8-26日 魅族面试-技术 Actions

2014年8月26日

23:34

1.ANR问题的分析流程，ANR产生原因；

导致ANR的原因很复杂

* 我们最常遇到的应用长时间没有处理InputDispatcher下放的输入事件  
  导致这种情况的原因为在主线程上做了耗时动作，或当前线程/进程没有足够的CPU资源，导致回调一直没有返回  
  耗时动作一般包括 IO操作和数据库操作，当然也包括同步锁使用不当和过于发杂的（跨进程）调用
* 其次还有一种ANR我们可能也会碰到：焦点切换时间过长导致的ANR  
  此ANR一般发生于大型应用或大内存应用进行切换时的问题，例如，Contacts退到后台，Launcher启动，这期间如果Launcher启动时间过长，那么Contacts是有可能发生ANR的，反之，上一个应用pause超时也会对当前应用造成影响
* 还有一些误报问题，Kevin.Yang之前有分析过此类问题

如何分析ANR

* 首先是看anr时的trace，碰碰运气，如果正好发现当前进程的主线程正在做耗时操作，或者等一把锁，那么就要考虑重构了
* 看看主线程输出的log，在system log发生anr的点向前看，搜索main log中的not respond，如果主线程的log在这附近过之前断掉了，那么要根据最后一句主线程log分析问题
* 如果还是看不到异常，那么还有可能是低内存引起的，看一看在之前是否有很多进城died，然后看一下buddy info，如果此时8K和16K内存个数都很少，那有可能是低内存
* 其他进程优先级非常高

我记得anr的trace中有几种状态，native suspend 科普一下。

发生ANR时，AMS会从appNotResponding函数开始掉下去，其中一个函数为dumpStackTraces，实际上就是对某些进程使用了 andriod.os.Process.sendSignal，signal为3，main log的关键字是 Sending signal. PID: <pid> SIG: <sig>。这些进程的SignalCatcher在收到signal时，会将当前进程的所有线程挂起，打trace 。当线程正好调用到native方法，例如jni调用，挂起的线程的状态为NATIVE，当应用在java层调用时，线程会在执行到某个checkpoint时挂起，状态显示为SUSPEND状态。如果线程被同步锁阻塞，一般会显示成WAIT状态。 WAIT状态要看架构是否合理；SUSPEND的参考意义不大，不过要看看这周围是否有耗时操作；NATIVE状态要看被调用者，如果是binder的IPC调用，那么binder另一端的操作是否阻塞住，剩下的问题要具体问题具体分析。

2.Contacts涉及的批量操作，批量操作相比单个操作快的关键点，跨进程I/O传输；对BInder的了解，两种操作的Binder的counts差别。

* 这就好比 IO操作 有20KB 数据，如果每次只写 1KB，那么IO会执行20次，如果每次写4KB数据，那么只需要执行5次IO就可以完成。 5次比20次强的地方在于可以减少IO负担，每次的IO流程比较复杂，依赖于iowait的也比较多，所以要尽可能的缩短IO次数。每次IO写入一定量的数据，这种东西叫做buffer。buffer太小，IO的次数会增多，IO负担大；buffer太大，内存的占用量会增大，数据也不安全。

* IO这部分说完了，跨进程调用的原因和IO是一样的，数据库一般是 IO + 跨进程操作，跨进程操作需要进程间调度和进程同步，会占用很多CPU的资源，浪费很多时间，所以如果跨进程的次数越多，效率就越低。Contacts使用的是Batch，就和Buffer类似，Batch的内容越多，跨进程的次数越少，但如果数据太多有可能会导致Binder大小超出限制

3.Telephony SIM 这块大致都会涉及哪些业务，好像问题，SIM卡联系人加载流程问题，讲的还OK；

这个就帮不了你了，可以问问张旭，给他再讲一遍

4.回归到基本的问题，Activity有几种模式，Service几种启动方式，Start 与bind ，各自的使用场景，我是结合联系人的BatchOperationService的实现讲的，讲的还好；

Activity有几种模式我就不确认了，我也没系统看过

* Start service，一般不关注Service的返回结果，适合应用做繁重工作，在service的onStartCommand内开始分配任务
* Bind service一般属于功能的共享，比如某个Service有一定的权限，只有这个service或所在的用户组/进程才能做，一般应用做不了。那么这些应用就要通过bind service的方式来做

5.真实的场景，数据库升级的问题，兼容性。

onUpgrade方法，具体怎样操作还不清楚

6.ContactsProvider的表结构；

这个问伟哥更合适

7.视频壁纸的WallpaperService什么时候启动，由谁来触发。

如果一个Service被调用起来，那么肯定是有一些方式，一般情况有两种，一种是注册Meta-data，还有一种是注册intent-filter，注册intent-filter后，通过PMS queryIntentXXX的方式就可以拿到所有注册 相应Action的 component（service、activity等等），然后通过对这些查出来的component做判断，进行后续操作，WallpaperManagerService肯定是这样找到你的service，但具体怎样启动明天看过代码再论

另外提一句，Engine其实一般是解析RenderScript的在，这个东西很犀利，有点类似OpenGL编程，然后通过写RenderScript可以写出很炫的效果。

说到Render，给你大致梳理一下 绘制一个图片资源的流程：

图片资源在编译时，aapt 制作ResourceTable数据结构，将图片id索引和configuration配置保存到apk的resources.arsc 数据结构中。 上层应用读取资源时需要通过AssetManager去解析resources.arsc，解出ResourceTable，拿到最佳图片。拿出图片后将图片数据传送给 BitmapFactory，BitmapFactory通过调用skia将文件解析成bitmap数据（C++层），将C++的bitmap做处理后，返回到Java层。Java在绘制图片时会调用到 Canvas的draw方法，这个draw会调用到hwui的HardwareRenderer中，它会将C++层的bitmap数据通过hwui的内置的EGL接口（EGL为OpenGL的本地化接口）将bitmap用显卡转换为Texture（Texture为显卡画图的基本单位）。render完成之后，这些texture会放到一个layer上，传送到surfaceflinger中去做compose变化，包括叠加和透明化等等，最终生成framebuffer （fb），最终将framebuffer放到屏幕的设备节点上，这个画好的图片才能显示到屏幕上。

这一块是不是还有layout加载流程？这块研究过么？

上面说的是从一个bitmap从被decode出来到被画到屏幕上经历的过程，和layout加载流程关系不大，或者说不是一个体系。layout加载流程，一组layout实际上是需要从xml转换成java实例的，程序先从AssetManager中找到对应的layout文件，然后将文件(流)传入LayoutInflater，LayoutInflater中几乎都是xml的parser，它会将layout解析，通过反射实例化layout对象，然后按照layout描述的结构放置child和parent关系。完成后，ViewRootImpl的doTraversal里会分别调用解析好的layout实例的onMeasure、onLayout和onDraw实现layout的绘制。

视频壁纸配置文件，

AndroidManifext.xml:

<uses-permission android:name="android.permission.SET\_WALLPAPER\_HINTS" />

<uses-permission android:name="android.permission.SET\_WALLPAPER\_COMPONENT" />

service-> android:permission="android.permission.BIND\_WALLPAPER"

WallpaperManager wpm = WallpaperManager.getInstance(this);

wpm.getIWallpaperManager().setWallpaperComponent(

mWallpaperIntent.getComponent());

//用sharePreferences存放视频路径:

SharedPreferences sp = getApplicationContext().getSharedPreferences(

SHAREDPREFERENCE\_FILE,

MODE\_PRIVATE);

sp.edit().putString(SHAREDPREFERENCE\_PATH\_KEY, mVideoInfo.path).apply();

setResult(RESULT\_OK);

视频壁纸WallpaperService加载流程:

1.user手动设置壁纸:

使用 WallpaperManager 通过aidl方式跨进程调用 WallpaperManagerService.setWallpaperComponent()里面执行bindWallpaperComponentLocked()，先是做一些简单的componentName的适配，再通过PMS.getServiceInfo获取ServiceInfo判断permission权限，调用PMS.queryIntentService，取出ServiceInfo扔进list返回，然后创建wallpaperConnection(wi,wallpaper),执行bindServiceAsUser()完成service的绑定;

？2.开机视频壁纸子启动流程还没研究清楚（WallpaperObserver这个类继承自FileObserver，用法介绍：观察壁纸更改并通知所有IWallpaperServiceCallbacks该墙纸已经改变。当没有壁纸被设置且是第一次被创建时，CREATE会被触发。而每次墙纸改变时，CLOSE\_WRITE触发。）​

8. ANR的时候分散出来的，耗时的操作在Service里面做的话，需要什么特殊的处理么？还有BroadCast里面能做耗时操作么？最好不能，Why？

* Broadcast 的 onReceive在各个进程间为串行操作，所谓串行就是要等上一个ActivityThread处理完onReceive之后，才能分发给下一个进程。如果有20个应用，每个应用都处理10s，那么广播将花费200s，这样就失去了广播的实时性