IPC aidl

进程a和进程B通讯

当两个进程都运行在内存中时 可以通过aidl 然后会产生一个ibinder 类

stub存根 proxy

渲染机制和优化

app会卡主要是造成了ui卡顿，（内存和渲染机制分析）不是线程卡顿而是ui卡顿。

卡主线程（卡UI线程）

1 外部引起

比如：Activity 里面进行网络访问／大文件的io操作

主要是内存这块需要注意什么

内存抖动的问题：

主要是在短时间内 大量产生了对象，并gc了对象就会照成内存抖动

（当这个gc所用的时间超过一般指，或者一大堆一直执行会消耗庞大的帧象时间，gc（一般运行在主线程中）长时间会把主线程卡住）

google：如果用户感到画面流畅，一般gc优化时间为16ms（用户体验最好），每隔16ms（间隔）刷新一次ui

2 刷新频率和帧频率不一致

1. 了解渲染刷新机制

VSYNS：有两个概念

1. Refresh Rate：屏幕在一面时间内刷行屏幕的次数—由硬件的参数决定，比如60hz，
2. Frame Rate：GPU在一秒内绘制操作的帧数，比如60fps

GPU：进行数据渲染，这时候就涉及到两个概念，GPU渲染的机制还有硬件刷新屏幕 刷行频率和帧频率 如果刷新频率和帧频率不一致会出现掉帧

GPU刷新：GPU帮助我们将UI组件计算成纹理Texture（衣服）和三维图形PolyGons ，（形状）。同时会使用openGL（api）来绘制图形，openGL会将要渲染的纹理存储在GPU memory ，会将纹理和polygons换存在GPU内存里 ，如果想要高效率使用opneGL 我们有些可以省去一些操作，比如viewGroup 背景 颜色，如果你没有设置任何背景viewgroup就不会调用onDraw方法，一旦设置背景，就会调用ondraw 。这时候会产生一个View Tree，如果view tree 一个节点 发生改变整个view tree 会刷新。

View Tree树

60hz是屏幕刷行理想的屏幕

24帧／秒 电源胶卷（比较贵，节省成本）时代。这个在我们app肯定不行，因为我们操作非常快，必须频繁操作程序，用户体验要好肯定不是24帧每秒 更高

60fps：一秒内绘制的帧数

在60fps内，每隔一段时间系统会得到发送的vsync信号（垂直刷行），去进行渲染，就会正常绘制。如果低于卡顿比较明显，60fps 要求：每一帧要求16ms刷新。例如：第一帧很正常16ms绘制ondraw，当下一帧花费时间大于16ms，系统会按照16ms一帧进行读取，这时候发现，并没有完全绘制成功，这时候会造成卡顿，还是上一帧画面，下一帧再来可能会从第一镇跳到第三帧（掉帧）

2view本身卡顿

自定义view要注意，内否优化

优化：

1 渲染性能的优化

2防止过度绘制

在我们的绘制渲染机制里面比较耗时的

1cpu计算时间：

CPU的优化，从减轻加工View对象的polygons 和texture来下手

view hierarchy包含了大多数没用的view 这些view更本就不会显示在屏幕上面，一旦触发测量和布局操作，就会拖累应用的性能表现

工具：Hierarchy Viewer工具

优化：

1. 当我们的布局使用的FrameLayou的时候，我们可以把它改成merge，可以避免自己的帧布局和系统的ContentFrameLayou帧布局重叠造成重复计算measure和layout ）

viewstub：当加载的时候才会占用，不加载的时候就是隐藏，仅仅只是占用位置。

如和找出里面没用的view呢》或者减少不必要的view嵌套

三个圆点分别代表：测量、布局、绘制三个阶段的性能表现

1. 绿色：渲染的管道阶段，这歌诗图的渲染速度快于至少一半的其他视图
2. 黄色：渲染速度比较慢50%
3. 红色：渲染速度非常慢

优化思想：查看自己的布局，有没有层次是否很深以及渲染是否耗时，然后想办法能否减少层次以及优化每个view的渲染时间

2GPU进行删格化（比如你放大某个图形，他就是一格一格的像素）

优化：尽量不要过度绘制（指的是屏幕上的某一点像素在同一帧的时间的内被绘制了很多次。重叠像素）GPU如何优化：

1背景进场容易造成过度绘制。

手机开发者选项里找到工具：Debug GPU override

由于我们布局设置了背景，同时用到了MaterialDesign的主题会默认给一个背景。（解决办法，将主题添加到北京去掉 getWindow().setBackgroundDrawable(null)）。

2 可以通过裁减来处理

比如通过canvas .clips进行裁剪

3 cpu将计算好的polygons和texture传递到GPU的时候也需要时间

path和svg

我们知道一个UI对象转化为一系列多边形和纹理的过程肯定是相当耗时，从CPU上传处理数据到GPU同样也很耗时，所以很明显，我们需要尽量减少对象转化的次数，以及CPU上传的数据次数，。 OpenGl允许数据上传到GPU可以对数据进行保存（ displaylist）经常改变UI 缓存就没有意义了·（因为它会重新执行所有步骤，然后再把新的数据缓存到displaylist）

渲染性能优化就是尽可能地上传数据到GPU，然后尽可能长时间不修改情况下保存数据，应为每次上传资源到GPU时，我们都会浪费宝贵的时间。

电量优化：   
重要参数：WIFI、wake\_lock（CPU/屏幕）、conn、mobile\_ratio（蜂窝信号比wifi耗电）

1. 为了省电有些工作会频繁地请求唤醒cpu，为了省电，有些工作可以放当手机插上电源的时候去做，往往这样的情况非常多，想这些不需要及时和用户交互的操作可以放到后台处理

比如 ：360手机助手，当充上电的时候，才会自动清理手机垃圾，自动备份上传图片、联系人等到云端

提问：拍照（立马执行）和图片的处理（不一定立马执行），他们可以做一些电量的优化吗 ？  
假如现在没有充电，电量比较低，拍照动作是需要立马执行的，但是图片处理（需要消耗·大量的计算—电量的大量消耗）是否可以放在用户手机插上电源之后后来处理？

如何礼记获取手机当前充电状态，我们可以有针对性对一些代码做优化。

2 wake\_lock 唤醒锁

系统为了节省电量，CPU在没有任务忙的时候就会自动进入到休眠

有任务需要唤醒CPU高效执行的时候，就会给CPU加wake\_lock。

很容易唤醒CPU，但是很容易忘记释放wake\_lock 如果不是不释放锁 cpu不会进入休眠

wake\_lock锁主要是相对系统的休眠而言的，意思句是我的程序给CPU加了这个锁那个系统就不会休眠了，这样做的目的就为了权利配合我们的程序的运行。有的情况如果不这么做，就会出现一些问题，比如微信等即时通信心跳包会熄屏不久后停止网络访问等问题，所以微信里面是由大量使用到wake\_lock锁。

Wake lock因该是大家开发时用的手段把。所以没有特别特别仔细讲解

有一些意外情况，比如小米手机是做了同步心跳包（如果超过了这个同步的频率就会被屏蔽掉或者将降频），所有的app后台唤醒频率不能太高，比如每隔两秒钟去请求

AlarmManager.setRepeating();(模糊时间不是准确时间)

如果去改进，对任务调度的优化

网络优化—也解决电量问题