**Android基础题**

**1、导致内存泄露的原因有哪些？**

内存泄露的根本原因：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。

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。

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

注册/反注册未成对使用引起的内存泄漏。

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。

 内存分析工具的使用

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

inSampleSize：缩放比例，在把图片载入内存之前，我们需要先计算出一个合适的缩放比例，避免不必要的大图载入。

decode format：解码格式，选择ARGB\_8888/RBG\_565/ARGB\_4444/ALPHA\_8，存在很大差异。

IV.减少资源图片的大小，过大的图片可以考虑分段加载

**2、理解Activity，View,Window三者关系**

这个问题真的很不好回答。所以这里先来个算是比较恰当的比喻来形容下它们的关系吧。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。

1：Activity构造的时候会初始化一个Window，准确的说是PhoneWindow。

2：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。

3：“ViewRoot”通过addView方法来一个个的添加View。比如TextView，Button等

4：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、Handler的原理**

所以就有了handler，它的作用就是实现线程之间的通信。

handler整个流程中，主要有四个对象，handler，Message,MessageQueue,Looper。当应用创建的时候，就会在主线程中创建handler对象，

我们通过要传送的消息保存到Message中，handler通过调用sendMessage方法将Message发送到MessageQueue中，Looper对象就会不断的调用loop()方法

不断的从MessageQueue中取出Message交给handler进行处理。从而实现线程之间的通信。

**4、View，ViewGroup事件分发**

1. 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。

2.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。

3.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。

4.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。

5.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。

6.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。

7.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。

**5、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。

**6、mvc 和 mvp mvvm**

·  1.mvc:数据、View、Activity，View将操作反馈给Activity，Activitiy去获取数据，数据通过观察者模式刷新给View。循环依赖

1.Activity重，很难单元测试

2.View和Model耦合严重

·  2.mvp:数据、View、Presenter，View将操作给Presenter，Presenter去获取数据，数据获取好了返回给Presenter，Presenter去刷新View。PV，PM双向依赖

1.接口爆炸

2.Presenter很重

·  3.mvvm:数据、View、ViewModel，View将操作给ViewModel，ViewModel去获取数据，数据和界面绑定了，数据更新界面更新。

1.viewModel的业务逻辑可以单独拿来测试

2.一个view 对应一个 viewModel 业务逻辑可以分离，不会出现全能类

3.数据和界面绑定了，不用写垃圾代码，但是复用起来不舒服

**7、自定义控件**

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

第一步：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。

第二步：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。

第三步：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。

**8、Serializable和Parcelable 的区别**

[**https://www.jianshu.com/p/a60b609ec7e7**](https://www.jianshu.com/p/a60b609ec7e7)

[**https://www.cnblogs.com/leipDao/p/8022063.html**](https://www.cnblogs.com/leipDao/p/8022063.html)

1.P 消耗内存小

2.网络传输用S 程序内使用P

3.S将数据持久化方便

4.S使用了反射 容易触发垃圾回收 比较慢