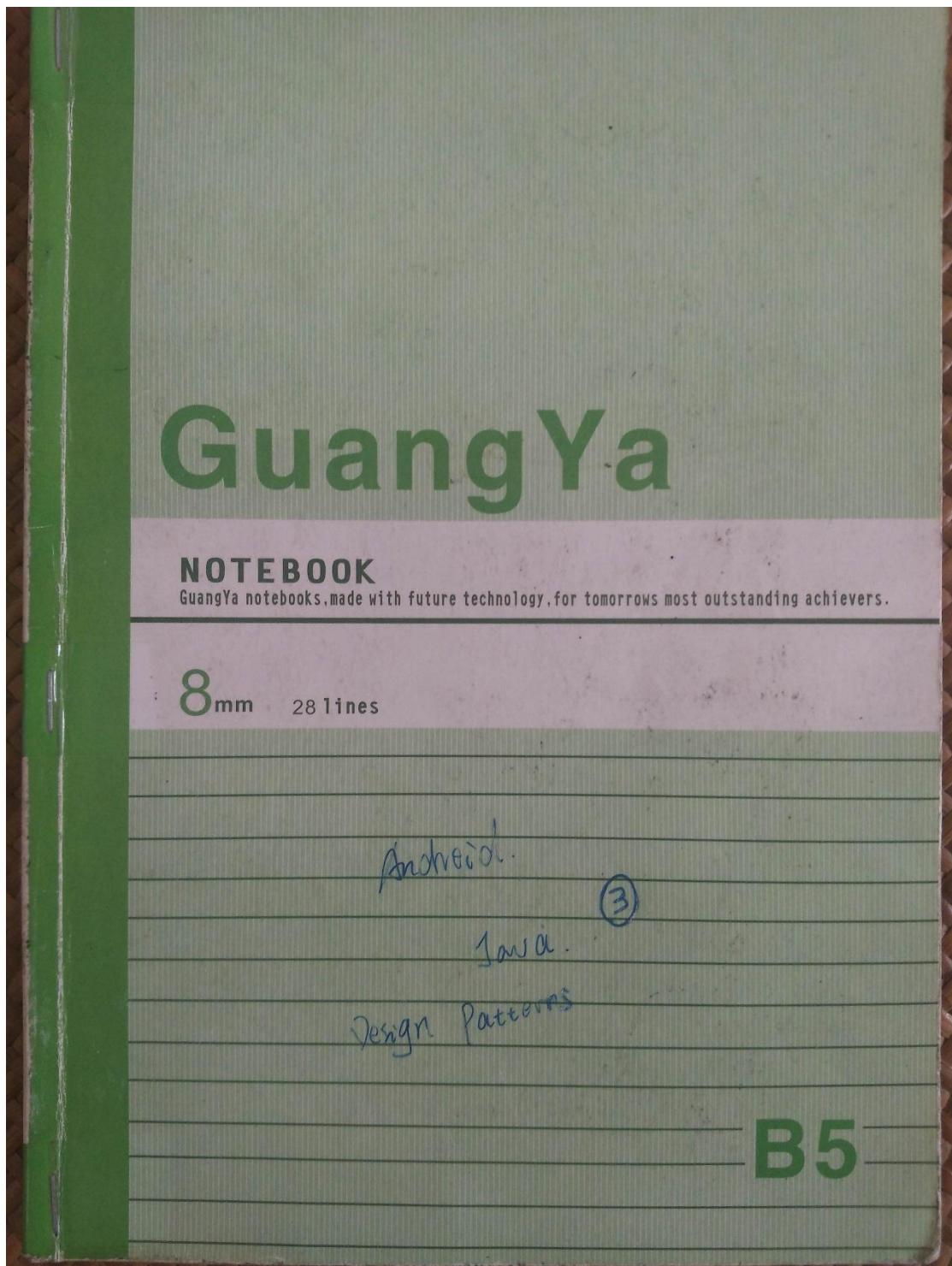


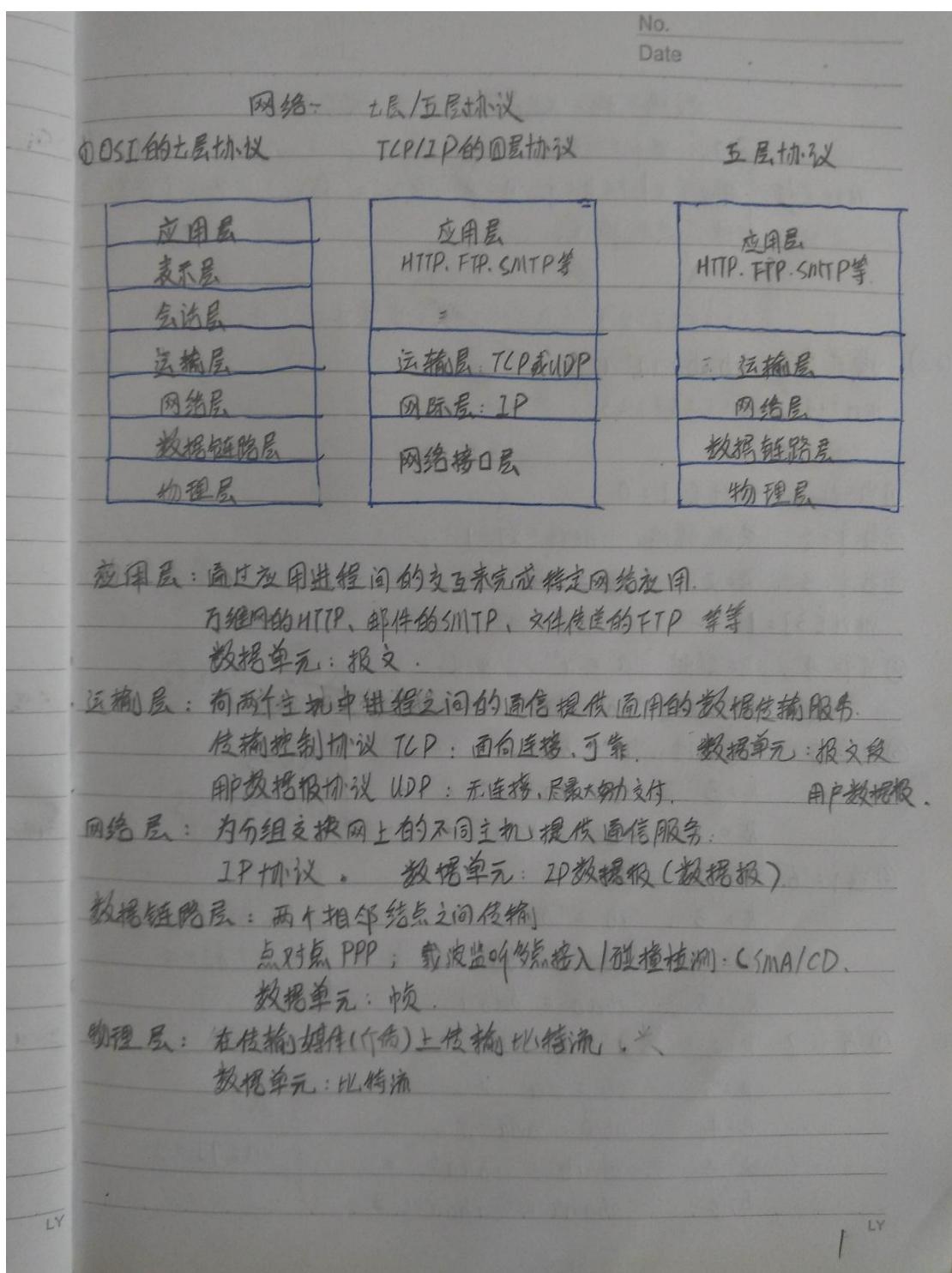
NoteBook-3



目录

301.网络 七层/五层协议	3
302.数据结构 KMP 算法 next 数组推导	4
304.Java ArrayList 和 LinkList 区别、线程安全、List、Set、Map	6
305.设计模式 六大原则.....	7
308.操作系统 生产者-消费者问题	10
310.Java HashMap 和 HashTable 区别.....	11
311.数据结构 散列.....	12
312.Android 群英传 第三章 自定义 View 笔记.....	13
313.Android 群英传 第三章 自定义 ViewGroup 笔记.....	14
314.Android 群英传 第三章 事件分发拦截 笔记.....	15
317.Android 自定义 view 常用构造函数 参数作用	18
318.Android 群英传 第四章 ListView 优化 笔记	19
319.Android 群英传 第四章 ListView 滑动监听 笔记	20
320.Android 群英传 第四章 ListView 扩展 笔记	21
321.Android RecyclerView 和 ListView 区别.....	22
322.Android 群英传 第五章 Scroll 分析	23
326.Android 群英传 第六章 绘图机制 笔记.....	27
329.Android 群英传 第六章 SurfaceView 笔记.....	30
331.Network TCP 三次握手，四次挥手.....	32
334.Java hashCode()和 equals()区别	35
335.Android 群英传 第七章 动画 笔记.....	36
339.Android 群英传 第八章 Activity 启动模式 笔记.....	40
341.Android 群英传 第九章 Android 系统消息，安全机制	42
345.Java 静态块，静态方法.....	45
346.Android 面试 Activity 退出保存数据 文件读写.....	46
348.Android 面试 ANR 及处理.....	48
350.Android 面试 RxJava 优缺点、序列化	50
351.Java 面试 序列化.....	51
353.Android 面试 View 绘制原理.....	53
357.算法 冒泡排序算法.....	56
358.算法 直接插入排序算法.....	57
359.数据结构 顺序表和链表区别.....	58
360.算法 快速排序算法.....	59
362.Android 面试 Looper 原理	61
363.算法 希尔排序算法.....	62
364.算法 二分查找.....	63
365.Android 面试 Bitmap OOM	64
366.Android 使用线程方式及基本原理.....	65
367.Network http 协议简单介绍	66
370. Android 面试题.....	68
373.Java 面试题.....	70
375.数据机构/算法 网络	71

301. 网络 七层/五层协议



302. 数据结构 KMP 算法 next 数组推导

No.
Date

数据结构 - KMP 算法 next 数组推导

② 当 $j=1$
 $\text{next}[j] = \begin{cases} 0, & \text{当 } p_1 \dots p_{k+1} = p_{j+k+1} \dots p_{j+k+1} \\ \max\{k | 1 \leq k \leq j, \text{且 } p_1 \dots p_{k+1} = p_{j-k+1} \dots p_{j+1}\}, & \text{其他情况} \end{cases}$

j 1 2 3 4 5 6 7 8 9 读：串下标从 1 开始
(3) 模式串 $p: ababaaaba$ ③ 当 $j=2$
 $\text{next}[j] = 0 1 1 2 3 4 2 2 3$

① 当 $j=1$, $\text{next}[1]=0$

② 当 $j=2$, 其他情况, $\text{next}[2]=1$

③ 当 $j=3$, $k=2$, ' $p_{2+1} = 'a'$ ', ' $p_{3+1} = 'b'$ ', ' $a = b$ '? 不等, 其他情况
 $\text{next}[3]=1$

④ 当 $j=4$, $k=2$ 时, ' $a = a$ ', $k=2$. 取 $k=2$, $\text{next}[4]=2$.
 $k=3$ 时, ' $ab = ba$ '? \neq , 总结

⑤ 当 $j=5$, $k=2$ 时, ' $a = b$ ' \neq . 取 $k=3$
 $k=3$, ' $ab = ab$ ', $k=3$. 取 $k=3$, $\text{next}[5]=3$
 $k=4$, ' $abc = bab$ ' \neq ,

⑥ 当 $j=6$, $k=2$, ' $a = a$ ', $k=2$. 取 $k=4$.
 $k=3$, ' $ab = ba$ ' \neq
 $k=4$, ' $aba = aba$ ', $k=4$. 取 $k=4$, $\text{next}[6]=4$
 $k=5$, ' $abab = babab$ ' \neq

⑦ 当 $j=7$, $k=2$, ' $a = a$ ', $k=2$. 取 $k=2$
 $k=3$, ' $ab = aab$ ', \neq
 $k=4$, ' $aba = baa$ ', \neq
 $k=5$, ' $bab = abaa$ ', \neq
 $k=6$, ' $ababa = babaa$ ', \neq 取 $k=2$, $\text{next}[7]=2$

空 况	⑧ 当 $j=8$, $k=2$, ' $\alpha' = '01'$	$k=2$	\rightarrow 取 $k=2$ $\text{next}[k] = 2$
	$k=3$, ' $ab' = '01a'$	\neq	
	$k=4$, ' $aba' = '01aa'$	\neq	
	$k=5$, ' $abab' = '01aaa'$	\neq	
	$k=6$, ' $ababa' = '01aaaa'$	\neq	
	$k=7$, ' $ababaa' = '01aaaaa'$	\neq	
	⑨ 当 $j=9$, $k=2$, ' $\alpha' = 'b'$	\neq	
	$k=3$, ' $ab = 'b'0'$	\neq	
况	$k=4$, ' $aba = 'b'a0'$	\neq	\rightarrow 取 $k=3$ $\text{next}[k] = 3$
	$k=5$, ' $abab = 'b'aab'$	\neq	
	$k=6$, ' $ababa = 'b'aabab'$	\neq	
	$k=7$, ' $ababaa = 'b'aabaaab'$	\neq	
	$k=8$, ' $ababaaa = 'b'aabaaab'$	\neq	
总结: ' $p_{j-k+1} \dots p_{j-1}$ ' = ' $p_{j-k+1} \dots p_{j-1}$ ' 是前缀与后缀的比较。			
3 LY			

304. Java ArrayList 和 LinkedList 区别、线程安全、List、Set、Map

Java.

① ArrayList 和 LinkedList 的实现和区别。

ArrayList：由数组实现的 List。

优点在于对元素进行快速随机访问；但插入删除元素较慢。

LinkedList：由链表实现的 List。

优点在于插入和删除元素很快；随机访问则相对较慢。

② 如何保证线程安全？

使用锁或者同步，还有 volatile，尽量少用共享变量。

③ List、Set、和 Map 的区别。

Java 的集合主要有三种，就是 List（列表）、Set（集）、Map（映射）。

其中 Set、和 List 都继承了 Collection、Map。Collection 是最基础的集合接口。

Set：是一种最简单的集合，集合中的对象不按特定的方式排序，并且没有重复对象。存放的是对象的引用。

List：特征是其元素以线性方式存储，集合中可以存放重复的对象。

Map：是一种把键对象和值对象映射的集合，它的每一个元素都包含一对键对象和值对象，结构键对象，返回值对象。

三者的区别：

1. List、Set 都继承 Collection 接口。

2. List，元素有放入顺序，且可重复。

Set，元素无放入顺序，不可重复。（注意其内部顺序由 hashCode 决定，顺序）

Map，元素按键值对存储，无放入顺序。

3. List 用来处理序列，Set 用来处理集，Map 是用来存储键值对。

305. 设计模式 六大原则

Date

设计模式 — 六大原则

1. 单一职责原则 SRP (Single Responsibility Principle)

定义：就一个类而言，应该只有一个引起它变化的原因。

一个类中应该是一组相关性很高的函数、数据的封装，即满足高内聚的要求。两个完全不一样的功能就不应该放在一个类中。

例如在Activity中可能要访问数据库，又要请求网络，我们不应该在Activity中编写实现代码，而应该把这些实现放到某个数据库类和网络请求类当中。

2. 开闭原则 OCP (Open Close Principle)

定义：软件中的对象（类、模块、函数等）应该对于扩展是开放的，对于修改是封闭的。

• 软件需求变化时，尽量通过扩展的方式实现变化，而不是通过修改已有代码。

例如：MemoryCache 和 DiskCache 的缓存实现不一样，但它们都实现了 ImageCache 这个接口，根据用户的需求，通过 setImageCache (ImageCache cache) 方法使用不同的缓存策略。假如某一天，要添加双缓存策略，就新增 DoubleCache，实现 ImageCache 接口，从而实现双缓存的具体实现，调用 setImageCache 就能轻松地更改缓存策略。

• 因此，遵循开闭原则的重要手段应该是通过抽象。

三

3. 里氏替换原则 LSP (Liskov Substitution Principle)

定义：所有引用基类的地方必须能透明地使用其子类对象。

• 里氏替换原则就是依赖于继承、多态两大特性。

• 父类能出现的地方子类就可以出现，而且替换子类不会产生任何错误或异常。

• 最终归结于，抽象。

• 开闭原则和里氏替换原则往往是生死相依、不离不弃的，通过里氏替换

未达到对扩展开放，对修改封闭的效果。

例如：Student 和 Teacher 都继承了 Person 类。

Person person = new Student();

也可以写为

Person person = new Teacher();

而不产生任何错误和异常。

再例如：Memory Cache 和 Disk Cache 都实现了 ImageCache 接口。

Memory Cache 中的一方法定义如下：

public void setImageCache (ImageCache cache) { ... }

我们可以这样

ImageCache.setImageCache (new MemoryCache());

也可以这样写为

ImageCache.setImageCache (new DiskCache());

4. 依赖倒置原则 DIP (Dependence Inversion Principle)

定义：依赖倒置原则指代了一种特定的解耦形式，使得高层模块不依赖于低层模块的实现、细节的目的，依赖模块被颠倒了。

关键点：高层模块不应该依赖低层模块，两者都应该依赖其抽象。
抽象不应该依赖细节；

细节应该依赖抽象。

抽象：接口或抽象类。

高层模块：调用端

细节：实现类

低层模块：具体实现类

实现类之间不发生直接的依赖关系，其依赖关系是通过接口或抽象类产生的。这就是面向接口编程或面向抽象编程，是面向对象编程精髓之一。

例如 ImageLoader 中的一个方法。

public void setImageCache (ImageCache cache) { ... }

就是依赖于抽象。

5. 接口隔离原则 ISP (Interface Segregation Principle)

定义：客户端不应该依赖它不需要的接口

类间的依赖关系应该建立在最小的接口上。

简单地说就是客户端依赖的接口尽可能少。

例如：public static void closeQuietly (Closeable closeable) {
if (closeable != null)

try {

closeable.close();

} catch (IOException e) {

e.printStackTrace();

}

}

这个方法的基本原理就是依赖于 Closeable 抽象（依赖倒置原则），并且建立在最小化依赖原则的基础上，它只需要知道这个对象是可关闭的，其他一概不依赖关心，也就是接口隔离原则。

* SOLID 原则：单一职责、开闭原则、里氏替换、接口隔离、依赖倒置

6. 迪米特原则 LOP (Law of Demeter)

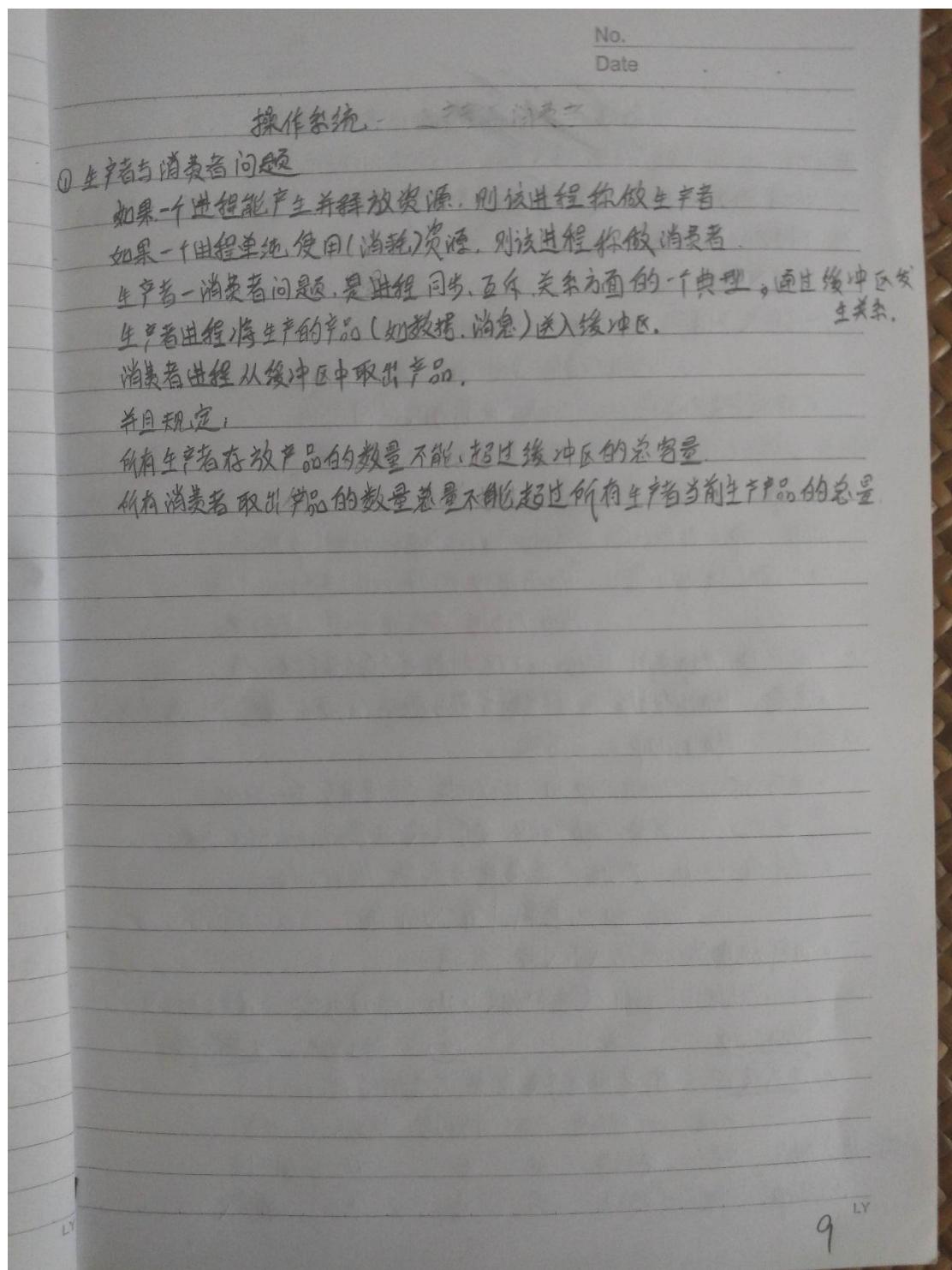
最少知识原则 (Least Knowledge Principle)

定义：一个对象应该对其他对象有最少的了解。

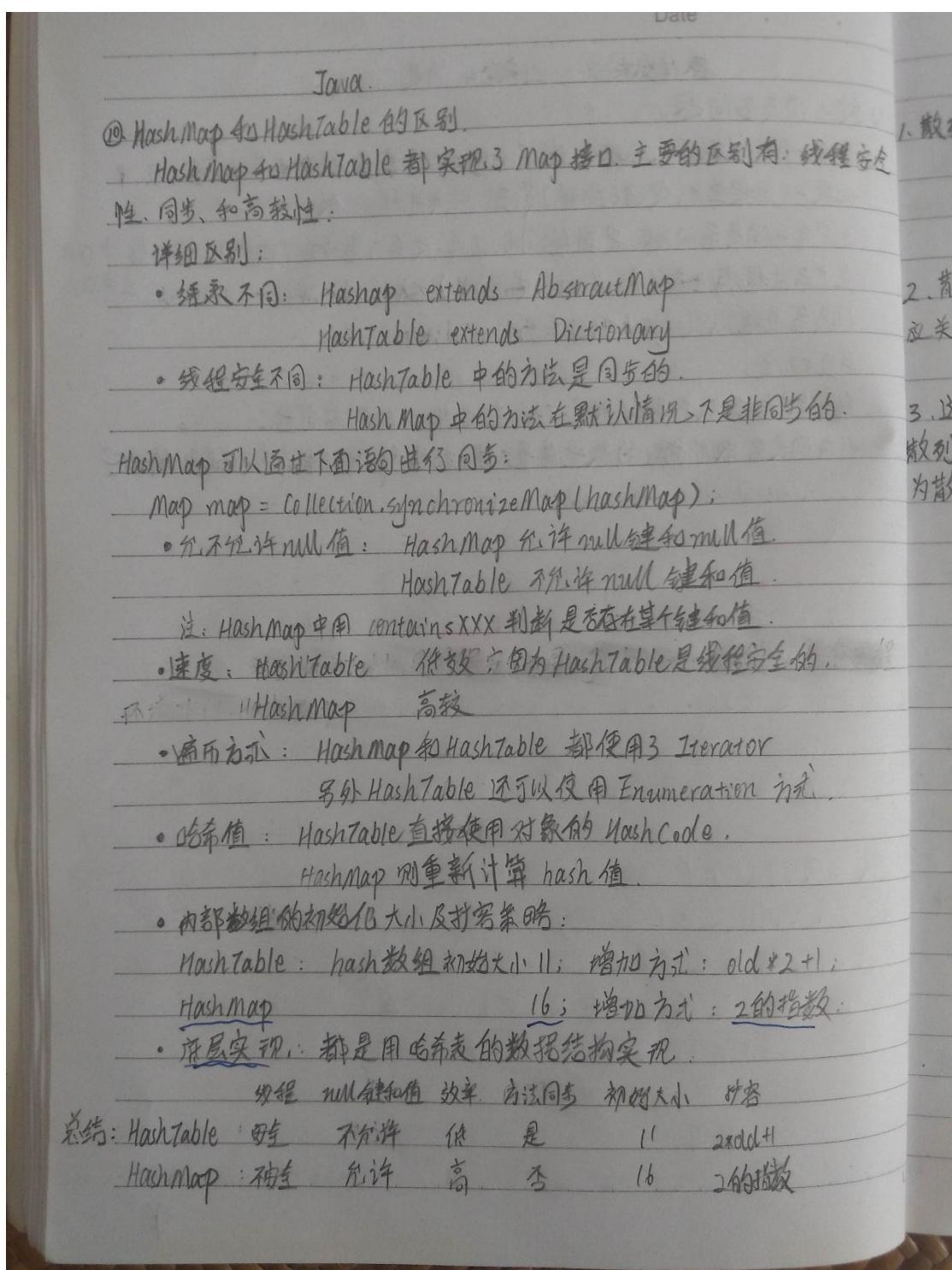
举例：我们买房子的时候，是通过中介找而找房子的工作应该是中介去完成的，我们只需把条件告知中介，中介就会把找到的房子归给我们。

或
精

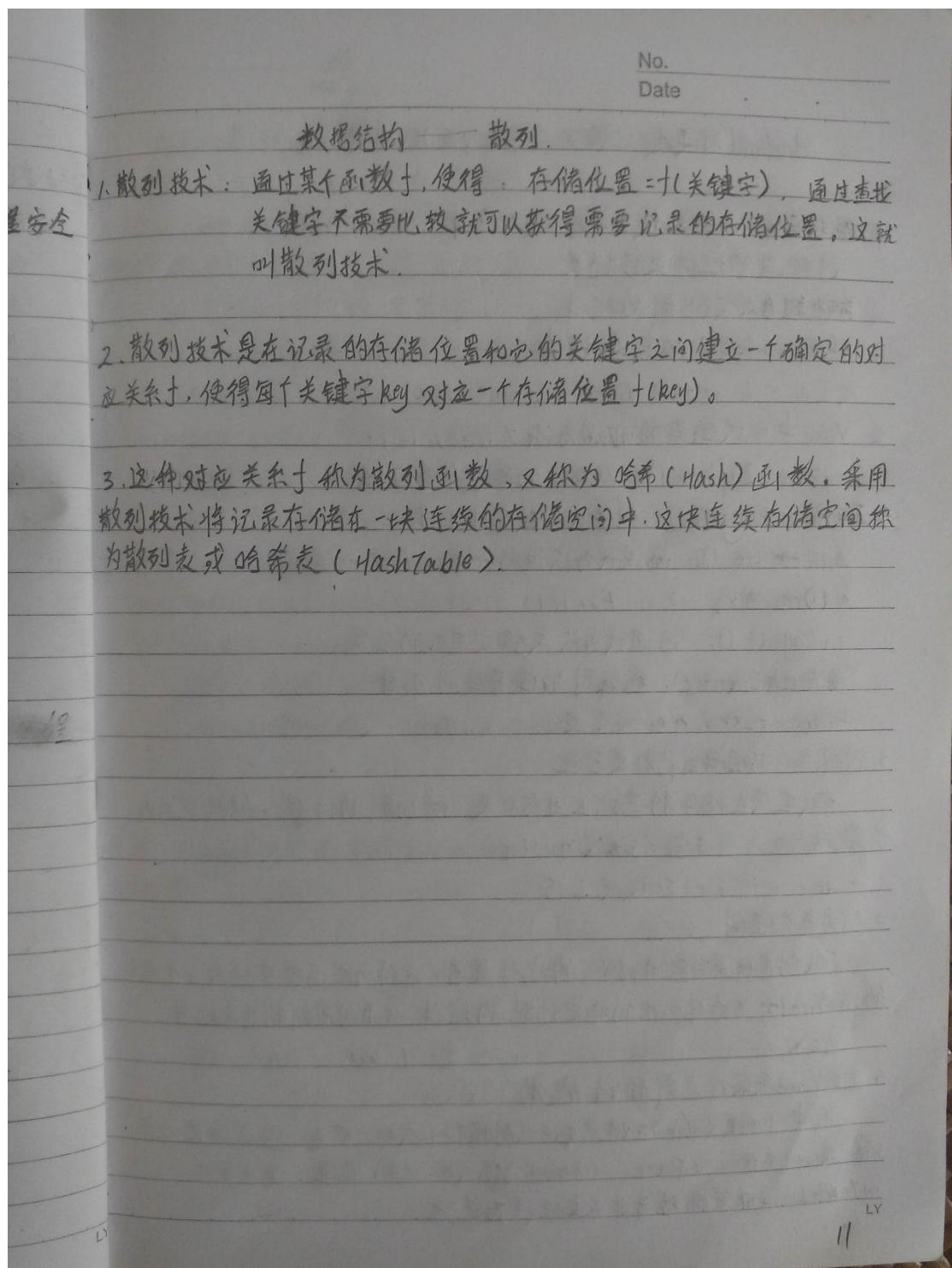
308. 操作系统 生产者-消费者问题



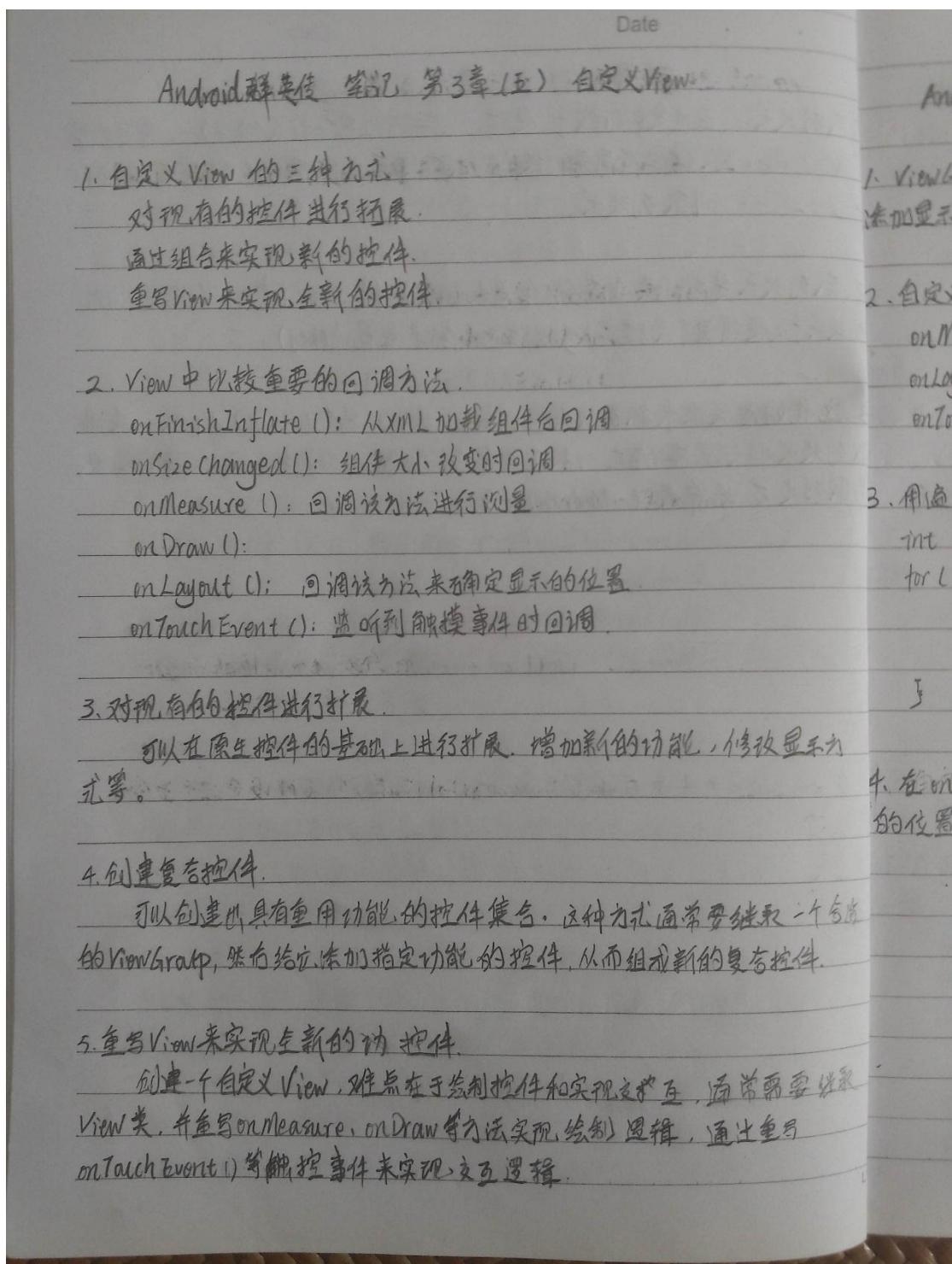
310. Java HashMap 和 HashTable 区别



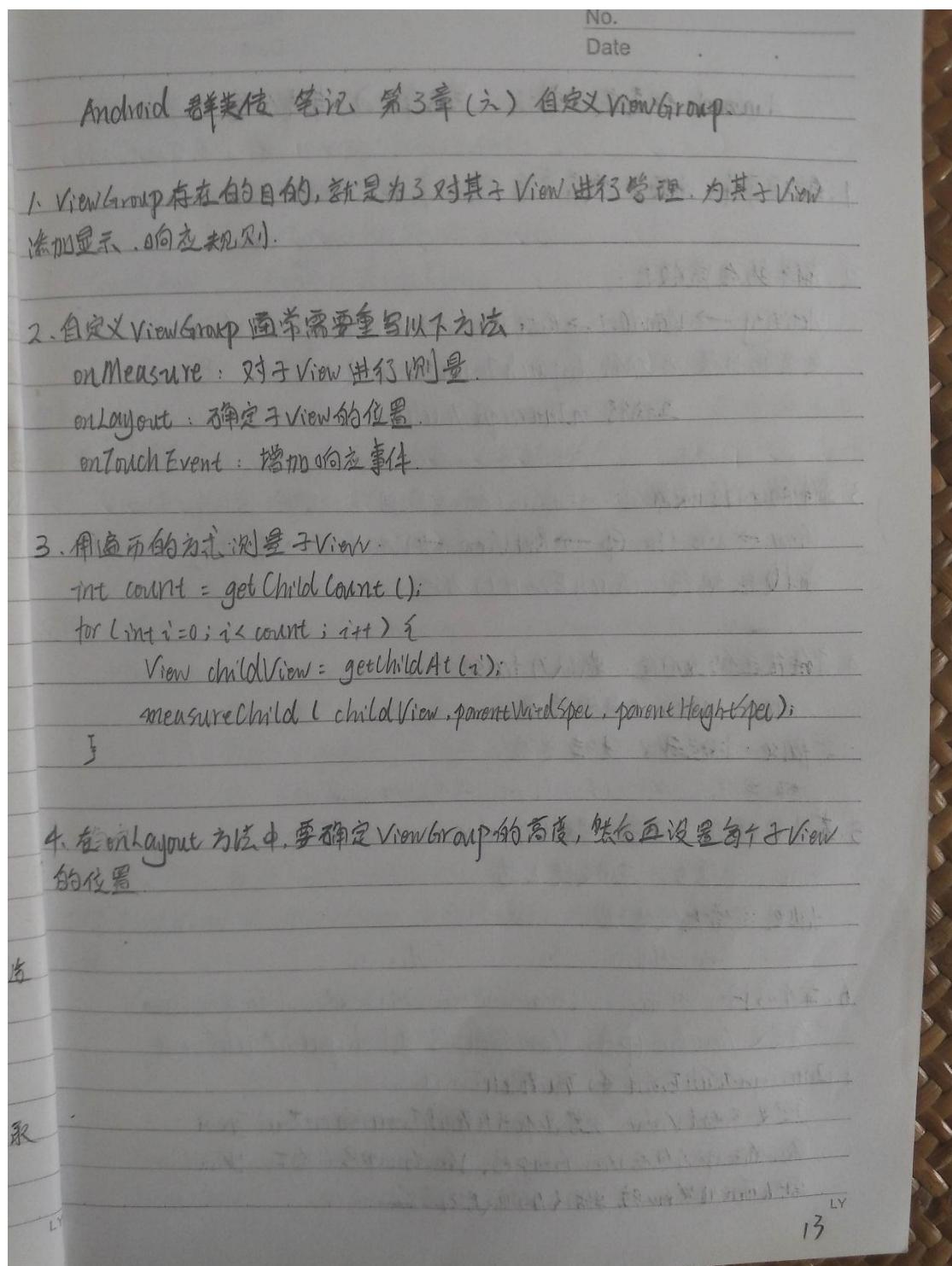
311. 数据结构 散列



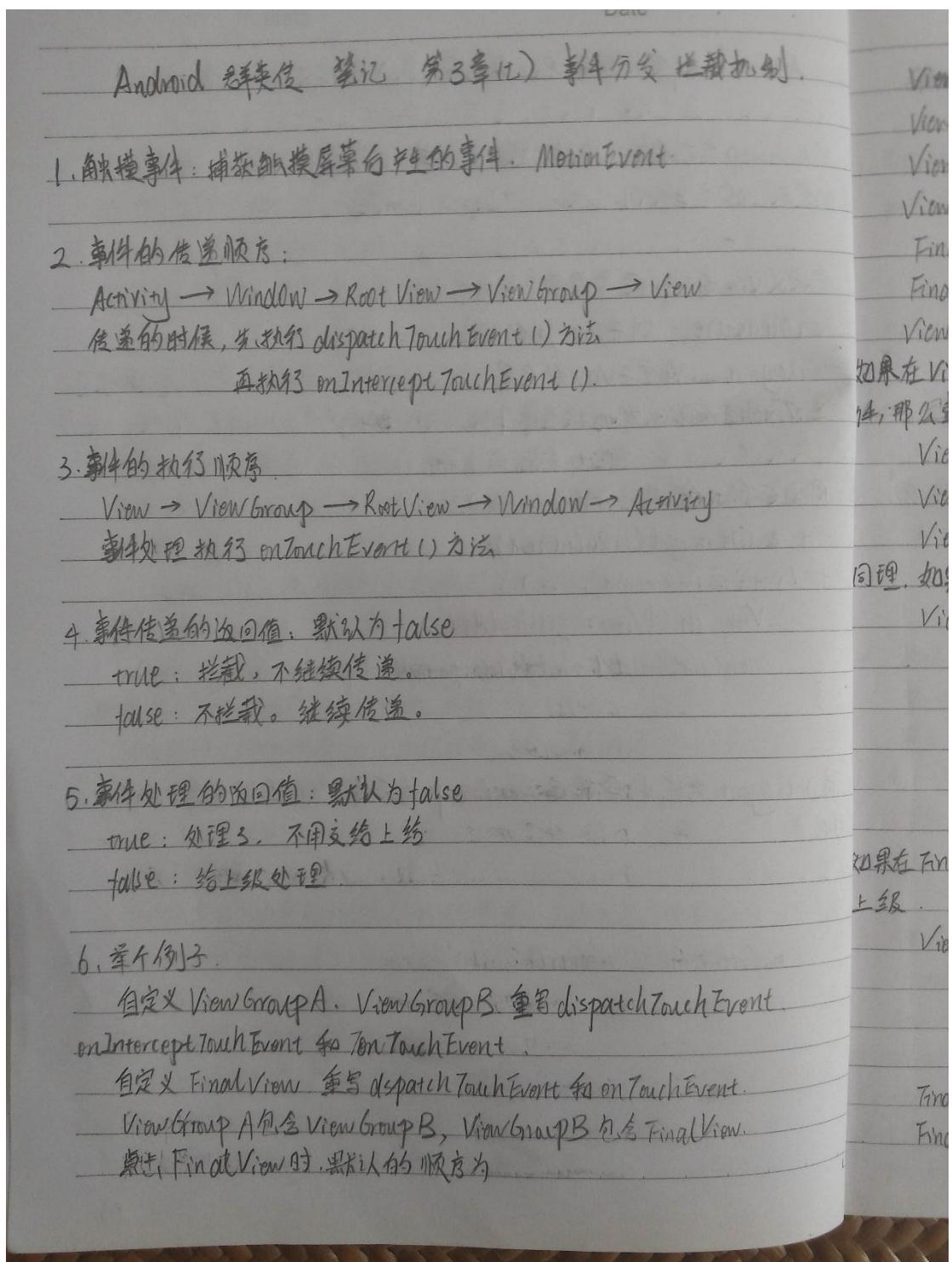
312.Android 群英传 第三章 自定义 View 笔记



313.Android 群英传 第三章 自定义 ViewGroup 笔记



314.Android 群英传 第三章 事件分发拦截笔记



ViewGroup A dispatchTouchEvent.
ViewGroup A onInterceptTouchEvent.
ViewGroup B dispatchTouchEvent.
ViewGroup B onInterceptTouchEvent.
Final View dispatchTouchEvent.
Final View onTouchEvent.

ViewGroup B onTouchEvent / ViewGroup A onTouchEvent

如果在 ViewGroup A 的 onInterceptTouchEvent 中返回 true，即拦截该事件，那么事件会在 ViewGroup B 的 onTouchEvent 中执行。

ViewGroup A dispatchTouchEvent
ViewGroup A onInterceptTouchEvent
ViewGroup A onTouchEvent

同理，如果在 ViewGroup B 的 onInterceptTouchEvent 中返回 true，

ViewGroup A dispatchTouchEvent.
A onInterceptTouchEvent
B dispatch
B onIntercept
B onTouchEvent
A onTouchEvent

如果在 Final View 的 onTouchEvent 中返回 true，即处理该事件行，不交给上级。

ViewGroup A dispatchTouchEvent
A onInterceptTouchEvent
B dispatch
B onIntercept
Final View dispatch
Final View onTouchEvent.

No.

Date

同理，如果 ViewGroupB 的 onTouchEvent 中返回 true，则

ViewGroupA dispatchEvent
A onInterceptEvent
B dispatch
B onIntercept
Final View dispatch
Final View onTouchEvent
B onTouchEvent

如果 Final View 设置了 onTouchListener，那么它的
onTouch 方法会调用。如果该方法返回 true：那么 onTouchEvent 不会调用
false： onTouchEvent 会调用

317. Android 自定义 view 常用构造函数 参数作用

Android 自定义view 常用构造函数，及参数作用

1. 常用的三个构造函数

```
public MyView (Context context) {  
    super (context);
```

}

```
public MyView (Context context, AttributeSet attrs) {  
    super (context, attrs, 0);
```

}

```
public MyView (Context context, AttributeSet attrs,  
        int defStyle) {  
    super (context, attrs, defStyle);
```

}

2. 调用时机

第一个构造：在代码中直接 new MyView 实例的时候。

第二个构造：在 XML 中使用 MyView。

在 XML 中使用 MyView 并且使用了自定义属性。

第三个构造：系统默认只调用前两个构造，这个构造通常是在自己的
函数中调用第三个构造函数，加上。

3. 各参数作用：

Context：没什么好解释，上下文环境。

AttributeSet：属性值集合，在 XML 中使用的属性会封装到这里，然后调

用 context.obtainStyledAttributes (attrs, R.styleable.MyView) 获得

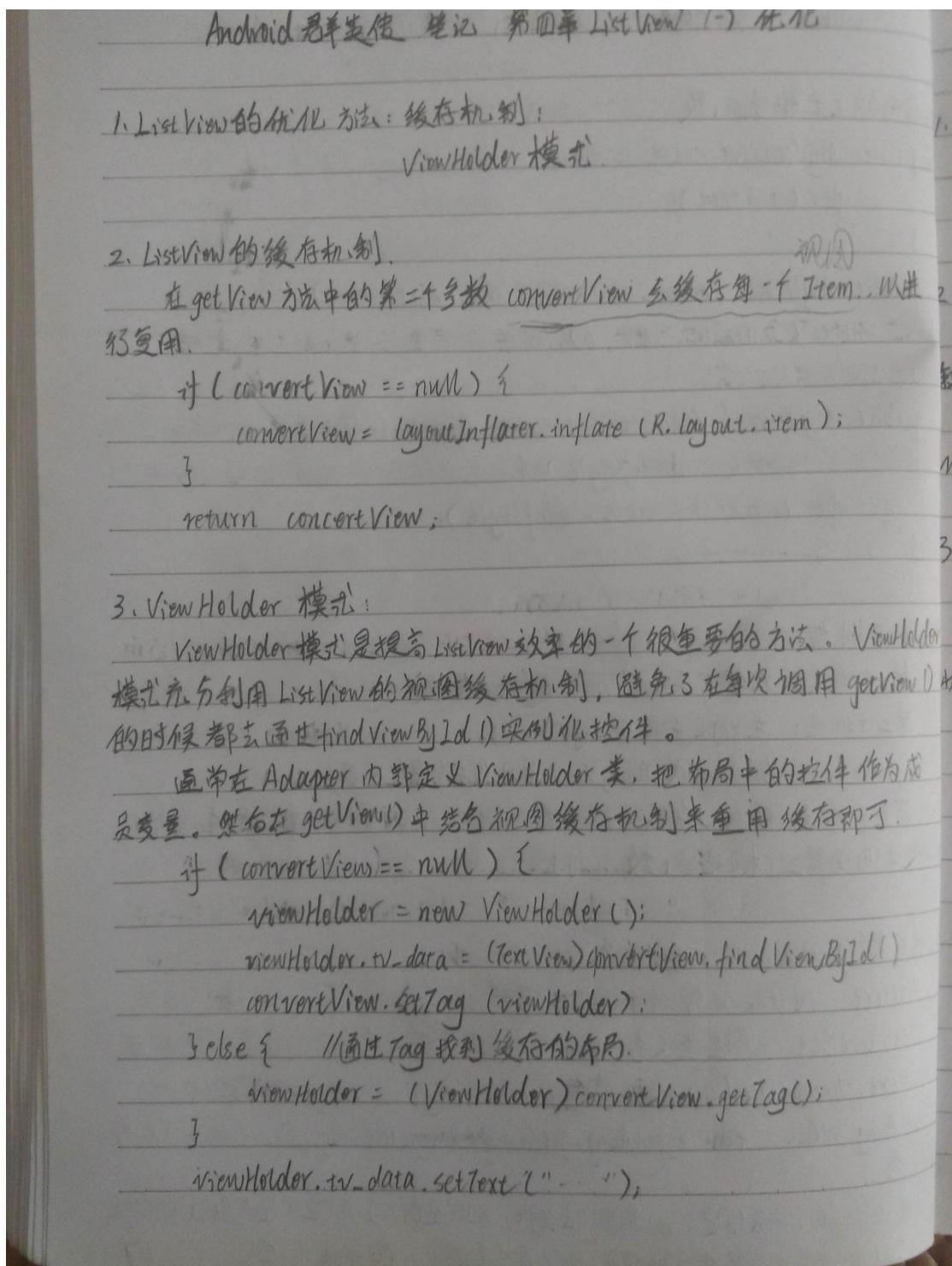
TypedArray 对象，调用 typedArray.getInt (R.styleable.MyView_attrName) 获得
属性值。

int defStyle：用来指定 View 的默认 style，如果是 0，则将不应用任何默认的 style。

该值可以是一个属性指定的 style 引用，也可以是一直接显示的 style 资源。

17

318.Android 群英传 第四章 ListView 优化笔记



319.Android 群英传 第四章 ListView 滑动 监听 笔记

Date

Android 群英传笔记 第四章 ListView (二) 滑动监听

1. 两种常用监听 ListView 滑动事件的方法

setOnTouchListener
setOnScrollListener.

2. setOnTouchListener 方式

根据触摸事件获取不同事件的触摸坐标值，对这些坐标值进行比较，实现不同的逻辑。

例如判断上滑、下滑：在 DOWN 事件中获取第一次触摸时的 y 坐标，在 MOVE 事件中获取移动时候的 y 坐标值，如果当前一次 > 0，则为下滑。

3. setOnScrollListener 方式：

(1) OnScrollListener 中有两个回调方法：

onScrollStateChanged(int scrollState)，根据它的参数 scrollState 来判定滚动的状态。scrollState 有三种值：

OnScrollListener.SCROLL_STATE_IDLE：停止滚动时。
- TOUCH_SCROLL：正在滚动时。
- FLING：由于惯性滚动时。

onScroll() 方法会在 ListView 滚动时一直调用，它有三个参数：

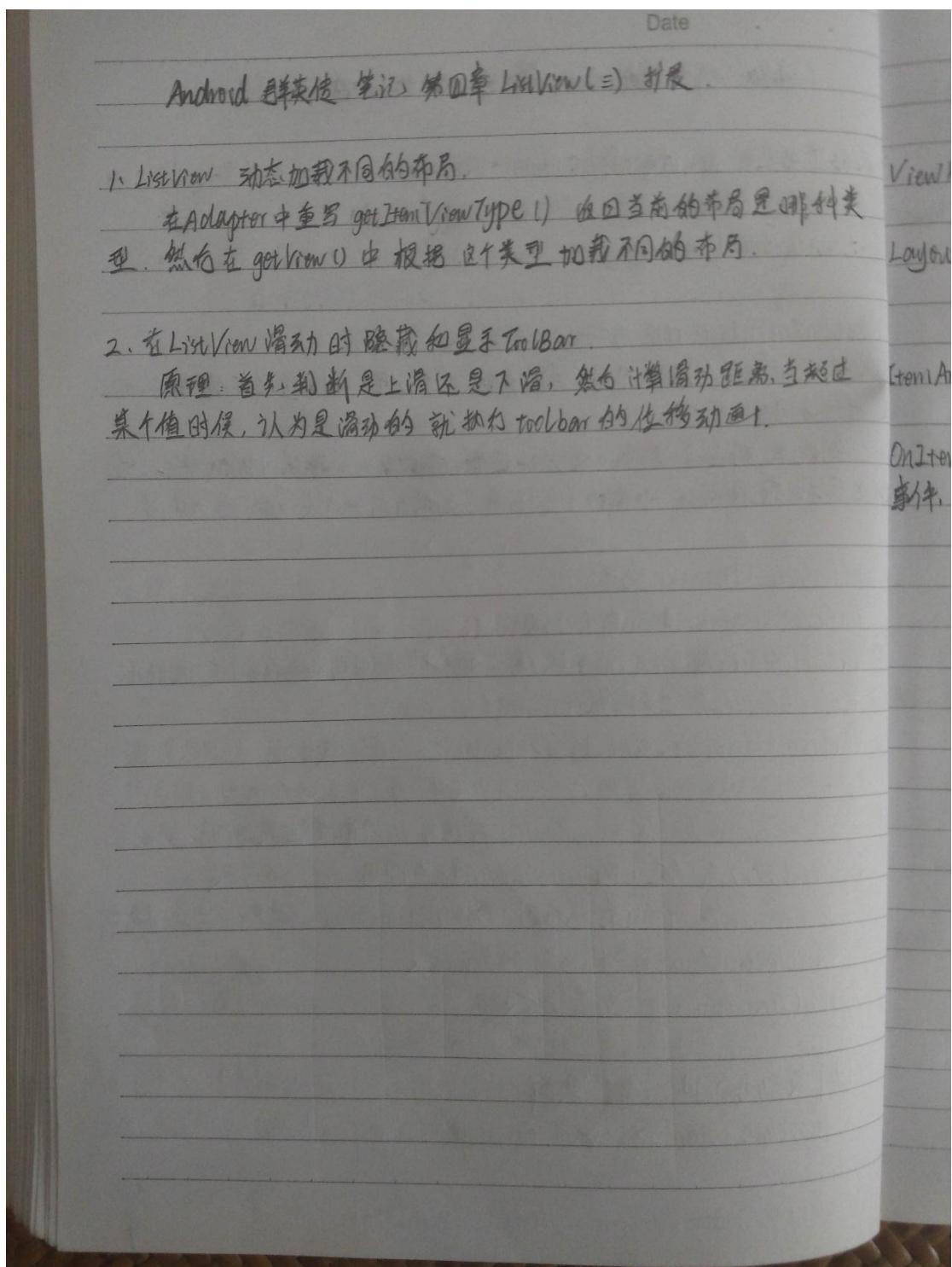
firstVisibleItem：当前能看到的第一个 Item 的 id，注意只显示一半也算。
visibleItemCount：能看到的 Item 数。
totalItemCount：Item 总数。

判断滚动方向时，只需要判断 firstVisibleItem > lastVisibleItemPosition 上滑。

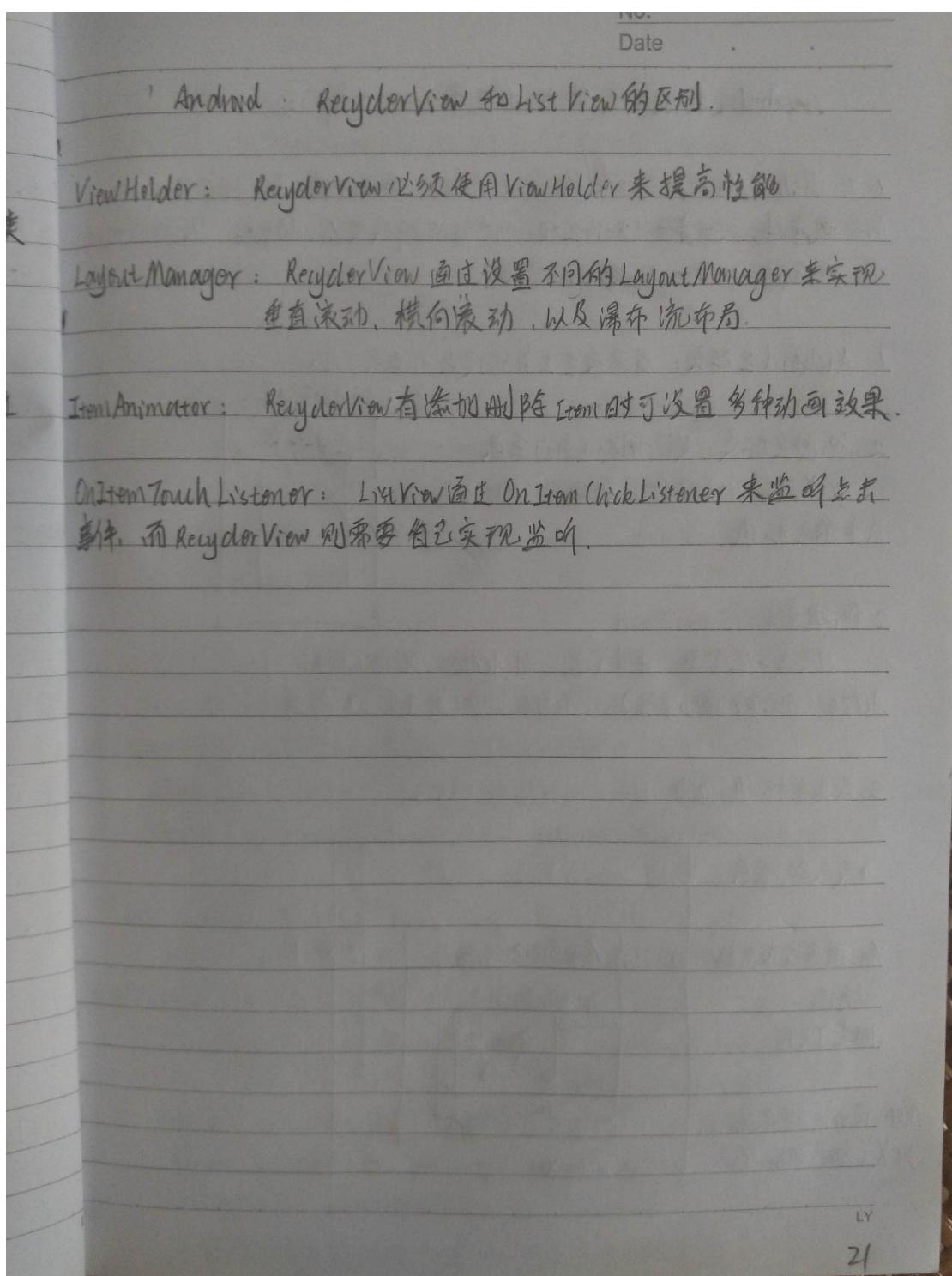
lastVisibleItemPosition = firstVisibleItem.

LY
19

320.Android 群英传 第四章 ListView 扩展笔记



321. Android RecyclerView 和 ListView 区别



322.Android 群英传 第五章 Scroll 分析

Date

牛宝生

坐标系

右

上

下

左

调用

改变

Android 群英传 笔记 第五章 Scroll 分析

滑动一个 View，本质就是移动一个 View。要实现滑动，就要监听用户的触摸事件，根据事件传入的坐标，动态且不断改变 View 的坐标，从而实现 View 的滑动。

1. Android 坐标系：屏幕最左上角的顶点为原点，

视图坐标系：父视图左上角为原点
getX()、getY() 就是获取视图坐标系中的坐标值。

2. 触控事件 - MotionEvent。
封装了一些常用的事件常量，如 ACTION_DOWN(按下事件)、ACTION_MOVE(移动事件)、ACTION_UP(离开事件) 等等。

3. 获取坐标值 方式。

表示触控点

View 提供获取坐标的方法
getL TRB

MotionEvent 提供的方法
getXX RawX RawY

The diagram illustrates the coordinate system hierarchy. At the top level is the 'Android 坐标系' (Android Coordinate System). Inside it is a '视图坐标系' (View Coordinate System), which contains a 'Layout'. Below the Layout is a 'View Group', which contains a 'View'. The View contains a point marked with an arrow and labeled 'getX'. Various other methods are shown with arrows pointing to their respective target components: 'getTop', 'getBottom', 'getRawY' (from View Group), 'getLeft', 'getRight' (from View Group), 'getRawX' (from View), and 'getL TRB' (from View).

4. 实现滑动的方式

基本思想：当触摸 View 时（DOWN），记录当前触摸点坐标。

手指移动时（MOVE），记录移动后触摸点的坐标，通过计算得到手指的偏移量。通过偏移量修改 View 的坐标。

在 onTouchEvent 中获取触摸点坐标。

```
float x = event.getX();
```

```
float y = event.getY();
```

在 ACTION_DOWN 中记录按下时候的坐标：

```
lastX = x; lastY = y;
```

在 ACTION_MOVE 中计算偏移量。

```
float movedX = x - lastX;
```

```
float movedY = y - lastY;
```

4.1 使用 layout 方式

View 的 layout() 方法就是设置控件的位置，在不移动的过程中不断

调用 layout 方法来修改 View 的位置，从而实现滑动。

```
float left = getLeft(); right = getRight(); top = getTop(); bottom = getBottom();
```

```
layout(left + movedX, top + movedY, right + movedX, bottom + movedY);
```

4.2. Params 参数方式

LayoutParams 保存了一个 View 的布局参数，因此在移动过程中通过改变 View 的 LayoutParams 的参数，从而实现移动。

```
LinearLayout.LayoutParams params = (LinearLayout.LayoutParams)
```

```
getLayoutParams();
```

```
params.setLeftMargin(getLeft() + movedX);
```

```
params.setTopMargin(getTop() + movedY);
```

```
setLayoutParams(params);
```

注意使用该方法必须知道 View 所在父布局的类型。

ViewGroup:
但是使用 MarginLayoutParams 就不需要知道父布局的类型。使用方式与
LayoutParams 是一样的。

4.3. ScrollBy 方式

注意两点：想要让 View 以 ScrollBy 的方式滚动，ScrollBy 方法就
要作用于 View 的父容器上；

想要让移动的方向与手指移动的方向一致，偏移量
就要取该方向的反方向。

参数内容是偏移量，不是坐标。

((View)getParent()).scrollBy(-movedX, -movedY);

4.4. Scroller 方式：实现平滑滚动。

初始化 Scroller : scroller = new Scroller(context);

重写 computeScroll() 方法：

@Override

public void computeScroll() {

super.computeScroll();

if (scroller.computeScrollOffset()) { // 判断是否执行完毕。

((View)getParent()).scrollTo(scroller.getCurrX(),

scroller.getCurrY());

invalidate();

}

}

scroller 的 getCurrX() 和 getCurrY() 来取得当前的滚动坐标。

computeScroll() 方法不会自动调用，只能通过

invalidate() → draw() → computeScroll() 来间接调用。

只能在 computeScroll() 方法获取滚动过程的 scrollX, scrollY。

开启模拟过程：scroller.startScroll (int scrollX, int scrollY, int dx, int dy);
case ACTION_UP:

int scrollX = ((View)getParent()).getScrollX();

int scrollY = ((View)getParent()).getScrollY();

scroller.startScroll (scrollX, scrollY, - scrollX, - scrollY);

invalidate();

break;

参数是起始坐标和偏移量，可以使用getScrollX() 和 getScrollY()
来获取父视图中于View所滑动到的点的坐标。

如果偏移量是坐标值的相反数，则返回原来位置。

4.5 属性动画

326.Android 群英传 第六章 绘图机制 笔记

Android 群英传 笔记 第六章 绘图机制

1. 屏幕信息

屏幕大小：屏幕对角线长度，寸。

分辨率：屏幕的像素点个数。720×1280. px

PPI：每英寸像素，又称 DPI (Dots Per Inch)

密度	ldpi	mdpi	hdpi	xhdpi	xxhdpi	xxxhdpi
密度值	120	160	240	320	480	640
分辨率	260×320	320×480	480×800	720×1280	1080×1920	1440×2560
1dp =	1px	1.5px	2px	3px	4px	

2. 2D绘图基础

Paint：画笔 Canvas：画布

通过这两个类的组合可以画出各种图案。

• Paint 的方法和功能

setAntiAlias(): 设置锯齿效果。

setColor():

setARGB():

setAlpha():

setTextSize():

setStyle(): 设置画笔的风格（空心或实心）

setStrokeWidth(): 设置空心边框的宽度。

• Canvas 的方法和功能

画点：drawPoint(x, y, paint)

画线段：drawLine(startX, startY, endX, endY, paint);

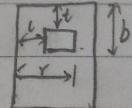
两点确定一线段。

Date

画多条直线：float[] pts = { startX1, startY1, endX1, endY1,
startX2, - - -
startXn, startYn, endXn, endYn };

drawLines(pts, paint);

画矩形：drawRect(l, t, r, b, paint)



画圆角矩形：drawRoundRect(l, t, r, b, radiusX, radiusY, paint);

画圆：drawCircle(x, y, r, paint); 圆心坐标和半径

画弧线：paint.setStyle(Paint.Style.STROKE);

//paint.setStrokeWidth(3);

drawArc(l, t, r, b, startAngle, sweepAngle, false, paint);

l, t, r, b 为圆、椭圆的外接矩形。

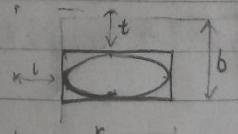


false：不与半径连接

true：与半径连接

stroke+false stroke+true fill+false fill+true

画椭圆：drawOval(l, t, r, b, paint)



画文字：drawText(text, x, y, paint);

画路径：Path path = new Path();

drawPath(path, paint);

path.moveTo(px1, py1);

path.lineTo(px2, py2);

path.lineTo(pxn, pyn).

3. XML 绘图

3.1 在 XML 中使用 Bitmap

```
<?xml version="1.0" encoding="utf-8"?>  
<bitmap xmlns:android="..." ...>  
    android:src="@drawable/ic_launcher"/>
```

代码中可以将图片直接转成 Bitmap.

3.2. shape :

用于绘制各种形状，无论是扁平化、拟物化，还是渐变，都能绘制。
使用各种各样的标签来实现各种效果。

gradient：渐变效果。

solid：填充颜色。

corners：当 shape 指定为 rectangle 时用，指定圆角。

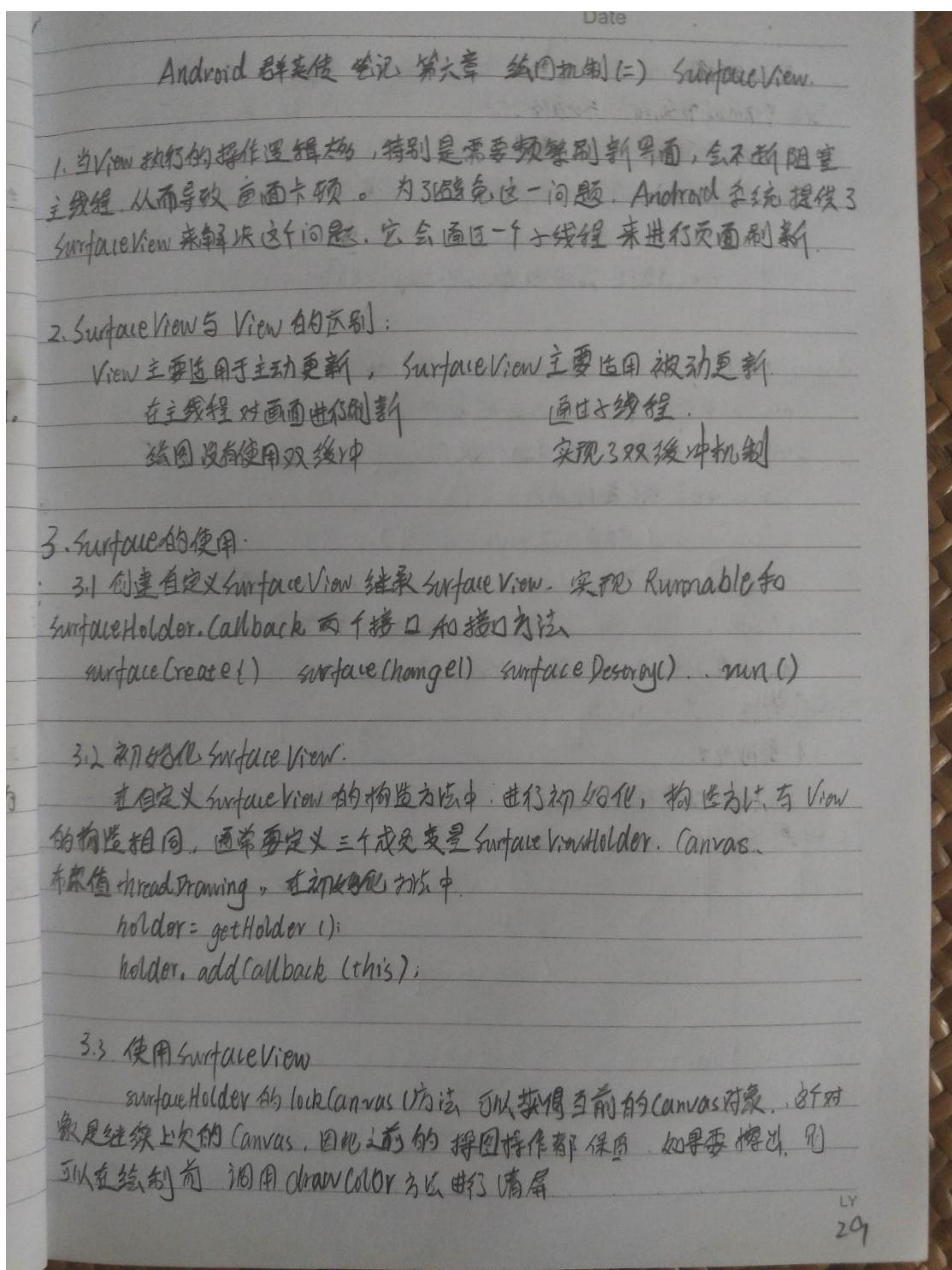
3.3. Layer:

使用 layer-list，把各种图片叠加，就像 Photoshop 的各个图层的叠加。每个图层是一个 item。

3.4. Selector:

selector 的作用在于帮助开发者实现事件反馈。例如正常状态下的
获得焦点，按下的反馈都可以。

329.Android 群英传 第六章 SurfaceView 笔记

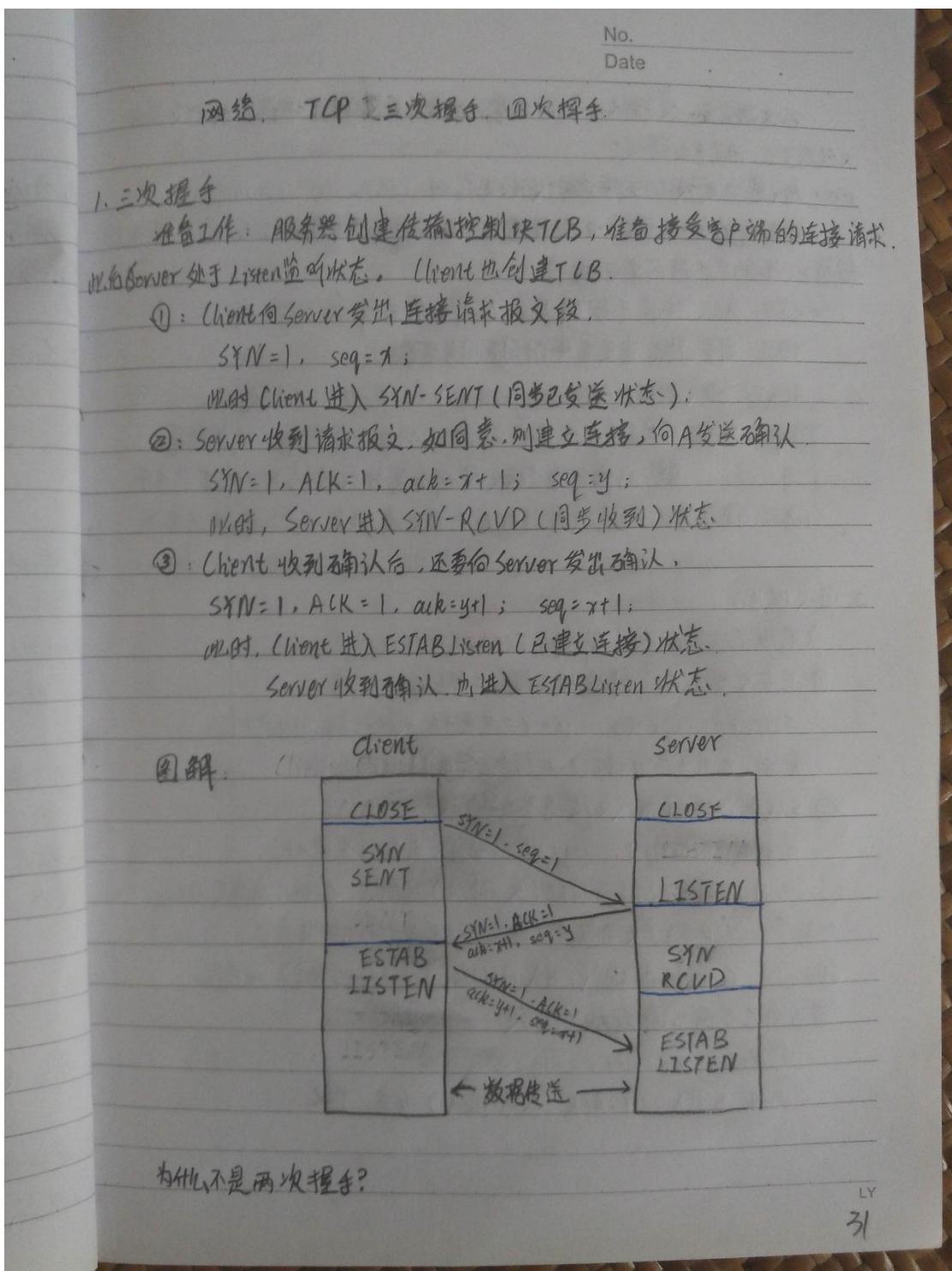


No.

Date

在 surfaceCreate 中开启子线程进行绘制，在 run 中使用
while (threadDrawing) 来循环绘制，在绘制的逻辑中，通过
lock (Canvas) 就取 canvas 进行绘制，通过 unlock (CanvasAndPost (1))
进行提交，且并提交放在 finally 语句块中，确保每次都能提高
1. 二
性能

331.Network TCP 三次握手，四次挥手



C还要发送一次确认，主要是为了防止已失效的连接请求报文段突然又收到了S，因而产生错误。

什么是已失效的连接请求报文段？

C发送的第一个请求在某网络结点上长时间滞留了，以致延误到直接释放以后的某个时间才到达S。

• SYN, ACK, seq, ack 都是TCP报文段首部的一些字段

SYN：同步 SYN，建立连接时用来同步序号。

ACK：确认 ACK (Acknowledgment)：当 ACK=1 才有效。

seq：序号：TCP连接中传送的字节流中的每一个字节都按顺序编号。

首部的值则指本报文段所发送的数据的第一个字节的序号。

ack：确认号：期望收到对方下一个报文段的第一个数据字节的序号。

2. 四次握手

当前状态：C 和 S 都处于 ESTABLISHED 状态。

① C 向 S 发送连接释放报文段，并停止发送数据

$FIN=1$, $seq=u$, $u=C$ 已发送过的数据一个字节的序号 +1。

此时，C 进入 FIN-WAIT-1 (终止等待 1) 状态。

② S 收到连接释放报文段后发出确认。

$ACK=1$, $ack=u+1$, $seq=v$, v 为 S 最后一个序号 +1。

此时，S 进入 CLOSE-WAIT (关闭等待) 状态。C → S 方向关闭

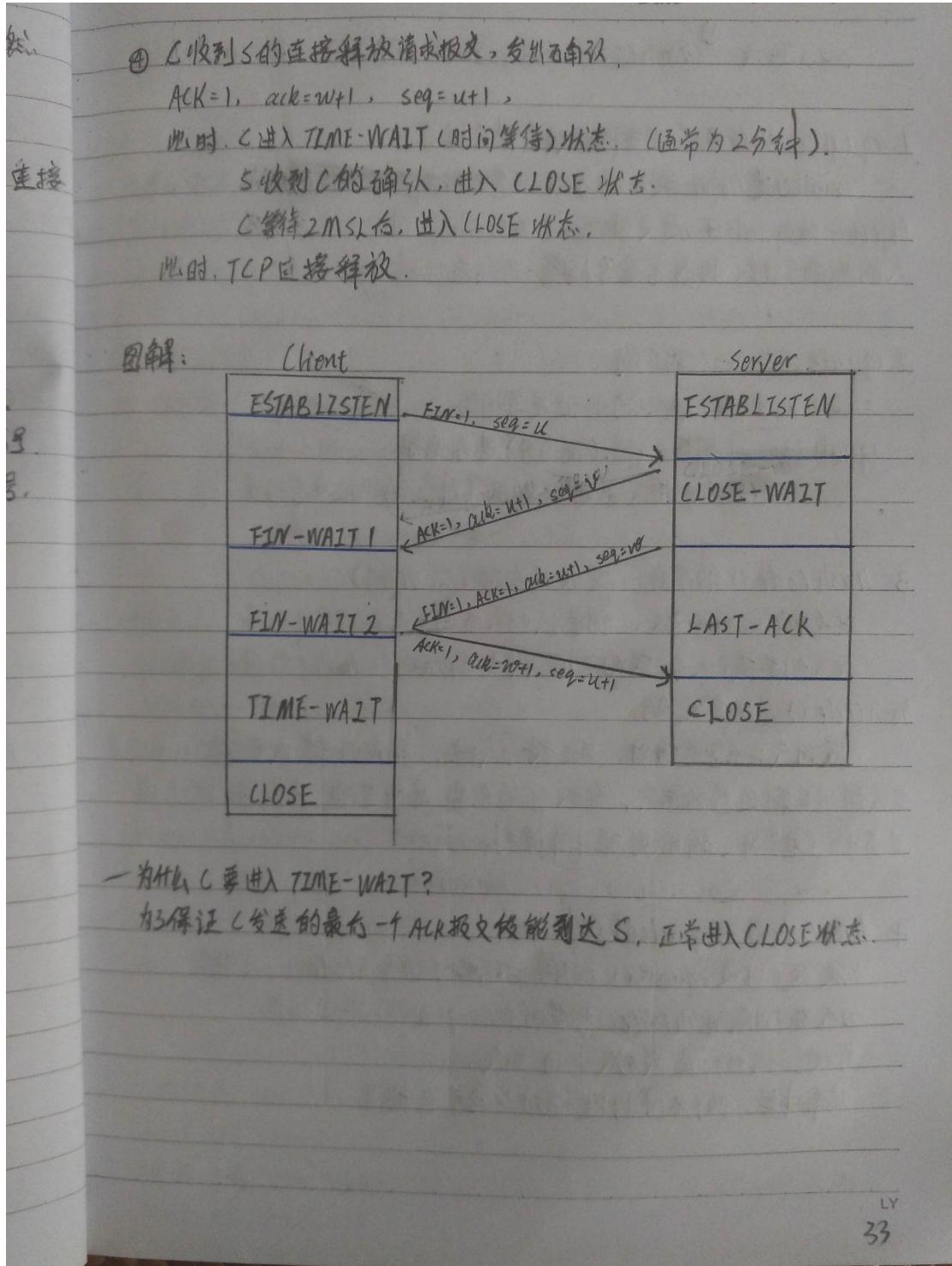
TCP 处于半关闭状态，即 S 向 C 发送，C 仍接收

C 收到 S 的确认，进入 FIN-WAIT-2 (终止等待 2) 状态。

③ S 向 C 发送连接释放报文段

$FIN=1$, $ack=u+1$, $seq=w$.

此时，S 进入 LAST-ACK (最后确认) 状态。



334. Java hashCode() 和 equals() 区别

Java: ① hashCode() 和 equals() 的作用与区别.

1. equals() 的作用：判断两个对象是否相等。

equals() 是 Object 类中的方法，通过判断两个对象的地址是否相等。
Object 中的 equals() 是使用 “==” 方法判断，因此，需要重写 equals() 方法
来判断两个对象的内容是否相等。否则两个对象的地址怎么可能相等。

2. equals() 与 “==” 的区别

==：判断两个对象的地址是否相等

equals()：判断两个对象（的内容）是否相等。

（默认情况下，也是判断两个对象的地址是否相等；即 ==）。

3. hashCode() 的作用：获取哈希码（散列码）

哈希码：int 整数，确定该对象在哈希表中的索引位置

仅当创建某个类的散列表时，该类的 hashCode() 才有用。否则
hashCode() 是无意义的。

散列表存储键值对，通过数组实现。当我们想要取某值时，其实是
取数组中某个位置的元素，而数组的位置，是通过“键”来获得。进一步说，
是通过“键”对应的散列码计算得到。

4. hashCode() 与 equals() 的关系

如果两个对象 equals() 相等，那么它们的 hashCode() 一定相等。

如果两个对象 hashCode() 相等，它们 equals() 不一定相等。

前提：该对象是散列表中的元素。

哈希冲突：两个不同的键值对，哈希值相等。

335.Android 群英传 第七章 动画 笔记

Android 群英传 笔记 第七章 动画

1. 视图动画

原理：每次绘制视图时 View 所在的 ViewGroup 中的 drawChild 方法会取该 View 的 Animation 的 Transformation 值，然后调用 canvas.concat (transformToApply.getMatrix ())，通过矩阵运算完成动画帧。如果动画没完成，就继续调用 invalidate ()，启动下次绘制来驱动动画，从而完成整个动画的绘制。

动画方式：AlphaAnimation：透明动画

RotateAnimation：旋转动画

TranslateAnimation：位移动画

ScaleAnimation：缩放动画

AnimationSet：动画集合

使用方法：

```
AlphaAnimation alphaAnimation = new AlphaAnimation (0, 1);  
alphaAnimation.setDuration (1000);
```

```
targetView.startAnimation (alphaAnimation);
```

第一个参数为：开始透明度，第二个参数为：结束时透明度

```
rotateAnimation = new RotateAnimation (0, 360, (0, 100));  
开门角、关门角，旋转中心点的坐标。
```

如果以自身的中心点为旋转中心，则

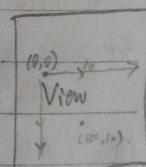
```
new RotateAnimation (0, 360,
```

```
    RotateAnimation.RELATIVE_TO_SELF, 0.5f,
```

```
    RotateAnimation.RELATIVE_TO_SELF, 0.5f);
```

位移动画：new TranslateAnimation (0.2f, 0, 300);
fromX, toX, fromY, toY。

缩放动画：new ScaleAnimation (0.2f, 0.2f); 从原点为缩放中心，
fromX, toX, fromY, toY。



No.
Date

老从自身的缩放中心，
new ScaleAnimation(0, 2, 0, 2,
Animation.RELATIVE_TO_SELF, 0.5f,
Animation.RELATIVE_TO_SELF, 0.5f);
动画集名： AnimationSet animationSet = new AnimationSet(true);
animationSet.addAnimation(alphaAnimation);
animationSet.addAnimation(rotateAnimation);

targetView.startAnimation(animationSet);

监听动画的运行状态：

animation.setAnimationListener(new AnimationListener() {
有三种状态： onAnimationStart()
onAnimationRepeat()
onAnimationEnd()

2. 属性动画

Animator框架使用最多的就是 AnimatorSet 和 ObjectAnimator 配合。
使用 ObjectAnimator 进行精细化控制，只控制一个对象的一个属性，而使用
多个 ObjectAnimator 组合到 AnimatorSet 形成一个动画。

属性动画通过调用属性的 get、set 方法来真实地控制一个 View 的属性
值。（内部通过 Java 反射机制来调用 set 方法修改对象属性值）

创建 ObjectAnimator 只需要通过他的静态工厂类直接使用一个
ObjectAnimator 对象，参数包括一个对象和对象的属性名，该属性必须重写
get、set 回数。

属性动画的监听：一个完整的动画具有 Start、Repeat、End、
Cancel 四个过程，分别对应 onAnimatorStart、Repeat、End、Cancel 方法。
objectAnimator.addListener(new AnimatorListener() { ... })；

AnimatorSet animatorSet = new AnimatorSet();

animatorSet.playTogether(animatorAlpha, animatorRotate...);

animatorSet.start();

在属性动画中，AnimatorSet 通过 playTogether, playSequentially, animatorSet.play(), with(), before(), after() 这些方法，控制多个动画协同工作。

3. 插值器

定义动画的变换速率，相当于物理中的加速度，例如先慢后快。

4. SVG矢量动画

最大的优势：放大不失真。

什么是SVG：Scalable Vector Graphics。可伸缩矢量图形。

使用XML格式定义图形。

<path>标签：用来创建SVG，就像用指令方式来控制一只画笔。例如将动画笔移到某一坐标位置，画一条线，画一条曲线，结束。

<path>支持的指令：

M = moveTo (M X, Y)：将画笔移动到点 (X, Y)，但未发生绘制。

L = lineTo (L X, Y)：画直线到 (X, Y)

H = horizontalLineTo (H X)：画水平线到指定 X 坐标

V = verticalLineTo (V, Y)：画垂直线到指定 Y 坐标

C = curveTo (C X1, Y1, X2, Y2, ENDX, ENDY)：三次贝塞尔曲线

S = smoothCurveTo (S, X2, Y2, ENDX, ENDY)：三次贝塞尔曲线

Q = quadraticBezierCurve (Q X, Y, ENDX, ENDY)：二次贝塞尔曲线

T = smoothQuadraticBezierCurveTo (T ENDX, ENDY)：映射前面路径向的终点

A = ellipticalArc (A RX, RY, XROTATION, FLAG1, FLAG2, X, Y)：椭圆弧

Z = closePath ()：关闭路径

No.

Date

注意：坐标轴以(0,0)为中心，X轴水平向右，Y轴垂直向下

指令大小写均可。大写绝对定位，参照全局坐标系；

小写相对定位，参照父容器坐标系。

指令和数据间的空格可省略。

同一指令出现多次可以只用一个。

SVG常用指令：M、L、V、H、A。

A指令参数解析：

RX, RY：椭圆半轴大小

XROTATION：椭圆的X轴与水平方向顺时针方向夹角。

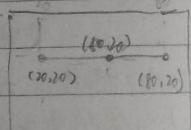
FLAG1：1表示大角度弧线，0表示小角度弧线。

FLAG2：从起点到终点的方向。1为顺时针，0为逆。

X, Y：终点坐标。

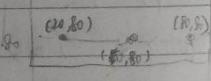
举例：pathData = "M 20,20

L 50,20,80,20"



pathData = "M 20,80

L 50,80,80,80"



两组宽高值：height = "200dp"，width = "200dp"

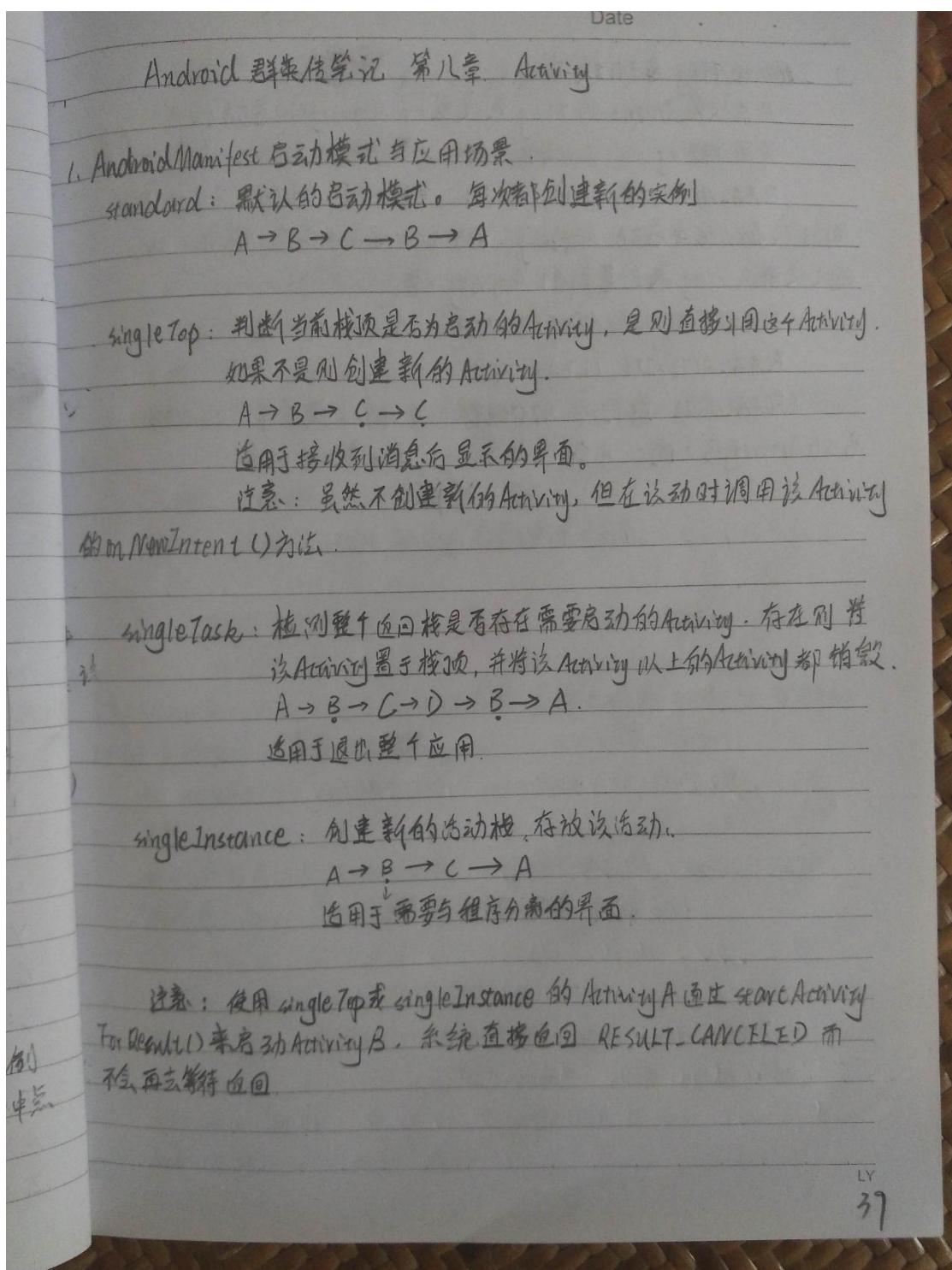
viewportHeight = "100"

viewportWidth = "100"

height, width 表示 SVG 图形具体大小。

viewportHeight, viewportWidth 表示 SVG 图形划分比例。加上手绘
表示 将 200dp 划分 100 份，而 (50,50) 表示正中间，(50, Y) 表示水平中点。

339. Android 群英传 第八章 Activity 启动模式 笔记



No.

Date

2. Intent Flag 启动模式

通过设置 Intent 的 Flag 来设置一个 Activity 的启动模式

常用的 Flag: Intent.

FLAG_ACTIVITY_NEW_TASK: 使用一个新的 Task 来启动 Activity

例如从 Service 中启动 Activity. 由于 Service 中不存在 Activity 栈.
所以使用该 Flag 来创建新的 Activity 栈.

FLAG_ACTIVITY_SINGLE_TOP: 与 singleTop 相同

FLAG_ACTIVITY_CLEAR_TOP: 与 singleTask 相同

FLAG_ACTIVITY_NO_HISTORY: 使用这种模式启动的 Activity A
启动 Activity B 之后. A 就消失了. 如 A → B → C → B

有:

Buu

2.

341.Android 群英传 第九章 Android 系统消息，安全机制

No.
Date

Android 群英传笔记 第九章 Android 系统信息与安全机制

1. 获取 Android 系统信息

方法: android.os.Build

String board = Build.BOARD;
SystemProperty
String os_version = System.getProperty("os.version");

Build 类中包含了系统编译时的大量配置信息，设备信息常用的有：

Build.BOARD; 主板
Build.BRAND; Android 系统定制商

其他的可自行查看官方文档。

SystemProperty 包含了很多系统配置属性值和参数，很多都和 Build 类中相同的，常用的有：

os.version OS 版本 os.name OS 名称
等等

这些信息可在 /system/build.prop 文件中查看，或者是在 /proc 目录中查看详细的系统信息。

2. Package Manager 应用包管理

ActivityInfo: 封装 Manifest.xml 中 activity 和 receiver 布置对之内的信息。
ServiceInfo: service

ApplicationInfo: application

PackageInfo: 包含以上的所有信息。

ResolveInfo: Intent 信息的上一级信息。

LY
41

PackageManager 的常用方法：

get PackageManager

get ApplicationInfo：返回指定包名的 ApplicationInfo

get ApplicationIcon

get Installed Applications

get Installed Packages

query Intent Activities：返回指定 Intent 的 ResolveInfo。Activity 集合

query Intent Services

resolve Activity

resolve Service

根据 ApplicationInfo 的各种 FLAG 判断应用的类型。

3. Activity Manager

Activity Manager. MemoryInfo {
availMem 可用内存

totalMem 总

threshold 低内存临界值

lowMemory 是否处于低内存

Debug. MemoryInfo

Running App ProcessInfo - pid, uid, processName

Running ServiceInfo

4. Packages.xml /data/system

<permissions> 标签：目前系统中所有权限

<package> 标签：代表一个 APK 的属性

<perms>：记录 APK 权限信息

No.

Date

5. Android 安全机制

代码混淆。

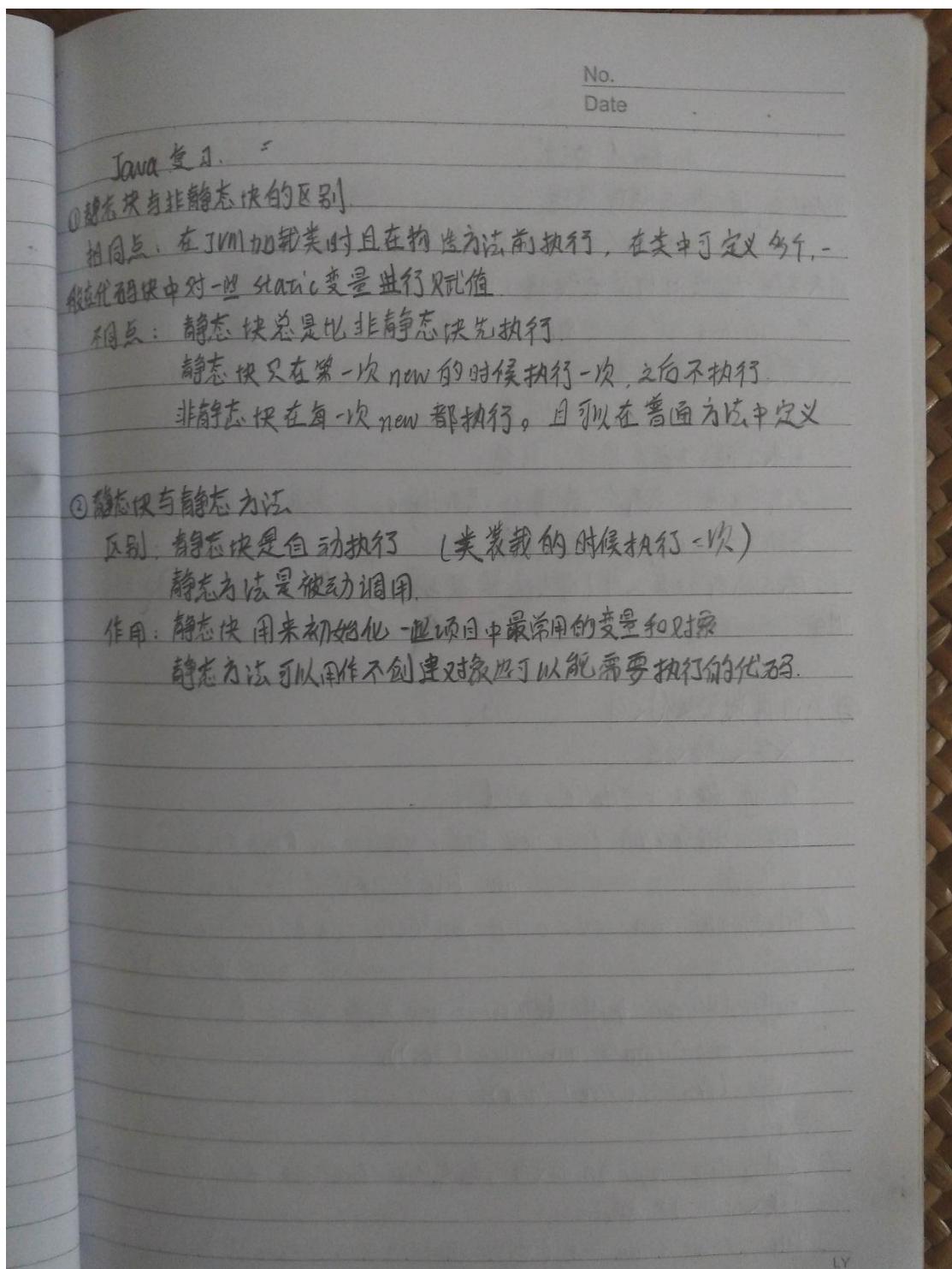
Manifest 文件权限声明、检查机制：应用接入权限控制

应用签名机制、数字证书

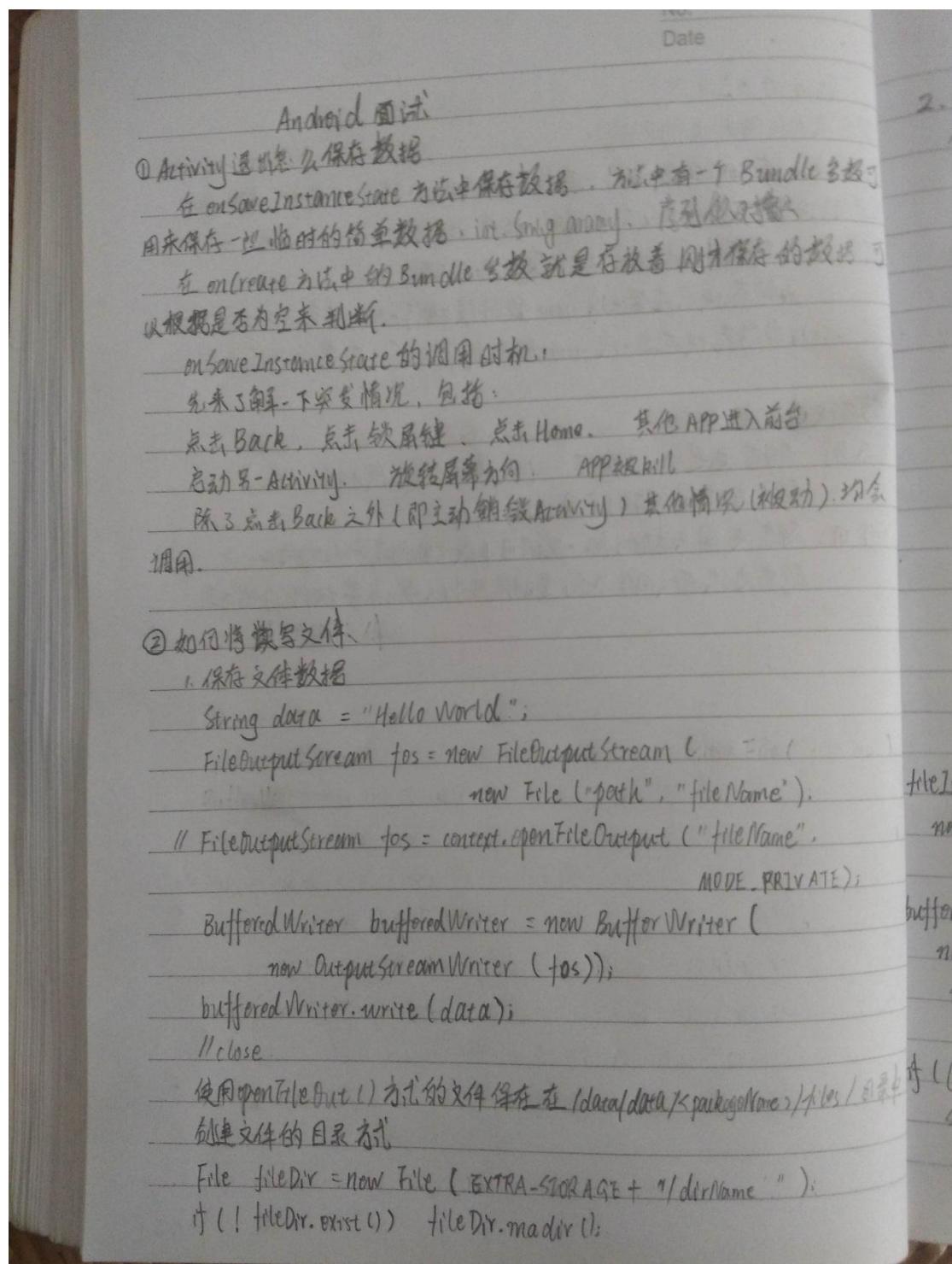
Linux 内核层安全机制：uid、访问权限控制

Android 虚拟机沙箱机制：沙箱隔离

345. Java 静态块，静态方法



346. Android 面试 Activity 退出保存数据 文件读写



2. 保存二进制数据

假设从网络(本地)获得的 InputStream 对象为 inputStream.

BufferedInputStream bufferedInputStream =

new BufferedInputStream (inputStream);

ByteArrayOutputStream byteArrayOutputStream =

new ByteArrayOutputStream ();

byte[] b = new byte [1024];

int len;

while ((len = bufferedInputStream.read (b)) > 0)

byteArrayOutputStream.write (b, 0, len);

FileOutputStream fileOutputStream =

new FileOutputStream (new File ("fileDir", "fileName"));

fileOutputStream.write (byteArrayOutputStream.toByteArray ());

//close .

3. 读文件

读文本文件

fileInputStream =

new FileInputStream (

new File ("fileDir", "fileName"));

bufferedReader =

new BufferedReader (

new InputStreamReader (

fileInputStream), "UTF-8");

if (strLine = bufferedReader.readLine ()) != null)

stringBuilder.append (strLine);

读二进制文件

fileInputStream =

new FileInputStream (

new File ("fileDir", "fileName"));

bufferedInputStream =

new BufferedInputStream (

fileInputStream);

if ((intLen = bufferedInputStream.read (b)) > 0)

byteArrayOutputStream.write (b, 0, intLen);

348. Android 面试 ANR 及处理

Android 面试

③ ANR 及如何处理

ANR: Application Not Responding 应用程序无响应。

出现场景：主线程被 10 操作阻塞。

主线程中存在耗时的计算。

主线程中错误的操作，如 Thread.sleep、wait 等。

Android 系统会监控程序的响应状况，一旦出现以下情况，会弹出 ANR 对话框。

应用在 5 秒内未响应用户的输入事件（如按键或触摸）。

BroadcastReceiver 未在 10 秒内完成相关的处理。

Service 在 20 秒内无法处理完成。

有些情况已经是 ANR，但不会弹出对话框。

如何避免？

UI 线程尽量只做跟 UI 相关的工作（如显示和更新）。

耗时的工作，如数据库操作、I/O、网络，或其他阻碍 UI 的操作
把它们放入单独的线程处理。

使用 AsyncTask 或者 Handler 处理耗时的操作。

BroadcastReceiver 中的 onReceive 方法尽量减少耗时，建议使用或
IntentService 处理耗时操作。

主线程有哪些？（UI 线程）

Activity 的生命周期、onKeyDown()、onClick 等。

AsyncTask 的 onPreExecute()、onProgressUpdate()、onPostExecute() 等。

Handler 的 handleMessage()、handler.post(runnable)

如何查看 ANR？

Log 日志。

traces.txt

/data/anr/traces.txt

③ Glide 的缓存策略

Glide 使用了内存缓存和硬盘缓存两种策略。

内存缓存主要是防止应用重复将图片数据读取到内存当中。

硬盘缓存主要是防止应用重复从网络或其他地方重复下载和读取数据。

缓存 Key : Engine Key .

把图片 url 作为 id , 和 width, height 等参数封装成 EngineKey 对象作为缓存 key .

内存缓存 :

Glide 自动开启了内存缓存功能 , 比如第一次加载图片会把图片加载进内存 , 只要内存没被清理 , 以后加载该图片就会首先从内存中获取 , 提高加载速度。

可以设置 skipMemoryCache(true) 来关闭内存缓存。

Glide 的内存缓存实现使用了 LRU (Least Recently Used) 最近最少使用算法 , 结合弱引用机制 共同完成内存缓存功能。

硬盘缓存 :

Glide 加载图片的时候 , 不会将原始图片展示出来 , 而是对图片进行压缩和转换 , 经过一系列操作得到转换后的图片。

Glide 默认情况下在硬盘缓存的就是转换后的图片。

可以通过 diskCacheStrategy (DiskCacheStrategy.NONE) 设置不缓存或者缓存不同类型的图片。

350.Android 面试 RxJava 优缺点、序列化

Android 面试

④ RxJava 优缺点

RxJava：一个用响应式编程和观察者模式实现异步操作的库
RxJava 中的响应式编程是被观察者拿到数据主动传递给观察者，将展示层和数据处理层分离，解耦了各个模块，通过不同线程操控代码运行配合变换过滤等 API 操作实现数据流传播。

RxJava 优点：异步、简洁

内部支持多线程操作。

强大的 map 和 flatMap 保证了依赖上一次接口数据进行二次处理时不会嵌套，将各模块

将各模块分离

支持 Lambda 表达式，保证 RxJava 代码在阅读上更加简洁。

随着程序逻辑的复杂，依然保持简洁。

⑤ Android 中的序列化？

什么是序列化、反序列化？

对象序列化：把 Java 对象转换为字节序列并存储至一个储存媒介的过程

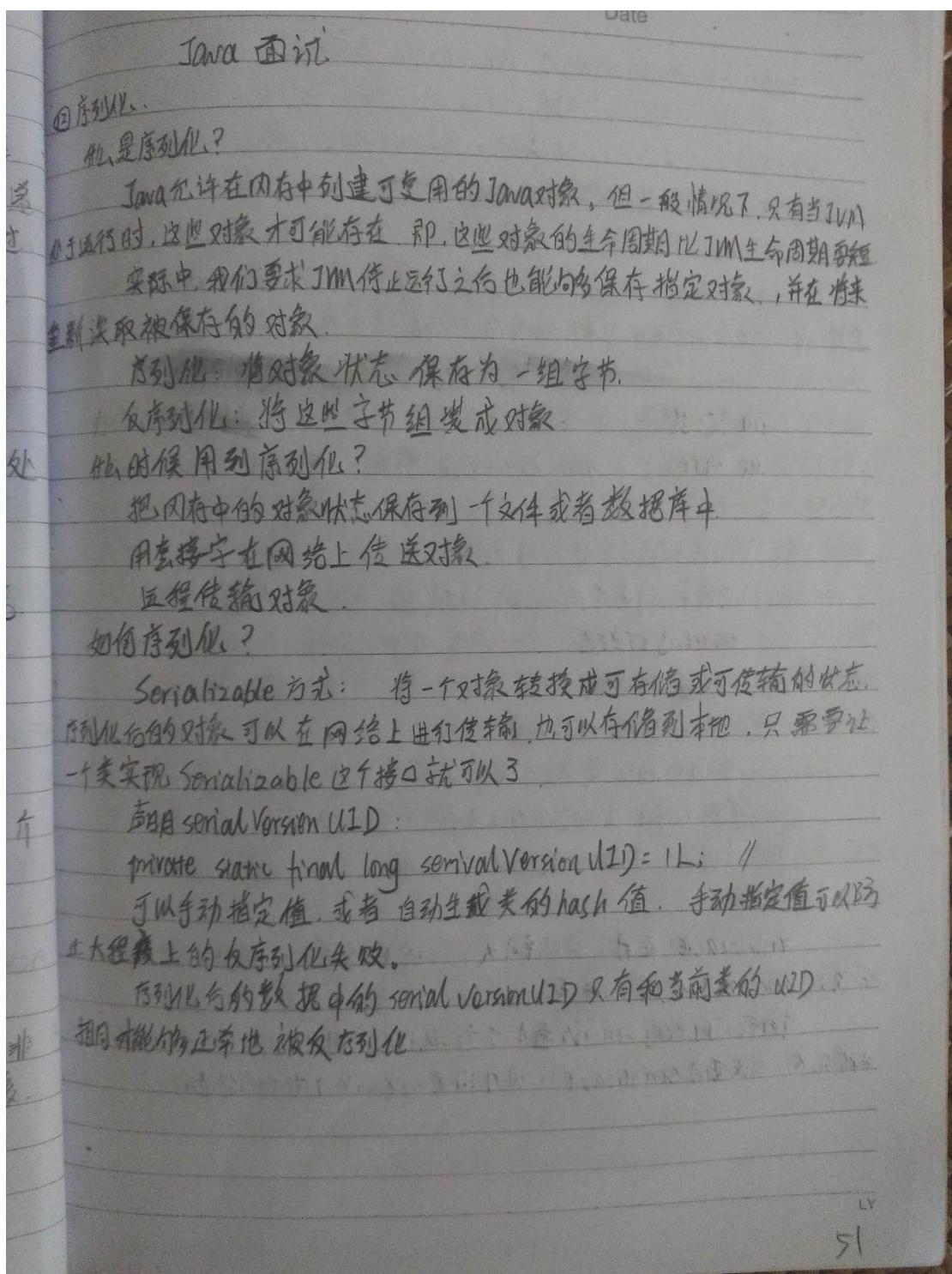
对象反序列化：把字节序列恢复为 Java 对象

Java 对象的组成：变量和方法。但是序列化和反序列化仅处理 Java 对象的变量而不处理方法。

原理：把 Java 对象的状态信息保存到存储堆栈，这样在 JVM 运行情况下，获取 Java 对象，也可以在其他机器 JVM 上获取指定 Java 对象，如使用 RMI（远程方法调用）、网络中传递对象。

续(52页)

351. Java 面试 序列化



Android 中 Serializable 与 Parcelable 的区别

相同点：都可以序列化对象

不同点：

• 首先肯定是实现方式不同。

Serializable 方式只要类实现 Serializable 接口就可以了。

Parcelable 方式比较复杂，首先实现 Parcelable 接口

重写 describeContents() 和 writeToParcel() 这两个方法，其中
describeContents() 直接返回 0；而 writeToParcel() 需要调用 parcel 的
writeXXX() 方法将对象中的字段一一写出。还要在类中创建一个名为
CREATOR 的静态常量，new Parcelable.Creator 接口的一个实现，并确保自
定义类型为当前序列化对象的类。在 createFromParcel() 方法中用 Parcel 对
象的 readXXX() 方法，读取序列化对象，并通过反序列化对象。而
在 newArray() 方法中，创建序列化对象的数组，把 size 作为数组的大小。
Parcelable 原理：将一个完成的对象分解，分解后的每一部
分都是 Intent 所支持的数据类型，这样也就实现传递对象功能。

• 最大区别：存储操作不同

Serializable 使用字节流存储在硬盘。

Parcelable 直接在内存中读写，速度大于 Serializable 方式。

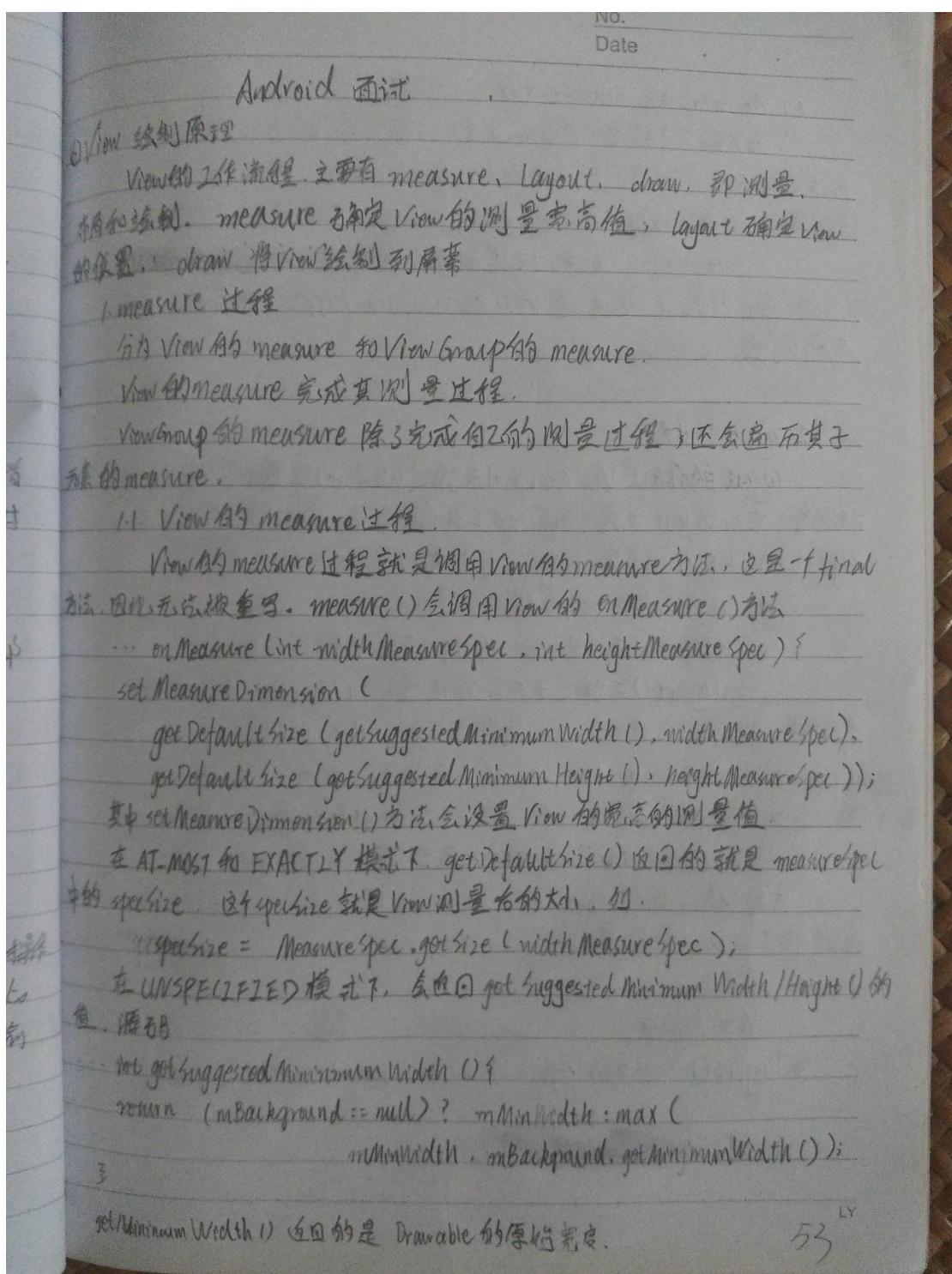
Android 中优先使用 Parcelable

如何选择？

Serializable 使用简单，开销大，序列化和反序列化需要大量 I/O 操作； Parcelable 使用稍麻烦，但高效，首选，主要用于内存序列化上。

将对象序列化到存储设备中或者将对象序列化后通过网络传输的情况下，建议使用 Serializable，因为用 Parcelable 方式会很复杂。

353. Android 面试 View 绘制原理



1.2 ViewGroup measure 过程

和 View 不同的是，ViewGroup 是一个抽象类没有重写 View 的 onMeasure 方法，但有一个 measure (children) 方法，会遍历每个子元素，然后调用 measureChild 方法。

在 measure (child) 中，根据子元素的 LayoutParams 来获取 MeasureSpec 对象，传递给 View 的 measure () 方法，从而完成子元素的测量。

2. layout 过程

layout 的作用是 ViewGroup 用来确定子元素的位置，当 ViewGroup 位置确定后，在 onLayout 中遍历所有子元素并调用其 layout 方法，在子元素的 layout 方法中又会调用其 onLayout 方法来确定子元素的位置。

layout 方法的大致流程：

setFrame () 来设定 View 的四个顶点位置。

onLayout () 来确定子元素的位置： onLayout () 的具体实现与布局有关，因此 View 和 ViewGroup 并没真正实现 onLayout。

① 遍历每个子元素并调用 → ② setChildFrame () 来确定子元素的相对位置。

setChildFrame () 仅调用子元素的 layout () 方法。

去到子 View 的 layout () 方法中，也是调用 onLayout () 方法，最终确定子 View 的位置。

ViewGroup

View

* layout () → onLayout () → layout () → onLayout ()

No.

Date

3. draw 过程

绘制过程分为以下几步：

drawBackground (canvas); // 绘制背景

onDraw (canvas); // 绘制自己

dispatchDraw (canvas); // 绘制 children

onDrawScrollBars (canvas); // 绘制装饰

View 的绘制过程的传递是通过 dispatchDraw 来实现。 dispatchDraw
会遍历所有子元素的 draw 方法，如此 draw 事件就一层层传递下去。

357. 算法 冒泡排序算法

No. _____
Date _____

算法：冒泡排序

1. 交换排序：两两比较待排序记录的关键字，一旦发现两个记录不满足要求则进行交换，直到整行列表全部满足要求为止。

2. 冒泡排序：比较相邻记录的关键字，如果逆序，则进行交换。从而使关键字小的记录如气泡一样通过（往上“漂浮”（左移），或者说使关键字大的记录则不断往下“下沉”（右移）。

3. 优化点：
每进行完一轮比较后，下一轮的比较数减1，因为最后那些已经排序好了。

4. 结束比较：
当一轮比较之后没有发生交换，则已经排序好了。因此，用一个标志位来记录是否有发生过交换。

5. Java实现：

```
private void popSort (int[] a) {  
    int len = a.length - 1;  
    boolean flag = true;  
    while (len > 0 && flag) { // flag=false;  
        flag = false;  
        for (int j=0; j < len; j++) {  
            if (a[j] > a[j+1]) {  
                int temp = a[j];  
                a[j] = a[j+1];  
                a[j+1] = temp;  
                flag = true;  
            }  
        }  
    }  
}
```

6. 时间复杂度： $O(n^2)$

7. 算法优缺点：

- (1) 稳定
- (2) 可用于链式存储
- (3) 缺：移动次数较多。
当很大时：不适用

LY
57

358. 算法 直接插入排序算法

算法：直接插入排序

1. 插入排序：每一趟对一个待排记录，按其关键字的大小插入到已经排序的一组记录的适当位置上，直到所有待排序记录全部插入。

2. 直接插入排序（最简单的插入排序）：

将一条记录插入到已排好序的有序表中，从而得到一个新的、记录数是增1的新表。

3. Java实现

```
private void insertSort(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = i; j > 0; j--) {
            if (a[j] < a[j - 1]) {
                int temp = a[j];
                a[j] = a[j - 1];
                a[j - 1] = temp;
            }
        }
    }
}
```

4. 时间复杂度 $O(n^2)$

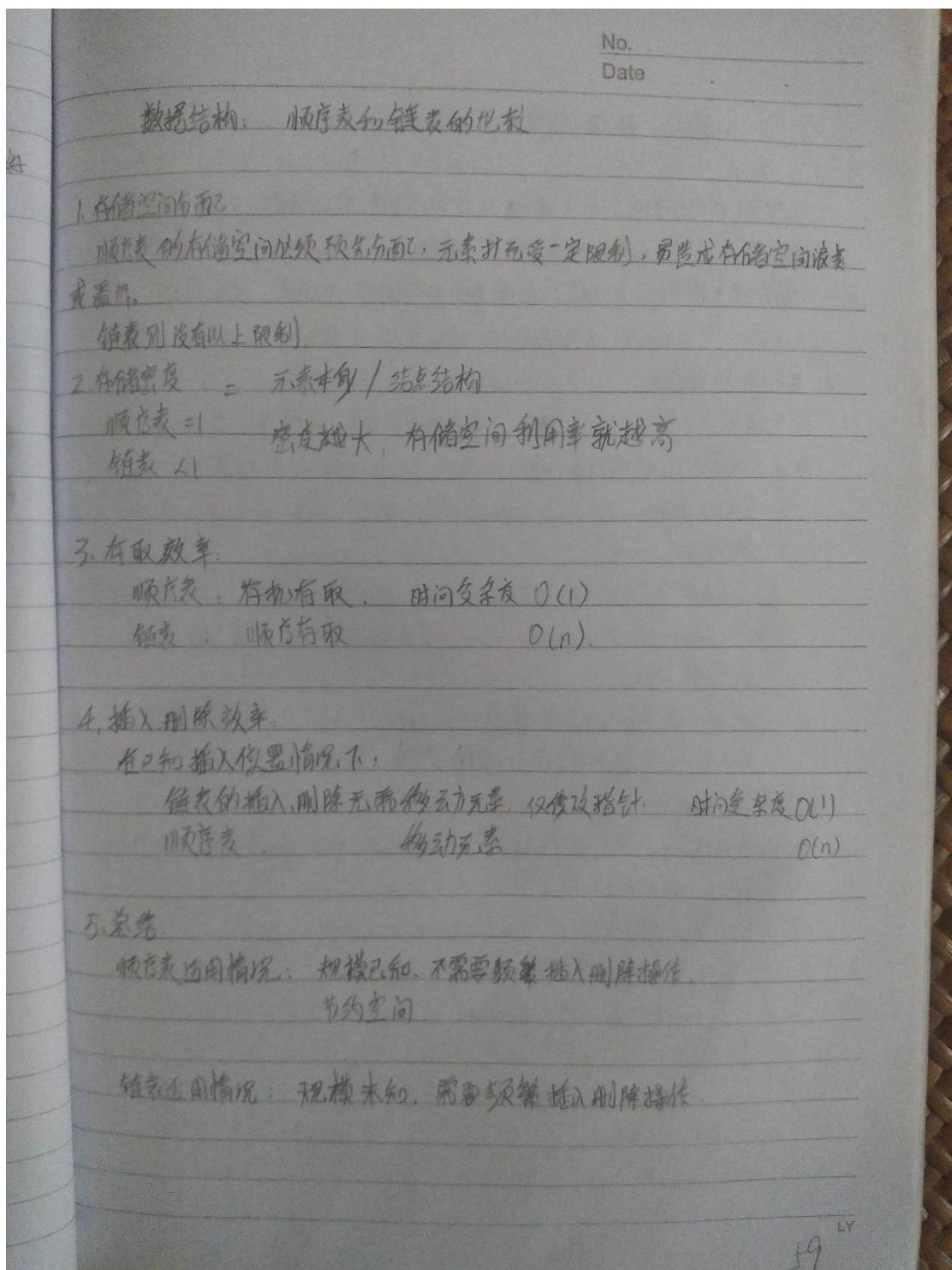
5. 特点：稳定

简单、简便、容易实现

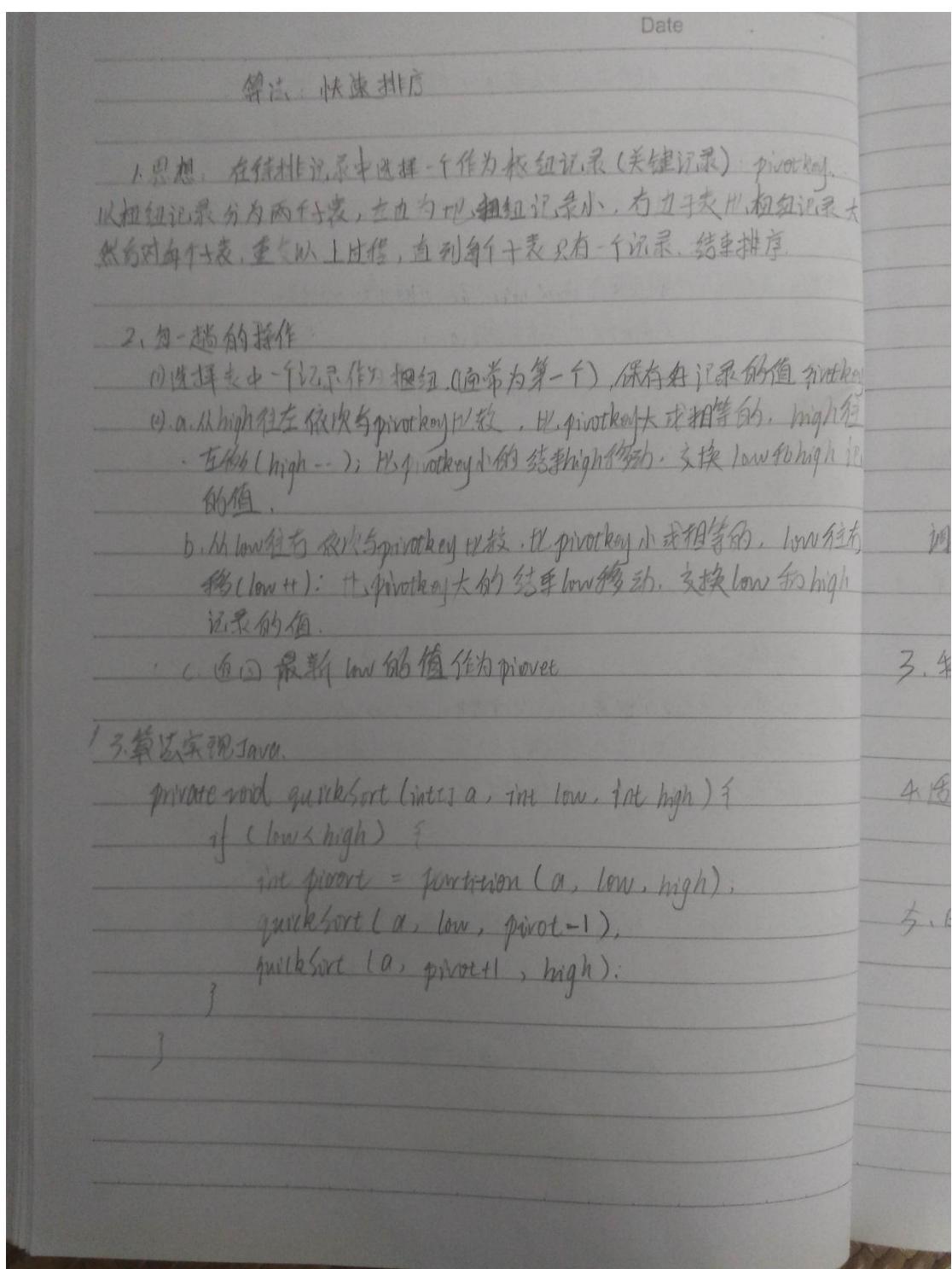
适用于链式存储结构

应用于基本排序，当初始无序，n较大时，不适合

359. 数据结构 顺序表和链表区别



360. 算法 快速排序算法



```
private int partition(int[] a, int low, int high) {
    int pivotkey = a[low];
    while (low < high) {
        while (low < high && a[high] >= pivotkey) high--;
        swap(a, low, high);
        while (low < high && a[low] <= pivotkey) low++;
        swap(a, low, high);
    }
    return low;
}
```

调用： quickSort(a, 0, a.length-1);

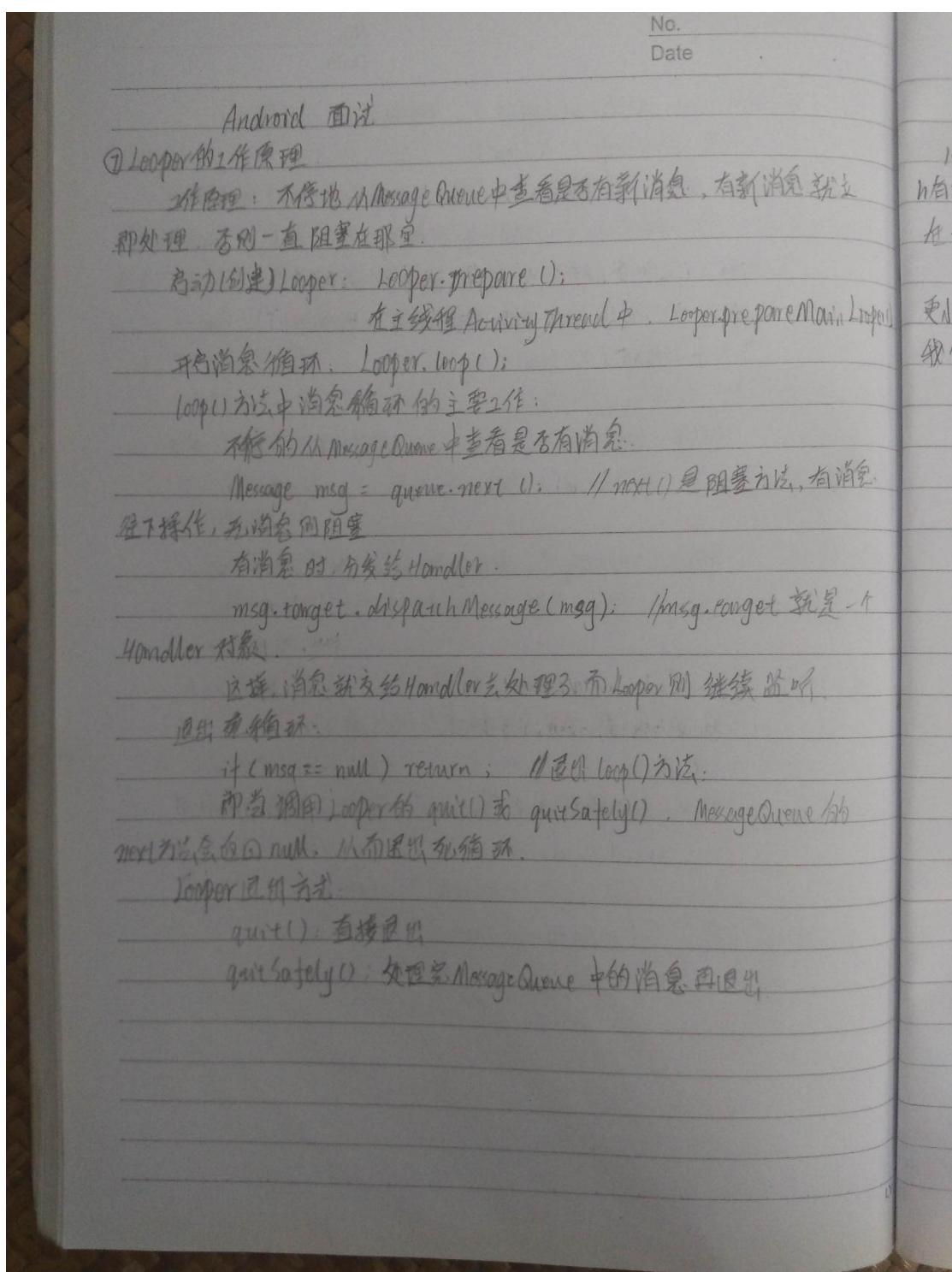
3. 特点：不稳定

适用于顺序存储，不适用于链式存储结构

4. 适用：n也较大，初始词条无序

5. 时间复杂度： $O(n \log n)$

362. Android 面试 Looper 原理



363. 算法 希尔排序算法

Date

算法：希尔排序。

1. 思想：使数组中任意间隔为 h 的元素都是有序的。这样的数组称为 h 有序数组。简单地说，一个 h 有序数组就是 h 个相互独立的有序数组编组在一起组成的一个数组。

在排序时，如果 h 很大，我们能将元素移动到很远的地方，为实现更小的 h 有序数组创造方便。用这种方式，对于任意以 1 结尾的 h 列，我们都能够将数组排序。

2. 实现（Java）

```
private void shellSort (int[] a) {  
    int N = a.length;  
    int h = 1;  
    while (h < N/3) h = h*3+1;  
    while (h>=1)  
        for (int i=h; i<N; i++) {  
            for (int j=i; j>=h; j=j-h) {  
                if (a[j]<a[j-h]) {  
                    int temp = a[j-h];  
                    a[j-h] = a[j];  
                    a[j] = temp;  
                }  
            }  
        }  
        h = h/3;  
}
```

3. 时间复杂度（平均） $O(n^{1.3})$

4. 特点：

- ① 不稳定
- ② 只适用于顺序存储
- ③ 不适用于链式存储
- ④ n 越大，效果越明显
- ⑤ 适用 n 较大
- 初学者易犯错

LY
63

364. 算法 二分查找

算法：二分查找（折半查找）

1. 思想：在有序表中，取中间记录作为比较对象。若关键字与中间记录相等则查找成功；

若关键字小于中间记录，则在中间记录左半区继续查找。

若关键字大于中间记录，则在中间记录右半区继续查找。

重复以上过程，直到查找成功，或查找区域无记录。

2. 算法实现 (Java)

```
private int binarySearch(int[] a, int key) {
    int low = 0; int high = a.length - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (key == a[mid]) return mid;
        else if (key < a[mid]) high = mid - 1;
        else if (key > a[mid]) low = mid + 1;
    }
    return -1; // 未记录
```

3. 时间复杂度 $O(\log n)$

特点：比较次数少，效率高。

只适用于顺序存储结构，且查找前要排序。

不适用于经常插入删除操作的顺序表。

365.Android 面试 Bitmap OOM

No.
Date

Android 面试

① Bitmap 如何避 OOM?

5 中

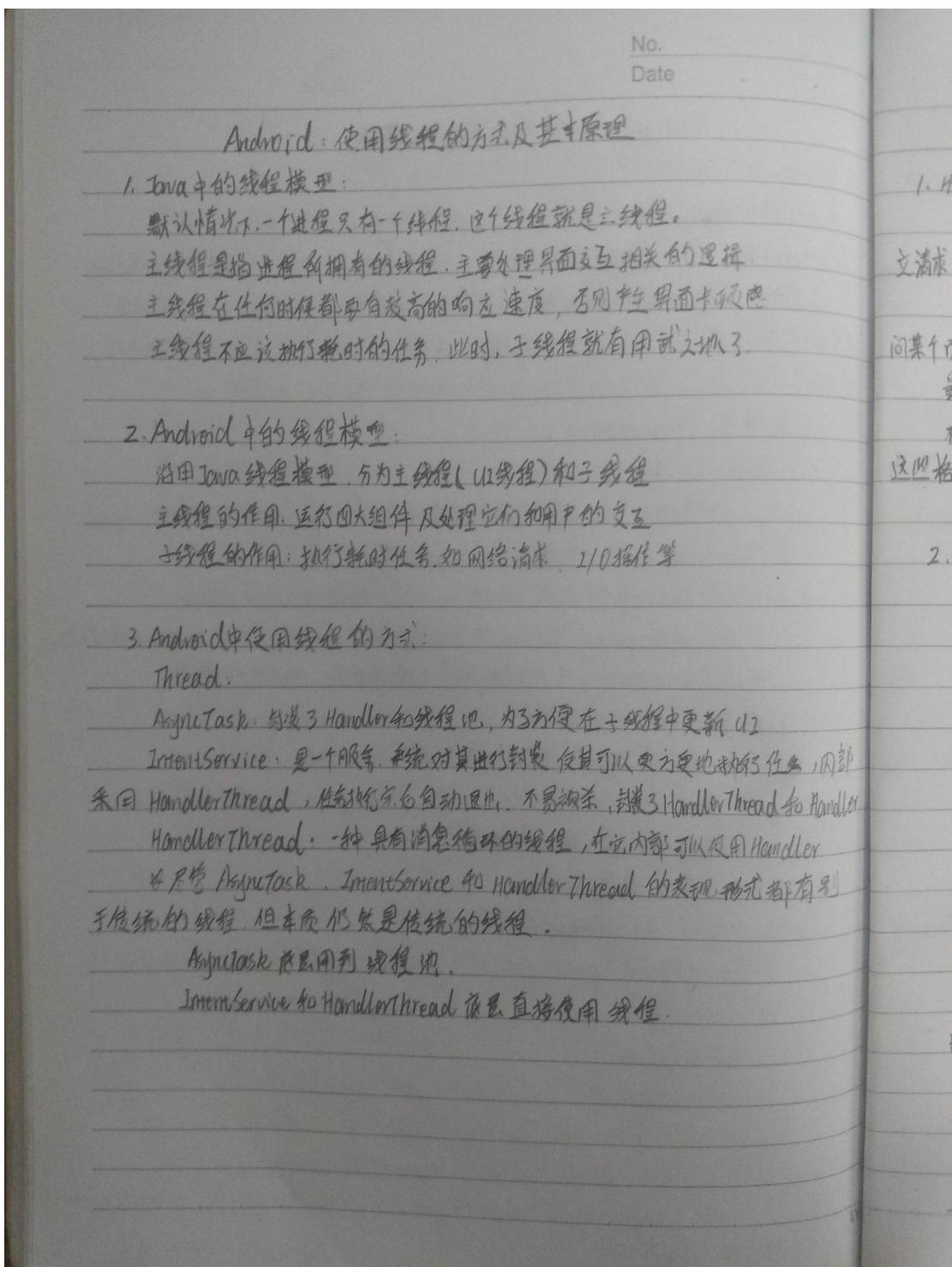
1. 加载 Bitmap 的四种方式: `BitmapFactory.decodeXXY`
`decodeFile`、`decodeResource`、`decodeStream`、`decodeByteArray`
2. 高效加载 Bitmap 的核心思想: 使用 `BitmapFactory.Option` 加载所需尺寸的图片, 即通过 `BitmapFactory.Option` 指定一定的采样率来加载缩小后的图片
3. 采样率: `inSampleSize`。
 $\text{inSampleSize} = 1$ 时, 不缩放。
 $\text{inSampleSize} = n$ 时, 缩小图片的宽高为原来的 $1/n$, 大小为原来的 $1/n^2$ 。
建议 `inSampleSize` 的值为 2 的指数倍。
4. 高效加载 Bitmap 的大致流程:
 - ① 将 `BitmapFactory.Option` 的 `inJustDecodeBounds` 设为 true 并加载图片
此时 `BitmapFactory` 只会解析图片的宽高信息, 而不会加载图片
 - ② 从 option 中取出图片宽度 `outWidth`、`outHeight`
 - ③ 通过一定的算法计算出适当的采样率
 - ④ 将 option 的 `inJustDecodeBounds` 设为 false 并重新加载图片
此时, 加载就是缩小后的图片

5. 获取采样率通用算法

```
public static int getInSampleSize()  
    BitmapFactory.Option option, int requestWidth, int requestHeight ) {  
    int width = option.outWidth; int height = option.outHeight;  
    int inSampleSize = 1;  
    if ( width > requestWidth || height > requestHeight ) {  
        int halfWidth = width / 2; int halfHeight = height / 2;  
        while ( (halfWidth / inSampleSize) >= requestWidth  
               && (halfHeight / inSampleSize) >= requestHeight ) inSampleSize *= 2;  
    }  
    return inSampleSize;  
}
```

63

366. Android 使用线程方式及基本原理



367. Network http 协议简单介绍

No. _____
Date _____

网络：Http 协议简单介绍

1. HTTP的工作流程：

服务器不断监听 TCP 的 80 端口，以便发现是否有客户端向它发送直接建立连接。

一旦监听到连接建立请求并建立了 TCP 连接之后，客户端向服务器发出访问某个页面的请求，服务器接着返回数据作为响应。

最后，TCP 连接被释放了。

在请求和响应的交互，必须按照规定的格式和遵循一定的规则。
这些格式和规则就是 HTTP 协议。

2. HTTP 协议的报文结构：

HTTP 有两类报文：请求报文 和 响应报文

其结构很相似：都包括 开始行、首部行、实体主体。

请求	回车换行
方法 URL 版本 CRLF	{ 请求行 状态行 }
首部字段名： 值 CRLF	首部行 { 首部字段名： 值 CRLF : 首部字段名： 值 CRLF CRLF }
实体主体	响应报文

请求报文 响应报文

开始行：用来区分是请求报文还是响应报文。

请求报文的开始行叫请求行。 响应报文的开始行为状态行

首部行：用来说明客户端、服务器或报文主体的一些信息。用键值对表示。
每一行结束的地方都要有回车换行

实体主体：

67 LY

3. 请求报文举例

开始行: GET /SocialServer/LoginServlet?username=xm&password=dm HTTP/1.1

Host: 39.108.9.138:8080

User-Agent: Mozilla/5.0 Firefox/54.0

Accept: text/html, application

Accept-Language: en-US, en; q=0.5

Accept-Encoding: gzip, deflate

Cookie: id=1327; SESSIONID=xxxx

Connection: keep-alive

Upgrade-Insecure-Requests: 1

回车换行

Content-Type: multipart/form-data; boundary=xxxxxxxxxx

Content-Length: 50389

回车换行

--boundary

参数 Content-Description: form-data; name="username"

回车换行

username的值

n个参数

Content-Description: form-data; name="photo"; filename="filename.png"

Content-Type: image/png.

回车换行

文件参数 文件的二进制数据.....

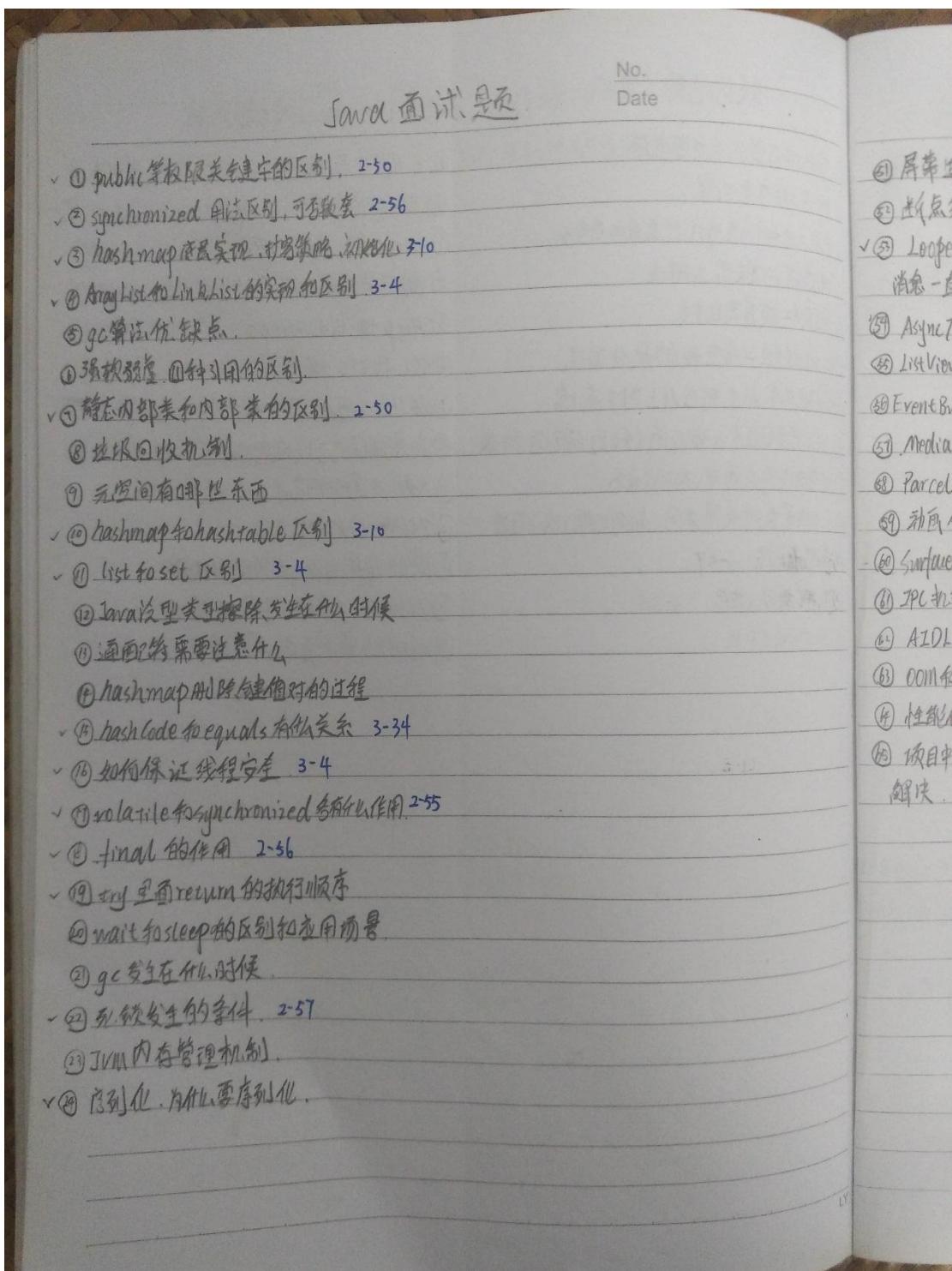
--boundary-- 回车换行

370. Android 面试题

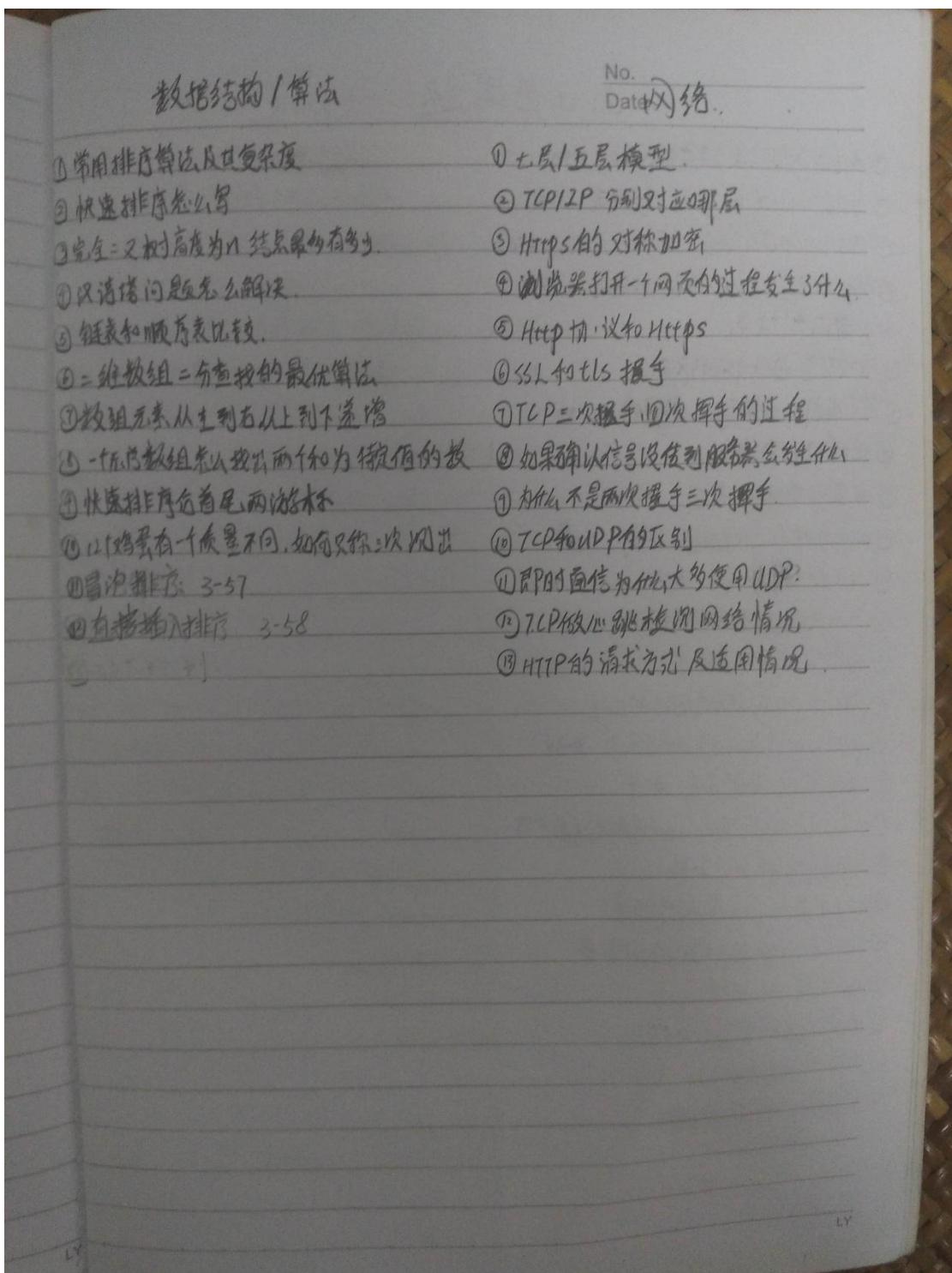
Android 面试题 (初、中级)	
✓ ① Activity退出怎么保存数据 3-46	⑥ 进程和线程的区别
✓ ② 将数据写入文件 3-46	⑦ 多进程和多线程的实现
✓ ③ Glide的缓存策略 3-49	⑧ map, flatmap 的原理
④ LruCache 原理实现	⑨ Singleton 启动 standard 的 Activity 在哪个栈
⑤ 缓存文件怎么命名	⑩ 可不可以多次 subscribe On? Observer 会有什么后果
✓ ⑥ RxJava 优缺点 3-50	⑪ Lambda 表达式和匿名内部类的区别
✓ ⑦ 自定义 View 的几个构造方法及参数的作用 3-17	⑫ 怎么解决内存泄露
✓ ⑧ ListView 中 convertView 的作用 3-18	⑬ 实现 IPC 的方法有哪些?
✓ ⑨ ViewHolder 为什么可以优化 3-18	⑭ 消息队列为空怎样的
⑩ App 被杀死怎么启动	⑮ 主题功能怎么实现
⑪ 怎么解决耗电量问题	⑯ APP 启动过程及堆栈分配
⑫ 怎么统计 crash.	⑰ 共享数据的方式有哪些
✓ ⑬ 事件分发机制 3-14	⑲ Service 和 IntentService 的区别
⑭ 怎么减少用户流量消耗	⑳ 自定义 View 的几种方式 3-12
✓ ⑮ 消息机制 Handler 原理 2-15	㉑ 动画的几种方式
✓ ⑯ View 绘制原理 3-53	㉒ 进程通信方式
⑰ 方法数超过 535 怎么办	㉓ 序列化, 为什么要序列化, 怎么序列化 3-50
⑱ Binder 机制	㉔ MVP (什么是 MVP, 优缺点, 如何解决缺点)
✓ ⑲ ANR 及如何避免 3-48	㉕ 和 MVC 的区别, 为什么要用 MVP.
✓ ⑳ ListView 优化 3-18	㉖ 当 Activity 退出时, 如何取消 (Volley) 请求
㉗ Bitmap 怎么避免 OOM	㉗ Glide 与 Picasso, UIL, Fresco 区别
㉙ Retrofit 原理 (OkHttp, Volley)	㉘ Glide 与生命周期
㉚ RecyclerView 和 ListView 的区别, 及其缓存原理 3-21	㉙ 图片加载框架的思路
㉛ Activity 生命周期 (Service, Fragment) 1-36, 1-41, 2-19	㉚ 动态加载的好处与原理
㉜ 四种启动模式及其区别, 应用场景	㉛ 热修复技术, 原理和优缺点
	㉜ 如何启动线程, 线程的管理, 线程组

- ④ 屏幕适配、方法、PX转换
- ⑤ 线程继承、多线程下我如何实现
- ⑥ Looper 怎么启动，启动后做了什么，没有消息一直循环么。3-62
- ⑦ AsyncTask 原理和优缺点，如何解决
- ⑧ ListView 和 RecyclerView 的优缺点
- ⑨ EventBus 原理机制
- ⑩ MediaPlayer 与 Activity 的生命周期
- ⑪ Parcelable 和 Serializable 区别
- ⑫ 动画分类，原理和区别
- ⑬ SurfaceView 和 View 的区别，为什么要用它
- ⑭ IPC 机制，为什么要用
- ⑮ AIDL 的原理
- ⑯ OOM 和 内存溢出 的区别，如何优化
- ⑰ 性能优化有哪些，工具有哪些
- ⑱ 项目中遇到最大难题是什么？如何解决

373.Java 面试题



375. 数据机构/算法 网络



No.

Date

设计模式

① 我所知道的设计模式

- ① 生产者和消费者问题
- ② 逻辑分页和物理分页的好处
- ③ 什么是虚内存
- ④ 死锁发生的条件，如何避免
- ⑤ 进程和线程的区别
- ⑥ 线程之间如何同步
- ⑦ 进程间的通信方式
- ⑧ LRU 算法原理