目录

1. ListView 中图片错位的问题是如何产生的?

2. ListView 图片加载错乱的原理和解决方案

https://blog.csdn.net/a394268045/article/details/50726560

https://blog.csdn.net/a394268045/article/details/50726560

图片错位原理：

如果我们只是简单显示 list 中数据，而没用 convertview 的复用机制和异步操作，就不会产生图片错位；

重用 convertview 但没用异步，也不会有错位现象。但我们的项目中 list 一般都会用，不然会很卡。

在上图中，我们能看到 listview 中整屏刚好显示 7 个 item，当向下滑动时，显示出 item8，而 item8 是重用的 item1，如果此时异步网络请求 item8 的图片，比 item1 的图片慢，那么 item8 就会显示 item1 的 image。当 item8 下载完成，此时用户向上滑显示 item1 时，又复用了 item8 的 image,这样就导致了图片错位现象(item1 和 item8 是用的同一块内存哦)。

解决方法：

对 imageview 设置 tag，并预设一张图片。

向下滑动后，item8 显示，item1 隐藏。但由于 item1 是第一次进来就显示，所以一般情况下，item1 都会比 item8 先下载完，但由于此时可见的 item8 的 tag，和隐藏了的 item1 的 url 不匹配，所以就算 item1 的图片下载完也不会显示到 item8 中，因为 tag 标识的永远是可见图片中的 url。

3.混合开发有了解吗？

https://blog.csdn.net/sinat_33195772/article/details/72961814

暂时不做，只做过 h5

4. 知道哪些混合开发的方式？说出它们的优缺点和各自使用场景？（解答：比如:RN，weex，H5，小程序，WPA 等。做 Android 的了解一些前端 js 等还是很有好处的)；

https://www.jianshu.com/p/22aa14664cf9

https://www.jianshu.com/p/52071a3d07b4

5. 屏幕适配的处理技巧都有哪些?

https://blog.csdn.net/wangwangli6/article/details/63258270

(1)根据屏幕的配置来加载相应的 UI 布局

在平板电脑和电视的屏幕（>7 英寸）上：实施“双面板”模式以同时显示更多内容

1) 在手机较小的屏幕上：使用单面板分别显示内容

2) 适配手机的单面板（默认）布局：res/layout/main.xml

3) 适配尺寸>7 寸平板的双面板布局（Android 3.2 后）

res/layout-sw600dp/main.xml

屏幕方向（Orientation）限定符

使用场景：根据屏幕方向进行布局的调整

取以下为例子：

1）小屏幕，竖屏：单面板

2）小屏幕，横屏：单面板

3）7 英寸平板电脑，纵向：单面板，带操作栏

4）7 英寸平板电脑，横向：双面板，宽，带操作栏

5）10 英寸平板电脑，纵向：双面板，窄，带操作栏

6）10 英寸平板电脑，横向：双面板，宽，带操作栏

7）电视，横向：双面板，宽，带操作栏

(2)百分比适配，也就是使用权限

6.服务器只提供数据接收接口，在多线程或多进程条件下，如何保证数据的有序到达？（暂无）

7.动态布局的理解

https://blog.csdn.net/ustory/article/details/42424313

先加载已经存在的 Layout 文件，然后再用代码动态的改变。

LayoutParams 继承于 Android.View.ViewGroup.LayoutParams.

LayoutParams 相当于一个 Layout 的信息包，它封装了 Layout 的位置、高、宽等信息。（假设在屏幕上一块区域是由一个 Layout 占领的，如果将一个 View 添加到一个 Layout 中，最好告诉 Layout 用户期望的布局方式，也就是将一个认可的 layoutParams 传递进去。）（可以这样去形容 LayoutParams，在象棋的棋盘上，每个棋子都占据一个位置，也就是每个棋子都有一个位置的信息，如这个棋子在 4 行 4 列，这里的“4 行 4 列”就是棋子的 LayoutParams。

）

但 LayoutParams 类也只是简单的描述了宽高还有位置，宽和高都可以设置成三种值：

1) 一个确定的值；

2) FILL_PARENT，即填满（和父容器一样大小）；

3) WRAP_CONTENT，即包裹住组件就好

```
    //添加子元素
    //rl.addView(bt);

      //定义参数，这个制定样式，也就是布局空间的位置
    RelativeLayout.LayoutParams Params = new RelativeLayout.LayoutParams(
                    ViewGroup.LayoutParams.WRAP_CONTENT,
                    ViewGroup.LayoutParams.WRAP_CONTENT
                    );
    Params.addRule(RelativeLayout.ALIGN_PARENT_TOP);
    Params.addRule(RelativeLayout.CENTER_HORIZONTAL,RelativeLayout.TRUE);
```

8.怎么去除重复代码？

https://blog.csdn.net/wangzhongshun/article/details/78738217

1）使用 include 标签引用重复布局

```
    <include layout="@layout/divider_view"/>
```

2）使用 style 定义样式

```
<style name="DemoBtn">

    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textSize">16dp</item>
    <item name="android:gravity">center</item>
    <item name="android:layout_marginTop">5dp</item>
    <item name="android:layout_marginBottom">5dp</item>
</style>
```

3）使用 ViewStub 减少整体布局的重复

ViewStub 是一个轻量级别的，不可见的 View，当 ViewStub 被设为 visible 时，或者显示调用 layout()时，才会去把它所指向的布局渲染出来，所以它非常适合处理整体相同，局部不同的情况。

```
<ViewStub
  android:id="@+id/viewstub_demo_text"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginLeft="5dip"
  android:layout_marginRight="5dip"
  android:layout_marginTop="10dip"
  android:layout="@layout/viewstub_demo_text_layout"/>
```

4）多使用应用资源

所有的资源的指定都可以用引用，而非直接写死，直接写死就会出现重复代码，比如颜色，背影，字串，布局，ID，度量(dimen)，风格等等。那么，我们在使用的时候，也尽可能的使用引用，这样非常易于复用，修改和定制，从而也就更方便复用。

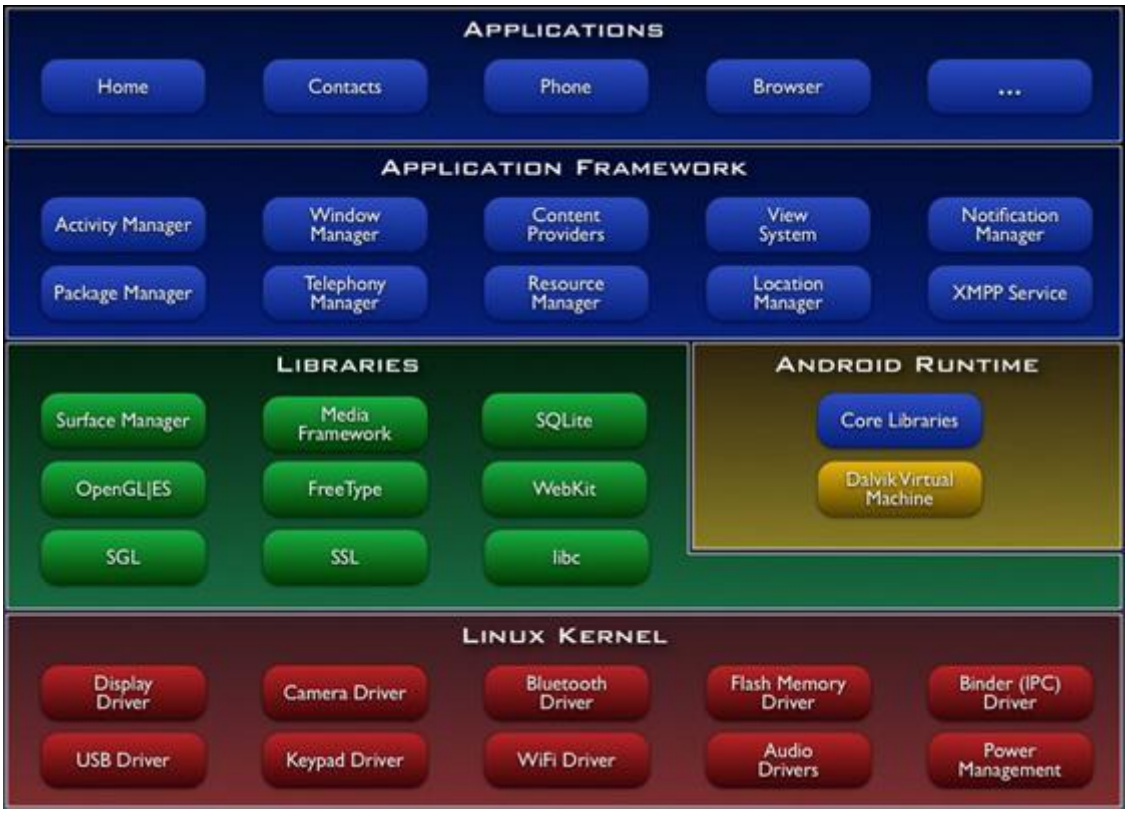
5）代码的抽象与继承

从代码上去除重复的代码就是用通用的重构技巧，比如提炼方法，抽象基类，提炼常量等。

9. 画出 Android 的大体架构图

https://blog.csdn.net/wang5318330/article/details/51917092

Android 的系统架构采用了分层架构的思想，如图 1 所示。从上层到底层共包括四层，分别是**应用层、应用框架层、系统库和 Android 运行时和 Linux 内核**。



一、应用层

该层提供一些核心应用程序包，例如电子邮件、短信、日历、地图、浏览器和联系人管理等。同时，开发者可以利用 Java 语言设计和编写属于自己的应用程序，而这些程序与那些核心应用程序彼此平等、友好共处。

二、应用程序框架层

该层是 Android 应用开发的基础，开发人员大部分情况是在和她打交道。应用程序框架层包括活动管理器、窗口管理器、内容提供者、

视图系统、包管理器、电话管理器、资源管理器、位置管理器、通知管理器和 XMPP 服务十个部分。在 Android 平台上，开发人员可以完全访问核心应用程序所使用的 API 框架。并且，任何一个应用程序都可以发布自身的功能模块，而其他应用程序则可以使用这些已发布的功能模块。基于这样的重用机制，用户就可以方便地替换平台本身的各种应用程序组件。

三、系统库和 Android 运行时

系统库包括九个子系统，分别是图层管理、媒体库、SQLite、OpenGLEState、FreeType、WebKit、SGL、SSL 和 libc。Android 运行时包括核心库和 Dalvik 虚拟机，前者既兼容了大多数 Java 语言所需要调用的功能函数，又包括了 Android 的核心库，比如 android.os、android.net、android.media 等等。后者是一种基于寄存器的 java 虚拟机，Dalvik 虚拟机主要是完成对生命周期的管理、堆栈的管理、线程的管理、安全和异常的管理以及垃圾回收等重要功能。

四、Linux 内核

Android 核心系统服务依赖于 Linux2.6 内核，如安全性、内存管理、进程管理、网络协议栈和驱动模型。Linux 内核也是作为硬件与软件栈的抽象层。驱动：显示驱动、摄像头驱动、键盘驱动、WiFi 驱动、Audio 驱动、flash 内存驱动、Binder（IPC）驱动、电源管理等。

10. Recycleview 和 ListView 的区别

https://blog.csdn.net/sanjay_f/article/details/48830311

Recycleview 是 ListView 的更高度定制版，也可以说是升级版，当你需要高效的展示大量数据时候，动态改变元素的列表的时候，就用这个。

如果只是动态展示数据，listview 也可以做到，用 RecyclerView 替代 listview 的原因有几个：

优点：

1.简介中提到的它封装了 viewholder 的回收复用。

2.RecyclerView 使用布局管理器管理子 view 的位置，也就是说你再不用拘泥于 ListView 的线性展示方式，如果之后提供其他 custom LayoutManager 的支持，你能够使用复杂的布局来展示一个动态组件。

3.自带了 ItemAnimation，可以设置加载和移除时的动画，方便做出各种动态浏览的效果。

4.分开的 view

缺点：

目前相对于我们对 listview 经常用到的方法，有下面两个问题：

1. 不能简单的加头和尾:

不能简单的添加 Head 和 Footer ，因为没有直接的 addHead 和 addFoot 的方法了

2. 不能简单的设置子 item 的点击事件:

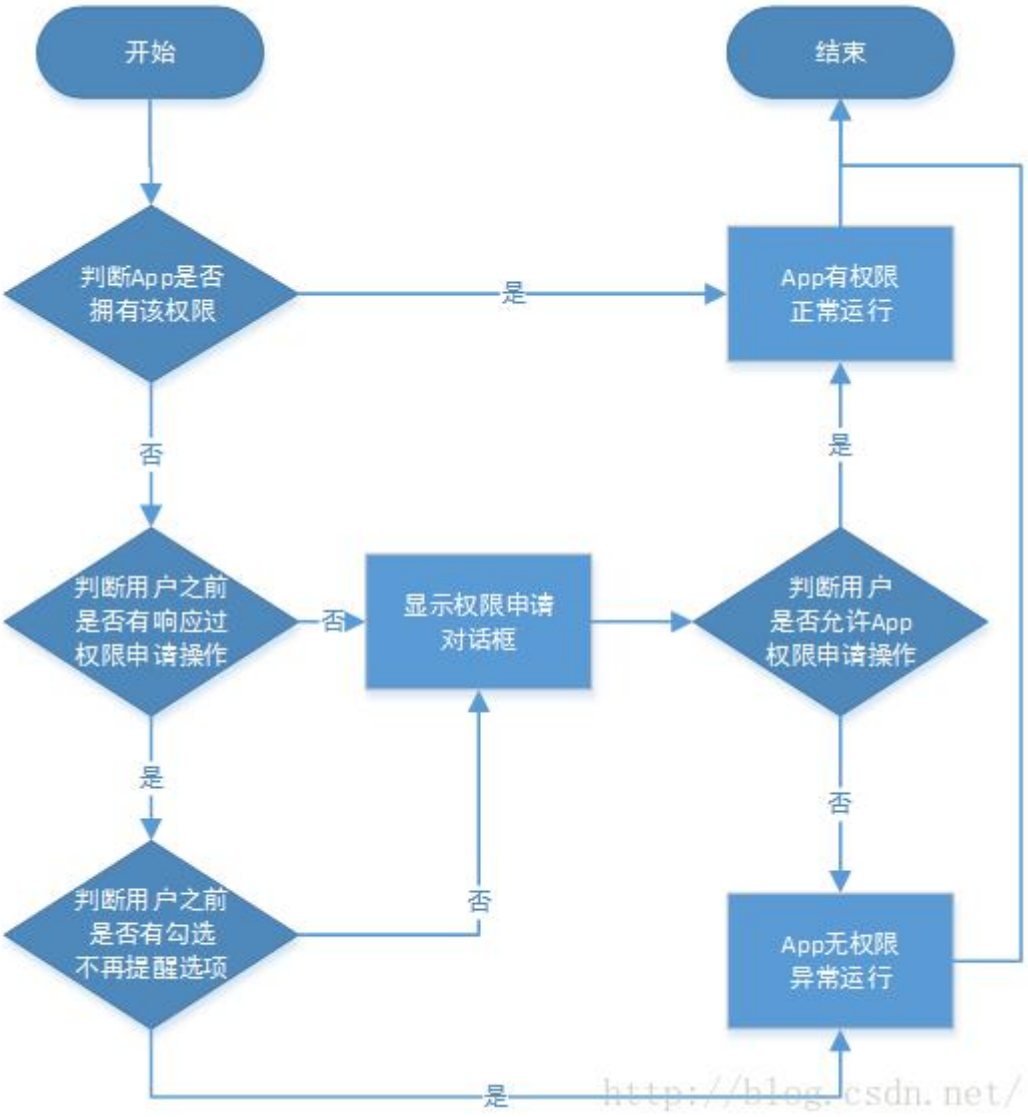
RecycleView，会发现没有这个接口了，解决办法如下：

让你的 viewholder 实现 onClickListener,然后在这个方法里面回调我们自己写的接口。

接着在你的 Adapter 里面加多个 set 方法，里面设置回调接口

11. 动态权限适配方案，权限组的概念

https://blog.csdn.net/xc765926174/article/details/49103483



动态权限申请流程图如上：对于在运行时需要申请权限的方法，在执行前需要做一系列的权限申请操作。

首先进行应用权限检查，检查应用是否有执行该方法所需要的权限，如有则可以正常运行，如没有则进行权限申请。

进行权限申请时，首先会检查一下用户之前是否有响应过该权限的申请，如没有，则显示权限申请对话框，如有，则检查用户之前是否有勾选“不再提醒”的选项，如没有，显示权限申请对话框，如有，则进行应用无权限异常运行。

显示权限对口框后，判断用户是否允许应用权限申请，如是，则进行应用有权限正常运行，如否，则进行应用无权限异常运行。

具体方法：

1) 使用 Context.checkSelfPermission()接口先检查权限是否授权。

2) 使用 Activity.shouldShowRequestPermissionRationale()接口检查用户是否勾选不再提醒。

3) 第 2 步返回为 true 时，表示用户并未勾选不再提醒选项，使用 Activity.requestPermissions()接口向系统请求权限。

4) 第 2 步返回为 false 时，表示用户已勾选不再提醒选项，则应用该弹框提示用户。

5) 第 3 步执行后，不论用户是否授予权限，都会回调 Activity.onRequestPermissionsResult()的函数。在 Activity 中重载 onRequestPermissionsResult()函数，在接收授权结果，根据不同的授权结果做相应的处理。

动态权限申请的适配方案：

对于低于 M 版本（安卓 6.0）的 Android 系统，没有动态权限的申请问题，动态权限的申请流程对于低于 M 版本的 Android 系统也

不再适用。所以适配方案，首先要考虑低于 M 版本的 Android 系统，因此对于 Android 版本小于 M 版本时，在检查权限时，直接返回 true 就 OK，直接屏蔽后续流程。

对于 M 版本（安卓 6.0）的 Android 系统，动态权限的申请流程会产生好几个分支处理逻辑，这样不善于管理和维护。所以对于此处，为了将写得代码更加整洁和更易于维护，我们可以将动态权限申请进行一次封装，

新建一个空白的 activity 用户权限申请和回调，然后在 activity 外包装一层管理内，限制一个调用的入口和出口，对于外部暴露唯一的入口和出口，这样对于外部逻辑代码需要调用权限时，将变得异常简单，并且由于将权限申请封装在了管理类中，

对于低于 M 版本的 Android 系统也将没有任何引用，在管理类中直接对于低于 M 版本的权限申请请求直接回调全部已授权即可。

实际 App 中动态权限申请代码如下：

1) 由于权限的动态申请与管理应该是伴随着整个 App 的生命周期的，所以 PermissionsManager 设计为单例的，且初始化保存着 applicationContext 作为默认跳转方式。

2) 在 App 代码中，应在自定义的 Application 中调用 PermissionsManager 中的 initContext()方法。

3) 在 App 代码中，在需要申请权限的地方调用 PermissionsManager 中的 newRequestPermissions()方法即可，不论 Android 版本是低于 M 版本还是高于 M 版本，根据实际 App 是否拥有对应权限，回调传入的该次权限申请的 PermissionsResultsCallback 接口的 onRequestPermissionsResult()方法。这样就将动态权限的申请结果处理流程给唯一化，将权限的申请的入口和出口都唯一化。

4) 这样将权限申请的细节和 Android 系统版本适配的细节都封装在了 PermissionsManager 内部，对于应用外部只需要知道申请权限的入口以及申请权限接口的出口即可。

12. Android 系统为什么会设计 ContentProvider？

https://www.cnblogs.com/zhainanJohnny/articles/3275908.html

ContentProvider 应用程序间非常通用的共享数据的一种方式，也是 Android 官方推荐的方式。Android 中许多系统应用都使用该方式实现数据共享，比如通讯录、短信等。

设计用意在于：

1) **封装。**对数据进行封装，提供统一的接口，使用者完全不必关心

这些数据是在 DB，XML、Preferences 或者网络请求来的。当项目需求要改变数据来源时，使用我们的地方完全不需要修改。

2) **提供一种跨进程数据共享的方式。**

3) **就是数据更新通知机制了。**因为数据是在多个应用程序中共享的，当其中一个应用程序改变了这些共享数据的时候，它有责任通知其它应用程序，让它们知道共享数据被修改了，这样它们就可以作相应的处理。

ContentResolver 接口的 notifyChange 函数来通知那些注册了监控特定 URI 的 ContentObserver 对象，使得它们可以相应地执行一些处理。ContentObserver 可以通过 registerContentObserver 进行注册。

1) ContentProvider 和调用者在同一个进程，ContentProvider 的方法（query/insert/update/delete 等）和调用者在同一线程中；

2) ContentProvider 和调用者在不同的进程，ContentProvider 的方法会运行在它自身所在进程的一个 Binder 线程中。

13. 下拉状态栏是不是影响 activity 的生命周期

不会影响

14. 如果在 onStop 的时候做了网络请求，onResume 的时候怎么恢复？（暂无）

15. Bitmap 使用时候注意什么？

https://blog.csdn.net/newbie_coder/article/details/9842995

1. 使用decodeStream获取Bitmap
ImageView.setImageBitmap、ImageView.setImageResource、BitmapFactory.decodeResource: 通过Java层的createBitmap完成，消耗更多内存
BitmapFactory.decodeStream: JNI >> nativeDecodeAsset，更加节约内存

```
1 InputStream is = getResources().openRawResource(R.drawable.pic);
2 Bitmap bitmap = BitmapFactory.decodeStream(is);
```

注：decodeStream方法得到的Bitmap长宽是其他方法的1/2，等于原图的长宽像素值。具体原因不清楚。

2. Decode时使用BitmapFactory.Options参数

a)Options.inSampleSize, 成比例放缩

```
BitmapFactory.Options ops = new BitmapFactory.Options();
ops.inSampleSize = 2; // 取样比例，得到的结果长宽是原图的1/2
Bitmap bitmap = BitmapFactory.decodeStream(is, null, ops);
```

b)Options.inJustDecodeBounds，只获取长宽，不获取图片

```
BitmapFactory.Options ops = new BitmapFactory.Options();
ops.inJustDecodeBounds = true; // 只解码边界值（长宽）
Bitmap bitmap = BitmapFactory.decodeStream(is, null, ops); // 此处得到的Bitmap为null
int bmpWidth = ops.outWidth; // 原始图片的宽度
int bmpHeight = ops.outHeight; // 原始图片的高度
```

登陆后复制

注：奇怪的是，只获取长宽的话，几种方法获得的大小都一样，等于decodeStream获取的长宽值，也是图片的原长宽值。

c)Options.inPreferredConfig，修改图片编码格式（默认是Bitmap.Config.ARGB_8888）

编码方式有：（ARGB分别代表透明度、红、绿、蓝）

Bitmap.Config ALPHA_8　占8位（具体情况不清楚）

Bitmap.Config ARGB_4444 占16位（ARGB各4位）

Bitmap.Config ARGB_8888 占32位（ARGB各8位）

Bitmap.Config RGB_565　占16位（R5位G6位B5位）

d)其他

```
/* 下面两个字段需要组合使用 */
options.inPurgeable = true;
options.inInputShareable = true;
```

3. 手动回收Bitmap

```
if(!bitmap.isRecycled()) {
        bitmap.recycle();
}
```

16. Bitmap 的 recycler()

https://blog.csdn.net/lonelyroamer/article/details/7569248

如果只是使用少量的几张图片，回收与否关系不大。可是若有大量 bitmap 需要垃圾回收处理，那必然垃圾回收需要做的次数就更多也发生地更频繁，会对系统资源造成负荷。所以，这个时候还是自己试用 recycle 来释放的比较好。

```
// 获得ImageView当前显示的图片
Bitmap bitmap1 = ((BitmapDrawable) imageView.getBackground()).getBitmap();

Bitmap bitmap2 = Bitmap.createBitmap(bitmap1, 0, 0, bitmap1.getWidth(),
                bitmap1.getHeight(), matrix, true);
// 如果图片还没有回收，强制回收
if (!bitmap1.isRecycled()) {
        bitmap1.recycle();
}
// 根据原始位图和Matrix创建新的图片
imageView.setImageBitmap(bitmap2);
```

登陆后复制

因为imageView.setImageBitmap()方法设置的是对应的src的图片，而不是background。而我却把background的图片给回收了。这样，就导致了异常的发生。

所以修改成如下的代码，去调用setBackGround()方法

```
// 获得ImageView当前显示的图片
Bitmap bitmap1 = ((BitmapDrawable) imageView.getBackground()).getBitmap();
Bitmap bitmap2 = Bitmap.createBitmap(bitmap1, 0, 0, bitmap1.getWidth(),
                bitmap1.getHeight(), matrix, true);
// 如果图片还没有回收，强制回收
if (!bitmap1.isRecycled()) {
        bitmap1.recycle();
}
// 根据原始位图和Matrix创建新的图片
//imageView.setImageBitmap(bitmap2);
imageView.setBackgroundDrawable(new BitmapDrawable(bitmap2));
```

所以，一定要注意ImageView图片的来源问题，然后在进行相应的recycle。

17. Android 中开启摄像头的主要步骤

https://www.yiibai.com/android/android_camera.html

主要思路：

(1) 获得摄像头管理器 CameraManager mCameraManager，mCameraManager.openCamera()来打开摄像头

(2) 指定要打开的摄像头，并创建 openCamera()所需要的 CameraDevice.StateCallback stateCallback

(3) 在 CameraDevice.StateCallback stateCallback 中调用 takePreview()，这个方法中，使用 CaptureRequest.Builder 创建预览需要的 CameraRequest，并初始化了 CameraCaptureSession，最后调用了 setRepeatingRequest(previewRequest, null, childHandler)进行了预览

(4) 点击屏幕，调用 takePicture()，这个方法内，最终调用了 capture(mCaptureRequest, null, childHandler)

(5) 在 new ImageReader.OnImageAvailableListener(){}回调方法中，将拍照拿到的图片进行展示

18. ViewPager 使用细节，如何设置成每次只初始化当前的 Fragment，其他的不初始化？

https://www.jianshu.com/p/e324e8378948

https://blog.csdn.net/linglongxin24/article/details/53205878

我们可不可以设置 ViewPager 的预加载为 0，不能解决问题

```
vp.setOffscreenPageLimit(0);
```

关键在于 setUserVisibleHint(boolean isVisibleToUser)方法

```
/**
 * Fragment 预加载问题的解决方案：
 * 1.可以懒加载的 Fragment
 * 2.切换到其他页面时停止加载数据（可选）
 * Created by yuandl on 2016-11-17.
 *blog :
http://blog.csdn.net/linglongxin24/article/details/53205878
 */
```

1. 初始化的时候去加载数据

2. 在 setUserVisibleHint()方法中设置加载数据方法，加载数据方法中设置检测方法，如果视图已经初始化而且视图对用户可见（getUserVisibleHint()为 true），则加载数据

3. 可选：当视图已经对用户不可见并且加载过数据，如果需要在切换到其他页面时停止加载数据，

```
public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
    view = inflater.inflate(setContentView(), container, false);
    isInit = true;
    /**初始化的时候去加载数据**/
    isCanLoadData();
    return view;
}

/**
* 视图是否已经对用户可见，系统的方法
*/
@Override
public void setUserVisibleHint(boolean isVisibleToUser) {
    super.setUserVisibleHint(isVisibleToUser);
    isCanLoadData();
}

/**
* 是否可以加载数据
* 可以加载数据的条件：
* 1.视图已经初始化
* 2.视图对用户可见
*/
private void isCanLoadData() {
    if (!isInit) {
        return;
    }

    if (getUserVisibleHint()) {
        lazyLoad();
        isLoad = true;
    } else {
        if (isLoad) {
            stopLoad();
        }
    }
}
```

azyLoadFragment 是一个抽象类，可以作为 BaseFragment,继承它。

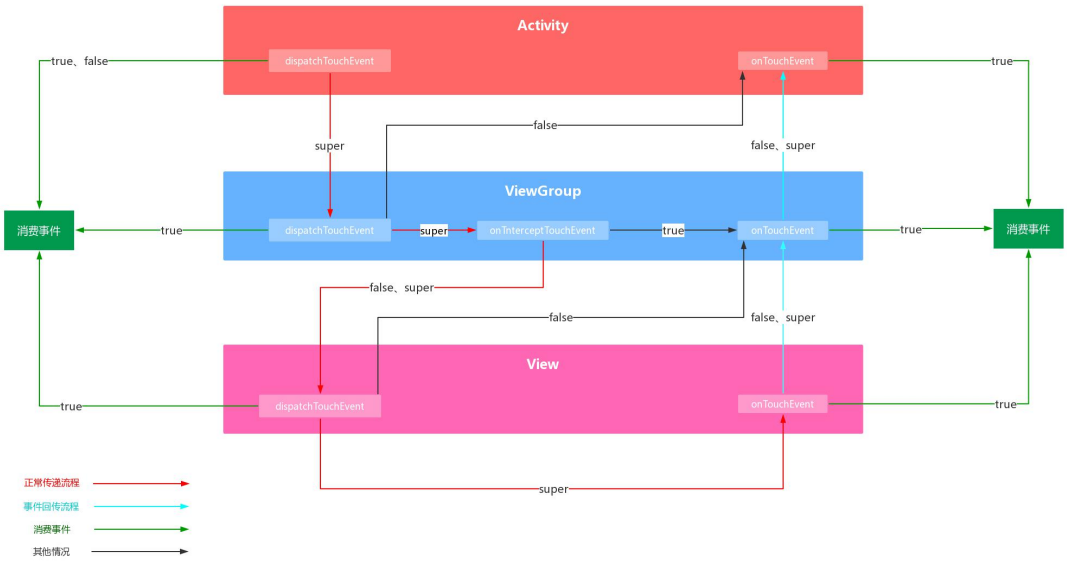
(1).用 setContentView()方法去加载要显示的布局

(2).lazyLoad()方法去加载数据

(3).stopLoad()方法可选，当视图已经对用户不可见并且加载过数据，如果需要在切换到其他页面时停止加载数据，可以覆写此方法

19. 点击事件被拦截，但是想传到下面的 View，如何操作？

https://www.jianshu.com/p/e99b5e8bd67b



图分 3 层，从上往下依次是 Activity、ViewGroup、View

他们返回值分别有 true、false、super.XXX

如果我们没有对控件里面的方法进行重写或更改返回值，而直接用 super 调用父类的默认实现，那么整个事件流向应该是从 Activity---->ViewGroup--->View 从上往下调用

dispatchTouchEvent 方法，一直到叶子节点（View）的时候，

再由 View--->ViewGroup--->Activity 从下往上调用 onTouchEvent 方法。

dispatchTouchEvent 和 onTouchEvent 一旦 return true,事件就停止传递了（到达终点）（没有谁能再收到这个事件）。看下图中只要 return true 事件就没再继续传下去了，对于 return true 我们经常说事件被消费了，消费了的意思就是事件走到这里就是终点，不会往下传，没有谁能再收到这个事件了。

有以下几种情况会消耗事件（停止事件传递）：

1) Activity 中的 dispatchTouchEvent（）返回 true、false

2) Activity 中的 onTouchEvent（）返回 true

3) ViewGroup 中的 dispatchTouchEvent（）返回 true

4) ViewGroup 中的 onTouchEvent（）返回 true

5) View 中的 dispatchTouchEvent（）返回 true

6) View 中的 onTouchEvent（）返回 true

对于 dispatchTouchEvent 返回 false 的含义应该是：事件停止往子 View 传递和分发同时开始往父控件回溯（父控件的 onTouchEvent 开始从下往上回传直到某个 onTouchEvent return true），事件分发机制就像递归，return false 的意义就是递归停

止然后开始回溯。

对于 onTouchEvent return false 就比较简单了，它就是不消费事件，并让事件继续往父控件的方向从下往上流动。

onInterceptTouchEvent 的作用: Intercept 的意思就拦截，每个 ViewGroup 每次在做分发的时候，问一问拦截器要不要拦截（也就是问问自己这个事件要不要自己来处理）如果要自己处理那就在 onInterceptTouchEvent 方法中 return true 就会交给自己的 onTouchEvent 的处理，如果不拦截就是继续往子控件往下传。默认是不会去拦截的，因为子 View 也需要这个事件，所以 onInterceptTouchEvent 拦截器 return super.onInterceptTouchEvent()和 return false 是一样的，是不会拦截的，事件会继续往子 View 的 dispatchTouchEvent 传递。

ViewGroup 怎样通过 dispatchTouchEvent 方法能把事件分发到自己的 onTouchEvent 处理呢，return true 和 false 都不行，那么只能通过 Interceptor 把事件拦截下来给自己的 onTouchEvent，所以 ViewGroup dispatchTouchEvent 方法的 super 默认实现就是去调用 onInterceptTouchEvent，记住这一点。

但是同样的道理 return true 是终结。return false 是回溯会父类的 onTouchEvent，怎样把事件分发给自己的 onTouchEvent 处理呢，那只能 return super.dispatchTouchEvent,View 类的 dispatchTouchEvent（）方法默认实现就是能帮你调用 View 自己的

onTouchEvent 方法的。

总结一下：

1) 对于 dispatchTouchEvent，onTouchEvent，return true 是终结事件传递。return false 是回溯到父 View 的 onTouchEvent 方法。

2) ViewGroup 想把自己分发给自己的 onTouchEvent，需要拦截器 onInterceptTouchEvent 方法 return true 把事件拦截下来。

3) ViewGroup 的拦截器 onInterceptTouchEvent 默认是不拦截的，所以 return super.onInterceptTouchEvent()=return false;

4) View 没有拦截器，为了让 View 可以把事件分发给自己的 onTouchEvent，View 的 dispatchTouchEvent 默认实现（super）就是把事件分发给自己的 onTouchEvent。

ACTION_DOWN 事件流向会先走整个事件流程，最后被某个控件中的某个事件消费

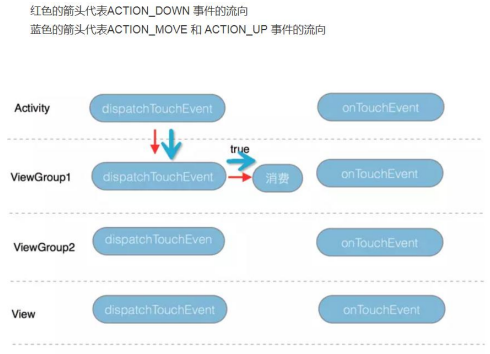
ACTION_MOVE 和 ACTION_UP 事件会依据 ACTION_DOWN 中事件分发，得知 ACTION_MOVE 被哪个控件消费哪个事件消费。

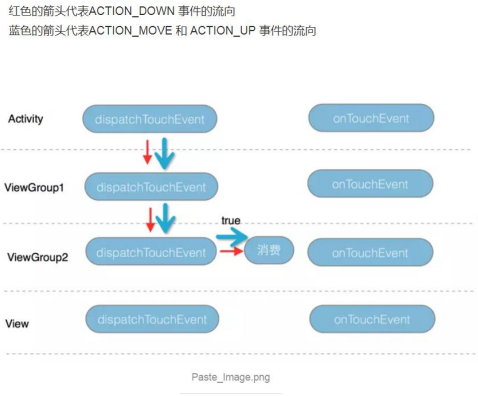
走事件流程的时候，走到该控件，便不再往下走，然后接着走到该控件中处理过 ACTION_DOWN 的消息事件中，交给其处理 ACTION_MOVE 和 ACTION_UP。

在哪个 View 的 onTouchEvent 返回 true，那么 ACTION_MOVE 和 ACTION_UP 的事件从上往下传到这个 View 后就不再往下传递了，而

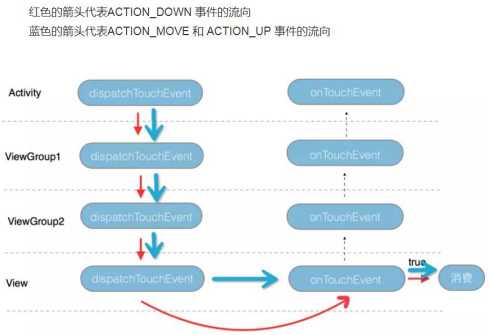
直接传给自己的 onTouchEvent 并结束本次事件传递过程。

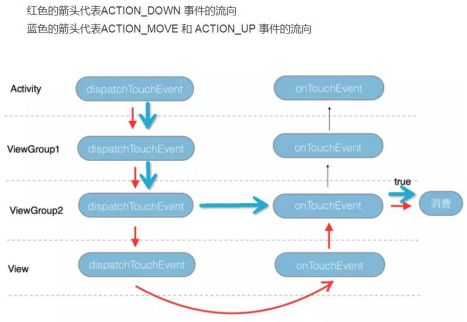
对于 ACTION_MOVE、ACTION_UP 总结：ACTION_DOWN 事件在哪个控件消费了（return true），那么 ACTION_MOVE 和 ACTION_UP 就会从上往下（通过 dispatchTouchEvent）做事件分发往下传，就只会传到这个控件，不会继续往下传，如果 ACTION_DOWN 事件是在 dispatchTouchEvent 消费，那么事件到此为止停止传递，如果 ACTION_DOWN 事件是在 onTouchEvent 消费的，那么会把 ACTION_MOVE 或 ACTION_UP 事件传给该控件的 onTouchEvent 处理并结束传递。



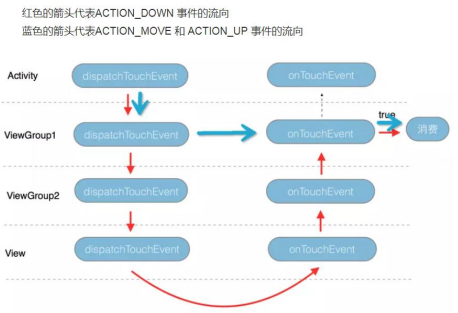


Paste_Image.png

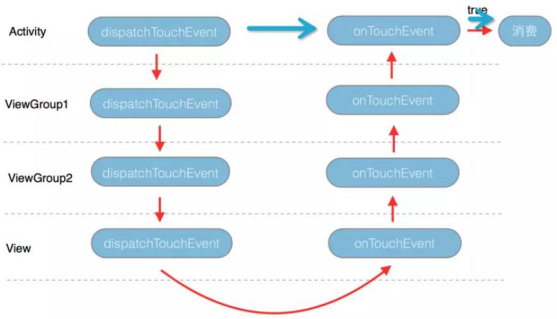




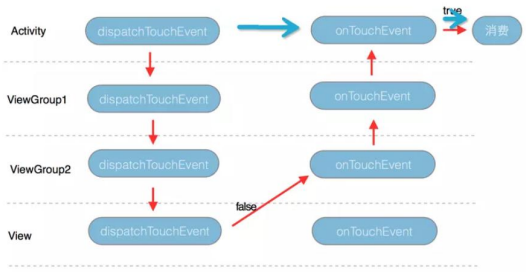
红色的箭头代表ACTION_DOWN 事件的流向
蓝色的箭头代表ACTION_MOVE 和 ACTION_UP 事件的流向



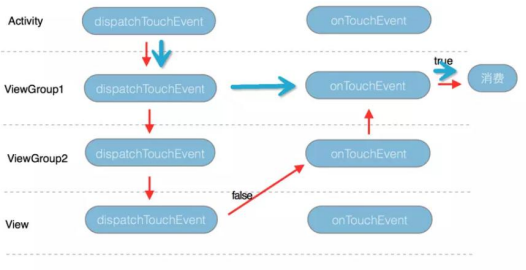
红色的箭头代表ACTION_DOWN 事件的流向
蓝色的箭头代表ACTION_MOVE 和 ACTION_UP 事件的流向



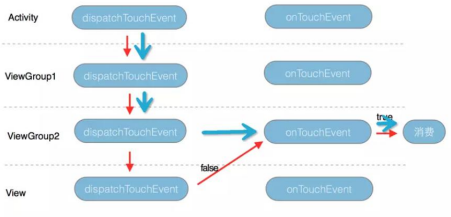
红色的箭头代表ACTION_DOWN 事件的流向
蓝色的箭头代表ACTION_MOVE 和 ACTION_UP 事件的流向



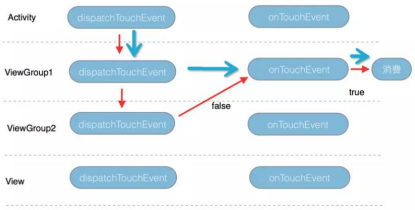
红色的箭头代表ACTION_DOWN 事件的流向
蓝色的箭头代表ACTION_MOVE 和 ACTION_UP 事件的流向



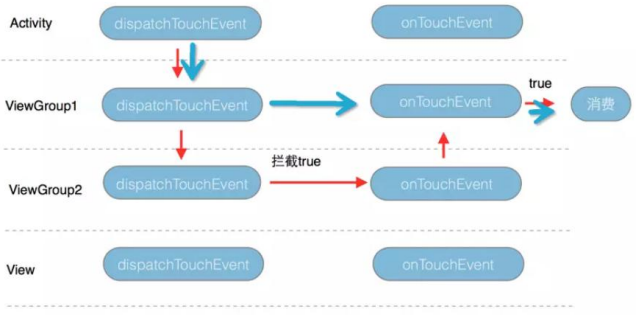
红色的箭头代表ACTION_DOWN 事件的流向
蓝色的箭头代表ACTION_MOVE 和 ACTION_UP 事件的流向



红色的箭头代表ACTION_DOWN 事件的流向
蓝色的箭头代表ACTION_MOVE 和 ACTION_UP 事件的流向



红色的箭头代表ACTION_DOWN 事件的流向
蓝色的箭头代表ACTION_MOVE 和 ACTION_UP 事件的流向



20. 微信主页面的实现方式

自己去研究下吧这个，无非 fragment 嵌套

21. 微信上消息小红点的原理（问哪个？自定义那种，还是桌面那种）

22. CAS 介绍（这是阿里巴巴的面试题，我不是很了解，可以参考博客:CAS 简介）

暂时不看