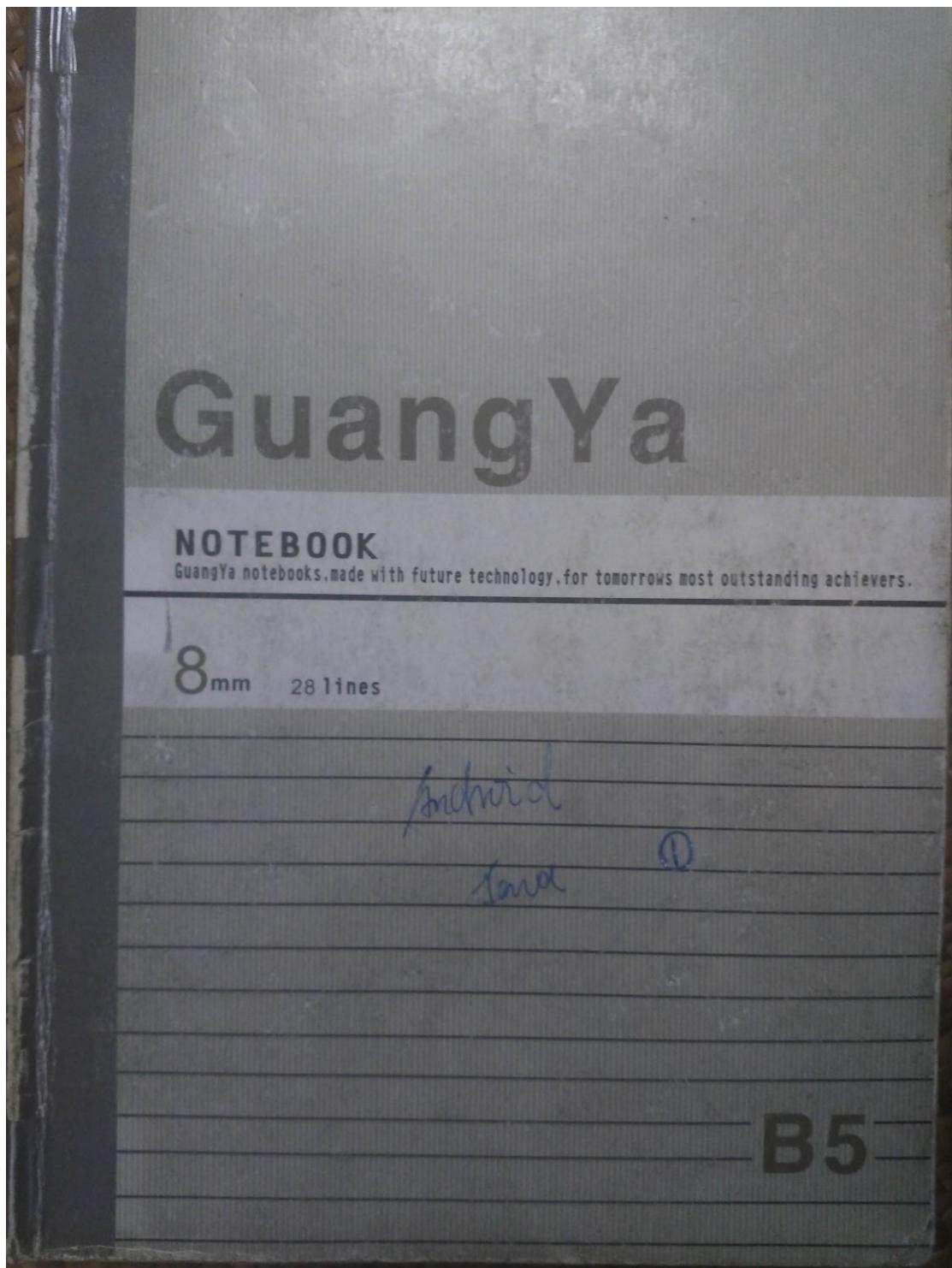


NoteBook-1



目录

101.Android 网络请求选择.....	3
102.MVC 在 Android 中的应用	4
104.自学 Android 到什么程度才找到工作.....	6
105.RecyclerView 的简单实用	7
107.ViewPager 的基本使用	9
108.ListView 的基本使用	10
111.Toolbar 的基本使用.....	13
112.属性动画的基本使用.....	14
123.Retrofit2 基本使用	17
133.第一行代码 第一章 Android 框架 笔记.....	24
134. 第一行代码 第二章 Activity 笔记.....	25
138.第一行代码 第三章 UI 笔记	28
141.第一行代码 第四章 Fragment 笔记	29
143.第一行代码 第五章 Broadcast Receiver 笔记.....	31
145.Java 基础.....	33
146.Java 常用类.....	35
150.第一行代码 第六章 持久化 File 笔记	38
151.第一行代码 第六章 持久化 SharePreferences 笔记.....	39
152.面相对象基本概念.....	40
155.Java 内部类.....	43
157.第一行代码 第六章 持久化 SQLite 笔记	45
159.Java List Map Set	47
161.第一行代码 第六章 持久化 LitePal 笔记	49
163.第一行代码 第八章 多媒体 Notification 笔记.....	51
165.第一行代码 第八章 多媒体 Camera 笔记	53
167.第一行代码 第八章 多媒体 选择图片 笔记.....	55
170.Java 异常.....	58
173.第一行代码 第八章 多媒体 音频视频 笔记.....	61
174.Java 多线程.....	62
176.Java 多线程 同步.....	64
179.Android Java HttpURLConnection	67

101. Android 网络请求选择

No.

Date

Android 网络请求

1. 原则：单一职责原则。专注：Volley、OkHttp、Retrofit

2. OkHttp：Square 公司开源的针对 Java 和 Android，封装的一个高性能 http 请求库。
对 HttpURLConnection 封装，支持 glog、http 2.0、websocket。
支持同步、异步。
封装了线程池、数据转换、参数使用、错误处理。

3. Volley：Google 的一套小而巧的异步请求库。

支持 HttpClient (已废弃)、HttpURLConnection、OkHttp、ImageLoader。
不支持 post 大数据、不适合上传文件。

4. Retrofit：Square 基于 OkHttp 封装的一套 RESTful 网络请求框架。

涉及一堆设计模式、注解、Json Converter 序列化数据。

支持 RxJava

Retrofit + OkHttp + RxJava + Dagger 2：

5. Volley VS OkHttp

VS

OkHttp

Volley：封装更好

支持 OkHttp

首选

OkHttp：需再次封装

基于 NIO 和 Okio

性能更好。

6. OkHttp VS Retrofit

Retrofit 基于 OkHttp，首选 Retrofit。

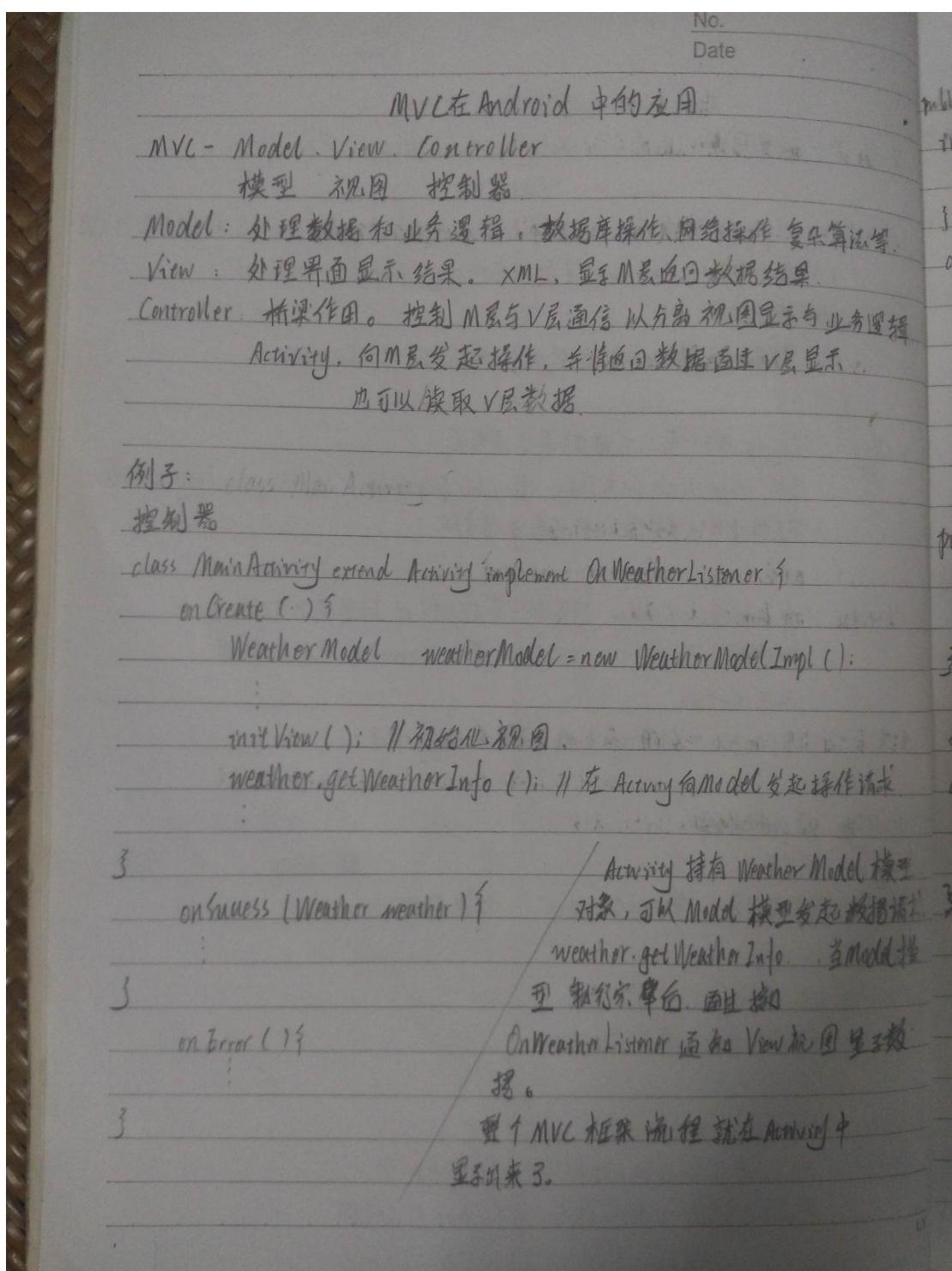
7. Volley VS Retrofit

Retrofit 默认使用 OkHttp，性能上比 Volley 有优势。

解耦更彻底，职责细分，与 RxJava 配合使用。

Retrofit > Volley > OkHttp

102. MVC 在 Android 中的应用



No.

Date

public

interface WeatherModel {

void getWeatherInfo();

}

class WeatherModelWithOkHttpImpl implements WeatherModel {

//重写接口的getWeatherInfo方法。

@Override

public void getWeatherInfo() {

}

public interface OnWeatherListener {

void onSuccess(Weather weather);

void onError();

}

如果以后把Okhttp改用Volley, 则新建

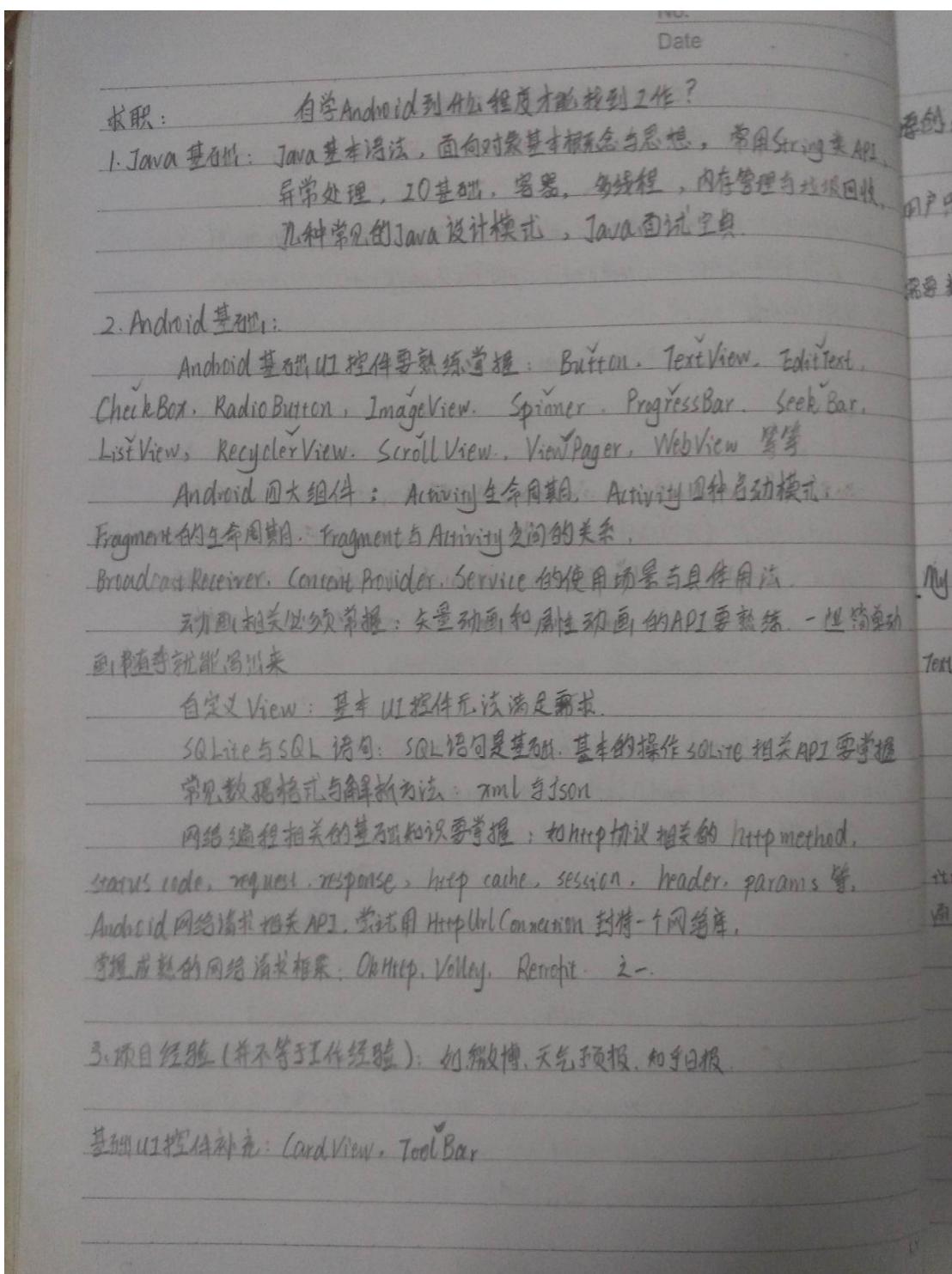
class WeatherModelWithVolleyImpl implements WeatherModel {

//重写getWeatherInfo方法

3.

3 LY

104. 自学 Android 到什么程度才找到工作



105.RecyclerView 的简单实用

原创： RecyclerView 使用方法：

1. RecyclerView 是 support-v7 的新组件，把 ViewHolder 的实现封装起来。
用户只要实现自己的 ViewHolder。

2. RecyclerView 的使用方法与 ListView 基本类似，需要 List item 的布局文件。
需要数据适配器 Adapter。重点在于对 Adapter 的代码编写。

3. 自定义 Adapter：

```
public class MyRecyclerViewAdapter extends RecyclerView.Adapter<MyViewHolder> {  
    MyViewHolder 是自定义的 ViewHolder;  
    MyRecyclerViewAdapter 的构造函数，一般需要传递上文的数据类 List<ObjectItem>;  
    MyRecyclerViewAdapter 需要重写父类的 3 个方法：  
    onCreateViewHolder：在这里做的是关联 Listitem 的布局，然后创建自定义  
    MyViewHolder 的对象，把 ViewHolder 与 Listitem 布局关联起来，并返回 ViewHolder。  
    onBindViewHolder：通过 ViewHolder 对象，关联视图数据，例如修改  
    TextView 的值，显示图片等等。  
    getItemCount：返回数据项的大小。
```

4. 定义 ViewHolder，通常在 Adapter 内部创建这个类。

```
public class MyViewHolder extends RecyclerView.ViewHolder {  
    重写构造方法 public MyViewHolder (View itemView) : super(itemView);  
    itemView 就是 Listitem 的布局，在这个构造方法中，找到各个控件，也就是  
    通过 findViewById 找到各控件。
```

5. 在 Activity 使用 recyclerView。

设置布局方式： recyclerView.setLayoutManager (new LinearLayoutManager (content));
或 .setLayoutManager (new GridLayoutManager (content, 3));

设置动画： .setItemAnimator (new DefaultItemAnimator ());

绑定 Adapter： .setAdapter (recyclerViewAdapter);

6.至此，RecyclerView 就能工作了。

7. 为 RecyclerView 的 Item 添加点击和长按的监听，实现方法有两种：
一：通过 recyclerView.addOnItemTouchListener 去监听然后去判断操作。
二：在 Adapter 中提供回调，下面介绍这种方法。

8. 为 RecyclerView 的 Item 添加事件监听：

在 MyRecyclerViewAdapter 内部创建接口。

public interface OnItemClickListener { 在这个接口中有两个方法。 }

 void onItemClick (View view, int position);

 void onItemLongClick (View view, int position);

在 MyRecyclerViewAdapter 中创建方法 setOnItemClickListener，提供给

Activity 中的 adapter 对象调用。

public void setOnItemClickListener (OnItemClickListener listener)

 MyRecyclerViewAdapter.this.itemClickListener = listener;

在 onBindViewHolder 方法中，为 itemView 设置监听，代码如下：

if (itemClickListener != null) {

 viewHolder.itemView.setOnClickListener (new OnClickListener (View view)

 itemClickListener.onItemClick (viewHolder.itemView, position));

};

 viewHolder.itemView.setOnLongClickListener (new OnLongClickListener () {

 itemClickListener.onItemLongClick (viewHolder.itemView, position));

});

} 然后为 Adapter 对象设置监听：

myRecyclerViewAdapter.setOnItemClickListener (new OnItemClickListener () {

 @Override

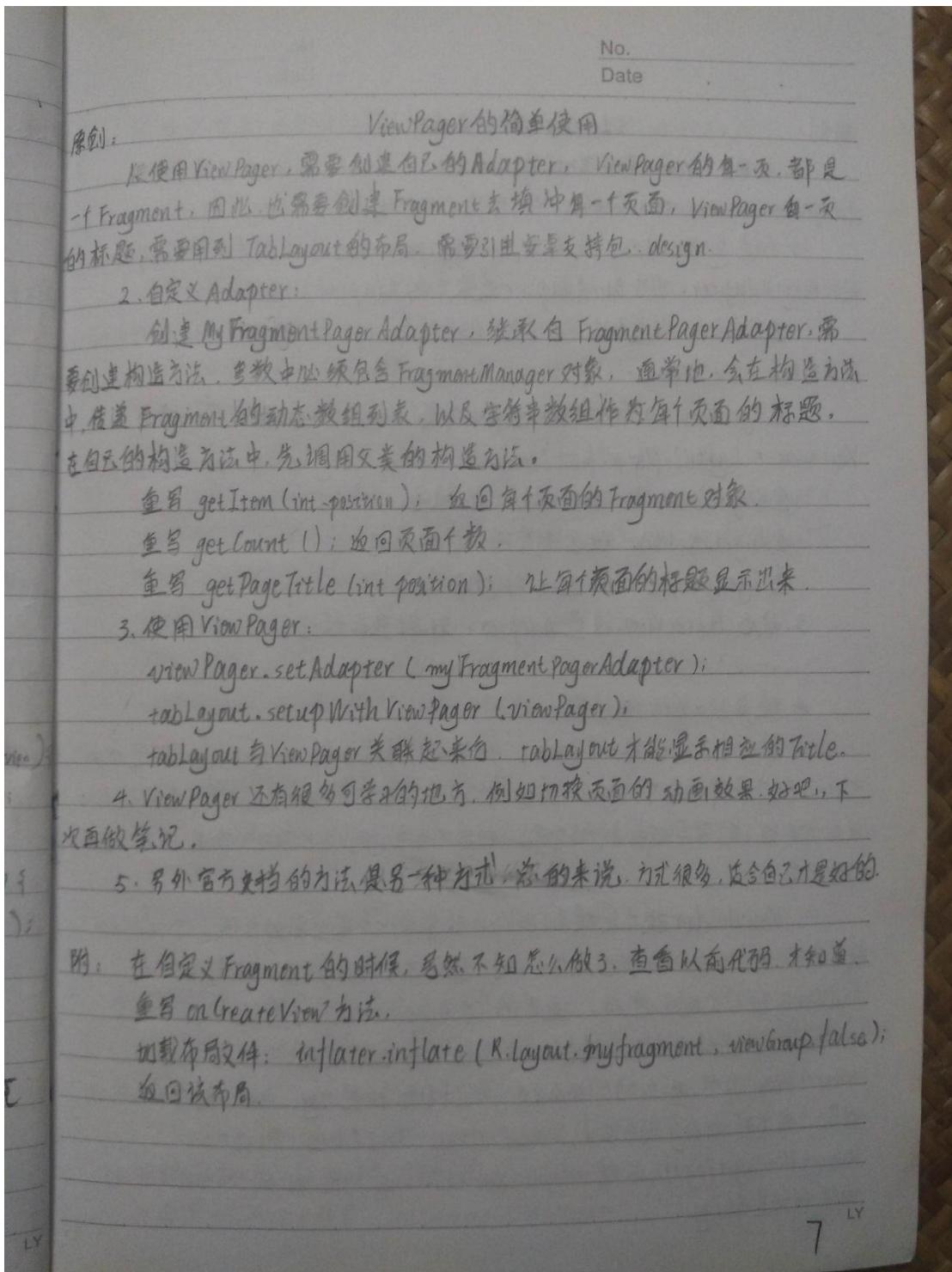
 public void onItemClick (View view, int position) { ... }.

 @Override

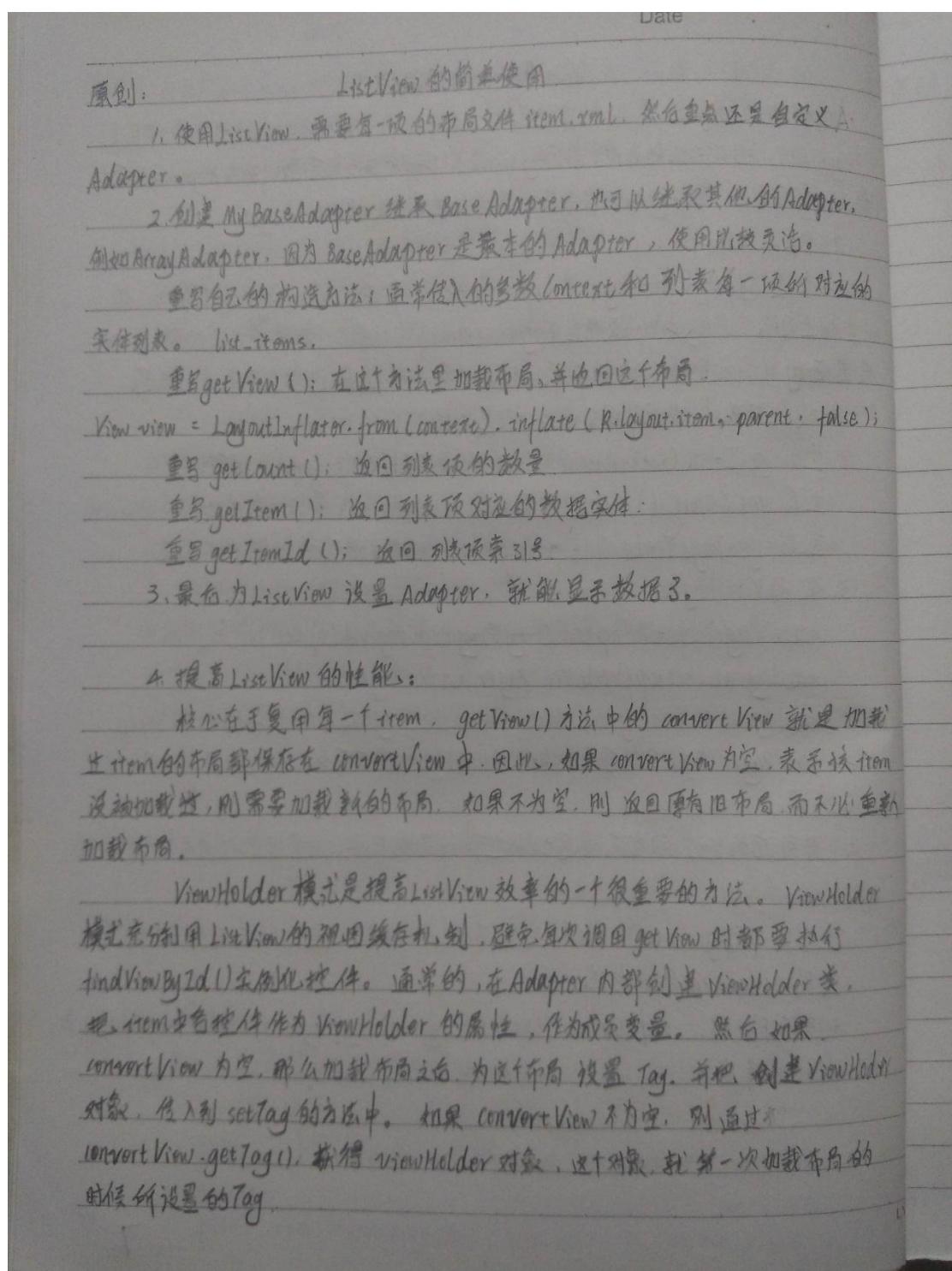
 public void onItemLongClick (View view, int position) { ... }.

};

107.ViewPager 的基本使用



108.ListView 的基本使用



比较完整的 getView() 方法的代码框架如下：

Override

```
public View getView(int position, View convertView, ViewGroup parent) {
    User user = list_user_items.get(position);
    View view;
    ViewHolder viewHolder;
    if (convertView == null) {
        view = LayoutInflater.from(context).inflate(...);
        viewHolder = new ViewHolder();
        viewHolder.iv_head = (ImageView) view.findViewById(...);
        viewHolder.tv_name = (TextView) view.findViewById(...);
        view.setTag(viewHolder);
    } else {
        view = convertView;
        viewHolder = (ViewHolder) view.getTag();
    }
    // 设置布局中控件要显示的数据 (user.getName());
    viewHolder.tv_name.setText(user.getName());
}
```

5. 关于 ListView 的其他使用技巧。

① 设置项目间的分割线：divider = "@color/white"，可设置颜色或图片资源。

dividerHeight = "10dp"

divider = "@null" 设为透明。

② 隐藏 ListView 的滚动条：scrollbars = "none"

③ 取消 Item 的点击效果：listSelector = "#00000000"

No.

Date

④ 设置 ListView 显示在第几项： 原创

listView.setSelection(position);

⑤ ListView 的平滑移动：（现在还不是很清楚）以后再详细解释 可定制

listView.smoothScrollBy(distance, duration);

listView.smoothScrollByOffset(offset);

listView.smoothScrollToPosition(position);

⑥ 动态增加 Item： list_user.add(userItem);

adapter.notifyDataSetChanged();

⑦ ListView 获取子 View：

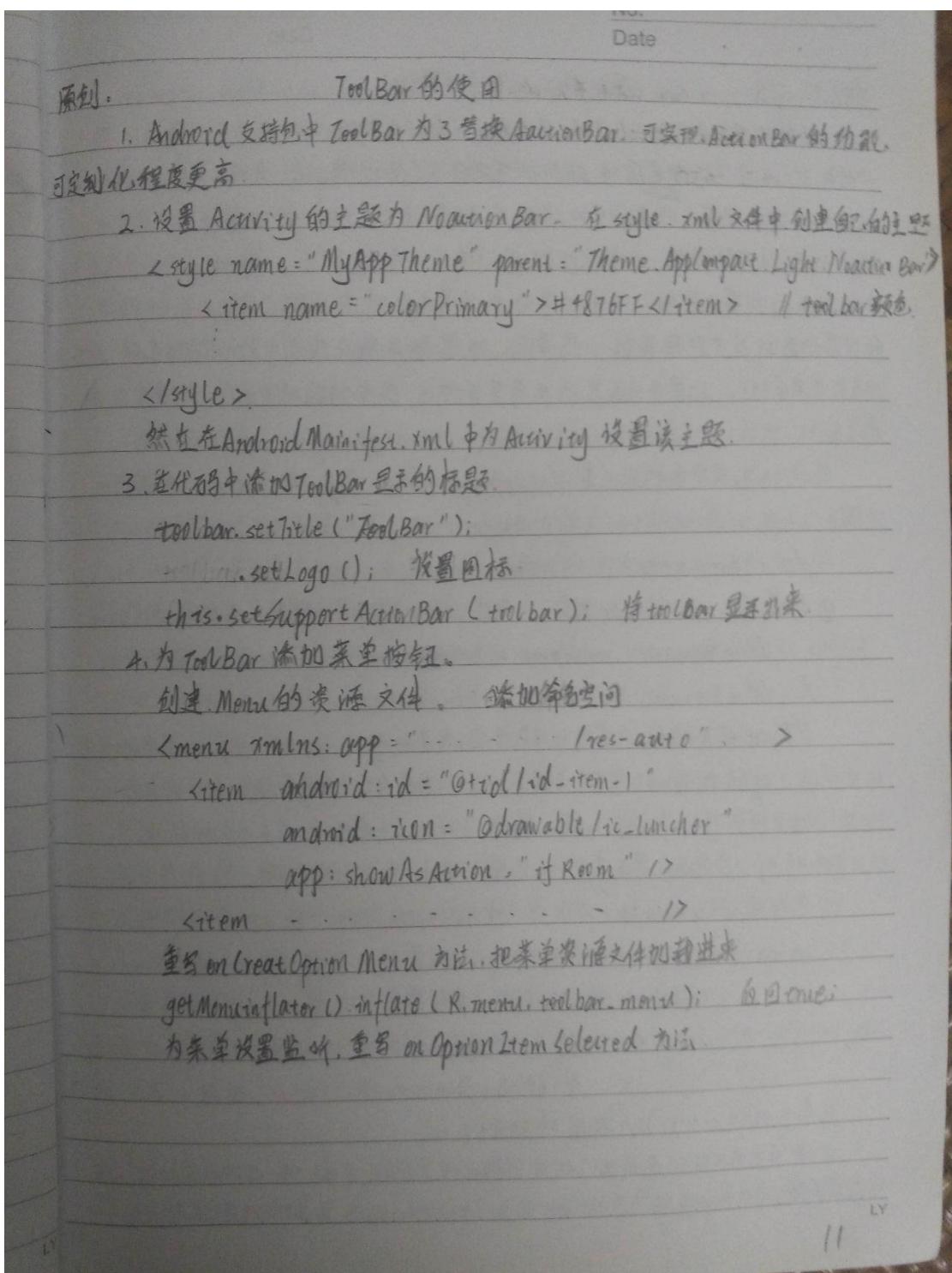
View view = listView.getChildAt(i); 获得第 i 项的 View。

⑧ 处理空的 ListView：

listView.setEmptyView(iv_no_data);

Listview 在空数据时显示了一张默认的图片，用来提示用户而在有数据时，则不会显示。这个布局通常在 ListView 所在的 XML 文件中定义。

111. Toolbar 的基本使用



112. 属性动画的基本使用

No.

Date

原句：属性动画的基本使用

动画能带来良好的用户体验，作为Android程序员，必须掌握最基本的动画的使用方法，这里总结的是属性动画的基本用法，包括透明度、位移、旋转、缩放，以及各种动画的集气效果。属性动画是Android 3.0 推出的动画框架，能实现视图动画所有的效果，并且完成视图动画不具备的功能。例如，一个原来在左上角的Button设置了监听，使用视图动画将其平移到右上角，当你点击他的时候，所设置的监听是不起作用了。简单说，视图动画仅仅改变了视图的绘制，并没有改变其属性。如果使用属性动画将其平移后，原来的监听还是有的，是真正的移动。

属性动画最重要的两个类 ValueAnimator 和 ObjectAnimator，下面逐一讲解。

一. ValueAnimator

① 创建 ValueAnimator 对象。通过 ValueAnimator 的静态方法创建该对象。

ValueAnimator animator = ValueAnimator.ofFloat(1f, 3f);

或 ValueAnimator animator = ValueAnimator.ofInt(1, 3);

ofFloat 传入的值的类型为浮点类型，表示该值从 1f 变化到 3f，1f 是初始值，3f 是终值。ofFloat 传入的是可变参数，就是可以往多个参数，表示值的变化过程。如 ofFloat(1f, 3f, 2f, 5f)；表示该值从 1f 逐步变化到 3f，再逐步变化到 2f，最终变化到 5f。这里逐步并不是从 1f 直接变到 5f，其间变化值很多。

② 然后用 ValueAnimator 对象的各种方法对动画进行定义：如

animator.setDuration(3000); // 设置动画持续时间为 3 秒

.setRepeatCount(2); // 设置重复次数 2

.setRepeatMode(ValueAnimator.RESTART); // 重复模式（restart 和

或 ValueAnimator.REVERSE）；例如

③ 最后调用 start() 方法播放动画。

也许，这里看不出动画效果，但是动画所设置的值是变化的，可以对 ValueAnimator 设置监听器，看其变化过程。

animator.addUpdateListener (new AnimatorUpdateListener () { ... });

float value = valueAnimator.getAnimatorValue ();

现在可以在日志中观察到该值的变化了。

2. Object Animator

Object Animator 继承了 Value Animator，所以也可以用父类的方法对动画进行一些通用的设置。不同的是创建 Object Animator 对象。

// 透明度动画

alphaAnimator = ObjectAnimator.ofFloat (view, "alpha", 1f, 0f, 1f);

view：为动画操作对象。

"alpha"：值变化的属性名，即 view 对象必须有 getAlpha 和 setAlpha 方法。

1f, 0f, 1f；表示变化过程，不透明 → 全透明 → 不透明。

// 旋转动画：旋转 360 度。

rotateAnimator = ObjectAnimator.ofFloat (view, "rotation", 0f, 360f);

// 水平移动：

translationXAnimator = ObjectAnimator.ofFloat (view, "translationX",

curTranslationX, -500f, curTranslationX, 500f, curTranslationX);

float curTranslationX = view.getTranslationX (); // 按件原来的 X 值。

// 垂直移动，类似水平移动，改变的属性名为 translationY

// 水平缩放：放大 3 倍再还原。

scaleAnimator = ObjectAnimator.ofFloat (view, "scaleX", 1f, 3f, 1f);

// 垂直缩放，类似水平缩放，改变的属性名为 scaleY

原理：属性动画作用于任何对象，变化的属性名必须有该对象中重写对应的 get 和 set 方法。

三、组合动画

组合动画，用到的类是 AnimatorSet，通过 new 的方式创建该类的实例对象。

No.

Date

然后用 ObjectAnimator 创建此类动画。

然后调用 animatorSet 对象的 play() 方法, with(), before() 等方法, 把动画组合起来一起运动。例如:

animatorSet.play(rotation).with(scaleY).before(alpha);

最后通过 animatorSet.start() 开始播放动画。

四、使用 XML 加载动画。

在 res 目录下创建 animator 的文件夹, 在里面创建 .xml 的动画资源文件。

使用最多的是 set 标签和 objectAnimator 标签, 如:

<set

 android: ordering = "sequentially"

 <objectAnimator

 android: propertyName = "alpha"

 : duration = "500"

 : valueFrom = "0"

 : valueTo = "1"

 : valueType = "floatType" >

 </objectAnimator>

 <set

 </set>

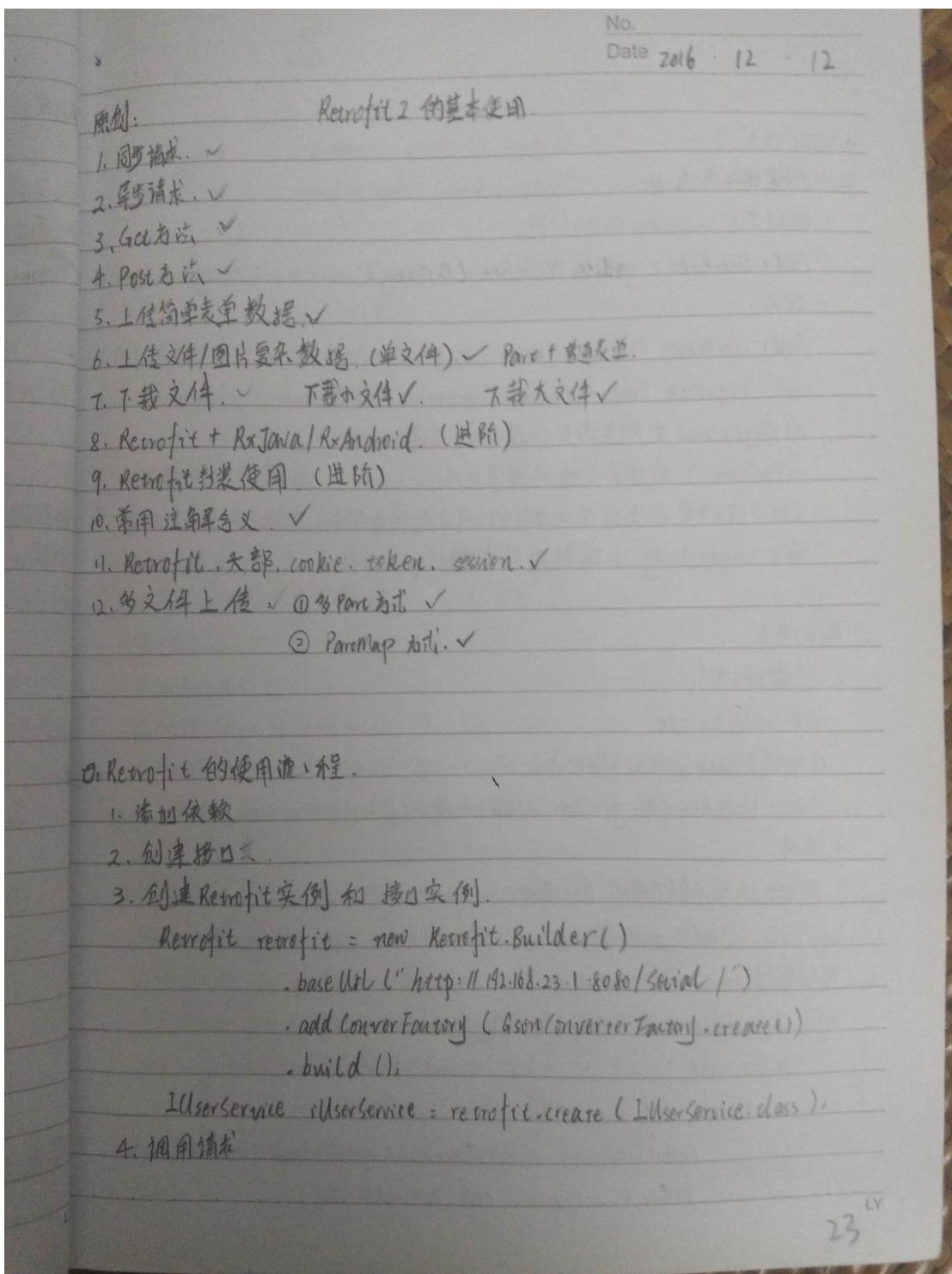
 </set>

ordering 为 sequentially 是顺序执行, together 为同时执行。

propertyName 为属性值, 动画作用的对象必须含 get, set 方法。

set 标签可以嵌套使用, 创造复杂的动画。

123.Retrofit2 基本使用



一、Get请求

1. 创建接口方法

```
@GET ("UserDataServlet")
```

```
Call<UserBean> getUserWithGet (@Query("username") String username);
```

2. 调用

```
Call<UserBean> call = iUserService.getUserWithGet ("xm");
```

```
call.enqueue (new Callback<UserBean> () { ... });
```

onResponse() 是网络请求成功的调用方法。

onFailure() 是请求失败的调用方法。

这两个回调是在主线程中进行的，可进行相应的UI操作。

通过 response.body(), 取到实体类对象。userBean = response.body();

二、Post请求

1. 创建接口方法

```
@FormUrlEncode
```

```
@POST ("UserDataServlet")
```

```
Call<UserBean> getUserDataWithPost (@Field ("username") String name);
```

2. 调用

同 get 请求的调用完全一致。

三、同步请求

```
new Thread (new Runnable () {
```

```
    @Override
```

```
    public void run () {
```

```
        Call<UserBean> call = iUserService.getUserDataWithGet ("xm");
```

```
        Response response = call.execute ();
```

```
userBeen = response.body();
handler.post(new Runnable() {
    public void run() {
        // 转到主进程处理
    }
}).start();
}

// 启动请求
call.enqueue(new Callback<UserBean>() {
    @Override
    public void onResponse(Call<UserBean> call, Response<UserBean> response) {
        UserBean userBean = response.body();
        if (userBean != null) {
            Intent intent = new Intent();
            intent.putExtra("user", userBean);
            LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
        }
    }

    @Override
    public void onFailure(Call<UserBean> call, Throwable t) {
        Log.e("TAG", "onFailure: " + t.getMessage());
    }
});
```

四、启动请求

```
call.enqueue(new Callback<UserBean>() {
    @Override
    public void onResponse(Call<UserBean> call, Response<UserBean> response) {
        UserBean userBean = response.body();
        if (userBean != null) {
            Intent intent = new Intent();
            intent.putExtra("user", userBean);
            LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
        }
    }

    @Override
    public void onFailure(Call<UserBean> call, Throwable t) {
        Log.e("TAG", "onFailure: " + t.getMessage());
    }
});
```

五、上传简单表单数据
同 post、或 get 请求的接口方法

六、上传单文件 + 简单表单数据

1. 创建方法

```
@Multipart
@POST("ModifyHeadServlet")
Call<UserBean> modifyHead(@Part MultipartBody.Part part,
                           @Part("description") RequestBody description);
```

2. 调用

```
RequestBody partBody = RequestBody.create(MediaType.parse("multipart/form-data"), file);
MultipartBody.Part part = MultipartBody.Part.createFormData("key", fileName, partBody);
RequestBody descriptionBody = RequestBody.create(null, "Description content");
Call<UserBean> call = iUserService.modifyHead(part, descriptionBody);
call.enqueue(new Callback<UserBean>() {
    @Override
    public void onResponse(Call<UserBean> call, Response<UserBean> response) {
        UserBean userBean = response.body();
        if (userBean != null) {
            Intent intent = new Intent();
            intent.putExtra("user", userBean);
            LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
        }
    }

    @Override
    public void onFailure(Call<UserBean> call, Throwable t) {
        Log.e("TAG", "onFailure: " + t.getMessage());
    }
});
```

七、多文件上传

方法一：Part 方式

创建方法：

@Multipart

@POST ("AddNewsServlet")

Call < Bean > addNews (@Part MultiPartBody. Part part 1,

@Part MultiPartBody. Part part 2,

@Part MultiPartBody. Part part 3,

@Part ("content") RequestBody content);

调用：与单文件上传类似，要学会举一反三。

方法二：Part Map 方式

创建方法：

@Multipart

@POST ("AddNewsServlet")

Call < Bean > addNews (@PartMap Map< String , RequestBody > partMap,

@Part ("content") RequestBody content);

调用：

RequestBody body = RequestBody.create (MediaType.parse ("multipart/form-data"), fileOnly);

{ X MultiPartBody. Part part1 = MultiPartBody. Part.createFormData ("key", body);

X part2 = ...;

Y part3 = ...;

Map< String , RequestBody > partMaps = new HashMap<>();

partMaps.put ("part1", filename = "filename", body);

partMaps.put (- - -);

put (- - -); //有多少就 put 多少。

Call < Bean > call = iUserService.addNews (partMaps, RequestBody.create (null, "content"));

RequestBody.create (null, "content"));

call.enqueue (new CallBack< Bean > () { ... });

八. 下载小文件.

1. 创建方法

@GET()

Call<ResponseBody> downloadPic(@Url String url);

2. 调用.

Call<ResponseBody> call = iUserService.downloadPic(urlString);

call.enqueue(new Callback<ResponseBody>() { ... });

不知为什么在 onResponse方法中直接操作 response.body() 获取不到内容. 需要在新的方法中使用. 例如. handleDownloadPic(response.body());

然后通过 response.body().byteStream(); 可获得 InputStream 对象.

进而进行相应的操作. 当然通过 responseBody 也可获得其他类型数据.

九. 下载大文件.

如果用上面方法去下载大文件, 会导致 OOM; 为此, 有如下方法.

创建方法

@Streaming

) ; @GET()

Call<ResponseBody> downloadAPK(@Url String url);

调用该方法要在非主线程中进行. 因此, 可以放到 AsyncTask 中执行.

new AsyncTask<Void, Long, Void>() { ... }.execute();

在 doInBackground 中调用.

Call<ResponseBody> call = iUserService.downloadAPK(urlString);

call.enqueue(new Callback<ResponseBody>() { ... });

在 onResponse 方法中转到主线程执行相应 UI 操作.

十一、添加头部 cookie 信息

```
OkHttpClient.Builder builder = new OkHttpClient.Builder();
builder.addInterceptor(new Interceptor() {
    @Override
    public Response intercept(Chain chain) throws IOException {
        Request originalRequest = chain.request();
        Request.Builder builder = originalRequest.newBuilder()
            .addHeader("Cookie", "SESSIONID=xxxxxx");
        Request request = builder.build();
        return chain.proceed(request);
    }
});
```

```
OkHttpClient client = builder.build();
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("http://192.168.23.1:8080/socialServer/")
    .addConverterFactory(GsonConverterFactory.create())
    .client(client)
    .build();
```

此后每次网络请求都会添加头部 cookie 信息了。当然也能添加其他头部信息。

十二、常用标注

@GET：用 get 方法请求

@POST：用 post 方法请求

@Query：url 上的参数，与 @GET 配合使用

@FormUrlEncoded：使用表单，通常与 @POST 配合使用

@Field：表单项

No.
Date 20

① Multipart : 使用 Multipart 技术请求

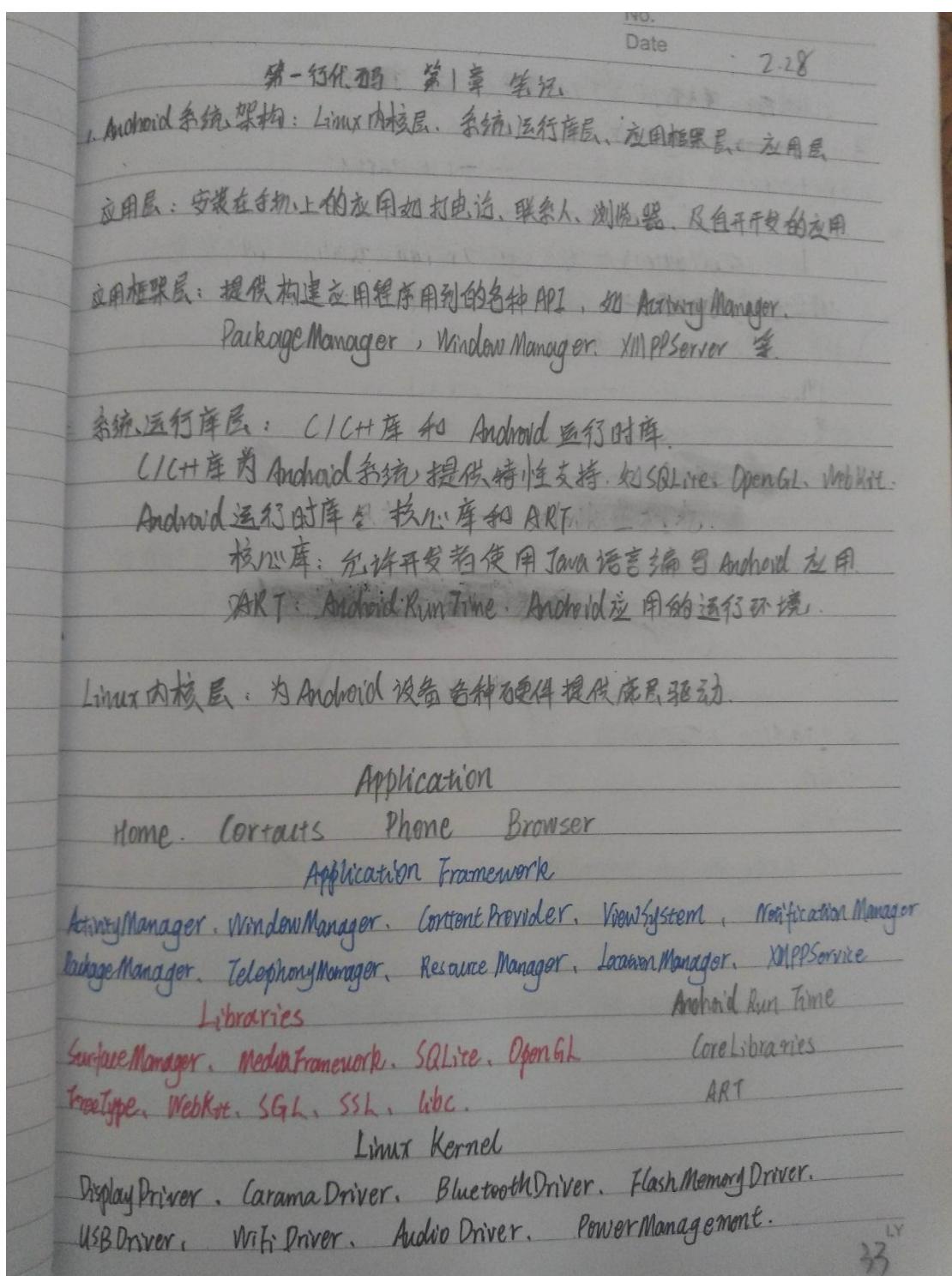
② Part : 指明 Multipart 中的 part 信息

③ Part Map : 使用 part Map 多文件上传

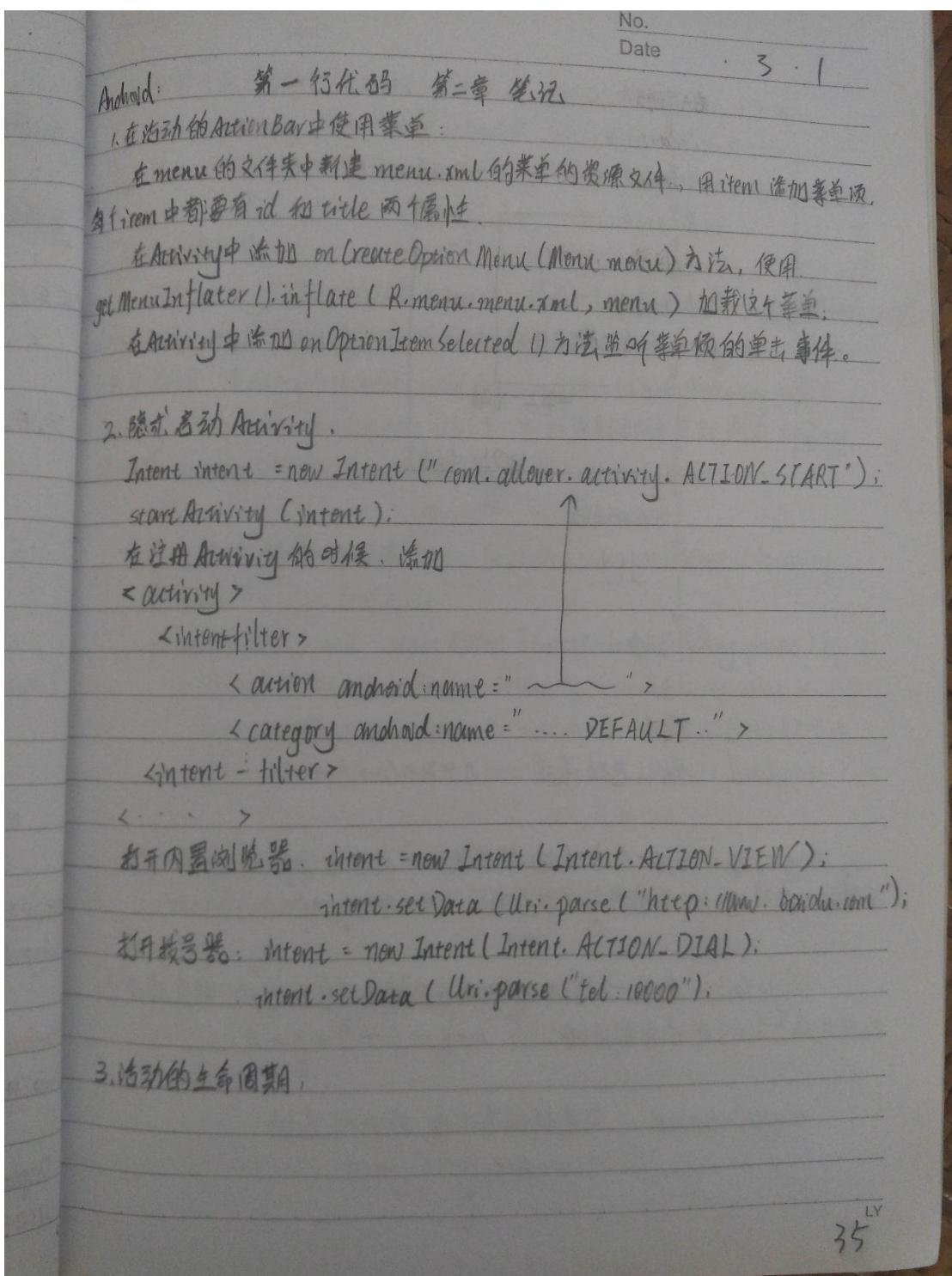
④ Streaming : 下载大文件

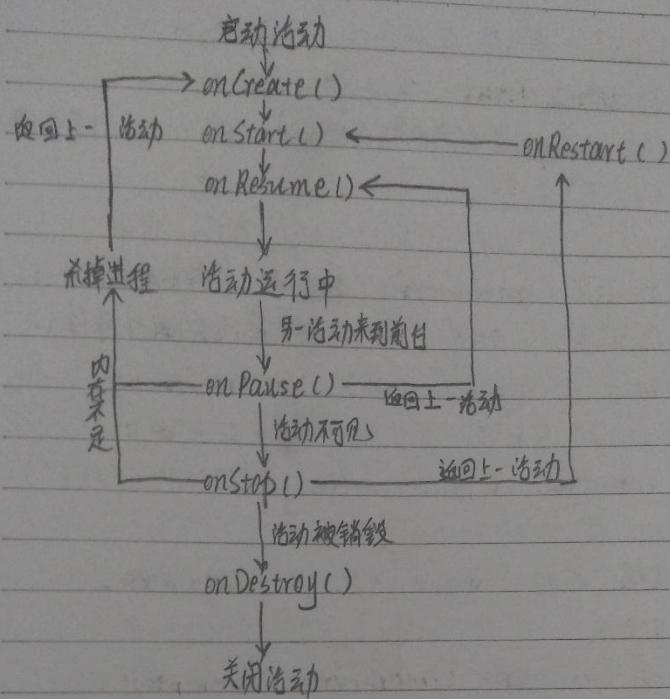
⑤ Url : 动态指定 url, 会覆盖 BaseUrl

133. 第一行代码 第一章 Android 框架 笔记



134. 第一行代码 第二章 Activity 笔记





4. 活动的启动模式：

standard：默认启动模式， $A \rightarrow B \rightarrow C \rightarrow B \rightarrow A$

singleTop：启动一个活动时，如果该活动处于返回栈的栈顶，则不需要创建该活动，而是直接使用当前。 $A \rightarrow B \rightarrow C \rightarrow C$

singleTask：启动一个活动时，如果返回栈中存在该活动，则直接使用它，该活动之前的活动全部出栈。 $A \rightarrow B \rightarrow C \rightarrow D \rightarrow B \rightarrow A$

singleInstance：创建新的启动栈，存放该活动。

$A \rightarrow B \rightarrow C \rightarrow A$

5. 知道当前是哪个活动：

在 BaseActivity 的 onCreate() 方法中，加上语句：

```
Log.d("BaseActivity", getClass().getSimpleName());
```

6. 活动管理器

创建 ActivityCollector 的普通 Java 类，持有一个动态数组管理活动。包括
添加一个活动、移除一个活动、移除所有活动。在 BaseActivity 的 onCreate()
方法中调用 ActivityCollector.addActivity(this); 在 onDestroy() 中调用
ActivityCollector.removeActivity(this); 在任意情况下退出所有 Activity
则调用 ActivityCollector.removeAll();

7. 启动活动的最佳写法： FirstActivity → SecondActivity

在 SecondActivity 创建方法

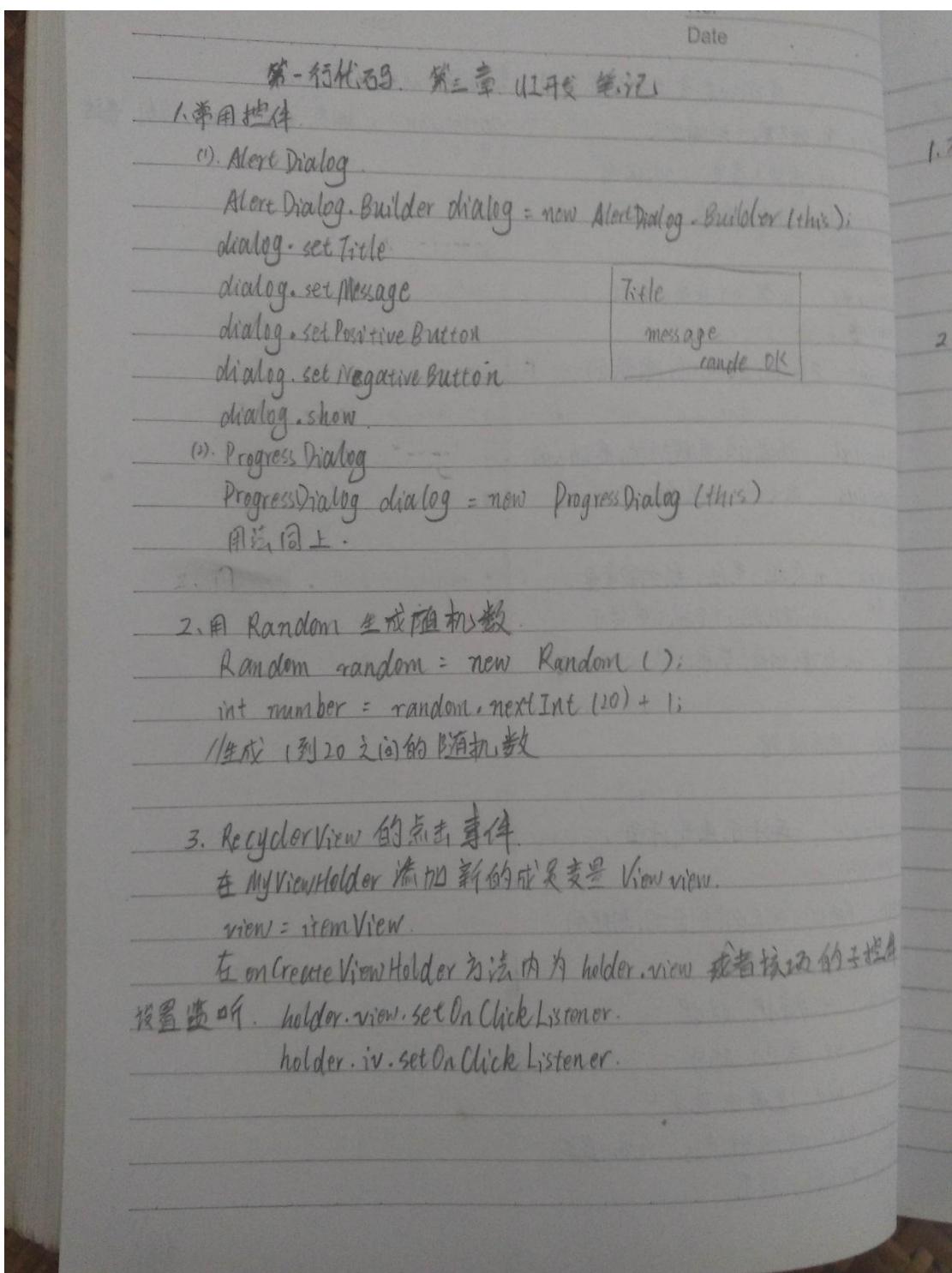
```
public static void startAction(Context context, String parme){  
    Intent intent = new Intent(context, SecondActivity.class);  
    intent.putExtra("parme", parme);  
    context.startActivity(intent);  
}
```

在 FirstActivity 中启动 SecondActivity

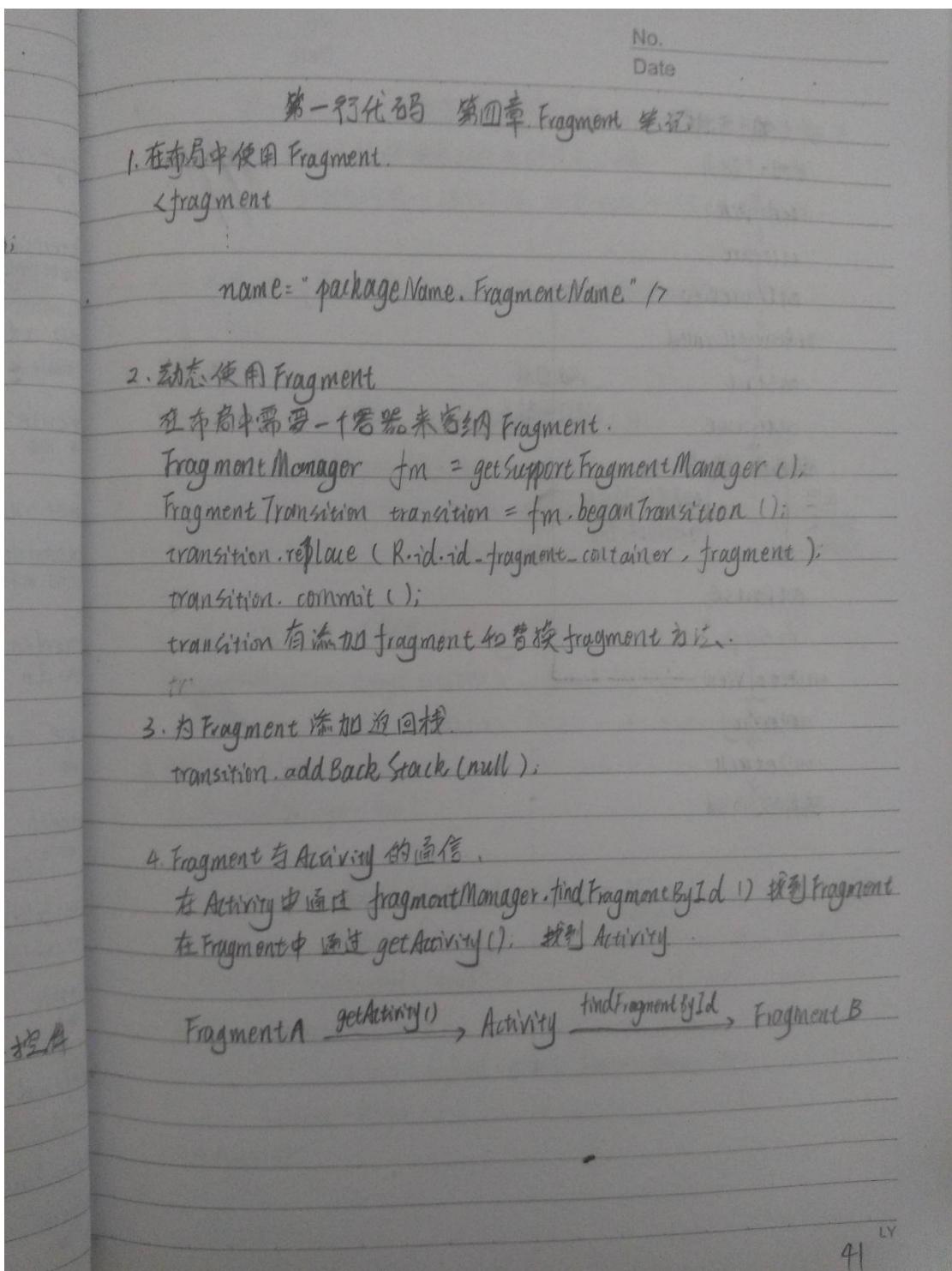
```
SecondActivity.startAction(FirstActivity.this, parme);
```

使用

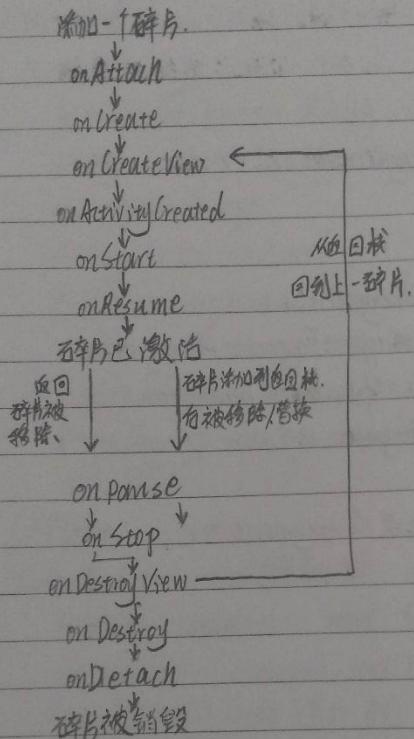
138. 第一行代码 第三章 UI 笔记



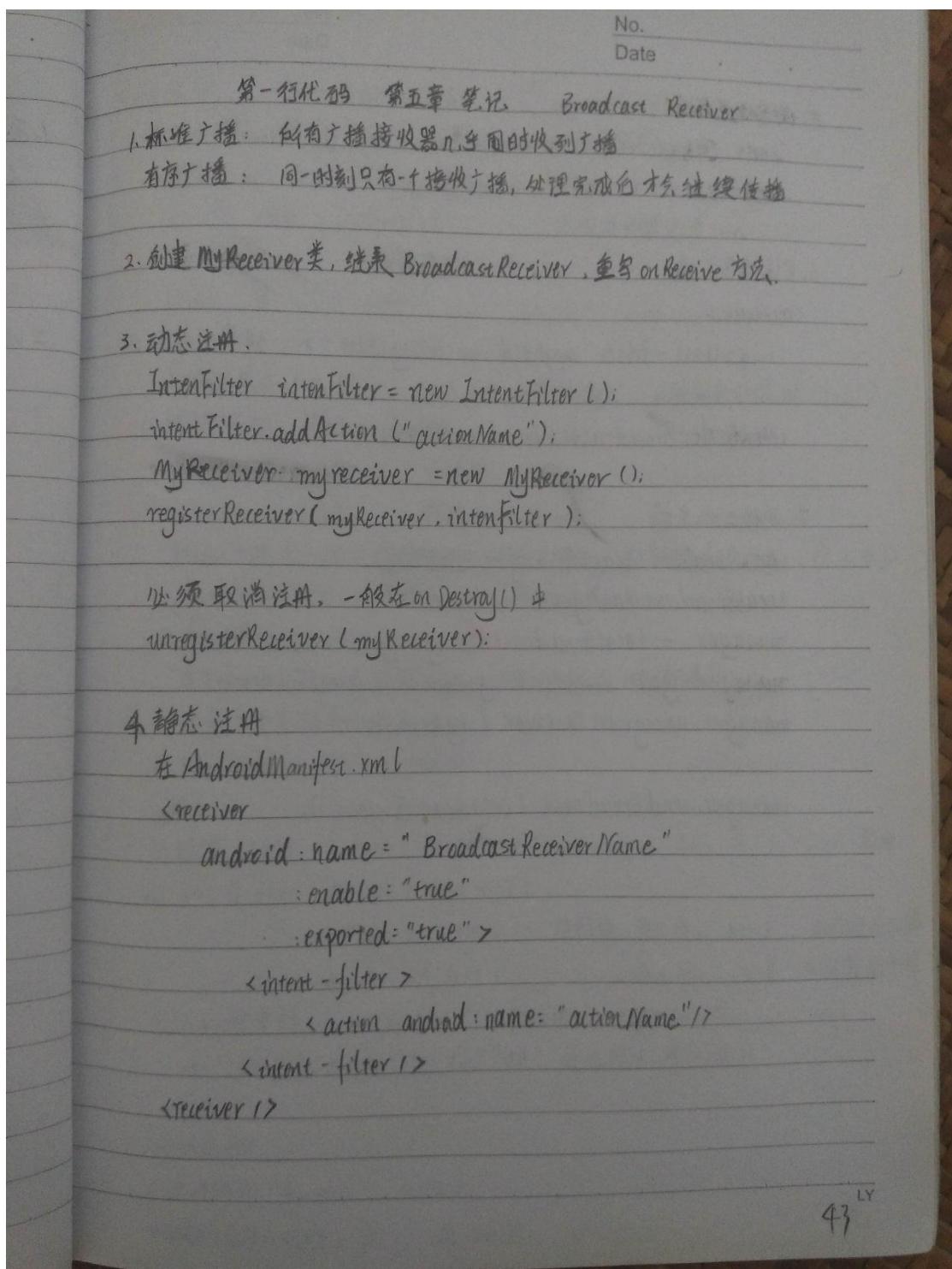
141. 第一行代码 第四章 Fragment 笔记



5. 碎片的生命周期



143. 第一行代码 第五章 Broadcast Receiver 笔记



Date

5. 发送标准广播

```
Intent broadcastIntent = new Intent("actionName");
sendBroadcast(broadcastIntent);
```

6. 发送有序广播

<receiver>

```
<intent-filter android:priority="100"> 设置高优先级
```

```
sendOrderBroadcast(broadcastIntent);
```

7. 使用本地广播

通过 LocalBroadcastManager 进行管理

```
LocalBroadcastManager manager;
```

```
manager = LocalBroadcastManager.getInstance(this);
```

```
manager.registerReceiver(myReceiver, intentFilter);
```

```
manager.unregisterReceiver(myReceiver);
```

```
manager.sendBroadcast(broadcastIntent);
```

145. Java 基础

No.
Date

Java 笔记：Java 基础

1. 数据类型：整型、浮点、boolean 型、char 型。

(1) 整型：int、short、long、byte。

int 4 字节 $-2^{31} \sim 2^{31}-1$ 范围与机器无关。

short 2 $-32768 \sim 32767$

long 8

byte 1 $-128 \sim 127$

长整数值有一个后缀 L，十六进制有前缀 0x，二进制有前缀 0b (Java7)。

(2) 浮点类型：float、double。

float 4 字节 有效位 6~7 位

double 8 字节 有效位 15 位。

float 类型有后缀 F (如 3.14F)，没有 F 的 3.14 默认为 double，或 3.14D

BigDecimal 类，计算不会有任何舍入误差。

(3) char 类型

在 Java 中，char 类型用 UTF-16 编码描述一个代码单元。

尽量不要在程序中使用 char。

(4) boolean：true、false。

2. 运算符：+、-、*、/、%、++、--、<、>、<=、>=、!=、&、|、^、~、枚举。

(1) ++i 与 i++；int m=7; int n=7;

int a = 2 * ++m; 执行后，a=16, m=8，先自增后运算

int b = 2 * n++; 执行后，b=14, n=8，先运算后自增

(2) 三元操作符：condition ? expression1 : expression2

如 $x < y ? x : y$ x 小于 y 吗？是选择 x，否选择 y。

3. 控制流程。

(1) 条件语句：if...else；switch。

(2) 循环语句：while；for；do...while；for...each；

45 LY

(3) 多重选择的 switch 语句

case 可以是：char, byte, short, int, 指针常量、String (java7).

(4) 中断控制流程语句：break; continue

break: 退出当前条件，如每个 case 分支都有一个 break.

退出循环体，执行循环体之外的语句

continue: 退出本次循环，执行下一次循环。

4. 数组

(1) 宣告： int a[]; 或 int[] a; 常用 int[] a;

(2) 初始化： int[] a = {1, 2, 3, 4, 5};

int[] a = new int[] {1, 2, 3, 4, 5};

(3) 数组拷贝： int[] b = Arrays.copyOf(a, a.length);

一种错误的方式： int[] b = a;

b[2] = 10; //修改后 a[2] 的值为 10.

(4) 数组排序： Arrays.sort(a);

146. Java 常用类

No.
Date

java 基础： 常用类：

1. String

- (1) String strA = "Hi"; String strB = "Hello";
- (2) 拼接字符串： String greet = "Hello"; String s = greet.substring(0, 3); // s=Hel，即从开始到3,不包含3
- (3) 拼接： String strC = strA + strB; // strC=HiHello;
- (4) 不可变字符串： 把 Hello 修改为 Help。
String strHelp = strB.substring(0, 3) + "p";
- (5) 判断相等： strA.equals(strB); 返回 boolean 值。
strA.equalsIgnoreCase(strB); 忽略大小写
- (6) 空串 != null串；
- (7) 长度： int length = strB.length(); // length = 5;
- (8) 大小写转换： strB.toLowerCase(); // hello
strB.toUpperCase(); // HELLO
- (9) 其他： boolean startWith(String prefix); 判断开始字符串

2. StringBuilder

- (1) 构建对象： StringBuilder builder = new StringBuilder();
- (2) 追加内容： builder.append("Hello");
builder.append("Hi");
- (3) 打印字符串： String str = append.toString(); // HelloHi

3. Math

- (1) 取算术平方根： double y = Math.sqrt(4.0); // 2.0
- (2) 幂运算： double y = Math.pow(2, 3); // 2³
- (3) 常用的三角函数： 多数为弧度制。
Math.sin、Math.cos、Math.tan、Math.atan
如 Math.sin(Math.PI/6); // 0.5

(4) 指数函数: Math.exp

(5) 对数函数: Math.log

(6) 10为底的对数: Math.log10

(7) 0-1 的随机特点数: Math.random.

4. Arrays.

(1) 数组拷贝: int[] b = Arrays.copyOf(a, a.length);

(2) 数组排序: Arrays.sort(a); // 优化的快速排序算法.

(3) 二分查找: int index = Arrays.binarySearch(a, 5); // 返回下标,

(4) 数组比较: boolean result = Arrays.equals(a, b); // 比较值

5. Date 类

```
Date date = new Date();
```

Format format:

```
format = new SimpleDateFormat("yyyy-mm-dd hh:mm:ss");
```

```
System.out.println(format.format(date));
```

```
format = new SimpleDateFormat("yyyy年mm月dd日 hh时mm分ss秒")
```

```
format.format(date);
```

6. GregorianCalendar 类

```
GregorianCalendar calendar = new GregorianCalendar(); // 通过构造方法
```

```
int year = calendar.get(Calendar.YEAR);
```

```
int month = calendar.get(Calendar.MONTH);
```

```
int day = calendar.get(Calendar.DAY_OF_MONTH);
```

```
Calendar.DAY_OF_WEEK, HOUR, MINUTE, SECOND
```

```
calendar = new GregorianCalendar(2016, 11, 30);
```

```
GregorianCalendar(2016, Calendar.DECEMBER, 30)
```

设置日期.

No.

Date

calendar.set(Calendar.YEAR, 2016),

calendar.set(Calendar.MONTH, Calendar.APRIL);

calendar.set(2016, Calendar.APRIL, 21),

次。

回下标、
可变值

(=53"),

miss秒"),

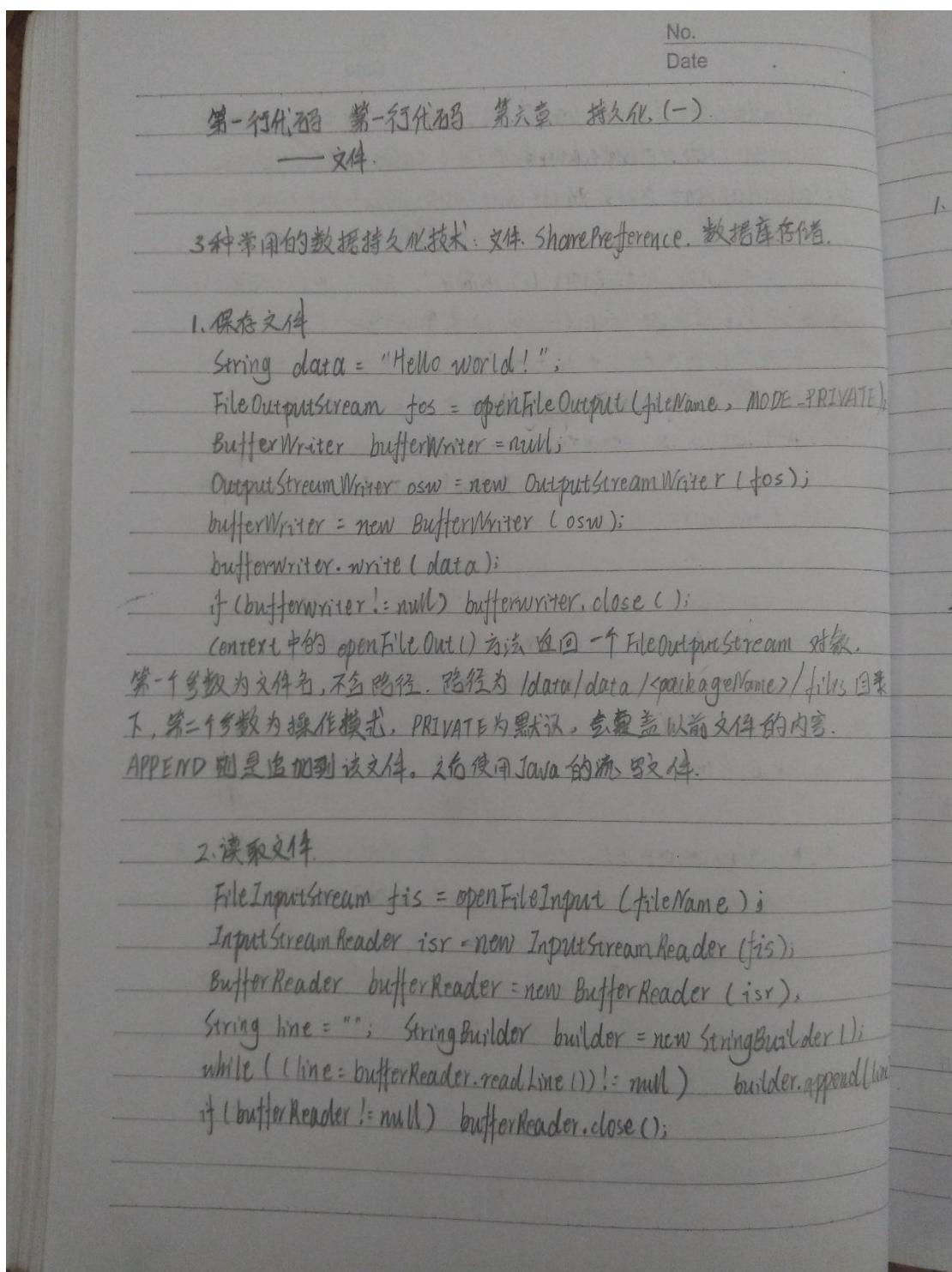
1, 115001
102

10

(BER, 30)

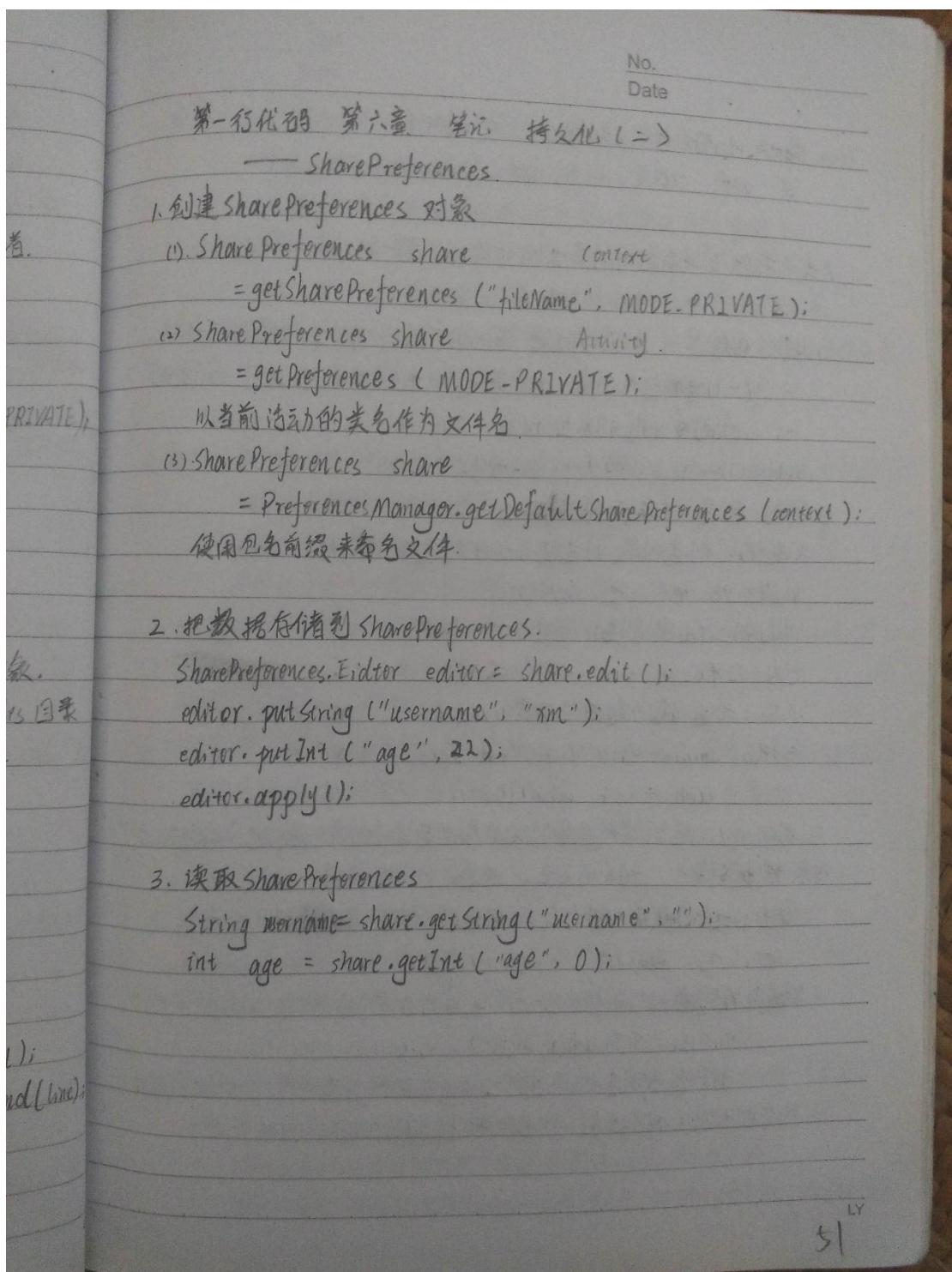
49 LY

150. 第一行代码 第六章 持久化 File 笔记

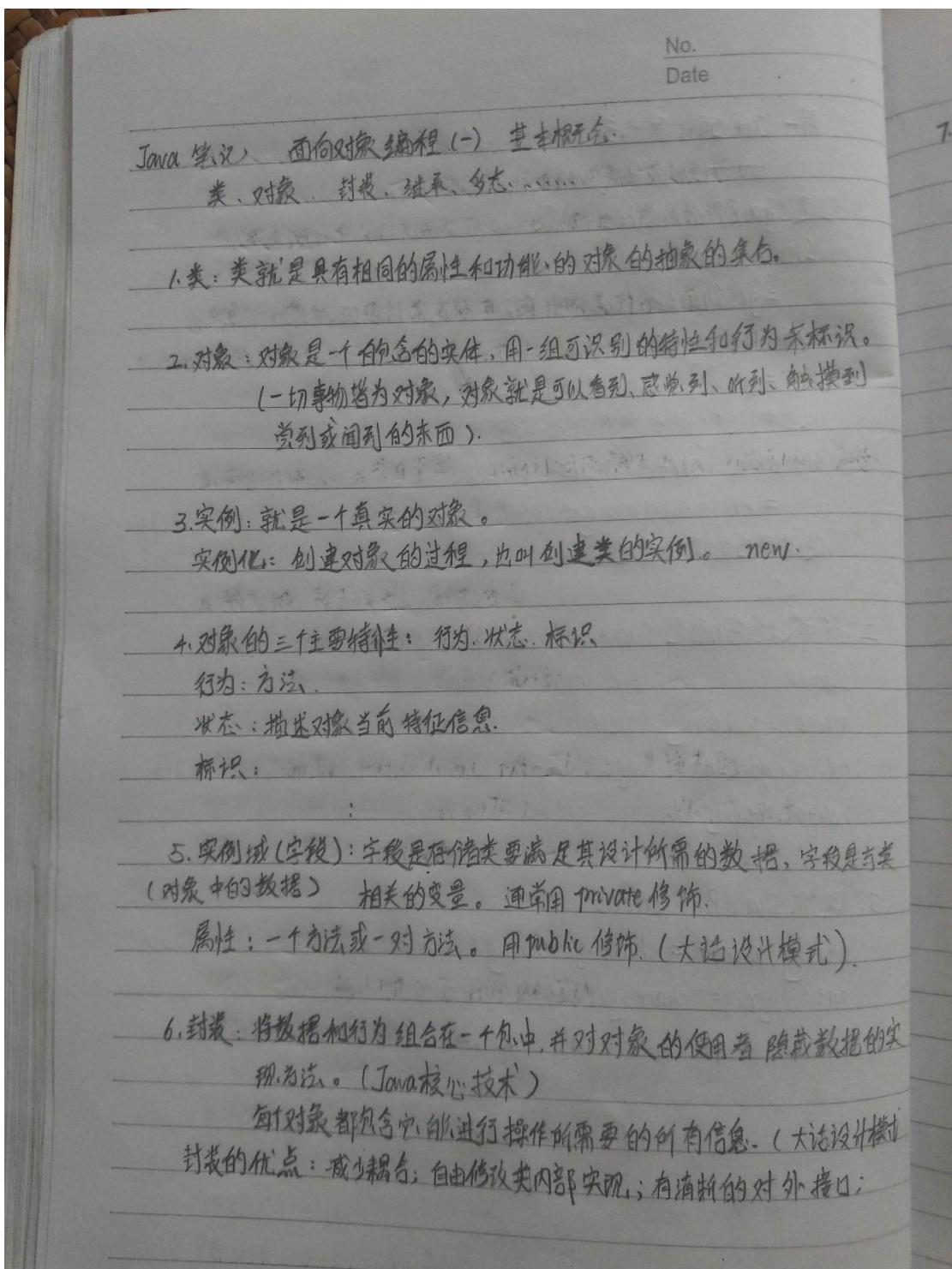


151. 第一行代码 第六章 持久化

SharedPreferences 笔记



152. 面相对象基本概念



7. 继承：子类拥有父类非 private 的属性和功能；

子类可以扩展父类没有的属性和功能；

子类可以以自己的方式实现父类的功能（方法重写）；

继承的优点：使子类公共部分放在父类，使代码得到共享，避免重复。

继承可使得修改或扩展继承而来的实现较容易实现。

继承的缺点：父类变子类不得不改变；破坏封装；父类实现细节需结合子类。

强制耦合：

8. 多态：不同的对象可以执行相同动作，但要通过它们自己的实现代码执行。

子类以父类的身份出现；

子类在工作时以自己的方式实现。

子类以父类身份出现时，子类特有的属性和方法，不可以使用。

方法重写（方法覆盖）：子类可以选择使用 override 关键字 将父类实现替换为自己的实现。

9. 方法重载：提供创建多个同名方法的能力；但这些方法需使用不同的参数类型。

方法名相同，参数类型或个数不相同。

方法重载可以在不改变原方法的基础上新增功能。

10. 抽象类：抽象类不能实例化；

abstract 抽象方法是必须被子类重写的方法。

如果类中包含抽象方法，那么类就必须定义为抽象类。

11. 接口：接口是把魔术方法和属性组合起来，以封装特定功能的一个集合。

实现：接口就必须实现接口中所有的方法。接口不能实例化。

接口中的方法或属性前面不能有修饰符，方法没有方法体。

接口不是类，不能实例化（new），但能声明，必须引用实现了该接口的
类对象。

12. 抽象类与接口:

抽象类可以给出一些成员的实现，接口却不包含成员的实现。

抽象类的抽象成员可被子类部分实现，接口的成员需要实现类完全实现。

一个类只能继承一个抽象类，但可实现多个接口。

类是对对象的抽象；抽象类是对类的抽象；接口是对行为的抽象。

如果行为跨越不同类的对象，可用接口；对于一些相似的类对象，用抽象类。

抽象类：

13. 接口(补充)：接口不能包含实例域或静态方法，但可以含常量。

接口可继承接口。

14. 静态域、静态常量、静态方法

```
public class Employee {  
    private int id; //实例域  
    private static int nextId; //静态域  
    public static final int ID; //静态方法  
    public static int getID() { } //静态方法
```

静态域：将实例域定义为 static，一个类中只有一个这样的域（还有其他类型的静态域）。例如 100 个 employee 对象，它们有各自的 id 域，但只有一个 nextId 域，它们是共用 nextId。

静态常量：调用方式：Employee.ID

静态方法：调用方式：Employee.getID();

静态方法不能访问类中的实例域，可以访问静态域

155. Java 内部类

Java 笔记 内部类

1. 内部类：定义在一个类中的类。

内部类方法可以访问该类定义所在作用域中的数据，包括私有数据。
内部类可以对同一包中的其他类隐藏起来。

当想要定义一个回调函数且不想编写大量代码时，使用匿名内部类比较便捷。

内部类既可以访问自身的数据域，也可以访问创建它的外围类对象的数据域；

当内部类没有声明构造方法时，编译器会自动为内部类生成一个默认的构造方法

```
public InClass ( OutClass out ) {  
    outer = out;  
}
```

内部类访问外部类的数据：`OuterClass.this.field`

2. 局部内部类：在外部类的方法里定义局部类
可以访问外部类，也可以访问被 final 修饰的局部变量。

3. 匿名内部类：只创建这个类的一个对象，就不必命名，这种类被称为匿名内部类。
如：

```
Listener listener = new OnClickListener () {  
    @Override  
    public void onClick ( View view ) {  
        //  
    }  
}
```

一般形式：`new SuperType (construction parameter) {
 inner class · method or data
}`

No.

Date

Person person = new Person ("xm"); //person 对象

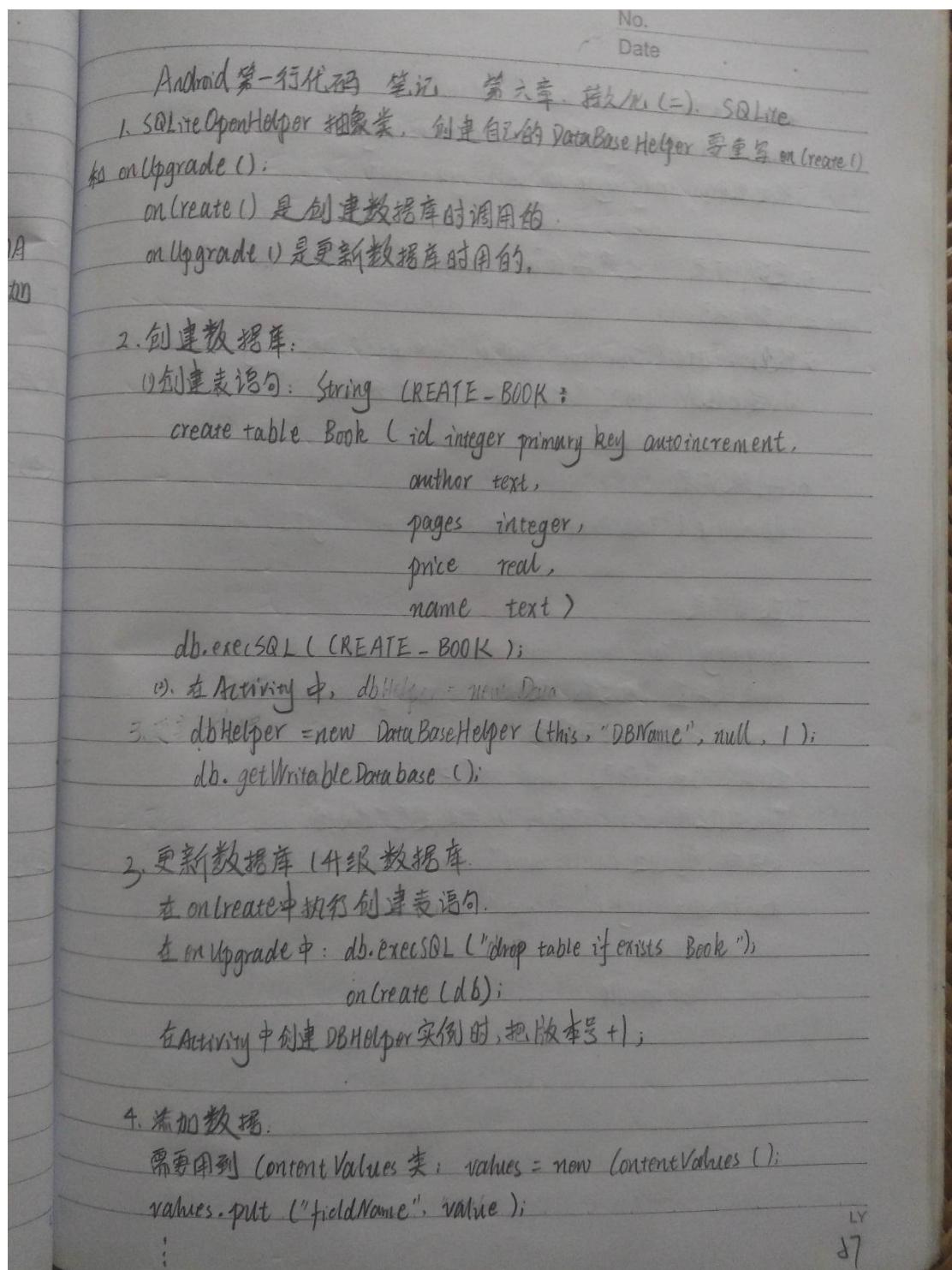
Person person = new Person ("allover"); //对象

//an object of an inner class extending Person

4. 静态内部类：内部类不需要引用外围类对象，可以将内部类声明为 static。外部类中通过静态方法构造内部类，因此内部类必须添加 static 修饰。调用方式：

OuterClass.InnerClass innerClass = OuterClass.getInnerClass();

157. 第一行代码 第六章 持久化 SQLite 笔记



No.
Date

db.insert ("tableName", null, values);

添加另一记录时，需要 values.clear();

然后重新 put(key, value) 和 insert.

5. 更新记录，也是需要 ContentValues对象，只需要更新的内容

put进去就好了。

db.update ("TableName", values, "name=?", new String[] {"Android"});

? 是占位符，有多少个？数组就有多少个元素。

6. 删除记录

db.delete ("TableName", "name=?", new String[] {"Android"});

7. 查询记录

db.query ("TableName", null, null, null, null, null, null);

null 表示查询指参数，如 where, orderBy, having 等等。

全 null 表示查询所有记录。

查找的结果保存到 Cursor 对象中。

可以通过 cursor.moveToFirst() 判断是否有数据。

如果有，则通过 cursor.moveToNext 指向下一条记录。

cursor.getString (cursor.getColumnIndex ("name"));

· getInt

· getDouble

159. Java List Map Set

No.
Date

Java 笔记：List、Map、Set.

1. 泛型数组列表：ArrayList

ArrayList 是一个采用表型数据的泛型类。指定了数组列表保存元素的数据类型。

- (1) 声明和构造：ArrayList<Employee> staff = new ArrayList<>();
或 ArrayList<Employee> staff = new ArrayList<>(100); // 指定初始大小。
- (2) 向数组列表添加元素：staff.add(new Employee());
- (3) 访问数组列表元素：Employee e = staff.get(0); // 首个。
- (4) 获取数组元素数量：int count = staff.size();
- (5) 一旦确定数组大小不再变化，可调用 staff.trimToSize(); // 回收空间。
- (6) 设置第 i 个元素：staff.set(i, employee);
- (7) 删除第 n 个元素：staff.remove(n);

2. 映射表：Map

用来存放键/值对。根据某些键信息，查找与之对应的元素。

常用的映射表有 HashMap、TreeMap。它们都实现了 Map 接口。

散列映射表（HashMap）对键进行散列。

树映射表（TreeMap）用键的整体顺序对元素进行排序，并将其组织成搜索树。

如果不按照排列顺序，最后使用散列映射表。

(1) 建立散列映射表：

```
Map<String, User> map = new HashMap<>();
```

- (2) 放入元素：map.put("xm", new User("xm", "XM"));
- (3) 删除元素：map.remove("xm"); //
- (4) 更新/替换元素：map.put("xm", new User("allever", "Allever"));
将替换键为 xm 所对应的那个元素。
- (5) 获取元素：User user = map.get("xm");

6) 遍历 Map:

```
for (Map.Entry<String, User> entity : map.entrySet()) {  
    String key = entity.getKey();  
    User value = entity.getValue();  
}
```

3. 散列表. (具体定义了看数据结构)

散列表中最简单的就是 `Set` 类型。 `Set` 是没有重复元素的元素集合。

`HashSet` 是实现了基于散列表的集。

建立散列表: `Set<String> set = new HashSet<>();`

放数据: `set.add("Hello");`

`set.add("Hi");`

使用迭代器遍历元素. (其访问顺序是随机的)

`Iterator<String> iter = set.iterator();`

`while (iter.hasNext()) String str = iter.next();`

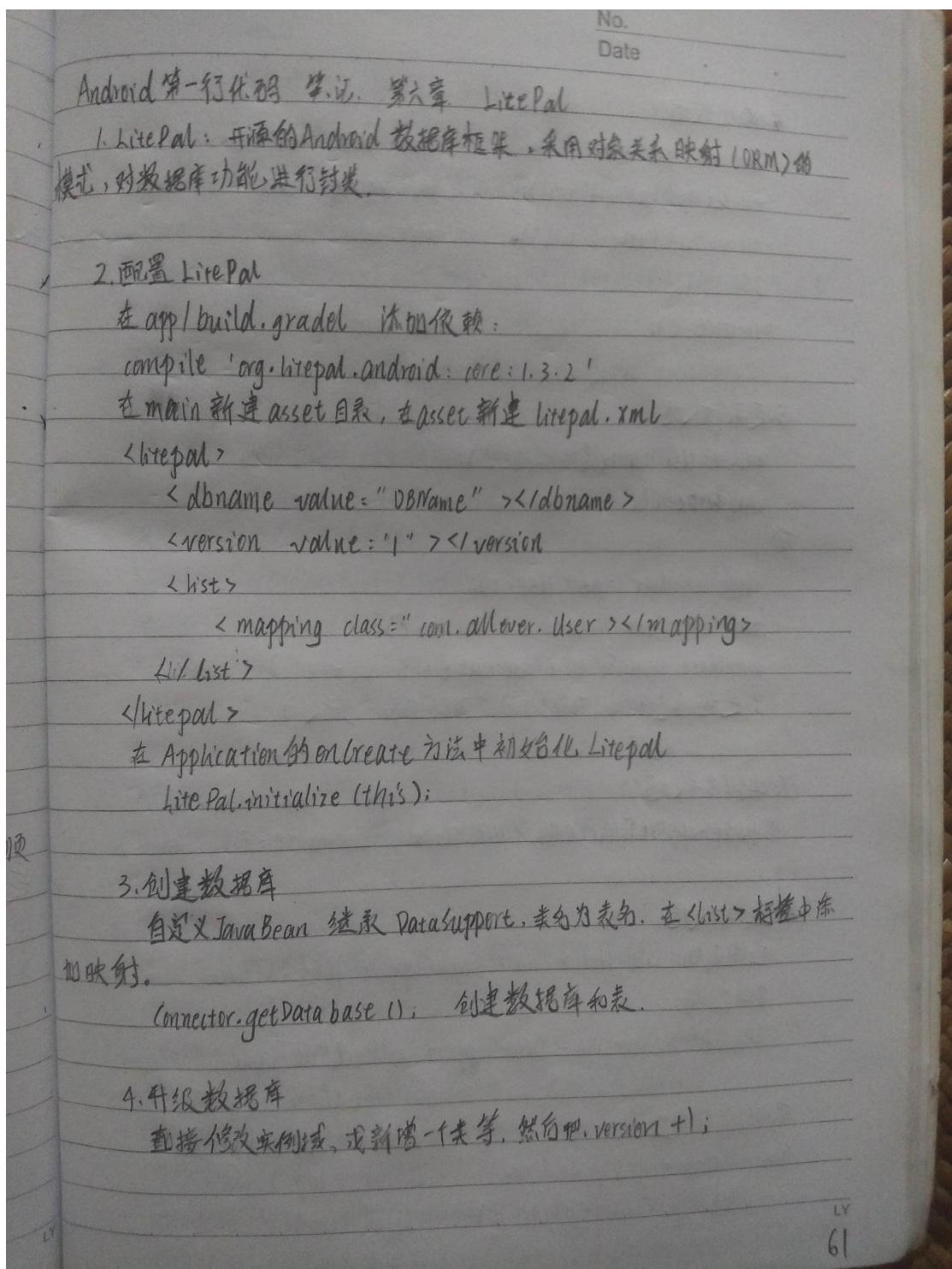
4. 树集 `TreeSet`. (红黑树实现)

可以以任意顺序将元素插入集合中, 每个值将自动地按照排序后的顺序呈现。

例如 `treeSet` 分别 `add`: Bob, Amy, Carl,

遍历顺序为: Amy, Bob, Carl

161. 第一行代码 第六章 持久化 LitePal 笔记



5. 添加数据:

```
User xm = new User();
xm.setUsername("xm");
xm.setNickname("xm");
xm.setSave();
User user = new User();
user.setUsername("allever");
user.setNickname("Allever");
user.setSave();
```

6. 更新数据:

```
xm.setUsername("Winchen");
xm.setSave();
```

或

```
User newUser = new User();
newUser.setUsername("NewUser");
newUser.setNickname("NewUser");
newUser.updateAll("username = ? ", "xm");
? 是相当占位符, 有多少个? 前面就有多少个参数);
```

7. 删陈数据

```
DataSupport.deleteAll(User.class, "username = ? ", "xm");
```

8. 查询数据

查询所有: List<User> list = DataSupport.findAll(User.class);

条件查询:

```
List<User> userList = DataSupport.select("username", "nickname")
```

```
.where("username = ? ", "xm")
```

```
.order("nickname")
```

```
.limit(10)
```

```
.offset(10)
```

```
.find(User.class);
```

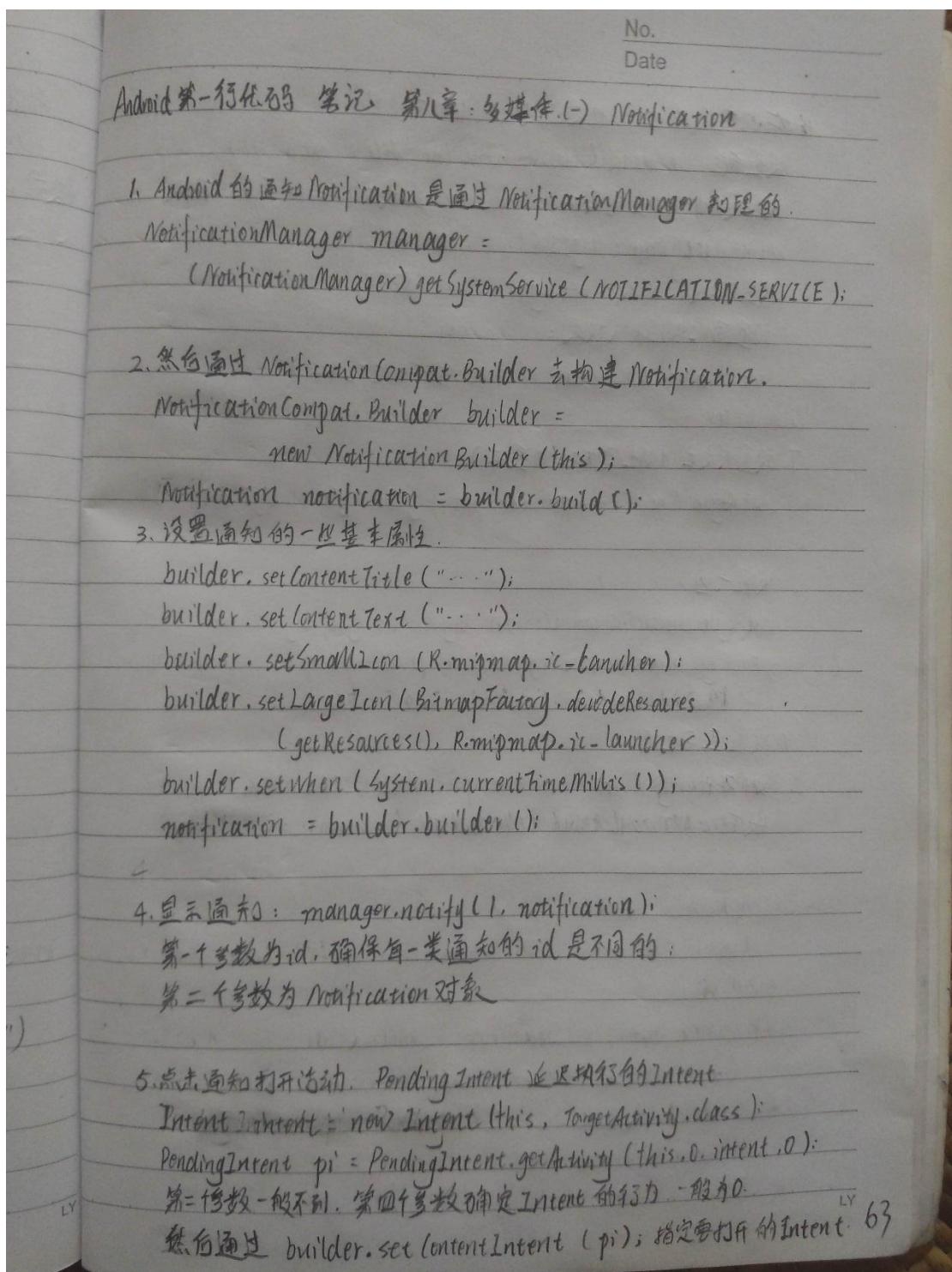
或纯SQL语句查询

cursor cursor =

DataSupport.findBysql("...");

163. 第一行代码 第八章 多媒体

Notification 笔记



No.
Date

6. 其他高级特性 也是通过builder 来设置

设置音效: setSound (Uri. fromFile (new File (" path ")))

设置震动: 如立刻震动 1s, 停止 1s, 再震动 1s, 要声明权限

setVibrate (new long[] { 0, 1000, 1000, 1000 });

设置呼吸灯闪烁: setLight (Color. GREEN, 1000, 1000) 一闪一闪效果

设置通知的默认效果:

setDefaults (NotificationCompat. DEFAULT_ALL);

7. 设置长文本 (继续下按时显示).

setStyle (new NotificationCompat. BigTextStyle () . bigText ("...."));

8. 设置大图

setStyle (new NotificationCompat. BigPictureStyle () . bigPicture
(BitmapFactory.decodeResource (getResources (), R. mipmap. ic_launcher)));

9. 设置重要程度.

setPriority (NotificationCompat. PRIORITY_MAX);

最高: MAX > HIGH > LOW > MIN

165. 第一行代码 第八章 多媒体 Camera 笔记

第一行代码 笔记 第八章 多媒体 (二) Camera

1. 创建 File 对象，用来保存拍照后的图片

```
File outputImage = new File (getExternalCacheDir (), "output-image.jpg").  
if (outputImage.exists ()) outputImage.delete ();  
outputImage.createNewFile ();
```

把拍照后的文件存放到应用关联缓存目录

2 /sdcard/Android/data/<package name>/cache

从 Android 6.0 开始 (API 23)，读写 SD 卡列为危险权限，需要用到运行时权限，放到该目录可以跳过这一步。

2. 把 File 转换成 Uri 对象

```
if (Build.VERSION.SDK_INT >= 24)
```

```
imageUri = FileProvider.getUriForFile (this,  
"xxxx.fileprovider,  
outputImage);
```

else imageUri = Uri.fromFile (outputImage);

在 7.0 以前，通过 Uri.fromFile () 把文件转成 Uri 对象。

在 7.0 及以后，通过 FileProvider 的 getUriForFile () 把文件转换成封装过的 Uri 对象。

在 7.0 中，使用含真实路径的 Uri 是不安全的，会抛出 FileUriExposedException 异常。

3. 打开系统相机软件

```
Intent intent = new Intent ("android.media.action.IMAGE-CAPTURE");
```

```
intent.putExtra (MediaStore.EXTRA_OUTPUT, imageUri);
```

```
startActivityForResult (intent, REQUEST_CODE_TAKE_PHOTO);
```

MediaStore.EXTRA_OUTPUT 指定文件输出路径

4. 把拍到的照片显示出来

```
Bitmap bitmap = BitmapFactory.decodeStream(  
    getContentResolver().openInputStream(imageUri));  
iv.setImageBitmap(bitmap);
```

5. 在上述代码中 使用了 FileProvider，因此要在 AndroidManifest.xml 中注册。

```
<provider>
```

```
    android:name="android.support.v4.content.FileProvider"
```

```
    android:authorities="xxxx.fileprovider"
```

```
    android:exported="false"
```

```
    android:grantUriPermissions="true"
```

```
<meta-data>
```

```
    android:name="android.support.FILE_PROVIDER_PATHS"
```

```
    android:resource="@xml/file_paths" />
```

```
</provider>
```

authorities 的值与 getUriForFile() 方法中第二个参数一致。

meta-data 指定 Uri 的共享路径，引用了 xml 目录下的资源文件。

res/xml/file_paths.xml

```
<paths xmlns:android="http://schemas.android.com...../android" />
```

```
<external-path
```

```
    name="my-images" path="" />
```

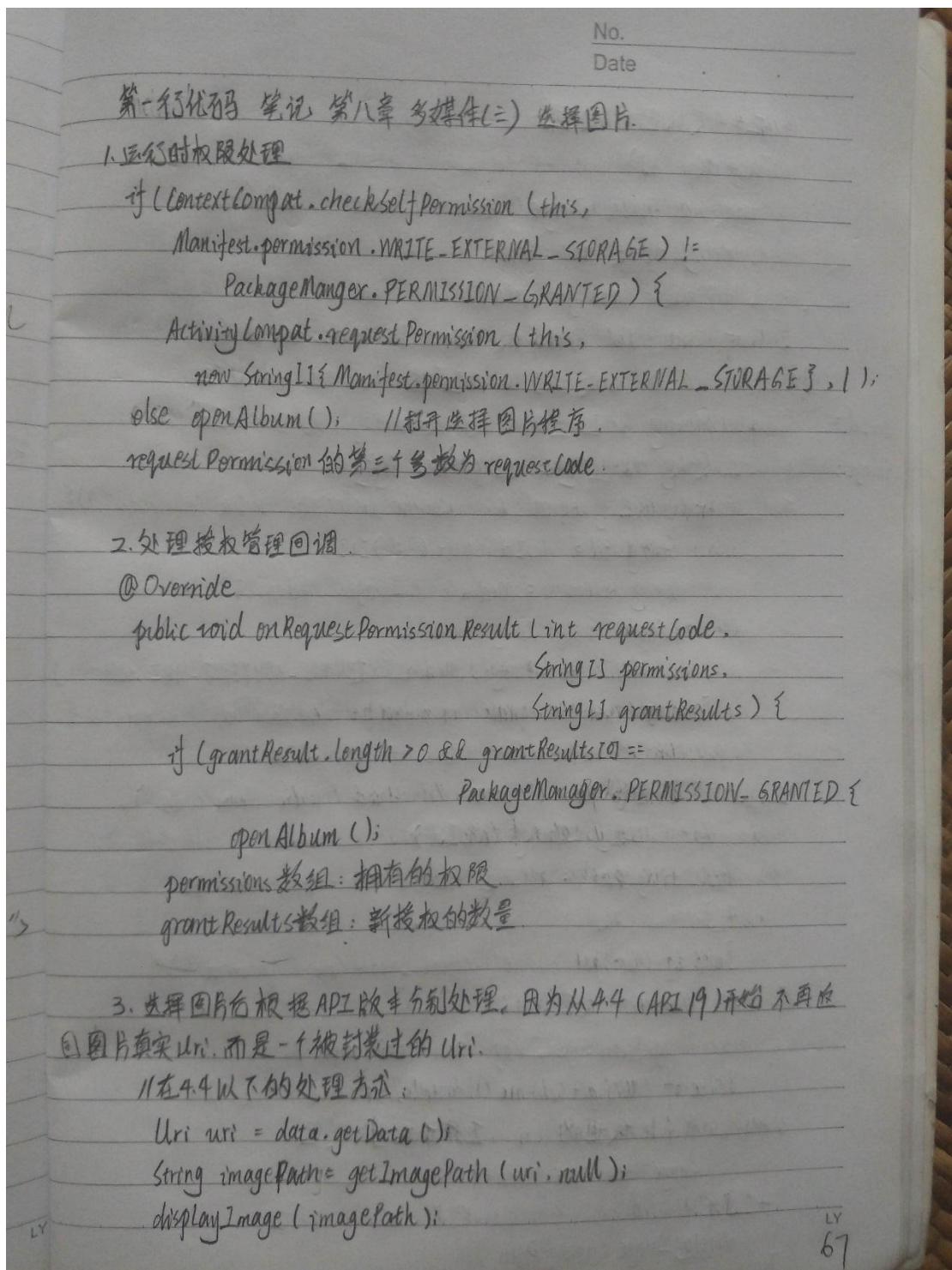
```
</paths>
```

external-path 就用来指定 Uri 的共享路径。

path 的值为具体路径，空表示挂载 SD 卡。

6. 为了兼容老版本手机，也要声明权限：WRITE_EXTERNAL_STORAGE

167. 第一行代码 第八章 多媒体 选择图片 笔记



//在4.4及以上的处理

@TargetApi(19)

```
private void handleImageOnKitKat(Intent data) {
    String imagePath = null;
    Uri uri = data.getData();
    if (DocumentsContract.isDocumentUri(this, uri)) {
        // document 类型 Uri 的处理: document_id 处理
        docId =
            String docId = DocumentsContract.getDocumentId(uri);
        image: 11589
        String authority = uri.getAuthority();
        if (authority.equals("com.android.providers.media.documents")) {
            String id = docId.split(":")[1];
            String selection = MediaStore.Images.Media._ID + "=" + id;
            imagePath = getImagePath(
                MediaStore.Images.Media.EXTERNAL_CONTENT_URI, selection);
        } else if (authority.equals("com.android.providers.downloads.documents")) {
            Uri contentUri = ContentUris.withAppendedId(
                Uri.parse("content://downloads/public-downloads"),
                Long.valueOf(docId));
            imagePath = getImagePath(contentUri, null);
        } else if (uri.getScheme().equalsIgnoreCase("content")) {
            // content 类型的 Uri
            imagePath = getImagePath(uri, null);
        } else if (uri.getScheme().equalsIgnoreCase("file")) {
            // file 类型的 Uri, 直接获取图片路径
            imagePath = uri.getPath();
        }
    }
    displayImage(imagePath);
}
```

No.

Date

Uri 的类型有：

//快图浏览器：content://com.alensw.PicFolder.FileProvider/document/xxxx.jpg
//最近：content://com.android.providers.media.documents/document/image%3A161
//下载：content://com.android.providers.downloads.documents
//文件浏览器 file:///sdcard/SDcard/Edirr/xx.jpg
//谷歌照片：content://com.google.android.apps.photos.contentprovider/-1/xxx...

4. 通过 Uri 和 selection 来获取真实的图片路径

```
Cursor cursor = getContentResolver().query(uri, null, selection, null, null);
if (cursor != null) {
    if (cursor.moveToFirst()) {
        String path = cursor.getString(cursor.getColumnIndex(
            MediaStore.Images.Media.DATA));
    }
    cursor.close();
}
```

5. 显示图片：

```
Bitmap bitmap = BitmapFactory.decodeFile(imagePath);
iv.setImageBitmap(bitmap);
```

170.Java 异常

No.
Date 3. - 31.

Java 笔记：异常

1. 异常分类：

Error类：描述了JAVA
运行时系统的内部错误
和资源耗尽错误。

Exception
IOException
Runtime
Exception

IOException类：I/O 错误导致的异常，包括：
试图在文件尾部后面读取数据；打开一个不存在的文件；等等

RuntimeException：由程序错误导致的异常，例如：
错误的类型转换；数组越界；空指针；等等

未检查异常：派生于Error或RuntimeException 的类。

已检查异常：其他异常

2. 声明已检查异常：

一个方法不仅需要告诉编译器要返回什么值，还要告诉编译器可能发生什么错误。如 `public FileInputStream (String name) throws FileNotFoundException` 在方法（或构造方法）首部用 `throws` 声明有可能抛出的异常。

应该抛出异常的情况：

- 调用一个抛出已检查异常的方法。如 `FileInputStream` 构造器
- 程序运行过程中发现错误，并且利用 `throw` 语句抛出一个已检查异常。
- 程序出现错误
- JVM 和运行时库出现的内部错误。

不要声明 Java 的内部错误，即从 Error 继承的错误；不应该声明从 `RuntimeException` 继承的未检查异常。

3. 抛出异常： `throw`

`throw new EOFException();`
一旦方法抛出异常，这个方法就不可能返回到调用者。

4. 创建异常类：

自定义一个派生于Exception的类，或派生于Exception子类的类。通常定义两个构造器，一个默认，另一个用来描述详细信息。如：

```
class MyFormatException extends IOException {
    public MyFormatException() {};
    public MyFormatException(String message) { super(message); }
}
```

然后抛出异常：throw new MyFormatException("description");

5. 捕获异常：

要捕获异常，必须设置try/catch语句。

如果try语句快抛出异常，那么跳过try语句快剩余代码，执行catch中的代码。

如果try语句块没抛异常，则跳过catch语句；如果抛出的异常没有在catch中声明，那么这个方法立即退出。

最好的做法是将异常传递给调用者，即在方法首部声明可能抛出已检查的异常。

应该捕获那些知道如何处理的异常，而将那些不知怎么处理的异常继续传递。

6. 捕获多个异常：

同一个try语句快可以有多个catch子句。

e.getMessage() 获取异常的更多信息。

e.getClass().getName() 得到异常对象的实际类型

7. finally子句的执行情况：

try：表示try语句快。

catch：表示catch子句。

finally：表示finally子句。

other：表示finally字句之后的代码。

- ① try → finally → other : try 中无抛出异常。
- ② try → catch → finally → other : try 抛出异常，catch 未抛出异常。
- ③ try → catch → finally : try 抛出异常，catch 抛出异常。
- ④ try → finally : try 抛出异常，catch 中无声明该异常。
try 语句可以只有 finally 子句，而没有 catch 子句。

8. try/catch 写法的建议指针

InputStream in = ...
try {
 try {

code that might throw exceptions

} finally {

in.close();

}

} catch (IOException e) {

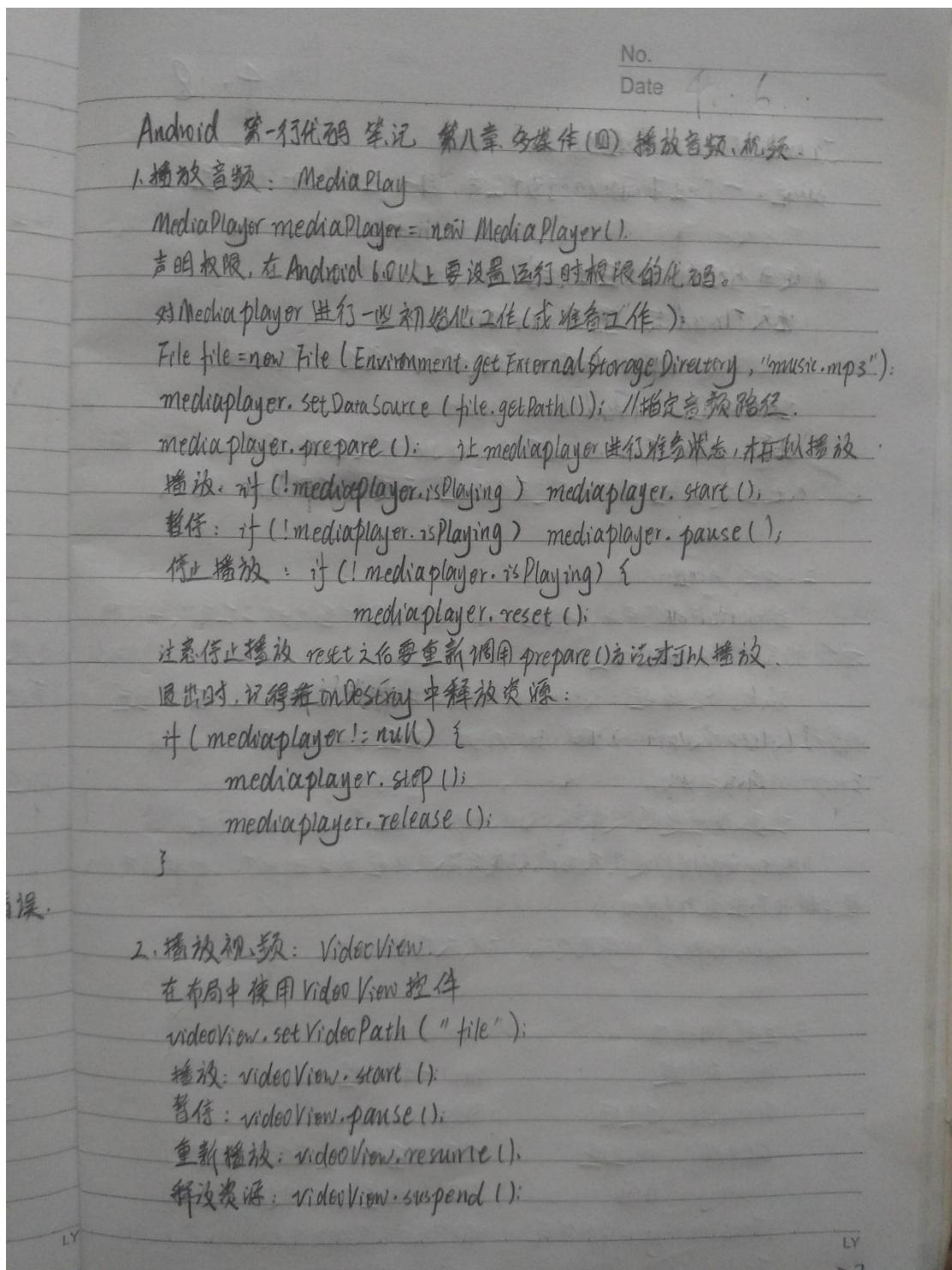
show error message

}

因此 try 语句块只有一个职责：关闭输入流。

外层 try 语句块职责：报告出现的错误；报告 finally 子句中出现的错误。

173. 第一行代码 第八章 多媒体 音频视频笔记



174. Java 多线程

No.
Date 9. 8

Java笔记：多线程 (-).

线程：一个程序同时执行多个任务，每一个任务称为一个线程。

1. 使用线程的两种基本方式：

继承 Thread 类，重写 run 方法。
实现 Runnable 接口，用该对象去实例化 Thread。
启动线程：threadObject.start();
注意：不能使用 Thread 或 Runnable 对象直接调用 run 方法，调用后仍然是当前线程而不会开启新线程，而应该使用 start() 方法。

2. 中断线程：

当 run 方法执行完毕，并经由 return 语句返回或者出现了在方法中没有捕获的异常时，线程终止。

强制终止线程：interrupt 方法。中断状态将被置位。在一个阻塞的线程（sleep 或 wait...）调用 interrupt，阻塞调用将会被 InterruptedException 异常中断。

检查线程是否被中断：interrupted 和 isInterrupted 方法。
interrupted() 是静态方法，会消除该线程的中断状态，即把当前线程中断状态重置为 false。
isInterrupted() 实例方法，不会改变中断状态。

3. 进程的状态：

NEW 新创建	new 一个线程。
Runnable：可运行	调用 start 方法。
Blocked：被阻塞	索取被其他线程持有的对象锁。
Waiting：等待	等待另一线程通知调度器一个条件。
TimeWaiting：计时等待	调用带超时参数的方法，如 sleep、wait、join。
Terminated：被终止	调用方法正常退出，没有捕获的异常。



4. 通用

4. 线程的属性：优先级、守护线程、线程组、处理未捕获异常的处理器

优先级：继承父线程的优先级，或通过 `setPriority` 设置优先级：

`MIN_PRIORITY : 1; MAX_PRIORITY : 10; NORM_PRIORITY : 5.`

中没有

守护线程：为其他线程提供服务，永远不去访问固有资源，如文件、数据源

`thread.setDaemon(true);`

阻塞

d

当前线

t.join() LY

LY

75

176. Java 多线程 同步

Java 笔记 多线程 (二) 同步

1. 先决条件：两个或两个以上的线程需要共享对同一数据的存取，并且每一个线程都调用了一个修改该对象状态的方法。

2. 锁对象：ReentrantLock类

```
myLock.lock();
```

```
try {
```

```
    // operation
```

```
} finally {
```

```
    myLock.unlock();
```

```
}
```

这一结构确保任何时刻只有 1 个线程进入临界区。当 1 个线程调用 lock() 后，其他线程无法通过 lock 调用，而进入阻塞状态，直到该线程释放锁对象。

如果使用锁就不能使用带资源的 try 块句？

锁保持一个持有计数来跟踪对 lock 方法的嵌套调用。线程在每一次调用 lock 都要调用 unlock 来释放锁。

3. 条件对象：myLock.newCondition();

```
myLock.lock();
```

```
try { while (...) myCondition.await(); }
```

```
...
```

```
myCondition.signalAll();
```

```
} finally {
```

```
    myLock.unlock(); // 如果临界区抛出异常，必须解锁，否则其他线程会阻塞
```

（如果临界区抛出异常，必须解锁，否则其他线程会阻塞）

当调用 `cancle()`，当前线程被阻塞，并放弃了锁，进入该条件的等待集。
 直到另一线程调用同一条件上的 `signAll()` 为止。解除等待线程的阻塞状态
 当调用 `signAll()`，重新激活因为这一条件等待的所有线程。
`signal()` 方法，随机解除等待集中的某个线程的阻塞状态。如果随机选择的线程发现自己仍然不能运行，再次被阻塞。如果没有其他线程调用 `signal`，那么系统就死锁。

4. synchronized 同步方法

锁和条件小结：

① 锁用来保护代码片段，任何时候只能有一个线程执行被保护的代码。

锁可以管理试图进入被保护代码段的线程

锁可以拥有一个或多个相关的条件对象

每个条件对象管理那些已经进入被保护代码段但不能运行的线程

Java 中每一个对象都有一个内部锁。如果用 `synchronized` 声明一个方法，那么对象的锁将保护整个方法。也就是说，要调用该方法，线程必须获得内部的对象锁。

次
`public synchronized void methodName() {
 // operation
 }`

等价于

```
public void methodName() {  

    lockObj.lock();  

    try {  

        // operation  

    } finally {  

        lockObj.unlock();  

    }  

}
```

synchronized 方法体内的 wait() 方法，相当于 myCondition.await();
将一个线程添加到等待集中；notifyAll / notify 相当于 myCondition.signalAll();
解除等待线程的阻塞状态。

wait, notifyAll, notify 都是 Object 类的 final 方法
内部对象锁只有一个相关条件。

也可以将静态方法声明为 synchronized。

内部锁和条件的局限性：

不能中断一个正在试图获得锁的线程。

试图获得锁不能设定超时。

每个锁仅有单一的条件，可能是不够的。

5. 同步阻塞

通过进入一个同步阻塞，可以获锁。如。

```
synchronized (obj) {  
    // operation  
}
```

3

客户

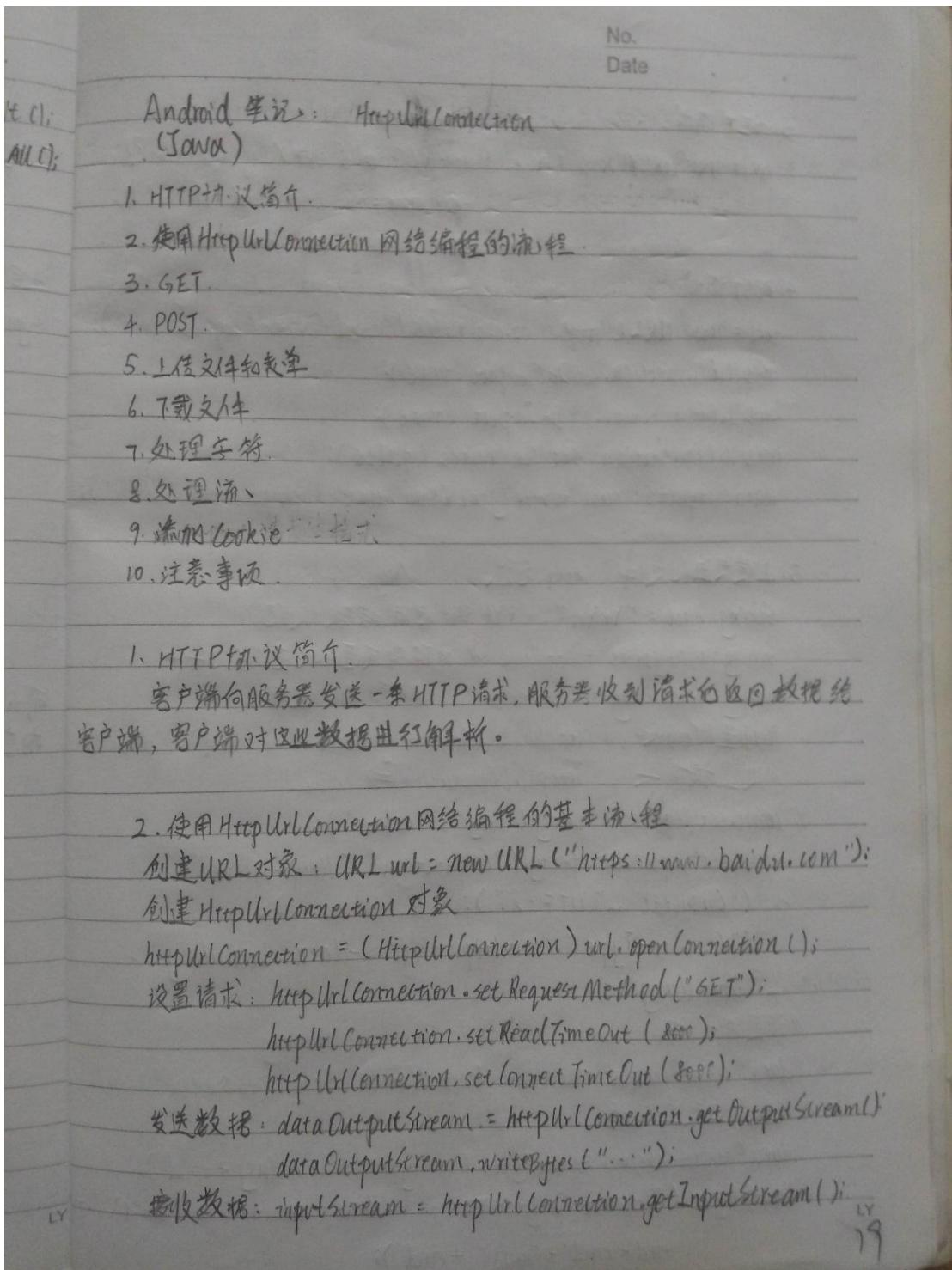
6. 阻塞队列

对于多线程问题，可以使用一个或多个队列。

生产者线程向队列插入元素，消费者线程则取出它们。

当试图向队列添加元素而队列已满，或想从队列移出元素而队列为空的时候，阻塞队列导致线程阻塞。

179. Android Java HttpURLConnection



3. GET 请求

```
url = new URL("http://172.0.0.1:8080/SerialServer/LoginServlet?" +
    "username=xm&password=dmxm");
```

4. POST 请求

```
url = new URL("http://172.0.0.1:8080/SerialServer/LoginServlet");
HttpURLConnection.setDoOutput(true);
HttpURLConnection.setDoInput(true);
HttpURLConnection.setUseCaches(false);
DataOutputStream.writeBytes("username=xm&password=dmxm");
```

5. 上传复杂文件：POST 方法：/RegistServlet

```
String end = "\r\n"; // 请求体的特殊换行，\r\n两个  

String twoHypens = "--"; // 分部符前面和后面附加字符串  

String boundary = "*****"; // 分部符，任意  

HttpURLConnection.setChunkedStreamingMode(128 * 1024); // 128KB  

// 设置请求头： HttpURLConnection <= conn.  

// conn.setRequestProperty("key", "value");  

conn.setRequestProperty("Connection", "Keep-Alive");  

~ ("Charset", "UTF-8");  

("Content-Type", "multipart/form-data; boundary=" +  

boundary);  

// 设置请求体  

DataOutputStream = new DataOutputStream(conn.getOutputStream());  

// 设置简单表单数据  

DataOutputStream.writeBytes(twoHypens + boundary + end);  

.writeBytes("Content-Disposition: form-data; name='username'" +  

end + end + "xm" + end);
```

```
    .writeBytes(twoHypens + boundary + end);
    .writeBytes("Content-Disposition: form-data; name=\"password\""
                + end + end + "dixm" + end);
    // 设置请求往的文件 附加
);
    .writeBytes(twoHypens + boundary + end);
    .writeBytes("Content-Disposition: form-data; name=\"photo\"";
                filename = "zm.jpg" + end + end);
    fis = new FileInputStream(path + filename); D��-13
");
    byte[] buffer = new byte[1024]; // 1KB
    int count;
    while ((count = fis.read(buffer)) != -1) {
        dataOutputStream.write(buffer, 0, count);
    }
    .writeBytes(end);
13
    .writeBytes(twoHypon + boundary + twoHyphen + end);
    dataOutputStream.flush();

```

6.7 下载文件 / 保存文件.

```
url = new URL("http://127.0.0.1:8080/SocialServer/apk/file_name.apk");
conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
byte[] b = new byte[1024];
bufferInputStream = new BufferedInputStream(conn.getInputStream());
byteArrayOutputStream = new ByteArrayOutputStream();
int len;
" + LY
LY  
8
```

或者直接使用
fileOutputStream.write(b, 0, len);
while ((len = bufferedInputStream.read(b)) > 0) {
 byteArrayOutputStream.write(b, 0, len);
}
fileOutputStream = new FileOutputStream(new File(pathName));
fileOutputStream.write(byteArrayOutputStream.toByteArray());
// 最后关闭流。

7. 处理字符

```
InputStreamReader = new InputStreamReader(inputStream);  
bufferedReader = new BufferedReader(InputStreamReader);  
StringBuilder = new StringBuilder();  
String line;  
while ((line = bufferedReader.readLine()) != null) {  
    StringBuilder.append(line);  
}  
String result = stringBuilder.toString();
```

8. 汉堡 Cookie

```
httpURLConnection.setRequestProperty("Cookie", "id=1;" +  
    "JSESSIONID=" + "session_id_value");
```

9. 注意事项:

- ① 使用 multipart 上传，Servlet 要有 @MultipartConfig。
- ② multipart 请求体最后的分隔符行要有"--"