简述

- Android Telephone
  - 第二章
    - adb的使用
  - 第三章
    - handle
    - service的AIDL
    - Broadcast
  - 第四章
    - 手机拨号流程分析

Android Telephone第二章

Android Debug Bridge (adb)

- Android 调试桥 (adb) 是一种功能多样的命令行工具，可让您与设备进行通信。adb 命令可用于执行各种设备操作（例如安装和调试应用），并提供对 Unix shell（可用来在设备上运行各种命令）的访问权限。它是一种客户端-服务器程序，组件：
  - **客户端**：用于发送命令。客户端在开发机器上运行。您可以通过发出 adb 命令从命令行终端调用客户端。
  - **守护程序 (adbd)**：用于在设备上运行命令。守护程序在每个设备上作为后台进程运行。
  - **服务器**：用于管理客户端与守护程序之间的通信。服务器在开发机器上作为后台进程运行。
-

命令	说明
adb logcat -vthreadtime	查看main日志的命令
adb logcat -vtime -b radio	查看radio日志的命令
adb logcat -vtime -b events	查看event日志的命令

通话流程

前置

同步和异步

- Synchronous (同步)和Asynchronous (异步)的概念最早来自通信领域。
  - 通信的同步指客户端在发送请求后,必须要在服务端有回应后客户端才继续发送其他请求,所以这时所有请求将会在服务端得到同步，直到服务端返回请求。
  - 通信的异步:指客户端在发送请求后,不必等待服务端的回应就可以发送下一个请求,对所有的请求动作来说将会在服务端得到异步,这条请求的链路就像是一个请求队列,所有的请求动作在这里不会得到同步。

Handler消息处理机制

- 每一个消息发送到主线程的消息队列中，消息队列遵循先进先出原则，发送消息并不会阻塞线程，而接收线程会阻塞线程。
- Handler允许发送并处理Message消息，Message对象通过主线程的MessageQueue消息队列相关联的Message和Runnable对象进行存取。每个Handler实例对Message消息发送和接收与对应主线程和主线程的消息队列有关。当创建一个新Handler时，Handler就属于当前主线程，主线程的MessageQueue消息队列也同步创建，即Handler会绑定到创建该Handler主线程/消息队列中。然后，Handler就可以通过主线程的消息队列发送和接收Message消息对象了。
- 特性
    - Android里没有全局Message Queue消息队列，每个Activity主线程都有一个独立的Message Queue消息队列，消息队列采用先进先出原 则。不同APK应用不能通过Handler进行Message通信，同一个APK应用中可通过Handler对象传递而进行Message通信。
    - 个Handler实例都会绑定到创建它的线程中（一般位于主线程，即Activity线程），但Handler实例均可在主线程或子线程中创建。
    - Handler发送消息使用Message Queue消息队列，每个Message发送到消息队列里面，遵循先进先出原则；发送消息采用异步方式不
      会阻塞线程，而接收线程采用同步方式会阻塞线程，所以当Handler处理完一个Message对象后才会接着去取下一个消息进行处理。
- 作用
    - Handler主要作用是异步处理较费时的逻辑，优先将界面返回给用户，异步处理完成后再去更新用户界面。

AIDL跨应用服务

- AIDL（Android Interface Definition Language, Android接口定义语 言）Android系统平台的接口定义语言与您可能已经使用过的其他IDLs接口定义语言相似。程序员可以利用AIDL自定义编程接口，在客户端和服务端之间实现进程间通信（IPC）。在Android平台上，一个进程通常不能访问另外一个进程的内存空间，因此，Android平台将这些跨进程访问的对象分解成操作系统能够识别的简单对象，并且为跨应用访问而特殊编排和整理这些对象。用于编排和整理这些对象的代码编写起来非常冗长，所以Android的AIDL提供了相关工具来自动生成这些代码供程序员使用。

广播

- 发送方仅需要将广播内容完成发送，而接收方过滤自己需要的广播信息和内容，然后进行处理，接收方信息是发送方不知道的。

流程分析

- Android Telephony主要的手机通信能力
    - Call（通话）
    - ServiceState（服务状态）
    - DataConnection（数据连接）
    - SMS（**S**hort **M**essage **S**ervice,短信）

拨号流程分析

1. 打开Nexus 6P手机的拨号盘
    1. 首先使用adb devices 命令查看和确认手机与计算机连接是否成功，然后使用adb logcat命令查看Nexus 6P手机的运行日志，最后操作手机，点击Home界面最下面一排最左边带有电话图标的应用按钮，打开拨号界面。

    2. ActivityManagerService将启动com.android.dialer包下的DialtactsActivity。system_server, 即ActivityManagerService所在的系统进程; 通过`adb shell ps -ef`查看进程信息命令可以确认相关的进程信息
2. 可在Android Studio连续两次按下右边的Shift 按键, 打开Search Everywhere对话框，输入DialtactsActivity,在输入过程中有逐个英文字母匹配的过程,
   输入完成后便可完整匹配DialtactsActivity.java文件。
    1. 快捷键Ctrl+F12快速打开当前类属性和方法列表浮动菜单
    2. 在Android Studio快速匹配过程中，可使用* (匹配多个字符)进行模糊匹配，并且输入的字符不区分大小写。
    3. DialpadFragment提供用户拨号的交互界面
    4. CalllntentBuilder创建拨号请求的intent对象
    5. TelecomManager继续传递拨号请求intent对象
3. lTelecomService接收拨号请求服务
    - lTelecomService的实现类TelecomServiceImpl的placeCall 方法，响应Dialer 应用发起的跨进程服务接口调用。将发出一个定向广播，由Telecom应用中的PrimaryCallReceiver 对象接收。
    - PrimaryCallReceiver的sendNewOutgoingCallIntent 方法，其调用过程是:sendNewOutgoingCallIntent→NewOutgoingCallntentBroadcaster.processIntent-→mCallsManager.placeOutgoingCall.这两个关键的处理逻辑最终是调用了CallsManager 对象的两个不同方法。
        - startOutgoingCall()：startOutgoingCall()将开始拨号前的准备工作
        - placeOutgoingCall():placeOutgoingCall将继续传递拨号请求，实现将拨号请求发送给BP Modem处理。
4. CallsManager的startOutgoingCall()
    - CallsManager.startOutgoingCall:主要逻辑是创建、更新和保存Call 对象
        - 如果不是以前保存在mCalls列表的Call对象，调用addCall (call) 方法保存并触发增加Call对象的通知，CallsManager对象将保存多个Call 对象到mCalls 集合中，Call对象则设置Listener 对象为CallsManager,对象之间相互引用。而CallsManager对象通过mListeners发出onCallAdded消息
          回调。
            - 在onCallAdded()方法中实现绑定服务。首先，创建InCallServiceBindingConnection对象，创建该对象的同时将同步创建一个mServiceConnection对象，此对象为匿名的ServiceConnection类型，重写了onServiceConnected和onServiceDisconnected方法;接着，创建action为InCallService.SERVICE_ INTERFACE的intent对象，并更新了PhoneAccount和Call的一些关键信息;然后，调用Android 系统的bindServiceAsUser 方法绑定服务;最后是绑定服务成功以后的onConnected系统回调，将发起对InCallController.this.onConnected的调用
                - InCallController.this.onConnected()将之前保存的Call对象通过inCallService发送出去
            - Telecom应用中完成了第一次绑定服务和对应服务的接口调用。绑定的SERVICE INTERFACE定义为“android.telecom.InCallService"，InCallController 通过绑定服务的方式，开启拨号流程中的第二次跨进程访问，从Telecom应用的system_server 进程再次回到Dialer应用的com.android.dialer进程。
5. IInCallService服务的响应过程

- InCallServiceImpl类继承于InCallService类，类代码文件在packages/apps/Dialer工程下，而InCallService类对应的代码文件则在framework 下，其服务接口的定义文件为:frameworks/base/telecomm/java/com/android/intermal/telecom/InCallService.aidl,主要定义了addCall setInCallAdapter、updateCall等接口方法。
- InCallController在拨号流程中，首先绑定服务，接着调用服务的setInCallAdapter、 addCall和onCallXXChanged接口。
  - onBind服务被绑定的响应方法
    - onBind()的返回Intent是InCallServiceBinder，InCallServiceBinder实现了IInCallService.aidl的接口，这些接口通过发送Handler消息，将服务接收到的服务请求转化为异步处理方式
  - setInCallAdapter设置Adapter
    - setInCallAdapter接口的响应逻辑，主要是创建Phone对象和设置Phone对象的Listener属性。
  - addCall增加主动拨号Outgoing Call
    - 在Telecom应用中，首先会创建Call对象，Dialer应用中也会创建Call对象，但这两个Call对象的定义是不同的。
    - Call对象的创建与转换。从Telecom应用中创建com.android.server.telecom.Call,并通过此对象创建跨进程传递android.telecom.parcelableCall对象(支持序列化和反序列化，因此可以跨进程传递此对象)，而Dialer应用中是接收到parcelableCall对象后，通过此对象相关信息创建android.telecom.Call对象。
    - 调用fireCallAdded(call)方法，使用多个监听器完成通话界面的展示和更新

6. CallsManager的placeOutgoingCall()
   - CallsManager 分别调用startOutgoingCall 和placeOutgoingCall。startOutgoingCall 方法将通过绑定服务和调用其服务接口，启动和更新Dialer应用中的InCallActivity, 展示出通话界面;但拨号请求并未发送到BP Modem处理。
   - placeOutgoingCall关键调用过程： sendNewOutgoingCallIntent->NewOutgoing->CallIntentBroadcaster. processIntent→mCallsManager.placeOutgoingCall->call.startCreateConnection→CreateConnectionProcessor.process→attemptNextPhoneAccount
     - 在CreateConnectionProcessor类中的定义是private ConnectionServiceWrapper mService服务,ConnectionServiceWrapper的createConnection 方法是拨号流程在Telecom应用中将发起第二次绑定服务的跨进程服务访问，绑定服务的服务对象为:SERVICE_INTERFACE，即“android.telecom.ConnectionService"
       - 流程
         - bind Service:创建intent的Action有一个比较隐含的设置， 在ConnectionServiceWrapper类的构造方法中调用了super构造方法，从而设置了绑定服务的对象为ConnectionService.SERVICE _INTERFACE。
         - addConnectionServiceAdapter:将传递实现IConnectionServiceAdapter.aidl接口Stub的跨进程访问binder对象。
         - createConnection:通过Call对象拨号请求相关信息创建ConnectionRequest对象，传递给packages/services/Telephony中对应的服务。
       - 拨号流程中，Telecom应用第一次跨进程服务调用，将与Call对象相关的拨号请求信息传递给了Dialer应用，去加载和展现通话界面;那么第二次跨进程服务调用，Call 拨号请求相关信息转换成了ConnectionRequest对象并传递给了TeleService应用。TeleService 将接收到的ConnectionRequest请求相关信息传递给BP Modem来发起电话拨号请求。