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)

https://blog.csdn.net/u013398960/article/details/74496242

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.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