1.导致内存泄漏的原因
2.Activity、View、Window三者关系
3.View，ViewGroup事件分发
4.onNewIntent()什么时候调用?(singleTask)
5.自定义控件
6.Serializable和Parcelable的区别
7.Android为什么要序列化？什么是序列化，怎么进行序列化
8.四大组件简介
9.四大组件的生命周期
10.Activity之间的通信方式
11.Activity各种情况下的生命周期
12.Activity与Fragment之间生命周期比较
13.Activity上有Dialog的时候按Home键时的生命周期
14.两个Activity 之间跳转时必然会执行的是哪几个方法?
15.前台切换到后台，然后再回到前台，Activity生命周期回调方法。弹出Dialog，生命值周期回调方法。
16.Activity的四种启动模式对比
17.Activity状态保存于恢复
18.Fragment各种情况下的生命周期
19.Fragment状态保存onSaveInstanceState是哪个类的方法，在什么情况下使用?
20.Fragment.startActivityForResult是和FragmentActivity的startActivityForResult?
21.如何实现Fragment的滑动?
22.fragment之间传递数据的方式?
23.service和activity怎么进行数据交互?
24.说说ContentProvider、ContentResolver、ContentObserver之间的关系
25.请描述一下广播BroadcastReceiver的理解
26.广播的分类
27.广播使用的方式和场景
28.**在manifest 和代码中如何注册和使用BroadcastReceiver?**
29.**本地广播和全局广播有什么差别?**
30.AlertDialog,popupWindow,Activity区别
31.**Application 和 Activity 的 Context 对象的区别**
32.LinearLayout、RelativeLayout、FrameLayout的特性及对比，并介绍使用场景。

1.导致内存泄漏的原因

> **根本原因:** 长生命周期的对象持有短生命周期的对象。短周期对象就无法及时释放。
>
> **场景:**
>
> - 静态内部类非静态内部类的区别(Handler 引起的内存泄漏。)
> - 静态集合类引起内存泄露
> - 单例模式引起的内存泄漏。
> - 注册/反注册未成对使用引起的内存泄漏。

- 资源未关闭，如cursor、I/O、bitmap等。

解决：

- Context是ApplicationContext，由于ApplicationContext的生命周期是和app一致的，不会导致内存泄漏。
- 集合对象没有及时清理引起的内存泄漏。通常会把一些对象装入到集合中，当不使用的时候一定要记得及时清理集合，让相关对象不再被引用。
- 注册与反注册成对出现
- 减少内存对象的占用

  I.ArrayMap/SparseArray代替hashmap

  II.避免在android里面使用Enum

  III.减少bitmap的内存占用。

  IV.减少资源图片的大小，过大的图片可以考虑分段加载

2.Activity、View、Window三者关系

Activity像一个工匠（控制单元），Window像窗户（承载模型），View像窗花（显示视图）LayoutInflater像剪刀，Xml配置像窗花图纸。

- Activity构造的时候会初始化一个Window，准确的说是PhoneWindow。
- 这个PhoneWindow有一个“ViewRoot”，这个“ViewRoot”是一个View或者说ViewGroup，是最初始的根视图。
- “ViewRoot”通过addView方法来一个个的添加View。比如TextView，Button等
- 这些View的事件监听，是由WindowManagerService来接受消息，并且回调Activity函数。比如onClickListener，onKeyDown等。

3.View，ViewGroup事件分发

- Touch事件分发中只有两个主角:ViewGroup和View。ViewGroup包含onInterceptTouchEvent、dispatchTouchEvent、onTouchEvent三个相关事件。View包含dispatchTouchEvent、onTouchEvent两个相关事件。其中ViewGroup又继承于View。
- ViewGroup和View组成了一个树状结构，根节点为Activity内部包含的一个ViwGroup。
- 触摸事件由Action_Down、Action_Move、Aciton_UP组成，其中一次完整的触摸事件中，Down和Up都只有一个，Move有若干个，可以为0个
- 当Acitivty接收到Touch事件时，将遍历子View进行Down事件的分发。ViewGroup的遍历可以看成是递归的。分发的目的是为了找到真正要处理本次完整触摸事件的View，这个View会在onTouchuEvent结果返回true。
- 当某个子View返回true时，会中止Down事件的分发，同时在ViewGroup中记录该子View。接下去的Move和Up事件将由该子View直接进行处理。由于子View是保存在ViewGroup中的，多层ViewGroup的节点结构时，上级ViewGroup保存的会是真实处理事件的View所在的ViewGroup对象:如ViewGroup0-ViewGroup1-TextView的结构中，TextView返回了true，它将被保存在ViewGroup1中，而ViewGroup1也会返回true，被保存在ViewGroup0中。当Move和UP事件来时，会先从ViewGroup0传递至ViewGroup1，再由ViewGroup1传递至TextView。
- 当ViewGroup中所有子View都不捕获Down事件时，将触发ViewGroup自身的onTouch事件。触发的方式是调用super.dispatchTouchEvent函数，即父类View的dispatchTouchEvent方法。在所有子View都不处理的情况下，触发Acitivity的onTouchEvent方法。

- onInterceptTouchEvent有两个作用：1.拦截Down事件的分发。2.中止Up和Move事件向目标View传递，使得目标View所在的ViewGroup捕获Up和Move事件。

4.onNewIntent()什么时候调用?(singleTask)

当Activity被设以singleTop启动，当需要再次响应此Activity启动需求时，会复用栈顶的已有Activity，还会调用onNewIntent方法。

并且，再接受新发送来的intent(onNewIntent方法)之前，一定会先执行onPause方法。

5.自定义控件

View的绘制流程：OnMeasure()——>OnLayout()——>OnDraw()

- OnMeasure()：测量视图大小。从顶层父View到子View递归调用measure方法，measure方法又回调OnMeasure。
- OnLayout()：确定View位置，进行页面布局。从顶层父View向子View的递归调用view.layout方法的过程，即父View根据上一步measure子View所得到的布局大小和布局参数，将子View放在合适的位置上。
- OnDraw()：绘制视图。ViewRoot创建一个Canvas对象，然后调用OnDraw()。六个步骤：①、绘制视图的背景；②、保存画布的图层（Layer）；③、绘制View的内容；④、绘制View子视图，如果没有就不用；⑤、还原图层（Layer）；⑥、绘制滚动条。

6.Serializable和Parcelable的区别

- P 消耗内存小
- 网络传输用S， 程序内使用P
- S将数据持久化方便
- S使用了反射 容易触发垃圾回收比较慢，频繁GC;
- S代码量少，P序列化复杂，代码量大；

7.Android为什么要序列化？什么是序列化，怎么进行序列化

> **why**
>
> 行Android开发的时候，无法将对象的引用传给Activities或者Fragments，我们需要将这些对象放到一个Intent或者Bundle里面，然后再传递。

> **what**
>
> 序列化，表示将一个对象转换成可存储或可传输的状态。序列化后的对象可以在网络上进行传输，也可以存储到本地。

> **how**
>
> Android中Intent如果要传递类对象，可以通过两种方式实现。
>
> - 方式一：Serializable，要传递的类实现Serializable接口传递对象，
> - 方式二：Parcelable，要传递的类实现Parcelable接口传递对象。

> **Serializable（Java自带）：**

> Serializable是序列化的意思，表示将一个对象转换成可存储或可传输的状态。序列化后的对象可以在网络上进行传输，也可以存储到本地。

> **Parcelable（android 专用）：**
>
> Parcelable方式的实现原理是将一个完整的对象进行分解，而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的功能了。

> **实现Parcelable的作用**
>
> - 永久性保存对象，保存对象的字节序列到本地文件中；
> - 通过序列化对象在网络中传递对象；
> - 通过序列化在进程间传递对象。

> **选择序列化方法的原则**
>
> - 在使用内存的时候，Parcelable比Serializable性能高，所以推荐使用Parcelable。
> - Parcelable不能使用在要将数据存储在磁盘上的情况，因为Parcelable不能很好的保证数据的持续性在外界有变化的情况下。尽管Serializable效率低点，但此时还是建议使用 Serializable 。

> **应用场景**
>
> 需要在多个部件(Activity或Service)之间通过Intent传递一些数据，简单类型（如：数字、字符串）的可以直接放入Intent。复杂类型必须实现Parcelable接口。

8.四大组件简介

> **Activity**
>
> 用户可操作的可视化界面，为用户提供一个完成操作指令的窗口。一个Activity通常是一个单独的屏幕，Activity通过Intent来进行通信。Android中会维持一个Activity Stack，当一个新Activity创建时，它就会放到栈顶，这个Activity就处于运行状态。

> **Service**
>
> 运行在手机后台，适合执行不需和用户交互且还需长期运行的任务。

> **BroadcastReceiver**
>
> 运用在应用程序间传输信息，可以使用广播接收器来让应用对一个外部事件做出响应。

> **ContentProvider**
>
> 使一个应用程序的指定数据集提供给其他应用程序，其他应用可通过ContentResolver类从该内容提供者中获取或存入数据。它提供了一种跨进程数据共享的方式，当数据被修改后，ContentResolver接口的notifyChange函数通知那些注册监控特定URI的ContentObserver对象。

9.四大组件的生命周期

> **Activity**
>
> onCreate()->onStart()->onResume()->onPause()->onStop()->onDestory()
>
> onCreate(): 为Activity设置布局，此时界面还不可见；

> onStart(): Activity可见但还不能与用户交互，不能获得焦点;
>
> onRestart(): 重新启动Activity时被回调;
>
> onResume(): Activity可见且可与用户进行交互;
>
> onPause(): 当前Activity暂停，不可与用户交互，但还可见。在新Activity启动前被系统调用保存现有的Activity中的持久数据、停止动画等;
>
> onStop(): 当Activity被新的Activity覆盖不可见时被系统调用
>
> onDestory(): 当Activity被系统销毁杀掉或是由于内存不足时调用

> **Service**
>
> - onBind方式绑定的
>
>   onCreate->onBind->onUnBind->onDestory（不管调用bindService几次，onCreate只会调用一次，onStart不会被调用，建立连接后，service会一直运行，直到调用unBindService或是之前调用的bindService的Context不存在了，系统会自动停止Service,对应的onDestory会被调用）。
>
> - startService启动的：
>
>   onCreate->onStartCommand->onDestory(start多次，onCreate只会被调用一次，onStart会调用多次，该service会在后台运行，直至被调用stopService或是stopSelf)。
>
> - 又被启动又被绑定的服务：
>
>   不管如何调用onCreate()只被调用一次，startService调用多少次，onStart就会被调用多少次，而unbindService不会停止服务，必须调用stopService或是stopSelf来停止服务。必须unbindService和stopService(stopSelf）同时都调用了才会停止服务。
>
> **BroadcastReceiver**
>
> - 动态注册
>
>   存活周期是在Context.registerReceiver和Context.unregisterReceiver之间，BroadcastReceiver每次收到广播都是使用注册传入的对象处理的。
>
> - 静态注册
>
>   进程在的情况下，receiver会正常收到广播，调用onReceive方法；生命周期只存活在onReceive函数中，此方法结束，BroadcastReceiver就销毁了。onReceive()只有十几秒存活时间，在onReceive()内操作超过10S，就会报ANR。
>
> 进程不存在的情况，广播相应的进程会被拉活，Application.onCreate会被调用，再调用onReceive。
>
> **ContentProvider**
>
> 应该和应用的生命周期一样，它属于系统应用，应用启动时，它会跟着初始化，应用关闭或被杀，它会跟着结束。

10.Activity之间的通信方式

- 通过Intent方式传递参数跳转
- 通过广播方式

- 通过接口回调方式
- 借助类的静态变量或全局变量
- 借助SharedPreference或是外部存储，如数据库或本地文件

11.Activity各种情况下的生命周期

- Activity各种情况下的生命周期

  onPause(A)->onCreate(B)->onStart(B)->onResume(B)->oStop(A)

  这时如果按回退键回退到A onPause(B)->onRestart(A)->onStart(A)->onResume(A)->oStop(B)

  如果在切换到B后调用了A.finish()，则会走到onDestory(A)，这时点回退键会退出应用
- 如果在切换到B后调用了A.finish()，则会走到onDestory(A)，这时点回退键会退出应用

  onPause(A)->onCreate(B)->onStart(B)->onResume(B)

  这时如果回退到A onPause(B)->onResume(A)->oStop(B)->onDestory(B)
- Activity(A)启动后点击Home键再回到应用的生命周期

  onPause(A)->oStop(A)->onRestart(A)->onStart(A)->onResume(A)
- 横竖屏切换的时候，Activity 各种情况下的生命周期**

  > - 切换横屏时：onSaveInstanceState->onPause->onStop->onDestory->onCreate->onStart->onRestoreInstanceState->onResume
  > - 切换竖屏时：会打印两次相同的log
  >
  >   onSaveInstanceState->onPause->onStop->onDestory->onCreate->onStart->onRestoreInstanceState->onResume->onSaveInstanceState->onPause->onStop->onDestory->onCreate->onStart->onRestoreInstanceState->onResume
  > - 如果在AndroidMainfest.xml中修改该Activity的属性，添加 android:configChanges="orientation"
  >
  >   横竖屏切换，打印的log一样，同1)
  > - 如果AndroidMainfest.xml中该Activity中的 android:configChanges="orientation|keyboardHidden"，则只会打印
  >
  >   onConfigurationChanged->

12.Activity与Fragment之间生命周期比较

> **Fragment生命周期:**
>
> onAttach->onCreate->onCreateView->onActivityCreated->onStart->onResume->onPause->onStop->onDestoryView->onDestory->onDetach
>
> **切换到该Fragment：**
>
> onAttach->onCreate->onCreateView->onActivityCreated->onStart->onResume
>
> **按下Power键：**
>
> onPause->onSaveInstanceState->onStop
>
> **点亮屏幕解锁：**

> onStart->onRestoreInstanceState->onResume
>
> **切换到其他Fragment:**
>
> onPause->onStop->onDestoryView
>
> **切回到该Fragment:**
>
> onCreateView->onActivityCreated->onStart->onResume
>
> **退出应用：**
>
> onPause->onStop->onDestoryView->onDestory->onDetach

13.Activity上有Dialog的时候按Home键时的生命周期

> AlertDialog并不会影响Activity的生命周期，按Home键后才会使Activity走onPause->onStop，AlertDialog只是一个组件，并不会使Activity进入后台。

14.两个Activity 之间跳转时必然会执行的是哪几个方法?

> 前一个Activity的onPause，后一个Activity的onResume

15.前台切换到后台，然后再回到前台，Activity生命周期回调方法。弹出Dialog，生命值周期回调方法。

> - 前台切换到后台，会执行onPause->onStop，再回到前台，会执行onRestart->onStart->onResume
> - 弹出Dialog，并不会影响Activity生命周期。

16.Activity的四种启动模式对比

> - **standard**
>
> 标准启动模式（默认），每启动一次Activity，都会创建一个实例，即使从ActivityA startActivity ActivityA,也会再次创建A的实例放于栈顶，当回退时，回到上一个ActivityA的实例。
> - **singleTop**
>
> 栈顶复用模式，每次启动Activity，如果待启动的Activity位于栈顶，则不会重新创建Activity的实例，即不会走onCreate->onStart，会直接进入Activity的onPause->onNewIntent->onResume方法。
> - **singleInstance**
>
> 单一实例模式，整个手机操作系统里只有一个该Activity实例存在，没有其他Actvity,后续请求均不会创建新的Activity。若task中存在实例，执行实例的onNewIntent()。应用场景：闹钟、浏览器、电话。
> - **singleTask**

> 栈内复用，启动的Activity如果在指定的taskAffinity的task栈中存在相应的实例，则会把它上面的Activity都出栈，直到当前Activity实例位于栈顶，执行相应的onNewIntent()方法。如果指定的task不存在，创建指定的taskAffinity的task。taskAffinity的作用，进入指写taskAffinity的task,如果指定的task存在，将task移到前台，如果指定的task不存在，创建指定的taskAffinity的task. 应用场景：应用的主页面

17.Activity状态保存于恢复

> Activity被主动回收时，如按下Back键，系统不会保存它的状态，只有被动回收时，虽然这个Activity实例已被销毁，但系统在新建一个Activity实例时，会带上先前被回收Activity的信息。在当前Activity被销毁前调用onSaveInstanceState(onPause和onStop之间保存)，重新创建Activity后会在onCreate后调用onRestoreInstanceState（onStart和onResume之间被调用），它们的参数Bundle用来数据保存和读取的。
>
> 保存View状态有两个前提：View的子类必须实现了onSaveInstanceState; 必须要设定Id，这个ID作为Bundle的Key。

18.Fragment各种情况下的生命周期

> - **正常情况下的生命周期**
>
>   onAttach->onCreate->onCreateView->onActivityCreated->onStart->onResume->onPause->onStop->onDestoryView->onDestory->onDetach
>
> - **Fragment在Activity中replace**
>
>   onPause(旧)->onAttach->onCreate->onCreateView->onActivityCreated->onStart->onResume->onStop(旧)->onDestoryView(旧)
>
>   如果添加到backStack中，调用remove()方法fragment的方法会走到onDestoryView，但不会执行onDetach()，即fragment本身的实例是存在的，成员变量也存在，但是view被销毁了。如果新替换的Fragment已在BackStack中，则不会执行onAttach->onCreate

19.Fragment状态保存onSaveInstanceState是哪个类的方法，在什么情况下使用?

> 在对应的FragmentActivity.onSaveInstanceState方法会调用FragmentController.saveAllState，其中会对mActive中各个Fragment的实例状态和View状态分别进行保存.当Activity在做状态保存和恢复的时候, 在它其中的fragment自然也需要做状态保存和恢复.

20.Fragment.startActivityForResult是和FragmentActivity的startActivityForResult?

> 如果希望在Fragment的onActivityResult接收数据，就要调用Fragment.startActivityForResult，而不是Fragment.getActivity().startActivityForResult。Fragment.startActivityForResult->FragmentActivitymHost.HostCallbacks.onStartActivityFromFragment->FragmentActivity.startActivityFromFragment。如果request=-1则直接调用FragmentActivity.startActivityForResult，它会重新计算requestCode，使其大于0xfffff。

21.如何实现Fragment的滑动?

> ViewPager+FragmentPagerAdapter+List

22.fragment之间传递数据的方式?

> - 在相应的fragment中编写方法，在需要回调的fragment里获取对应的Fragment实例，调用相应的方法；
> - 采用接口回调的方式进行数据传递；
> - 在Fragment1中创建一个接口及接口对应的set方法; b) 在Fragment1中调用接口的方法；c) 在Fragment2中实现该接口；
> - 利用第三方开源框架EventBus

23.service和activity怎么进行数据交互?

> - 通过bindService启动服务，可以在ServiceConnection的onServiceConnected中获取到Service的实例，这样就可以调用service的方法，如果service想调用activity的方法，可以在service中定义接口类及相应的set方法，在activity中实现相应的接口，这样service就可以回调接口言法；
> - 通过广播方式

24.说说ContentProvider、ContentResolver、ContentObserver之间的关系

> ContentProvider实现各个应用程序间数据共享，用来提供内容给别的应用操作。如联系人应用中就使用了ContentProvider，可以在自己应用中读取和修改联系人信息，不过需要获取相应的权限。它也只是一个中间件，真正的数据源是文件或SQLite等。
>
> ContentResolver内容解析者，用于获取内容提供者提供的数据，通过ContentResolver.notifyChange(uri)发出消息
>
> ContentObserver内容监听者，可以监听数据的改变状态，观察特定Uri引起的数据库变化，继而做一些相应的处理，类似于数据库中的触发器，当ContentObserver所观察的Uri发生变化时，便会触发它。

25.请描述一下广播BroadcastReceiver的理解

> BroadcastReceiver是一种全局监听器，用来实现系统中不同组件之间的通信。有时候也会用来作为传输少量而且发送频率低的数据，但是如果数据的发送频率比较高或者数量比较大就不建议用广播接收者来接收了，因为这样的效率很不好，因为BroadcastReceiver接收数据的开销还是比较大的。

26.广播的分类

> **1）普通广播**：完全异步的，可以在同一时刻（逻辑上）被所有接收者接收到，消息传递的效率比较高，并且无法中断广播的传播。
>
> **2）有序广播**：发送有序广播后，广播接收者将按预先声明的优先级依次接收Broadcast。优先级高的优先接收到广播，而在其onReceiver()执行过程中，广播不会传播到下一个接收者，此时当前的广播接收者可以abortBroadcast()来终止广播继续向下传播，也可以将intent中的数据进行修改设置，然后将其传播到下一个广播接收者。 sendOrderedBroadcast(intent, null);//发送有序广播

3）粘性广播：sendStickyBroadcast()来发送该类型的广播信息，这种的广播的最大特点是，当粘性广播发送后，最后的一个粘性广播会滞留在操作系统中。如果在粘性广播发送后的一段时间里，如果有新的符合广播的动态注册的广播接收者注册，将会收到这个广播消息，虽然这个广播是在广播接收者注册之前发送的，另外一点，对于静态注册的广播接收者来说，这个等同于普通广播。

27.广播使用的方式和场景

1）App全局监听：在AndroidManifest中静态注册的广播接收器，一般我们在收到该消息后，需要做一些相应的动作，而这些动作与当前App的组件，比如Activity或者Service的是否运行无关，比如我们在集成第三方Push SDK时，一般都会添加一个静态注册的BroadcastReceiver来监听Push消息，当有Push消息过来时，会在后台做一些网络请求或者发送通知等等。

2）组件局部监听：这种主要是在Activity或者Service中使用registerReceiver()动态注册的广播接收器，因为当我们收到一些特定的消息，比如网络连接发生变化时，我们可能需要在当前Activity页面给用户一些UI上的提示，或者将Service中的网络请求任务暂停。所以这种动态注册的广播接收器适合特定组件的特定消息处理。

28.在manifest 和代码中如何注册和使用BroadcastReceiver?

1）mainfest中注册:静态注册的广播接收者就是一个常驻在系统中的全局监听器，也就是说如果你应用中配置了一个静态的BroadcastReceiver，而且你安装了应用而无论应用是否处于运行状态，广播接收者都是已经常驻在系统中了。

```
<receiver android:name=".MyBroadcastReceiver">

    <intent-filter>

        <action
android:name="com.smilexie.test.intent.mybroadcastreceiver"/>

    </intent-filter>

</receiver>
```

2) 动态注册:动态注册的广播接收者只有执行了registerReceiver(receiver, filter)才会开始监听广播消息，并对广播消息作为相应的处理。

```
IntentFilter fiter = new
IntentFilter("com.smilexie.test.intent.mybroadcastreceiver");

MyBroadcastReceiver receiver = new MyBroadcastReceiver();

registerReceiver(receiver, filter);

//撤销广播接受者的动态注册

unregisterReceiver(receiver);

```

29.本地广播和全局广播有什么差别?

> 1）LocalBroadcastReceiver仅在自己的应用内发送接收广播，也就是只有自己的应用能收到，数据更加安全。广播只在这个程序里，而且效率更高。只能动态注册，在发送和注册的时候采用LocalBroadcastManager的sendBroadcast方法和registerReceiver方法。
>
> 2）全局广播：发送的广播事件可被其他应用程序获取，也能响应其他应用程序发送的广播事件（可以通过 exported–是否监听其他应用程序发送的广播 在清单文件中控制） 全局广播既可以动态注册，也可以静态注册。

30.AlertDialog,popupWindow,Activity区别

> - Popupwindow在显示之前一定要设置宽高，Dialog无此限制。
> - Popupwindow默认不会响应物理键盘的back，除非显示设置了popup.setFocusable(true);而在点击back的时候，Dialog会消失。
> - Popupwindow不会给页面其他的部分添加蒙层，而Dialog会。
> - Popupwindow没有标题，Dialog默认有标题，可以通过dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);取消标题
> - 二者显示的时候都要设置Gravity。如果不设置，Dialog默认是Gravity.CENTER。
> - 二者都有默认的背景，都可以通过setBackgroundDrawable(new ColorDrawable(android.R.color.transparent));去掉。
> - Popupwindow弹出后，取得了用户操作的响应处理权限，使得其他UI控件不被触发。而AlertDialog弹出后，点击背景，AlertDialog会消失。

31.Application 和 Activity 的 Context 对象的区别

> - Application Context是伴随应用生命周期；不可以showDialog, startActivity, LayoutInflation可以startService\BindService\sendBroadcast\registerBroadcast\loadResource values
> - Activity Context指生命周期只与当前Activity有关，而Activity Context这些操作都可以，即凡是跟UI相关的，都得用Activity做为Context来处理。
>
>   一个应用Context的数量=Activity数量+Service数量+1（Application数量）

32.LinearLayout、RelativeLayout、FrameLayout的特性及对比，并介绍使用场景。

> 1.RelativeLayout会让子View调用2次onMeasure，LinearLayout 在有weight时，也会调用子View2次onMeasure 2.RelativeLayout的子View如果高度和RelativeLayout不同，则会引发效率问题，当子View很复杂时，这个问题会更加严重。如果可以，尽量使用padding代替margin。 3.在不影响层级深度的情况下,使用LinearLayout和FrameLayout而不是RelativeLayout。 最后再思考一下文章开头那个矛盾的问题，为什么Google给开发者默认新建了个RelativeLayout，而自己却在DecorView自己是FrameLayout但是它只有一个子元素是属于LinearLayout。因为DecorView的层级深度是已知而且固定的，上面一个标题栏，下面一个内容栏。采用RelativeLayout并不会降低层级深度，所以此时在根节点上用LinearLayout是效率最高的。而之所以给开发者默认新建了个RelativeLayout是希望开发者能采用尽量少的View层级来表达布局以实现性能最优，因为复杂的View嵌套对性能的影响会更大一些。
>
> 4.能用两层LinearLayout，尽量用一个RelativeLayout，在时间上此时RelativeLayout耗时更小。另外LinearLayout慎用layout_weight,也将会增加一倍耗时操作。由于使用LinearLayout的layout_weight,大多数时间是不一样的，这会降低测量的速度。这只是一个如何合理使用Layout的案例，必要的时候，你要小心考虑是否用layout weight。总之减少层级结构，才是王道，让onMeasure做延迟加载，用viewStub，include等一些技巧。