安卓突破专题课程

Android 基础与底层机制

1. 数据库的操作类型有哪些，如何导入外部数据库？（Ricky）

   使用数据库的方式有哪些？

   （1） openOrCreateDatabase(String path);

   （2） 继承 SqliteOpenHelper 类对数据库及其版本进行管理(onCreate,onUpgrade)当在程序当中调用这个类的方法 getWritableDatabase() 或者 getReadableDatabase();的时候才会打开数据库。如果当时没有数据库文件的时候，系统就会自动生成一个数据库。

   1） 操作的类型：增删改查 CRUD

   直接操作 SQL 语句：SQliteDatabase.execSQL(sql);

   面向对象的操作方式 :SQLiteDatabase.insert(table, nullColumnHack, ContentValues);

如何导入外部数据库？

一般外部数据库文件可能放在 SD 卡或者 res/raw 或者 assets 目录下面。

写一个 DBManager 的类来管理，数据库文件搬家，先把数据库文件复制到” /data/data/包名/databases/” 目录下面，然后通过 db.openOrCreateDatabase(db 文件),打开数据库使用。我上一个项目就是这么做的，由于 app 上架之前就有一些初始数据需要内置，也会碰到数据的升级等问题，我是这么做的…… 同时我碰到最有意思的问题就是关于数据库并发操作的问题，比如：多线程操作数据库的时候，我采取的是封装使用互斥锁来解决……

2. 是否使用过本地广播，和全局广播有什么差别？（Ricky）

   引入本地广播的机制是为了解决安全性的问题：

   1） 正在发送的广播不会脱离应用程序，比用担心 app 的数据泄露；

   2） 其他的程序无法发送到我的应用程序内部，不担心安全漏洞。(比如：如何做一个杀不死的服务---监听火的 app 比如微信、友盟、极光的广播，来启动自己。)

   3） 发送本地广播比发送全局的广播高效。(全局广播要维护的广播集合表 效率更低。全局广播，意味着可以跨进程，就需要底层的支持。)

   本地广播不能用静态注册。----静态注册：可以做到程序停止后还能监听。

   使用：

   （1） 注册

   LocalBroadcastManager.getInstance(this).registerReceiver(new XXXBroadCastReceiver(), new IntentFilter(action));

   （2） 取消注册：

   LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver)

3. 是否使用过 IntentService，作用是什么， AIDL 解决了什么问题？ (小米) （Ricky）

   如果有一个任务，可以分成很多个子任务，需要按照顺序来完成，如果需要放到一个服务中完成，那么使用 IntentService 是最好的选择。

   一般我们所使用的 Service 是运行在主线程当中的，所以在 service 里面编写耗时的操作代码，则会卡主线程会 ANR。为了解决这样的问题，谷歌引入了 IntentService.

   IntentService 的优点：

   （1） 它创建一个独立的工作线程来处理所有一个一个 intent。

   （2） 创建了一个工作队列，来逐个发送 intent 给 onHandleIntent()

   （3） 不需要主动调用 stopSelf()来结束服务，因为源码里面自己实现了自动关闭。

（4） 默认实现了 onBind()返回的 null。

（5） 默认实现的 onStartCommand()的目的是将 intent 插入到工作队列。

总结：使用 IntentService 的好处有哪些。首先，省去了手动开线程的麻烦；第二，不用手动停止 service；第三，由于设计了工作队列，可以启动多次---startService(),但是只有一个 service 实例和一个工作线程。一个一个熟悉怒执行。

AIDL 解决了什么问题？

AIDL 的全称：Android Interface Definition Language，安卓接口定义语言。

由于 Android 系统中的进程之间不能共享内存，所以需要提供一些机制在不同的进程之间进行数据通信。

远程过程调用：RPC—Remote Procedure Call。 安卓就是提供了一种 IDL 的解决方案来公开自己的服务接口。AIDL:可以理解为双方的一个协议合同。双方都要持有这份协议---文本协议 xxx.aidl 文件（安卓内部编译的时候会将 aidl 协议翻译生成一个 xxx.java 文件---代理模式：Binder 驱动有关的，Linux 底层通讯有关的。）

在系统源码里面有大量用到 aidl，比如系统服务。

电视机顶盒系统开发。你的服务要暴露给别的开发者来使用。

讲解 Binder 机制。

4. Activity、 Window、 View 三者的差别， fragment 的特点？（360）（Ricky）

Activity、 Window、 View 三者如何协同显示界面的。---考点：显示的过程(view 绘制流程)源码的熟悉度。

Activity 剪窗花的人（控制的）；Window 窗户（承载的一个模型）；View 窗花（要显示的视图 View）；LayoutInflater 剪刀---将布局（图纸）剪成窗花。

（Alt+方向箭头）

fragment 的特点？(你用 fragment 有没有领略到一些乐趣,或者有没有踩过什么坑？)

fragment 的设计主要是把 Activity 界面包括其逻辑打碎成很多个独立的模块，这样便于模块的重用和更灵活地组装呈现多样的界面。

1） Fragment 可以作为 Activity 界面的一个部分组成；

2） 可以在一个 Activity 里面出现多个 Fragment, 并且一个 fragment 可以在多个 Activity 中使用；

3） 在 Activity 运行中，可以动态地添加、删除、替换 Fragment。

4） Fragment 有自己的生命周期的，可以响应输入事件。

踩过的坑：1.重叠；2. 注解 newAPI（兼容包解决）；3. Setarguement()初始化数据;4. 不能在 onsave...（）方法后，commit; 5. 入栈出栈问题；--事务。像 Activity 跳转一样的效果，同时返回的时候还能回到之前的页面(fragment)并且状态都还在。6. replace(f1, f2)严重影响生命周期:add()+show+hide

5. 描述一次网络请求的流程（新浪）（Jason）

6. Handler、 Thread 和 HandlerThread 的差别（小米）（Jason）

7. 低版本 SDK 实现高版本 api（小米）（Ricky）

两种情况：

1） 一般很多高版本的新的 API 都会在兼容包里面找到替代的实现。比如 fragment。Notification, 在 v4 兼容包里面有 NotificationCompat 类。5.0+出现的 backgroundTint,

minSdk 小于 5.0 的话会包检测错误，v4 兼容包 DrawableCompat 类。

2）没有替代实现就自己手动实现。比如：控件的水波纹效果—第三方实现。
或者直接在低版本去除这个效果。
3）补充:如果设置了 minSDK 但是代码里面使**用了高**版本的 API，会出现检测错误。需要在代码里面使用声明编译检测策略，比如：@SuppressLint 和@TargetApi 注解提示编译器编译的规则。@SuppressLint 是忽略检测；@TargetApi=23，会根据你函数里面使用的 API，严格地匹配 SDK 版本，给出相应的编译错误提示。
4）为了避免位置的错误，最好不要使用废弃 api。(一般情况下不会有兼容性问题，后面可能会随时删除这个 API 方法；性能方面的问题。)
5）http://chinagdg.org/2016/01/picking-your-compilesdkversion-minsdkversion-targetsdkversion/

8. launch mode 应用场景（百度、小米、乐视）（Ricky）
栈：先进后出
标准模式
SingleTop：使用场景：浏览器的书签；通讯消息聊天界面。
SingleTask：使用场景：某个 Activity 当做主界面的时候。
SingleInstance：使用场景：比如浏览器 BrowserActivity 很耗内存，很多 app 都会要调用它，这样就可以把该 Activity 设置成单例模式。比如：闹钟闹铃。
9. touch 事件传递流程（小米）（Ricky）

10. view 绘制流程（百度）（Ricky）
Measure：测量，测量自己。如果是 ViewGroup 就需要测量里面的所有 childview.
测量的结果怎么办？setMeasuredDimension(resolveSizeAndState(maxWidth, widthMeasureSpec, childState), heightSizeAndState);设置自己的大小。
Layout: 摆放，把自己摆放在哪个位置。如果是 ViewGroup 就需要发放里面的所有 childview.
怎么去具体摆放呢？
Draw:绘制

```
        /*
         * Draw traversal performs several drawing steps which must be executed
         * in the appropriate order:
         *
         *      1. Draw the background
         *      2. If necessary, save the canvas' layers to prepare for fading
         *      3. Draw view's content
         *      4. Draw children
         *      5. If necessary, draw the fading edges and restore layers
         *      6. Draw decorations (scrollbars for instance)
         */
```

11. 什么情况导致内存泄漏（美团）（Ricky）
1）什么是内存泄漏：最好解释清楚 GC 垃圾回收机制以及概念 GC Root。
2）为什么会有内存泄漏：因为内存泄漏是属于人为的失误造成的。而且面向对象开发

关系复杂、多线程的关系，很容易出现引用层级关系很深以及很混乱。

3）什么情况容易导致内存泄漏：

4）如何解决内存泄漏：

12. ANR 定位和修正（Ricky）

可以通过查看/data/anr/traces.txt 查看 ANR 信息。

根本原因是：主线程被卡了，导致应用在 5 秒时间未响应用户的输入事件。

很多种 ANR 错误出现的场景：

1） 主线程当中执行 IO/网络操作，容易阻塞。

2） 主线程当中执行了耗时的计算。----自定义控件的时候 onDraw 方法里面经常这么做。

（同时聊一聊自定义控件的性能优化：在 onDraw 里面创建对象容易导致内存抖动---绘制动作会大量不断调用，产生大量垃圾对象导致 GC 很频繁就造成了内存抖动。）内存抖动就容易造成 UI 出现掉帧卡顿的问题

3） BroadCastReceiver 没有在 10 秒内完成处理。

4） BroadCastReceiver 的 onReceived 代码中也要尽量减少耗时的操作，建议使用 IntentService 处理。

5） Service 执行了耗时的操作，因为 service 也是在主线程当中执行的，所以耗时操作应该在 service 里面开启子线程来做。

6） 使用 AsyncTask 处理耗时的 IO 等操作。

7） 使用 Thread 或者 HandlerThread 时，使用 Process.setThreadPriority(`Process.THREAD_PRIORITY_BACKGROUND`) 或者 java.lang.Thread.setPriority （int priority）设置优先级为后台优先级，这样可以让其他的多线程并发消耗 CPU 的时间会减少，有利于主线程的处理。

8） Activity 的 onCreate 和 onResume 回调中尽量耗时的操作。

13. 什么情况导致 oom（乐视、美团）（Ricky）

OOM 产生的原因：内存不足，android 系统为每一个应用程序都设置了一个硬性的条件：DalvikHeapSize 最大阀值 64M/48M/24M.如果你的应用程序内存占用接近这个阀值，此时如果再尝试内存分配的时候就会造成 OOM。

1)内存泄露多了就容易导致 OOM

2)大图的处理。压缩图片。平时开发就要注意对象的频繁创建和回收。

3）可以适当的检测：ActivityManager.getMemoryClass()可以用来查询当前应用的 HeapSize 阀值。可以通过命名 adb shellgetProp | grep dalvik.vm.heapxxxlimit 查看。

如何避免内存泄露：

1） 减小对象的内存占用：

a) 使用更加轻量级的数据结构：
考虑适当的情况下替代 HashMap 等传统数据结构而使用安卓专门为手机研发的数据结构类 ArrayMap/SparseArray。SparseLongMap/SparseIntMap/SparseBoolMap 更加高效。
HashMap.put(string,Object);Object o = map.get(string);会导致一些没必要的自动装箱和拆箱。

b) 适当的避免在 android 中使用 Enum 枚举，替代使用普通的 static 常量。（一般还是提倡多用枚举---软件的架构设计方面；如果碰到这个枚举需要大量使用的时候就应该更加倾向于解决性能问题。）。

c) 较少 Bitmap 对象的内存占用。
使用 inSampleSize:计算图片压缩比例进行图片压缩，可以避免大图加载造成 OOM; decodeformat : 图片的解码格式选择，ARGB_8888/RGB_565/ARGB_4444/ALPHA_8,还可以使用 WebP。
d) 使用更小的图片
资源图片里面，是否存在还可以继续压缩的空间。

2) 内存对象的重复利用：
使用对象池技术，两种：1.自己写；2.利用系统既有的对象池机制。比如 LRU(Last Recently Use)算法。
a) ListView/GridView 源码可以看到重用的情况 ConvertView 的复用。RecyclerView 中 Recycler 源码。
b) Bitmap 的复用
Listview 等要显示大量图片。需要使用 LRU 缓存机制来复用图片。
C) 避免在 onDraw 方法里面执行对象的创建，要复用。避免内存抖动。
D) 常见的 java 基础问题---StringBuilder 等

3) 避免对象的内存泄露：

4) 使用一些内存的优化策略：
看文档

14. Android Service 与 Activity 之间通信的几种方式（Ricky）
https://blog.csdn.net/xiaanming/article/details/9750689
1) 通过 Binder
2) 通过广播

15. Android 各个版本 API 的区别（Ricky）
https://blog.csdn.net/qq_21399461/article/details/80277472
把几个关键版本的特性记住：3.0/4.0、4.4、5.0、6.0/7.0

16. 如何保证一个后台服务不被杀死,比较省电的方式是什么？（百度）（Ricky）
看文档
https://www.cnblogs.com/dixonyy/p/5163880.html

17. Requestlayout, onlayout, onDraw, DrawChild 区别与联系（猎豹）（Ricky）
RequestLayout()方法：会导致调用 Measure()方法和 layout()。将会根据标志位判断是否需要 onDraw();
onLayout()：摆放 viewGroup 里面的子控件
onDraw()：绘制视图本身；(ViewGroup 还需要绘制里面的所有子控件)
drawChild(): 重新回调每一个子视图的 draw 方法。`child.draw(canvas, this, drawingTime);`

18. invalidate()和 postInvalidate() 的区别及使用（百度）（Ricky）
https://blog.csdn.net/weixin_41101173/article/details/79727304?utm_source=blogxgwz4

invalidate()：在主线程当中刷新；
postInvalidate()：在子线程当中刷新；其实最终调用的就是 invalidate，原理依然是通过工作线程向主线程发送消息这一机制。

```
    public void postInvalidate() {
        postInvalidateDelayed(0);
    }
    public void postInvalidateDelayed(long delayMilliseconds) {
        // We try only with the AttachInfo because there's no point in
invalidating
        // if we are not attached to our window
        final AttachInfo attachInfo = mAttachInfo;
        if (attachInfo != null) {
            attachInfo.mViewRootImpl.dispatchInvalidateDelayed(this,
delayMilliseconds);
        }
    }

    public void dispatchInvalidateDelayed(View view, long delayMilliseconds)
{
        Message msg = mHandler.obtainMessage(MSG_INVALIDATE, view);
        mHandler.sendMessageDelayed(msg, delayMilliseconds);
        }
public void handleMessage(Message msg) {
            switch (msg.what) {
            case MSG_INVALIDATE:
                ((View) msg.obj).invalidate();
                    break;
```

19. Android 动画框架实现原理（Ricky）

https://blog.csdn.net/harrain/article/details/53726960

传统的动画框架：View.startAnimation();

弊端：移动后不能点击。原因？跟实现机制有关系。

所有的透明度、旋转、平移、缩放动画，都是在 view 不断刷新调用 draw 的情况下实现的。调用的 canvas.translate(xxx),canvas.scaleX(xxx)…. Xxx:matrix 像素矩阵来控制动画的数据。记得看源码，结合多只缩放的 demo 看源码。

20. Android 为每个应用程序分配的内存大小是多少？（美团）（Ricky）

https://www.cnblogs.com/yaowen/p/6347682.html

看具体的手机平台，常见的有：64M/32M 等

22. LinearLayout 对比 RelativeLayout（百度）（Ricky）

https://blog.csdn.net/weixin_41101173/article/details/79730745

https://www.jianshu.com/p/b9bd08ffe921

https://www.jianshu.com/p/8a7d059da746

性能对比：LinearLayout 的性能要比 RelativeLayout 好。

因为 RelativeLayout 会测量两次。而默认情况下（没有设置 weight）LinearLayout 只会测量一次。

为什么 RelativeLayout 会测量两次？首先 RelativeLayout 中的子 view 排列方式是基于彼此依赖的关系，而这个依赖可能和布局中 view 的顺序无关，在确定每一个子 view 的位置的时候，就需要先给每一个子 view 排一下序。又因为 RelativeLayout 允许横向和纵向相互依赖，所

以需要横向纵向分别进行一次排序测量。

23. 优化自定义 view（百度、乐视、小米）（Ricky）

https://blog.csdn.net/whb20081815/article/details/74474736

1）减少在 onDraw 里面大量计算和对象创建和大量内存分配。

2）应该尽量少用 invalidate()次数。

3）view 里面耗时的操作 layout。减少 requestLayout（）避免让 UI 系统重新遍历整棵树。Mearsure。

4）如果你有一个很复杂的布局，不如将这个复杂的布局直接使用你自己的写的 ViewGroup 来实现。减少了一个树的层次关系 全部都是自己测量和 layout，达到优化的目的。（Facebook 就经常这么干）

24. ContentProvider（乐视）（Ricky）

提示：跨进程通信。进程之间进行数据交互共享。；源码来一剁。

25. fragment 生命周期（Ricky）

https://blog.csdn.net/jokeeeeee/article/details/46004931

26. volley 解析（美团、乐视）（Ricky）

https://blog.csdn.net/u012602304/article/details/79170137

https://www.cnblogs.com/caobotao/p/5071658.html

27. Android Glide 源码解析（Ricky）

https://www.cnblogs.com/guanmanman/p/7008259.html

https://www.cnblogs.com/guanmanman/p/7040942.html

28. Android 属性动画特性（乐视、小米）（Ricky）

https://www.jianshu.com/p/2412d00a0ce4

29.Activity 状态恢复

https://www.jianshu.com/p/715333d87738

https://blog.csdn.net/LovelyProgrammer/article/details/79812761