uri
    // 同一个地址下 裁剪的图片覆盖之前得到的图片
    intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
    startActivityForResult(intent, REQUEST_CROP_PHOTO);
}
```

保存图片：

```
//图片名称，以时间命名
filename = new SimpleDateFormat("yyyyMMddHHmmss").format(new Date());
//存储至DCIM文件夹
File path = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DCIM);
outputImage = new File(path, filename + ".jpg");
try {
    if(outputImage.exists()) {
        outputImage.delete();
    }
    outputImage.createNewFile();
} catch(IOException e) {
    e.printStackTrace();
}
```

```
FileOutputStream fileOutputStream = null;
try {
    fileOutputStream = new FileOutputStream(file);
    if (bitmap != null) {
        if (bitmap.compress(Bitmap.CompressFormat.JPEG, 100,
                fileOutputStream)) {
            fileOutputStream.flush();
        }
    }
} catch (FileNotFoundException e) {
    file.delete();
    e.printStackTrace();
} catch (IOException e) {
    file.delete();
    e.printStackTrace();
```

本地选择图片上传：

使用 Intent.ACTION_PICK，并设置类型为"image/*"。然后同样调用图片裁剪函数对图片进行裁剪并显示出来，然后使用 External Storage 方式将裁剪后的图片保存到本地。

```
Intent intent = new Intent(Intent.ACTION_PICK);
intent.setType("image/*");
startActivityForResult(intent, REQUEST_SELECT_PHOTO);
```

各个功能的调用流程处理：使用 startActivityForResult 启动功能，并在 onActivityResult 中根据 requestCode 进行对应的处理。

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == PictureActivity.RESULT_OK) {
        switch (requestCode) {
            case REQUEST_TAKE_PHOTO:       //拍照
                startPhotoZoom(imageUri, 300, 250);
                break;
            case REQUEST_SELECT_PHOTO:     //选择本地图片
                startPhotoZoom(data.getData(), 300, 250);
                break;
            case REQUEST_CROP_PHOTO:       //裁剪图片
                Bundle extras = data.getExtras();
                if (extras != null) {
                    Bitmap photo = extras.getParcelable("data");
                    //把图片显示到ImgeView
                    showImage.setImageBitmap(photo);
                    //发送广播更新图库
                    refreshAlbum();
                    saveImage(photo, outputImage);
                }
                break;
        }
    }
}
```

图片持久化的实现：

这里使用 SharedPreferences 来保存。使用 SharedPreferences 保存图片时，需要用 Base64 对 Bitmap 和 String 进行互相转换。

```
private void getBitmapFromSharedPreferences(){
    //第一步:取出字符串形式的Bitmap
    String imageString = sharedPreferences.getString(today, null);
    if (imageString == null) return;
    //第二步:利用Base64将字符串转换为ByteArrayInputStream
    byte[] byteArray = Base64.decode(imageString, Base64.DEFAULT);
    ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(byteArray);
    //第三步:利用ByteArrayInputStream生成Bitmap
    Bitmap bitmap = BitmapFactory.decodeStream(byteArrayInputStream);
    showImage.setImageBitmap(bitmap);
}
```

```
private void saveBitmapToSharedPreferences(Bitmap bitmap){
    //第一步:将Bitmap压缩至字节数组输出流ByteArrayOutputStream
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.JPEG, 100, byteArrayOutputStream);
    //第二步:利用Base64将字节数组输出流中的数据转换成字符串String
    byte[] byteArray = byteArrayOutputStream.toByteArray();
    String imageString = new String(Base64.encodeToString(byteArray, Base64.DEFAULT));
    //第三步:将String保持至SharedPreferences
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString(today, imageString);
    editor.commit();
}
```

8. 密码保护功能实现

设置密码时，密码经过 MD5 哈希加密后，将哈希值存入 SharedPreferences 中。之后打开日记时，需要先输入密码，将输入的密码的 MD5 哈希值与 SharedPreferences 中保存的哈希值进行比较，若相同则视为验证成功，否则验证失败。

设置密码代码实现：

```
final EditText newPwd = (EditText) findViewById(R.id.newPwd);
final EditText confirmPwd = (EditText) findViewById(R.id.confirmPwd);
final SharedPreferences sharedPreferences = getSharedPreferences("DiaryPwd", MODE_PRIVATE);
final String today = getIntent().getExtras().getString("today");

okBtn.setOnClickListener((view) → {
        String newPwdStr = newPwd.getText().toString();
        String confirmPwdStr = confirmPwd.getText().toString();
        if (TextUtils.isEmpty(newPwdStr)) {
            Toast.makeText(PasswordSettingActivityOne.this, "密码不能为空", Toast.LENGTH_SHORT).show();
        } else {
            if (newPwdStr.equals(confirmPwdStr)) {
                SharedPreferences.Editor editor = sharedPreferences.edit();
                editor.putString(today, MD5Utils.encode(newPwdStr));
                editor.commit();
                Toast.makeText(PasswordSettingActivityOne.this, "密码设置成功", Toast.LENGTH_SHORT).show();
                finish();
            } else {
                Toast.makeText(PasswordSettingActivityOne.this, "两次输入的密码不同，请重新输入", Toast.LENGTH_S
            }
        }
});
```

使用到的技术点总结如下：

1. 界面框架设计（用户友好，控件布局合理，美观）

2. 事件处理的实现（点击事件、activity 跳转等）

3. SQLite、ContentProvider、SharedPreferences 等数据存储方式

4. Notification、Widget 等显示方式

5. Broadcast

6. Service

7. 网络访问

8. 传感器使用

9. 动画应用

10. 文件访问和存储（包括内部存储和外部存储）

11. Viewpager、Drawerlayout、CardView、CalendarView 等组件的使用

12. 拍照功能应用

13. 图片裁剪功能