源码学习记录：一、在自定义view创建前传递进数据的方法除了静态变量调入之外，还可以有预设静态函数提前创建一个新对象，并对数据进行序列化，通过setArgument的方式导入，再在view被引用而建立时通过getArgument取出该序列化值的方式实现。（其实还是静态变量传递的方式，这两个函数在Fragment中存在，其他view还待考证）

二、其扫描图片的方式为直接寻找MediaStore数据库中的数据，搜索项包括

private final static String[] *IMAGE\_PROJECTION* = new String[]{  
 MediaStore.Images.Media.*DATA*,//图片路径  
 MediaStore.Images.Media.*DISPLAY\_NAME*,//图片文件名，包括后缀名  
 MediaStore.Images.Media.*TITLE*//图片文件名，不包含后缀

之后将图片文件夹单独归为一个数组集合，再以图片文件夹名为key，图片数组为value归为一个Map集合，其中保存方式都为File。（之后还有几个诸如图片信息之类的集合，感觉太为复杂）

三、在数据寻找成功后，直接在当前页面通过函数将当前的空白页换成了自定义的FragmentView，该View中的数据即刚才取出的数据。

四、具体的图片解析过程用了universal-image-loader开源库，其过程简略说明即为设置图片加载任务参数，尝试在内存寻找是否存在缓存，尝试建立线程池的任务，采用多线程方式来进行图片操作。（多线程基础还得恶补，这里直接搬运开源库过来用了）

四、可以通过getWindow().getDecorView().setOnSystemUiVisibilityChangeListener(new View.OnSystemUiVisibilityChangeListener() {}监听界面变化。

沉浸式界面private void toggleImmersiveMode() {

if (Build.VERSION.SDK\_INT >= 11) {

int uiOptions = getWindow().getDecorView().getSystemUiVisibility();

// Navigation bar hiding: Backwards compatible to ICS.

if (Build.VERSION.SDK\_INT >= 14) {

uiOptions ^= View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION;

}

// Status bar hiding: Backwards compatible to Jellybean

if (Build.VERSION.SDK\_INT >= 16) {

uiOptions ^= View.SYSTEM\_UI\_FLAG\_FULLSCREEN;

}

// Immersive mode: Backward compatible to KitKat.

if (Build.VERSION.SDK\_INT >= 18) {

uiOptions ^= View.SYSTEM\_UI\_FLAG\_IMMERSIVE\_STICKY;

}

getWindow().getDecorView().setSystemUiVisibility(uiOptions);

}

}

可以自定义一个style，然后在manifest里将该style作为现style的父类，从而取消标题：

<style name="AppTheme.Base" parent="Theme.AppCompat">

 <item name="windowActionBar">false</item>

 <item name="android:windowNoTitle">true</item>

<item name="colorPrimary">@color/mColorPrimary</item>//主色  
<item name="colorPrimaryDark">@color/mColorPrimaryDark</item>//深主色，状态栏颜色

<item name=”navigationBarColor”>@color/mColorNavigationBar</item>//底部导航栏颜色，5.0以后的属性

</style>

控制图片缩放效果的开源库 uk.co.senab.photoview.PhotoView，可以直接当ImageView用。

运行时权限的授权方法：

ActivityCompat.*checkSelfPermission*(this,Manifest.permission.*NFC*)!=

PackageManager.*PERMISSION\_GRANTED*);//检测目标权限是否处于授权状态

ActivityCompat.*requestPermissions*(this, new String[]{Manifest.permission.*NFC*},1);//如果未授权,则尝试请求权限，此时1为该次请求的Id

public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults){super.(requestCode,permissions,grantResults)/\*判断过程\*/}

//重写请求权限的回调函数，根据grantResults的值确认是否申请成功

Service复习：在startService()方法被调用时，如果service未创建，则会同时调用onCreate()和onStartCommad(),但如果调用的是bindServie(),则被调用的只有onCreate()，且service的生命周期取决于其何时被stopService()或者stopSelf(),如果被绑定的话，则需要unbindService()也一起被调用。当service被绑定时，先调用onCreate()（如果Service未建立）,然后会调用onBind()函数返回一个IBinder对象（通过传参到ServiceConnection类的onServiceConnected()函数中，该函数处于另外一个线程，不是即时调用的方法），该对象可以用来取得service的对象引用（如自定义继承IBinder的类添加取得对象的方法），以及完成一些服务与调用方的信息交互。一种简单的用法为在service中创建一个Messenger对象并定义自己的Handler方法，通过IBinder开放给绑定类，而绑定类可以取得该Messenger对象（通过官方定义的Messenger(IBinder)方法），通过该对象将自己的Messengr发给service的Handler（通过messahe机制）,这样便完成了两者的信息交互。

AIDL学习