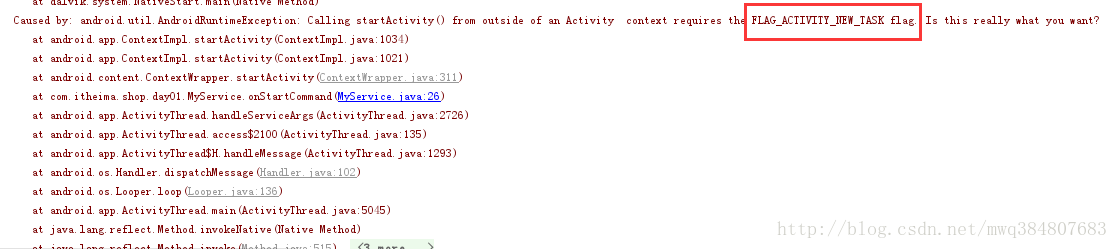
**Android面试题-源码分析为什么service里面startActivity抛异常？activity不会**

我们有时候需要在service里面启动activity，但是会发现报如下异常：



img

必须添加FLAG\_ACTIVITY\_NEW\_TASK这个标记就可以了，那么为什么在activity里面不需要呢？接下来通过从源码角度带大家分析。

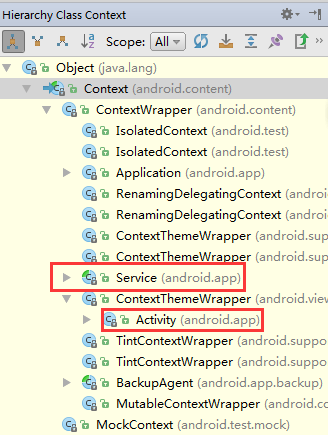
### **启动activity有两种形式**

1）直接调用Context类的startActivity方法；这种方式启动的Activity没有Activity栈，因此不能以standard方式启动，必须加上FLAG\_ACTIVITY\_NEW\_TASK这个Flag,服务就是通过Context调用。

2）调用被Activity类重载过的startActivity方法，通常在我们的Activity中直接调用这个方法就是这种形式；

### **Context.startActivity源码分析**

我们查看Context类的startActivity方法，发现这竟然是一个抽象类；查看Context的类继承关系图如下：



img

我们看到诸如Activity，Service等并没有直接继承Context，Activity继承了ContextThemeWrapper，Service而是继承了ContextWrapper；

#### **现在从源码分析ContextWrapper的实现：**

@Override  
public void startActivity(Intent intent) {  
 mBase.startActivity(intent);  
}

这个mBase是什么呢？这里我先直接告诉你，它的真正实现是ContextImpl类；至于为什么，有一条思路：在任意mBase打一个断点就能看到实现。

Context.startActivity最终使用了ContextImpl里面的方法，代码如下：

@Override  
 public void startActivity(Intent intent, Bundle options) {  
 warnIfCallingFromSystemProcess();  
 if ((intent.getFlags()&Intent.FLAG\_ACTIVITY\_NEW\_TASK) == 0) {  
 throw new AndroidRuntimeException(  
 "Calling startActivity() from outside of an Activity "  
 + " context requires the FLAG\_ACTIVITY\_NEW\_TASK flag."  
 + " Is this really what you want?");  
 }  
 mMainThread.getInstrumentation().execStartActivity(  
 getOuterContext(), mMainThread.getApplicationThread(), null,  
 (Activity) null, intent, -1, options);  
 }

#### **源码分析：**

1）大家看看抛出来的异常是不是还是熟悉的味道。

2）通过判断可知当前的intent.getFlags是否带有FLAG\_ACTIVITY\_NEW\_TASK这个标记，如果没有抛出异常，因为源码使用了&运算符,只有两个位都是1，结果才是1，所以可知service没有带FLAG\_ACTIVITY\_NEW\_TASK标记，才抛出异常。

3）真正的startActivity使用了Instrumentation类的execStartActivity方法；继续跟踪：

public ActivityResult execStartActivity(  
 Context who, IBinder contextThread, IBinder token, Activity target,Intent intent, int requestCode, Bundle options) {  
 ......  
try {  
 ......  
 int result = ActivityManagerNative.getDefault()  
 .startActivity(whoThread, who.getBasePackageName(), intent,  
 intent.resolveTypeIfNeeded(who.getContentResolver()),  
 token, target != null ? target.mEmbeddedID : null,  
 requestCode, 0, null, options);  
 checkStartActivityResult(result, intent);  
 } catch (RemoteException e) {  
 throw new RuntimeException("Failure from system", e);  
 }  
 return null;

#### **源码分析：**

1）到这里我们发现真正调用的是ActivityManagerNative的startActivity方法；

### **Activity.startActivity源码分析**

@Override  
public void startActivity(Intent intent) {  
 this.startActivity(intent, null);  
}

#### **源码可知：**

1）调用当前类的startActivity方法，代码如下：

@Override  
public void startActivity(Intent intent, @Nullable Bundle options) {  
 if (options != null) {  
 startActivityForResult(intent, -1, options);  
 } else {  
 // Note we want to go through this call for compatibility with  
 // applications that may have overridden the method.  
 startActivityForResult(intent, -1);  
 }  
 }

#### **源码可知**

1）调用了startActivityForResult

public void startActivityForResult(Intent intent, int requestCode, @Nullable Bundle options) {  
 ......  
 Instrumentation.ActivityResult ar =  
 mInstrumentation.execStartActivity(  
 this, mMainThread.getApplicationThread(), mToken, this,  
 intent, requestCode, options);  
 ......

#### **源码可知**

1）可以看到，其实通过Activity和ContextImpl类启动Activity并无本质不同，他们都通过Instrumentation这个辅助类调用到了ActivityManagerNative的方法。

2）只是Activity不会去检查标记，所以并不会抛出异常。

至此源码分析完毕。