深入理解ActivityRecord和Task---Activity管理系列

原创 牛晓伟 牛晓伟 2024年12月05日 11:46 山西

戳蓝字“**牛晓伟**”关注我哦！

用心坚持输出**易读、有趣、有深度、高质量、体系化**的技术文章，技术文章也可以有温度。

牛晓伟

专注Android framework、app开发多年，用心持续输出易读、有趣、高质量、体系...

55篇原创内容

公众号

前言

从本篇开始介绍**Activity管理**系列的文章，Activity的管理主要是**ActivityTaskManagerService** (以下简称ATMS)负责的，Activity的管理是一个复杂的模块，它的复杂度体现在代码量大、并且还与**WindowManagerService**有很大的交集。我在想如何才能让大家更好的理解**Activity的管理**模块呢，经过深思熟虑我决定先从最基础的**ActivityRecord和Task**作为开篇，只有先有了基础知识后 (比如ActivityRecord作用是啥，Task作用是啥) ，再去理解后面的知识时才会易如反掌。Activity管理系列文章的基调是层层递进，前面文章是后面文章的铺垫。

四大组件管理系统系列文章：

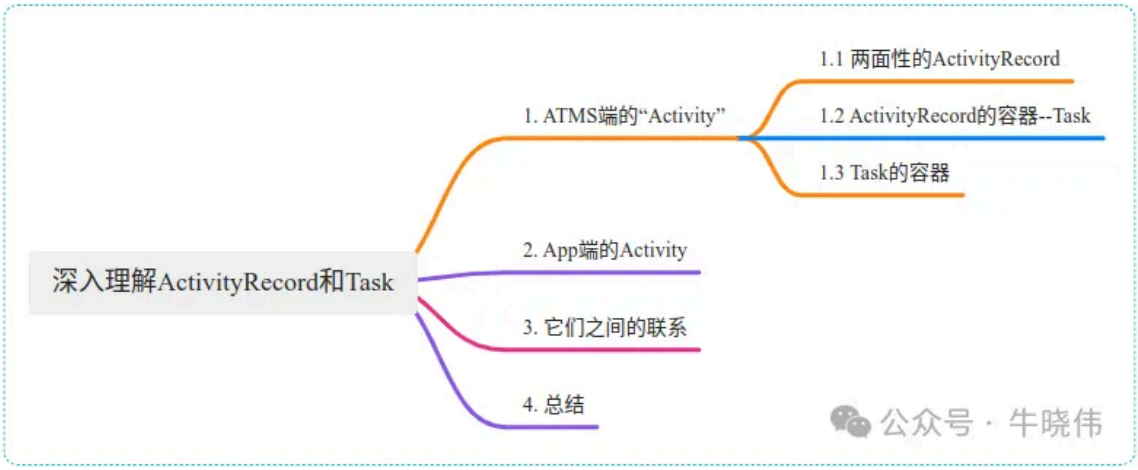
深度解读ActivityManagerService----Android四大组件系统系列

本文摘要

本文采用自述的方式分别从ActivityTaskManagerService端的Activity和App端的Activity，来介绍**ActivityRecord**和**Task**。(文中代码基于Android13)

注：文中提到的ATMS是ActivityTaskManagerService的简称。

本文大纲



image

1. ATMS端的“Activity”

大家好，我是**ActivityTaskManagerService**，大家可以称呼我的小名**ATMS**，我像**ActivityManagerService**一样也是一个binder服务，**ActivityManagerService**为了“专注于管理工作”特将**Activity管理**相关的事情全权交给了我，为了能让大家对我所做的工作有一个更深入的理解，因此我会先把最基础的内容介绍给大家，而后慢慢深入。

ATMS端的“Activity”其中Activity加引号的原因是，这里的Activity可并不是真正的Activity，它只是把启动的Activity记录下来了。把启动Activity信息记录下来的主要原因是为了更好的管理Activity，而这个记录者是*ActivityRecord*，那我就把它介绍给大家认识。

1.1 两面性的ActivityRecord

大家是不是疑惑ActivityRecord为啥是两面性的？那我就给大家讲讲，它的两面性表现在ActivityRecord既可以划分为**ActivityManagerService**的一部分，也可以划分为**WindowManagerService**的一部分。用更直白的话说就是ActivityRecord既有记录Activity的功能，又具有窗口功能。不管是从它的名字还是从它具有的功能都应该更和**ActivityManagerService**亲近些，但是它这个“叛徒”却投靠了**WindowManagerService**，可以从它所在的包名*com.android.server.wm*得出结论。那就从两面性来介绍下ActivityRecord。

1.1.1 记录功能

ActivityRecord的记录功能主要是**把启动的Activity记录下来**，一个启动Activity会在ATMS端对应一个ActivityRecord对象，ActivityRecord主要记录了**基础信息**、**Activity信息**、**Activity状态信息**、**启动者信息**等信息，那就逐一来介绍下这些信息吧。

基础信息

如下表格列出了一些主要的基础信息

属性	说明
mUserId:int	Activity是运行在哪个用户下的
packageName:String	Activity所属的包名
processName：String	配置Activity运行在哪个进程下
mActivityType：int	Activity类型，类型有standard、home、recents等，其中launcher的Activity是home类型；普通App的Activity是standard类型

上面只是列出了主要的基础信息，还有一些没有列出来。

Activity信息

Activity信息存放在类型为*ActivityInfo*的*info*属性中，而ActivityInfo则存储了在*AndroidManifest.xml*文件中配置的Activity相关的信息，如下表格是ActivityInfo的主要属性：

属性	说明
launchMode:int	Activity的启动模式
name:String	在AndroidManifest中使用android:name配置的Activity的类名信息
exported：String	该Activity是否可以让别的App使用
enabled：boolean	该Activity是否可以使用
applicationInfo：ApplicationInfo	指向在AndroidManifest中配置的Application信息

上面只是列了一些Activity信息，还有很多的Activity信息没有列出。

Activity状态信息

因为Activity是有各种生命周期方法的，比如onResume、onPause、onStop等，而Activity状态信息就是代表记录的Activity处于哪种状态，如下表格是Activity状态信息：

属性	说明
mState:State	State是一个枚举类型，它的值有INITIALIZING、STARTED、RESUMED、PAUSING、PAUSED、STOPPING、STOPPED、FINISHING、DESTROYING、DESTROYED、RESTARTING_PROCESS
finishing：boolean	代表记录的Activity是否finish

启动者信息

启动者指的是谁启动了Activity，如下表格是启动者信息：

属性	说明
launchedFromPid:int	启动者的进程id
launchedFromUid:int	启动者的uid
launchedFromPackage:String	启动者的包名

ActivityRecord包含的信息非常多，上面只是列出了一些常用的信息，ActivityRecord的记录功能就是**把启动Activity的信息记录**下来，比如是谁启动了Activity、启动时间是啥、启动Activity的详细信息等等都记录下来。启动Activity的状态发生任何变化也会把这些状态值告知ActivityRecord，比如启动Activity执行了onResume方法，则ActivityRecord会把*mState*值置为*RESUMED*。

那再从窗口功能的角度来看ActivityRecord吧。

1.1.2 窗口功能

在介绍窗口功能之前，先来简单介绍下**WindowManagerService**，WindowManagerService也是一个binder服务，正如其名它的作用是用来管理所有的窗口的(在后面会有专题来介绍它)，而ActivityRecord又可以被看成WindowManagerService中的一个容器窗口，啥叫容器窗口呢？就是说它不会包含真正的内容，它的作用就是容纳别的窗口。如果你不相信ActivityRecord是一个容器窗口的话，请看下面铁证：

image

上图展示了ActivityRecord的类图结构，它继承了WindowToken，而WindowToken又继承了WindowContainer，WindowContainer可是WindowManagerService管理的所有窗口的父类，而泛型就是说ActivityRecord是可以容纳多个WindowState窗口的。而WindowState可以理解为与Activity中承载的根View是一一对应关系，因此WindowState可以理解为是一个非容器类型的窗口，因为它包含了真正的绘制内容。

不管是ActivityRecord还是WindowState在**surfaceflinger进程**中都会对应各自的**layer**，而layer也是分为容器和非容器类型。当然在后面Surfaceflinger专题会介绍这些知识。

ActivityRecord是一个容器类型的窗口，它可以容纳多个WindowState。开发者经常会给Activity配置一些比如从左到右的进入动画，而这个动画的实现原理，就是不断的改变ActivityRecord这个容器窗口的位置信息。

1.1.3 小结

我从**记录功能**和**窗口功能**两个方面介绍了ActivityRecord，而ActivityRecord也是有容器容纳它的，那就来介绍下它的容器吧。

1.2 ActivityRecord的容器--Task

我ATMS会把所有启动的Activity都用ActivityRecord记录下来，进而会有一个问题就是ActivityRecord会越来越多，那该如何管理这些ActivityRecord呢？答案就是**Task**，这个Task有一个特性**先进后出**，这个特性是不是与*栈*这个数据结构很相似啊，凡是先进入Task的ActivityRecord会最后从Task中出来。**Task也是一个容器类型的窗口，它在surfaceflinger进程中也有对应的layer**。

下面是Task的类图结构：

image

如上图，Task继承了TaskFragment，而TaskFragment又继承了WindowContainer，关于Task的类图结构就简单介绍到此。

为了让大家对Task、ActivityRecord有一个直观的认识，我特意绘制了一幅图：

image

如上图App进程中的Activity处于显示状态，而systemserver进程中的某Task的*栈*底存放了一个ActivityRecord，而它记录了App进程中处于显示状态的Activity，该ActivityRecord现在的状态也是*resume*状态。

当App进程中启动另外一个*Activity1*后，Task、Activity、ActivityRecord之间的关系变为如下图所示：

image

systemserver进程中的某Task的*栈*底是先前的ActivityRecord，它的状态是*stop*，它依然记录了App进程中的Activity，只不过这时的Activity处于*stop*状态；Task的*栈*顶是*ActivityRecord1*，它的状态是*resume*状态，它记录了App进程中的*Activity1*。

当App进程中的*Activity1*被用户按*back*键返回后，它的记录者*ActivityRecord1*也会从Task的*栈*中消失，当App进程中的最后一个Activity被用户按back键返回后，它的记录者ActivityRecord同样从Task中消失，因为这时候Task已经为空了，因此它也会被销毁。

一般一个App进程中的Activity对应的ActivityRecord是放在同一Task中，当然也有例外比如该App进程中的Activity配置了*singleTask*或者*singleInstance*或者在启动该Activity的时候增加了*FLAG_ACTIVITY_NEW_TASK*这样的flag，该Activity对应的ActivityRecord就会被放在一个新的Task中。

不同App进程中的Activity对应的ActivityRecord是可以放在同一个Task中的，如下图：

image

如上图，*App1*进程和App进程中的Activity对应的ActivityRecord放在了同一个Task中，**凡是处于Task栈底的ActivityRecord是该Task的根Activity**。以上就是关于Task的介绍，接下来再简单介绍下**Task的容器**。

1.3 Task的容器

从launcher第一次启动App的某个主Activity时，都会增加一个增加*FLAG_ACTIVITY_NEW_TASK*这样的flag参数，这样该App的主Activity对应的ActivityRecord就会被放入一个新的Task。因此在打开很多App后，就会存在非常多的Task，而我ATMS如何存放这些Task呢？或者说Task的容器是啥呢？

TaskDisplayArea就是Task的容器，当然TaskDisplayArea也是一个容器类型的窗口，下面是它的类图结果：

image

关于TaskDisplayArea的类图就不做过多的解释了，因为它不是咱们的重点，在WindowManagerService专题的时候会介绍它。

TaskDisplayArea在存放Task的时候，并没有采用*栈*的数据结构，而是采用了*数组*，栈结构在此并不合适，因为会涉及到Task所处位置的变动，因此使用数组是最简单的。同样为了让大家更容易理解，特意绘制了一幅图：

image

如上图，TaskDisplayArea采用数组结构来存放Task，数组中尾部的Task是属于被显示的Task，而Task中*栈*顶的ActivityRecord对应的Activity是处于显示状态，它是可以与用户进行交互的；而数组尾部之前的Task一般是不被显示的，当然这些Task中的ActivityRecord对应的Activity也是处于stop状态。如上图中，App进程中的Activity1处于显示状态，而launcher和

App1进程中的Activity处于stop状态，因此它们对应的Task位于TaskDisplayArea的中部和首部位置。

1.4 小结

ATMS端的“Activity”指的是ActivityRecord，ActivityRecord有两个作用**记录启动的Activity**和作为一个**容器类型的窗口**被WindowManagerService管理。而Task是ActivityRecord的容器，它也是一个**容器类型的窗口**，它是以*栈*这个数据结构来存储ActivityRecord的。而TaskDisplayArea是Task的容器，它也是一个**容器类型的窗口**，它是以数组来存储Task的，位于数组尾部的Task中栈顶的ActivityRecord对应的Activity是出于显示状态，它是可以与用户进行交互的。

2. App端的Activity

介绍完ATMS端的“Activity”，那我再来介绍下App端的Activity，App端的Activity也是指已经启动的Activity，而这些启动的Activity是如何存储的呢？

要想知道答案，请看下面代码：

```
//ActivityThread类

final ArrayMap<IBinder, ActivityClientRecord> mActivities = new ArrayMap<>();
```

如上图所有启动的Activity对象是保存在ActivityThread的*mActivities*属性中的，该属性是ArrayMap<IBinder, ActivityClientRecord>类型的，它的key是一个IBinder对象，这个IBinder对象会在下面介绍到，而它的value是ActivityClientRecord对象，ActivityClientRecord对象除了包含启动Activity的对象外，还包含了一些其他信息，比如Activity的状态等。App端的Activity就介绍到此。

3. 它们之间的联系

它们之间的联系指的是ATMS端的ActivityRecord与App端的Activity是如何保证它们能找到对方。比如App端的Activity“到达”了ATMS端，ATMS是根据啥来找到它对应的ActivityRecord；再比如ATMS端的ActivityRecord“到达”了App端，App端又是根据啥来找到它对应的Activity的。这里的到达加了引号，主要想表达到达并不是它们的对象真正到达了对方，它们的对象实例即使到达了对方，对于对方来说也是没有的。

而答案就是**IBinder对象**，在**App端的Activity**时介绍过存储启动Activity用的是一个ArrayMap<IBinder, ActivityClientRecord>对象，而它的key值是IBInder对象，那这个IBinder对象是啥呢？请看下面代码：

```
```

```
//ActivityRecord类

//Token就是保证它们连接的类
private static class Token extends Binder {
        @NonNull WeakReference<ActivityRecord> mActivityRef;

        @Override
        public String toString() {
            return "Token{" + Integer.toHexString(System.identityHashCode(this)) + " "
                    + mActivityRef.get() + "}";
        }
    }

 //该方法从IBinder中找到ActivityRecord
 static @Nullable ActivityRecord forTokenLocked(IBinder token) {
        if (token == null) return null;
        final Token activityToken;
        try {
            activityToken = (Token) token;
        } catch (ClassCastException e) {
            Slog.w(TAG, "Bad activity token: " + token, e);
            return null;
        }
        final ActivityRecord r = activityToken.mActivityRef.get();
        return r == null || r.getRootTask() == null ? null : r;
    }
```

如上代码，其中的**Token**类它继承了Binder类，看到没它Binder的子类，而它的对象就是上面提到的**IBinder对象**，而Binder对象有这样的特性：**Binder对象被传递到别的进程时，在binder驱动层会对该Binder对象进行转换，当到达别的进程时，这时候已经是BinderProxy对象了；而该进程中的BinderProxy对象再次被传递到原先进程时，在binder驱动层会对该BinderProxy对象进行转换，当到达原先进程时就转换为原先的Binder对象了**。Binder对象还有一个特性：**同一Binder对象在被多次传递给同一进程时，该Binder对象对应的BinderProxy对象是唯一的一个**。

基于以上Binder对象的特性，Token对象被传递到App端后，它作为ArrayMap<IBinder, ActivityClientRecord>对象的key值，而传递过来的Activity信息则保存在ActivityClientRecord对象中，这时候就就可以把启动的Activity保存起来了。而后每次交互的时候，ATMS都会把Token对象传递到App端，而App端根据Token对象对应的BinderProxy对象从ArrayMap<IBinder, ActivityClientRecord>对象那就到对应的ActivityClientRecord对象，进而拿到Activity。

而App端在传递信息给ATMS时候，也会带上Token对象对应的BinderProxy对象，它被传递到ATMS后，会使用上面的*forTokenLocked*方法，该方法根据IBinder找到对应的ActivityRecord。

4. 总结

到此关于ActivityRecord和Task的介绍就结束了，我带大家认识了ActivityRecord有两个作用**记录启动的Activity**和作为一个**容器类型的窗口**被WindowManagerService管理。而Task是ActivityRecord的容器，它也是一个**容器类型的窗口**，它是以*栈*这个数据结构来存储ActivityRecord的。而TaskDisplayArea是Task的容器，它也是一个**容器类型的窗口**，它是以数组来存储Task的，位于数组尾部的Task中栈顶的ActivityRecord对应的Activity是出于显示状态，它是可以与用户进行交互的。

而App端的Activity与ActivityRecord建立联系是依据Token类，而该类是一个Binder类。这只是Activity管理系列的第一篇文章，敬请期待后面的系列文章。

#Android framework系列文章 25　#android 50

Android framework系列文章 · 目录 ≡

‹上一篇
深度解读ActivityManagerService----Android四大组件系统系列

下一篇 ›
App端框架之谜---Android四大组件系统系列

个人观点，仅供参考