# 深入底层：当你安装apk时，到底发生了什么？

Pingred  [Carson带你学Android](javascript:void(0);)  2023-02-27 08:05 发表于广东

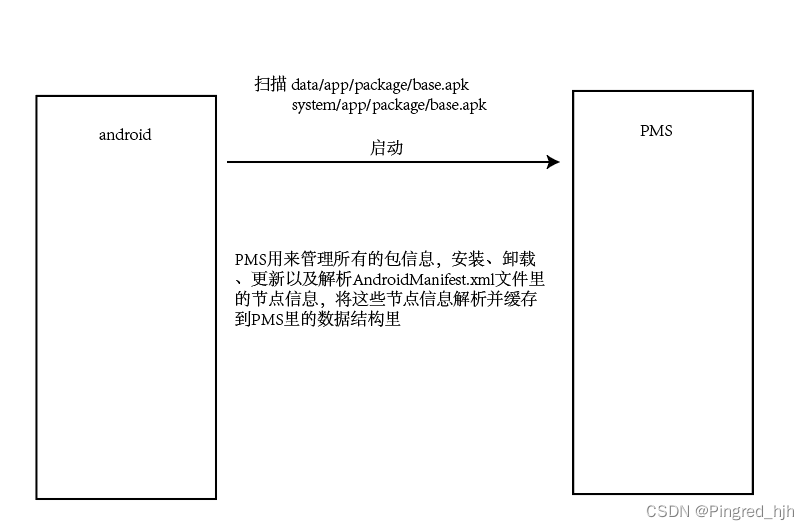
# **前言**

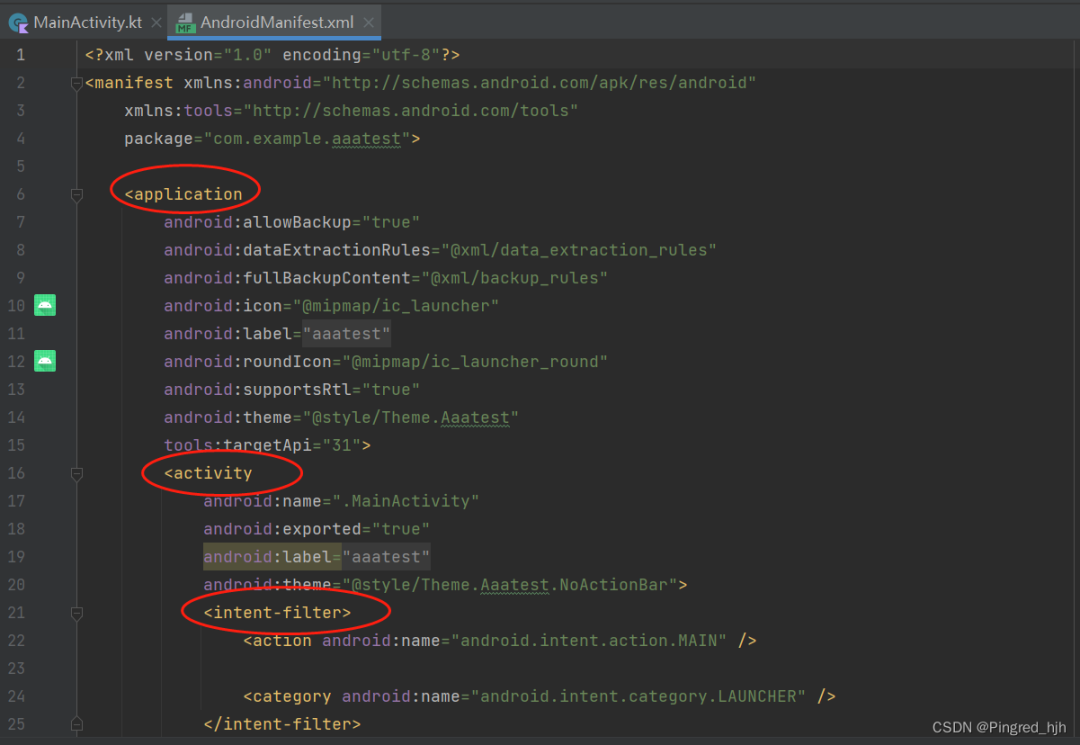
下载安卓apk ->点击安装 -> 点击图标进入app，这个过程相信大家都很熟悉了。

当点击apk进行安装时，**「到底发生了什么呢？」**

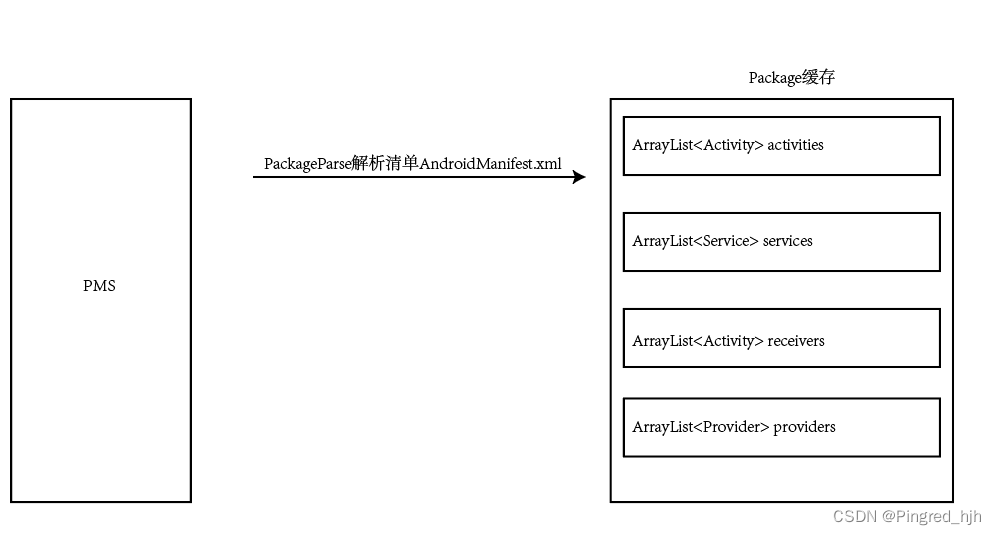
# **PackageManagerService**

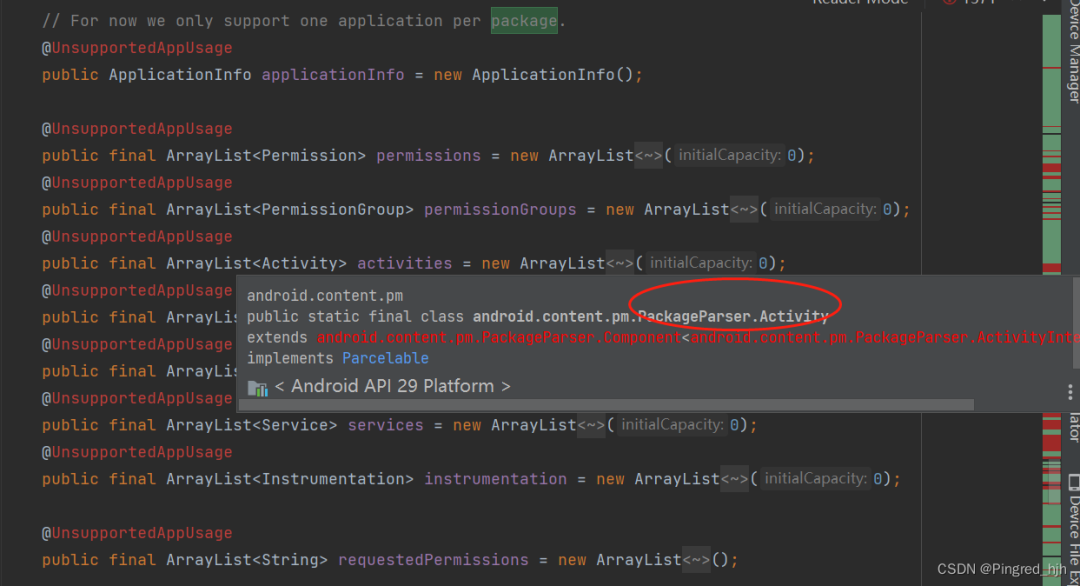
# 简称PMS，当安卓系统要安装某个应用时，apk文件会被加载到/data/app目录下，然后PMS就对该目录下的apk缓存文件进行扫描：IMG_256

也就是说PMS就会在/data/app目录里对这个apk文件进行解析，apk文件里是有很多文件的，因此要让系统迅速定位到这么多类（四大组件等），就应该有个服务事先把它们的包名路径等信息加载好，这样就可以快速地定位到这些类(比如入口Activity等)，而这个服务就是PMS，然后AMS即ActivityManagerService就能根据PMS里面的这些类信息来进行创建入口Activity等四大组件类并启动。另外，在安卓手机开机时，PMS也会去扫描加载/data/app目录和system/app里每个apk里的所有类信息，然后缓存到PMS它自己里面定义好的集合里，供AMS使用。

PMS加载每个apk（包括系统apk）里的清单文件，然后把里面相应的类（节点）信息进行分类存储。

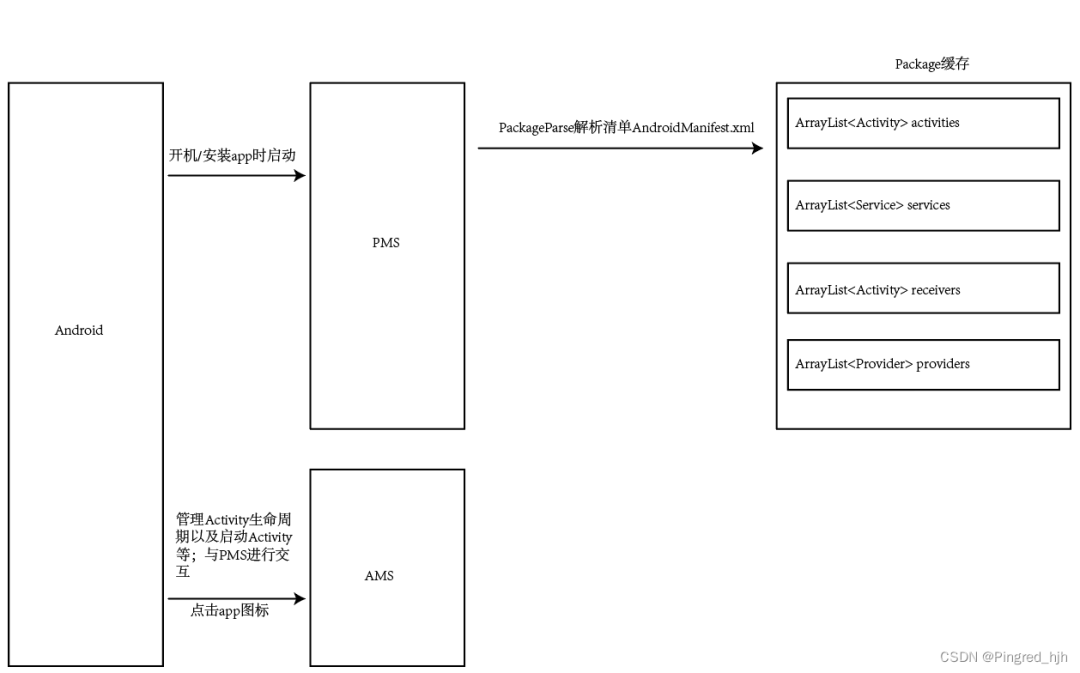
像上图AndroidManifest.xml清单文件里的节点信息，它是一个xml文件，PMS将这些节点信息缓存起来不用占太多容量，而且节点信息又清晰明了，有了这些信息足以让AMS快速去反射创建类对象，然后进行启动。

PMS使用的是一个工具类PackageParse去解析清单文件的，将里面的节点信息（Activity、Service、provider等）解析出来缓存到PMS里面的一个Package里的不同集合里：

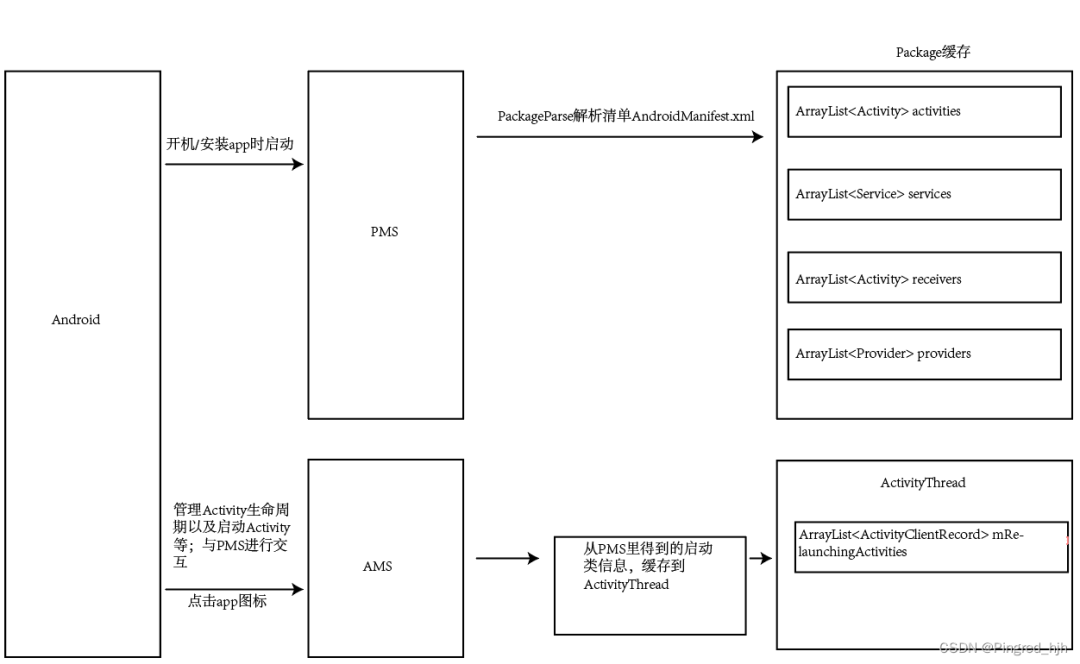
但这里要注意的是，收集Activity信息的集合里的这个Activity类它不是我们平常见到的活动类Activity类，它是Package类里的自己定义的特殊Activity，里面存储的只是清单文件里关于Activity类的一些路径、是否是启动活动等的一些配置信息，因此它不是正常的完整活动类Activity：

上面这些集合就是Package类里对于清单文件里配置的四大组件而定义的缓存集合，这些集合就是用来缓存清单文件里配置的四大组件，但并不是真正的四大组件的类。

每个apk对应一个缓存对象Package，而PMS是一个单独进程，来管理这些Package对象。

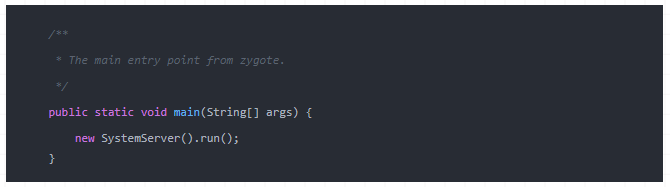
之后当用户点击app图标的那一刻，AMS就根据PMS里Package里的这些集合里的类信息以及intent里设置的意图信息，然后定位到要显示的类，创建然后启动它：

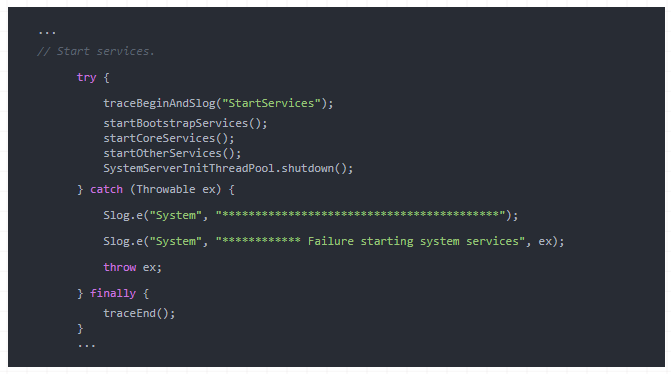
AMS从PMS里拿到这个启动类的信息数据之后，就缓存到它自己的ActivityThread里的ActivityClientRecord集合里，也就是跳转记录集合里：

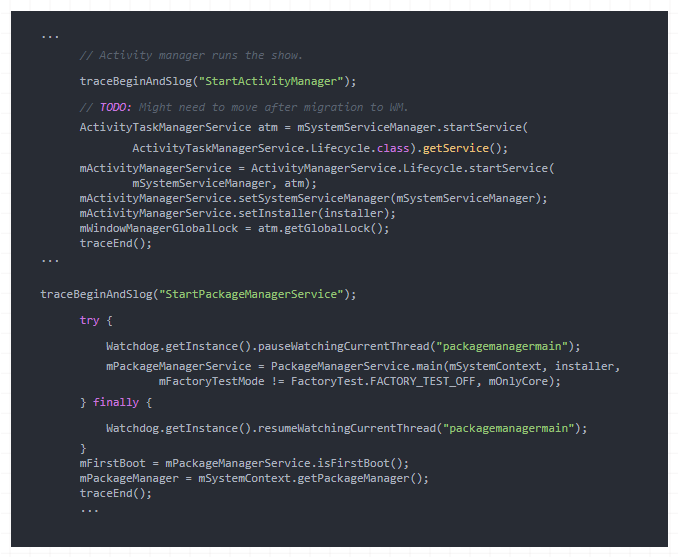


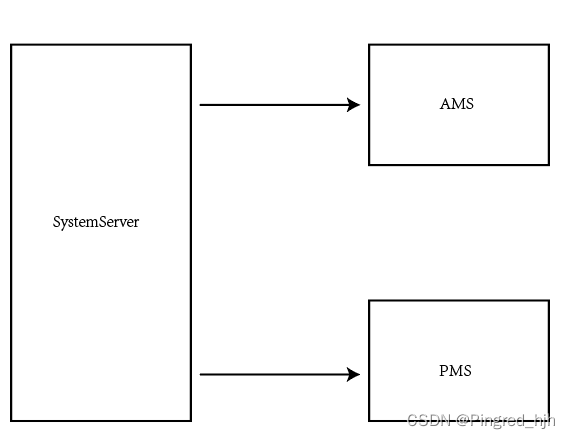
# **二、SystemServer**

SystemServer是管理和启动所有服务的类，PMS和AMS就是它负责管理和启动的。而它自己的启动方法run方法则是在手机开启时就会被调用：

可以看到，它是在main里调用的，而main方法又是被zygote进程调用的，也就是说SystemServer进程是被zygote进程启动的。重新看回SystemServer的run方法，往下看：

这里面启动了很多服务，看startBootstrapServices()方法：



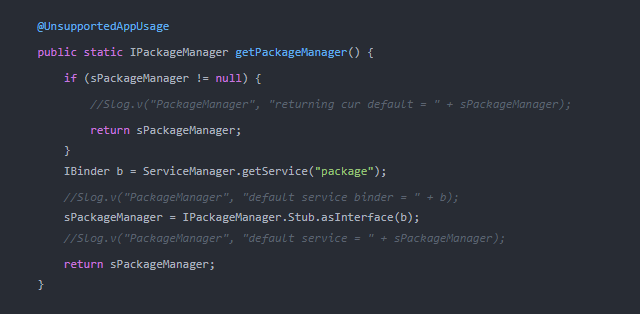
可以看到，构建了AMS对象也就是mActivityManagerService ，然后启动了AMS，继续往下看就是调用了PackageManagerService的main方法，即启动了PMS，所以我们也可以知道，当开机时，SystemServer会去启动AMS和PMS等这些服务：

而且从run方法来看，这个方法里这么多代码，相信里面也是不少耗时功能的，所以这也是手机开机时比较耗时的原因之一。

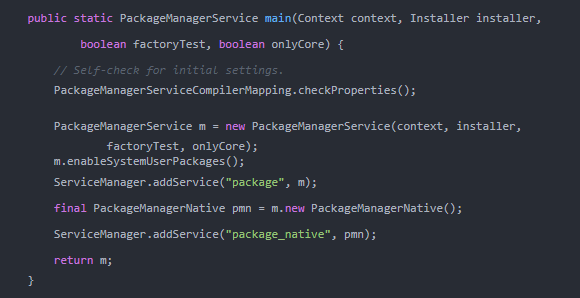
如果我们想拿到PMS对象，则可以通过context的getPackageManager()方法，但它返回的是PackageManager抽象对象，因此要调用它的实现对象ContextImpl的实现方法：

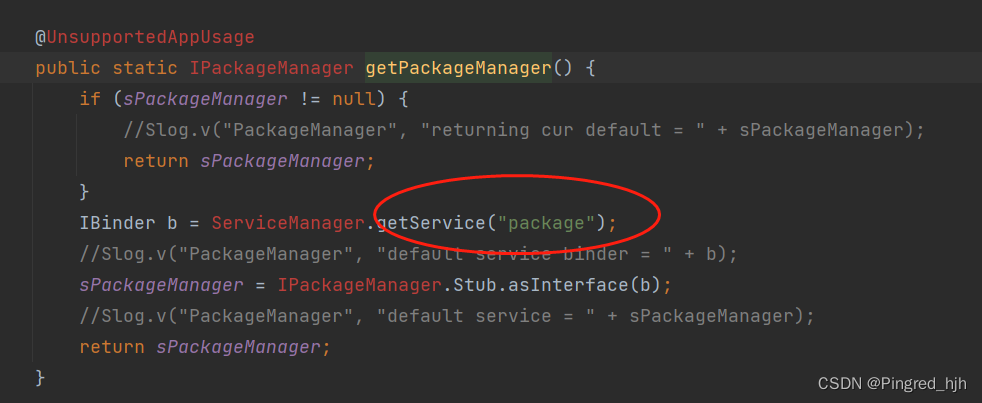


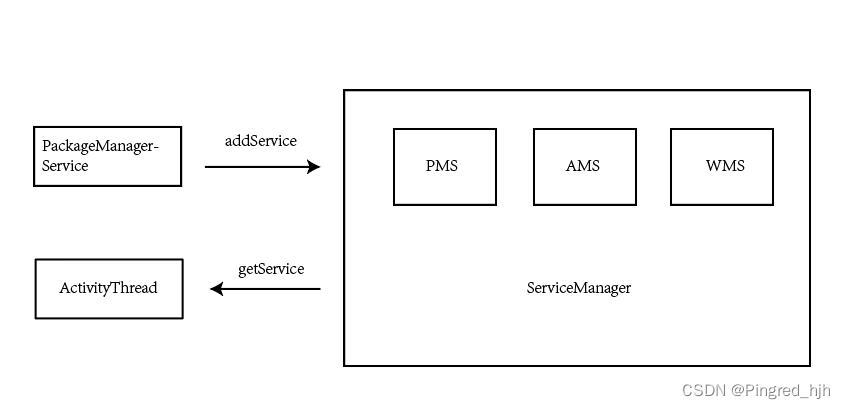
可以看到，通过ActivityThread的getPackageManager方法去获取，然后返回的是IPackageManager抽象对象，它是个Binder对象，因为此时我们是在app里使用context去获取PMS，这相当于app进程和SystemServer进程的通信(SystemServer拥有PMS)，也就是跨进程间通信，因此要使用Binder机制来进行获取进程对象的：



既然PMS也是服务，那么我们来看它的main方法：

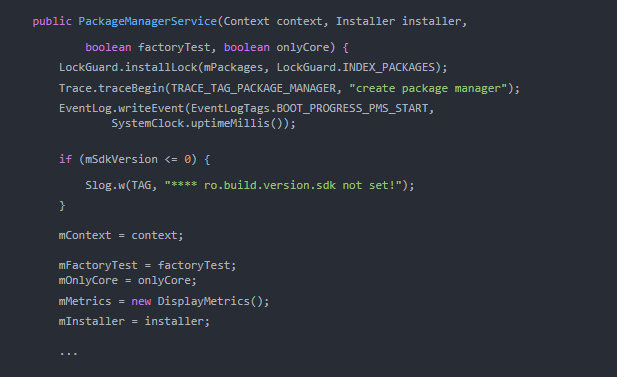


先是构造了自己PackageManagerService对象m，然后按照key-value方式存放到ServiceManager里，其实就是缓存，下次再来取PMS对象时，就可以直接按照key方法来取：

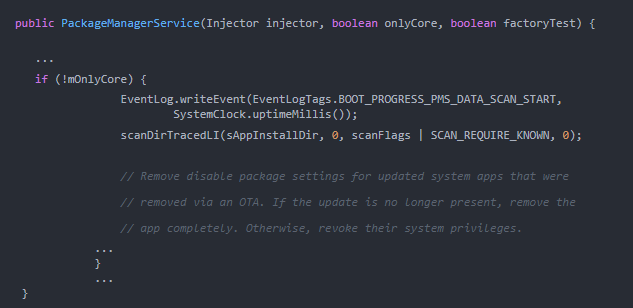
所以可以知道ServiceManager存储了很多服务的对象：

App应用就可以去调用ServiceManager去取这些服务对象了，当然取出来的是Binder对象，存的时候则是真实的服务对象。

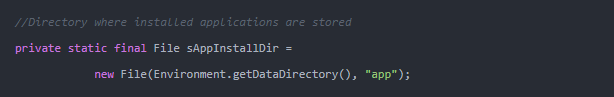
我们继续看回PMS的main方法里，它构造了自己，因此来看它的构造方法：



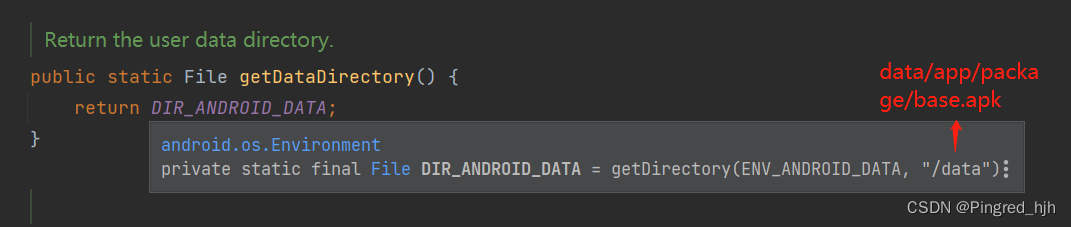
代码很长，所以这也解释了为什么手机开机时要花费这么长时间了，我们往下看这段代码：



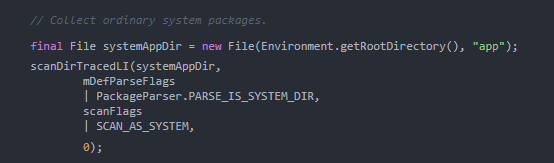
这里调用了一个scanDirTracedLI()方法，里面有个变量sAppInstallDir，可以看看它是什么：

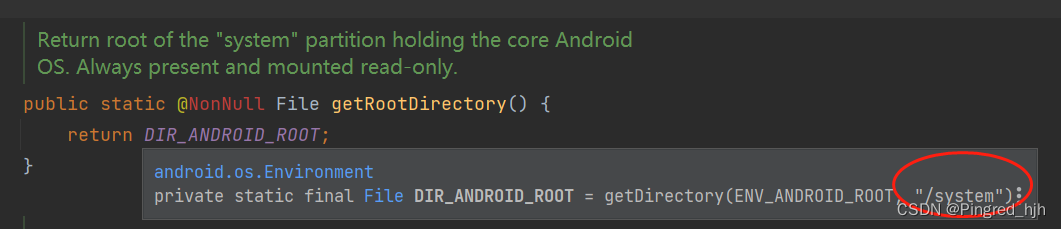


通过注释可以知道，这个sAppInstallDir就是缓存的要安装的应用的路径，也就是安装app时它的apk文件缓存的目录：

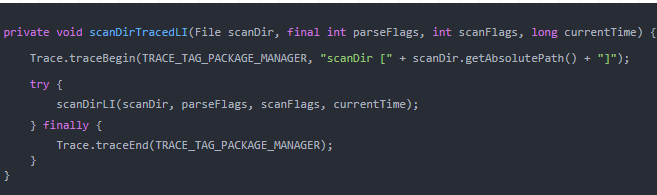


DIR\_ANDROID\_ADTA的值就是data/app目录，刚刚可以看到sAppInstallDir被传到这个scanDirTracedLI方法里，这个方法就是去扫描data/app目录下每个apk方法，而系统app的apk缓存文件（system/app）也是在PMS构造方法里去扫描的：



而这里的systemAppDir跟踪一下可以知道就是system/app目录：

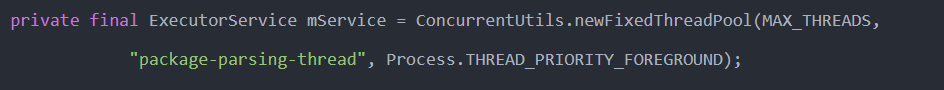
接下来我们来看看scanDirTracedLI()这个扫描方法的详情：

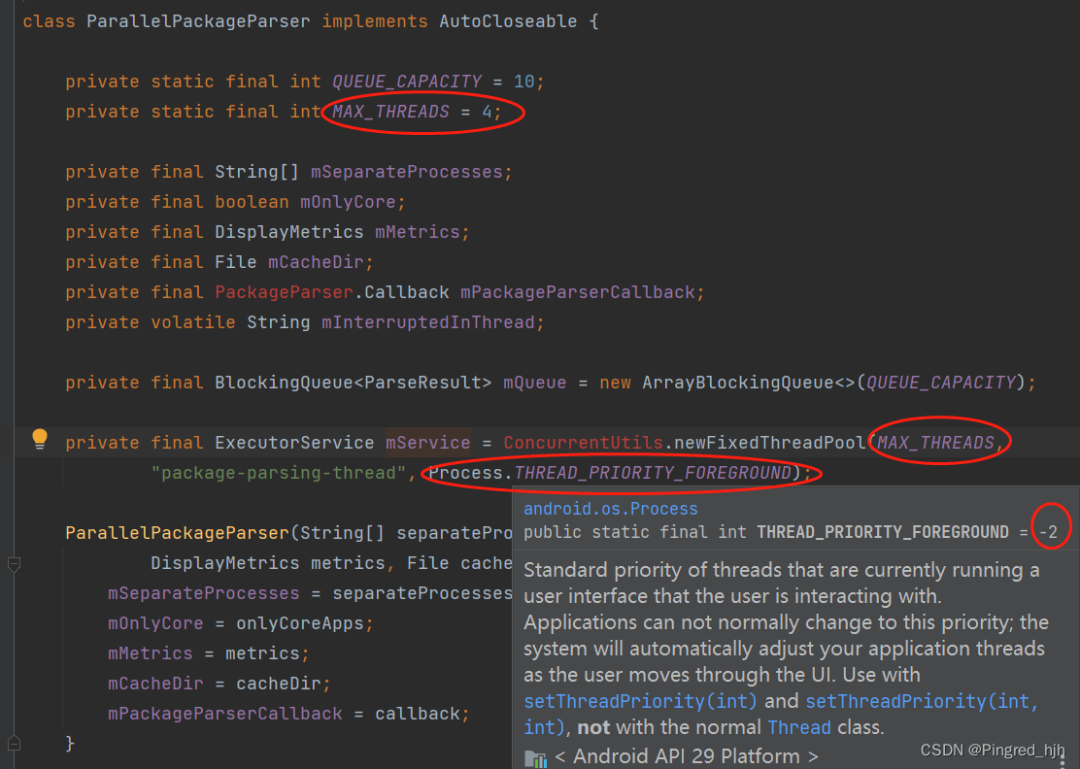


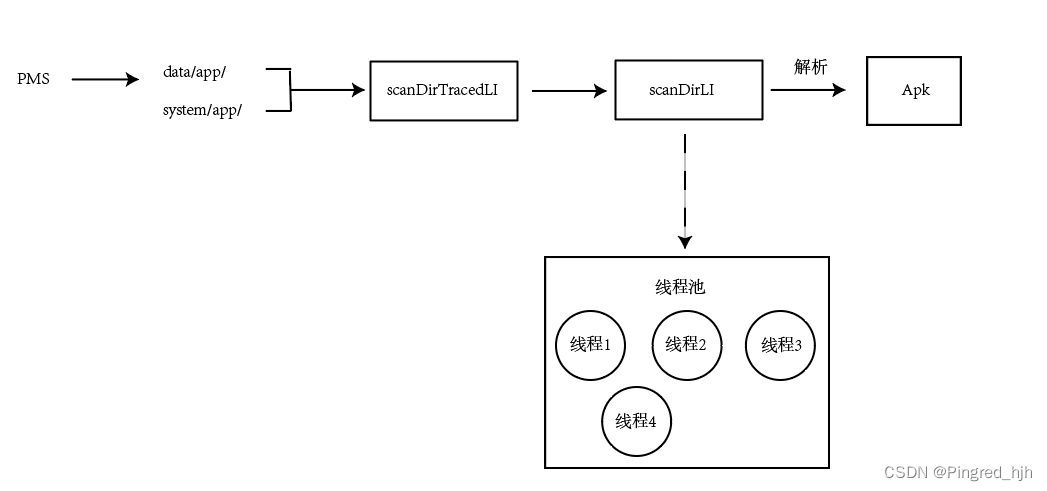
它里面调用了scanDirLI()，那么继续跟踪此方法：

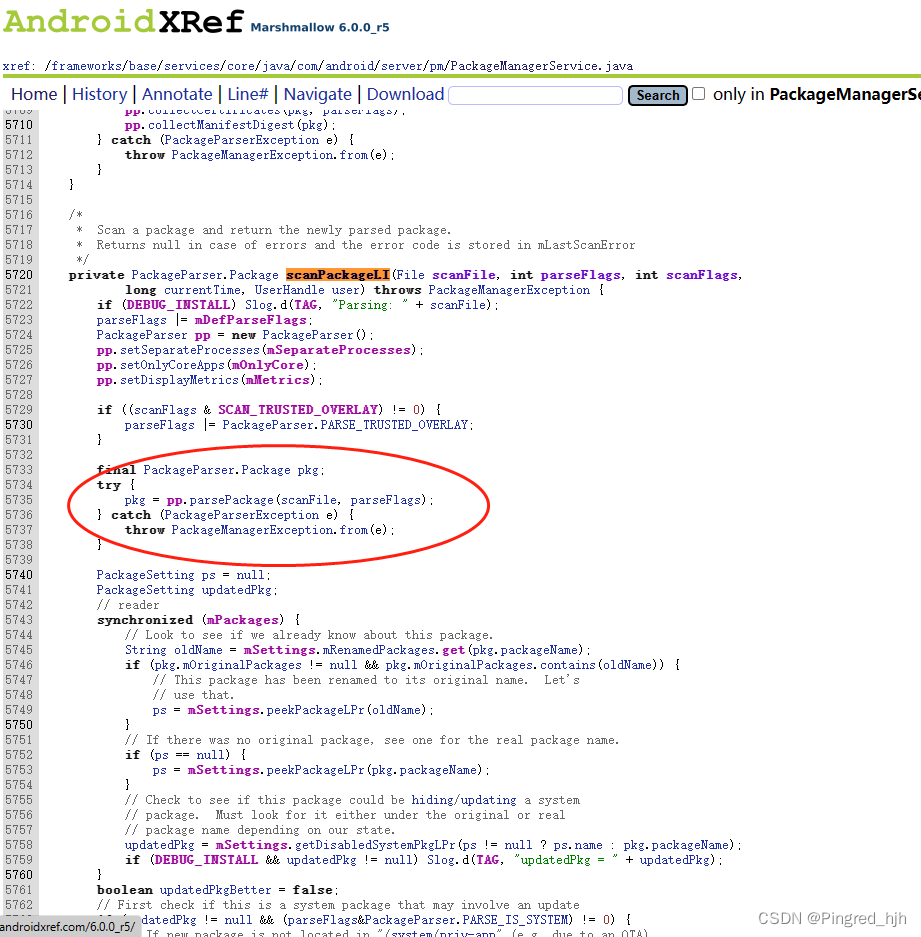
可以看到关键代码for循环里遍历每个apk文件，然后判断是否是apk文件才去解析，然后把apk文件传到了parallelPackageParser的submit方法里，来看看这个方法的详情：

实质上是调用了mService的submit方法，这个mService是一个线程池：



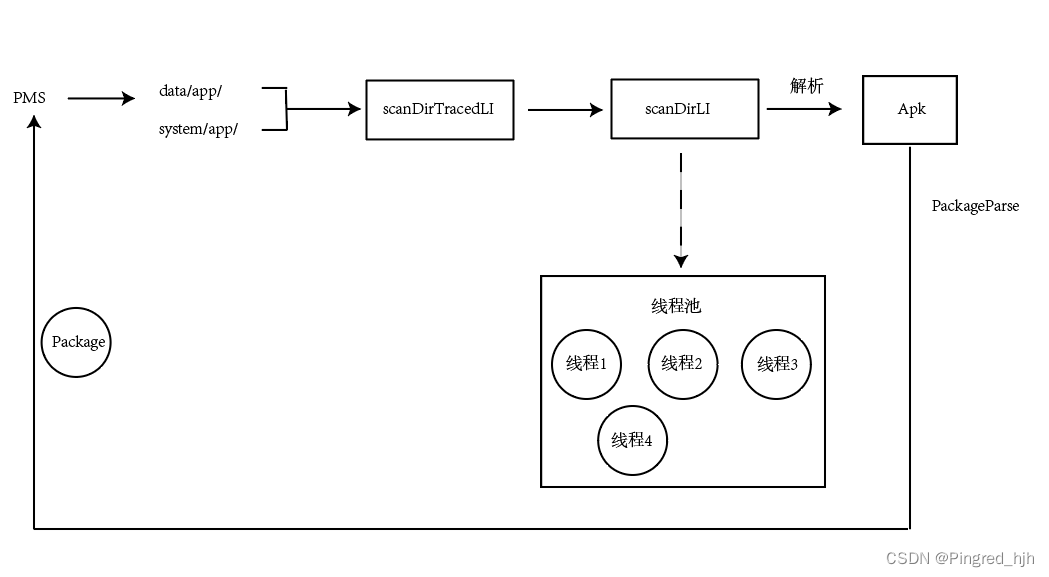
在这里插入图片描述

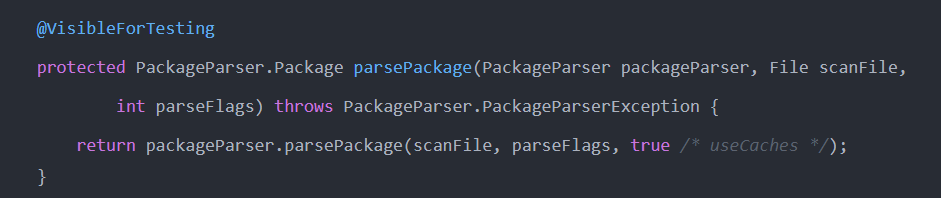
线程核心数为4，也就是说，解析apk其实是启动了一个线程池去解析的：

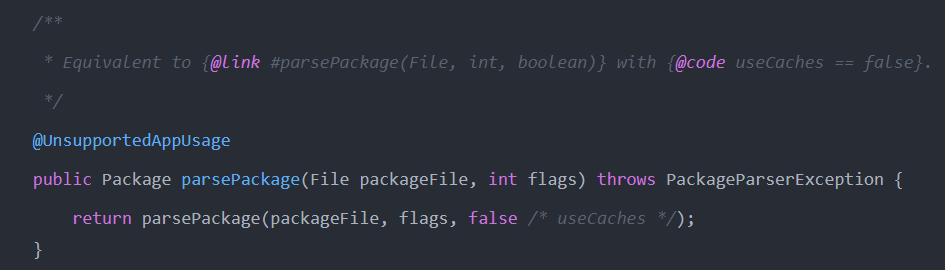
这是安卓29（10.0）的版本下的解析代码，而以前版本比如6.0-8.0的版本下是没有使用线程池去解析的，直接在扫描方法里解析的：

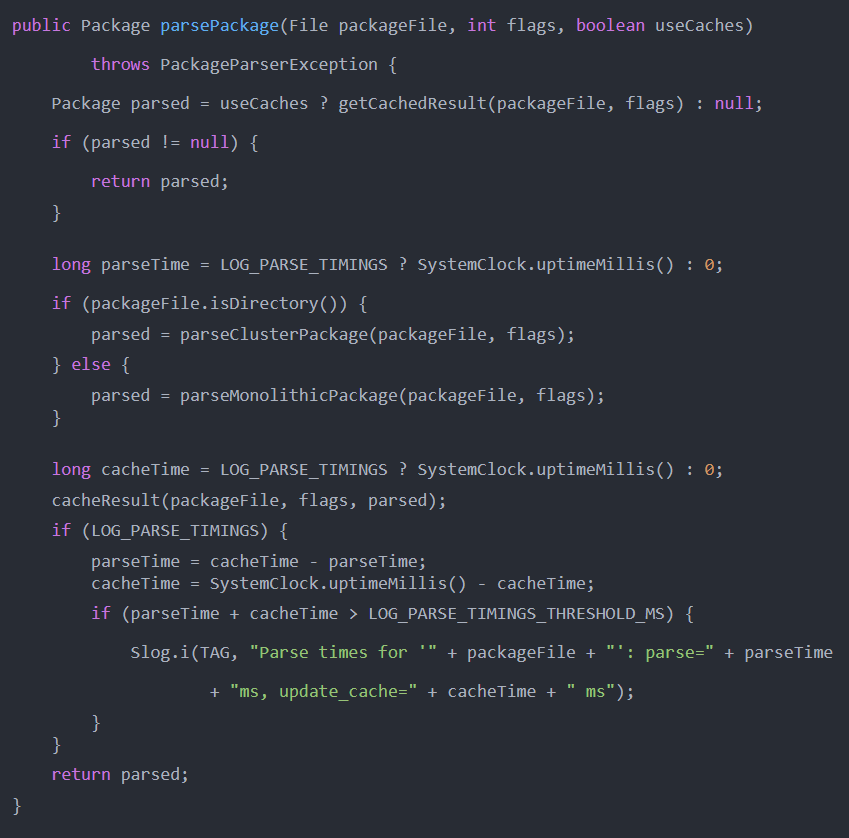
可以看到，6.0版本的解析过程中就没有启动线程池，而是直接就使用PackageParse去解析了，所以对比10.0系统，谷歌很明显优化了PMS在解析apk文件时的操作，启动了线程池去解析，因此现在新版的安卓手机开机时间会比以前快很多。而9.0和11.0也是启动了线程池去解析的，这里就不展示源码，感兴趣可自行搜索查看。

# **三、解析**

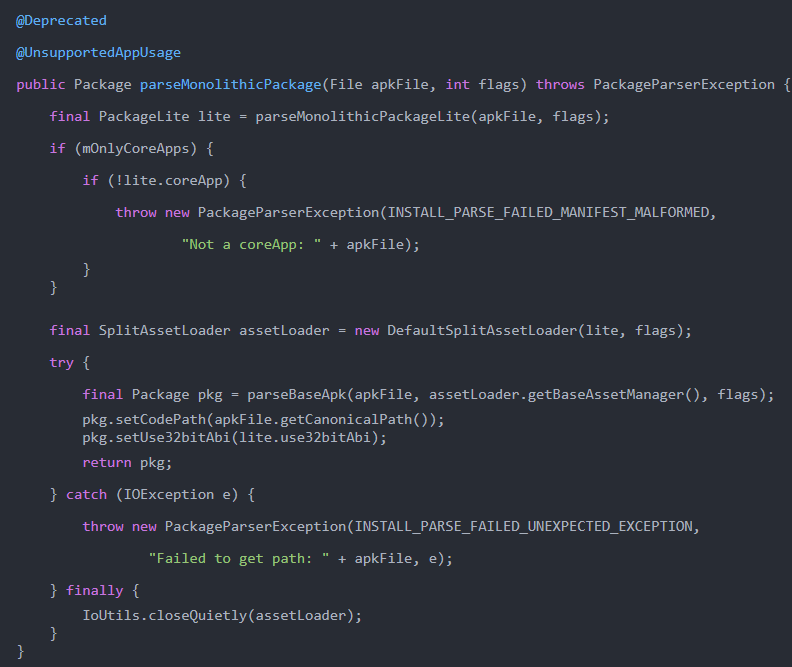
上文parallelPackageParser的submit()里已经知道，里面会使用线程池，然后调用parsePackage()方法，该方法就是进行解析，把清单文件里的类信息都解析成Package对象返回给PMS：现在来看看parsePackage()方法里的详情：

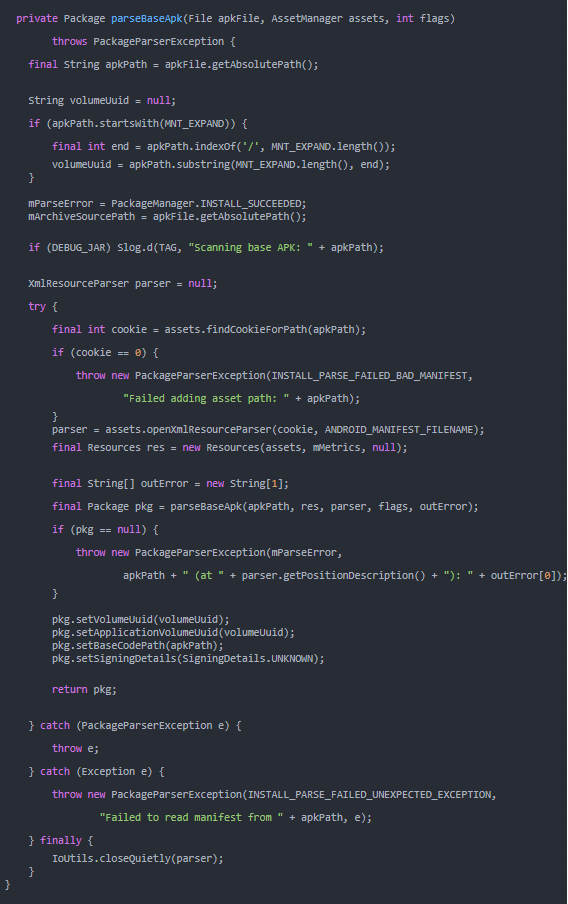
接着跟踪packageParser的parsePackge()方法：

返回的是Package的parsePackage()方法，所以继续跟踪：

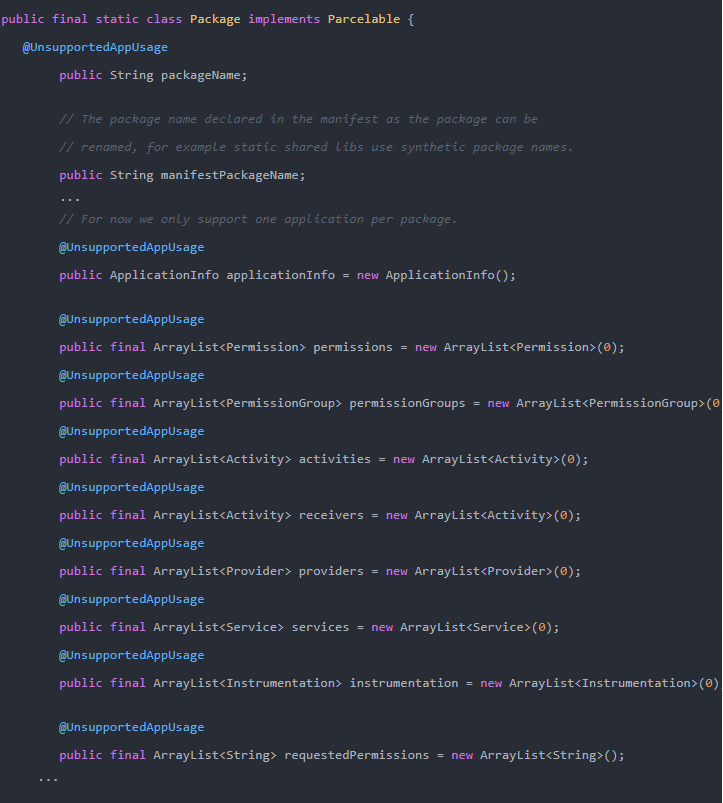
首先就是判断有无缓存useCaches，有则从缓存里直接调用getCachedResult()获取这个解析对象Package，没有缓存则重新解析，这种缓存机制也是9.0之后才有，9.0之前的版本是没有缓存，每次都需重新解析(所以这也是优化解析性能的一个点，感兴趣可自行查看源码对比)：

继续往下看解析方法可以看到，里面有个判断packageFile.isDirectory()，如果不是目录，也就是有apk文件的，则调用parseMonolithicPackage()，因此继续看该方法的详情：

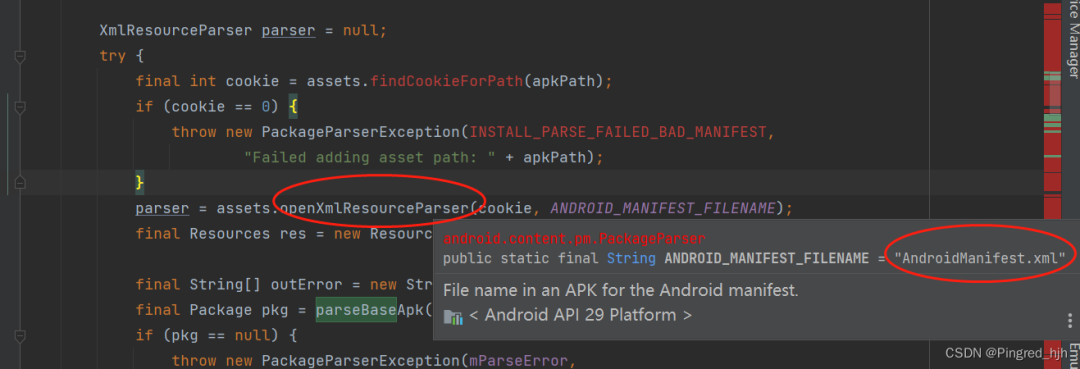
代码虽然有点长，但还是可以看到中间调用了parseBaseApk()方法：

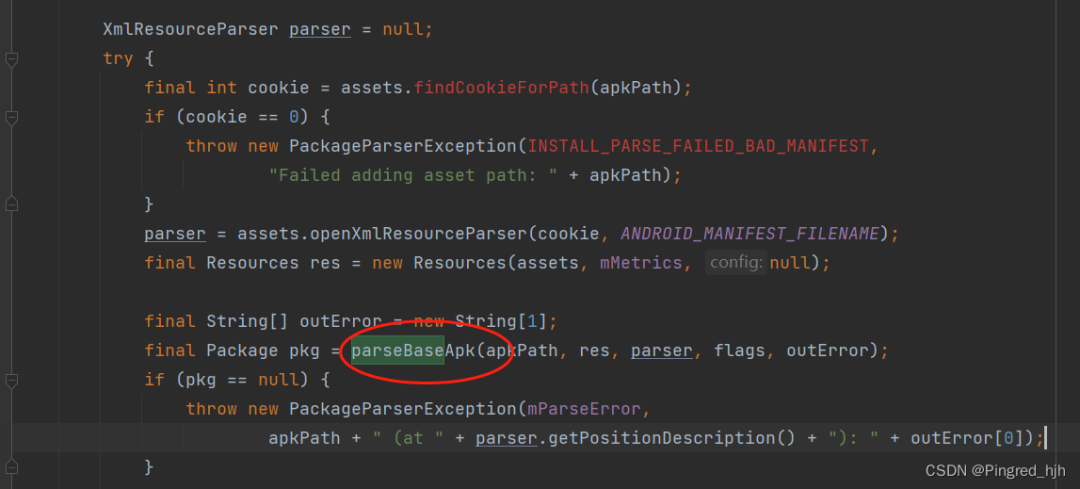


看到这里相信大家都能知道，parseBaseApk这个方法就是去解析清单文件AndroidManifest.xml，最终得把解析的节点信息都封装在Package对象，一个apk对应一个Package对象，然后返回给PMS，看看这个Package类：



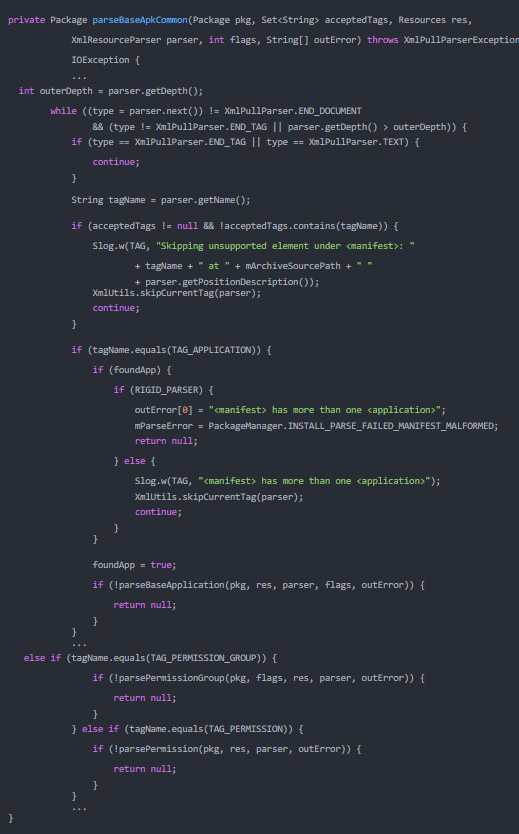
可以看到，有包名信息，还有权限集合，以及四大组件对应的存储集合，每个集合存放对应的类型（Activity、Service、Provider和PermissionGroup等）。

我们继续看回parseBaseApk()方法：

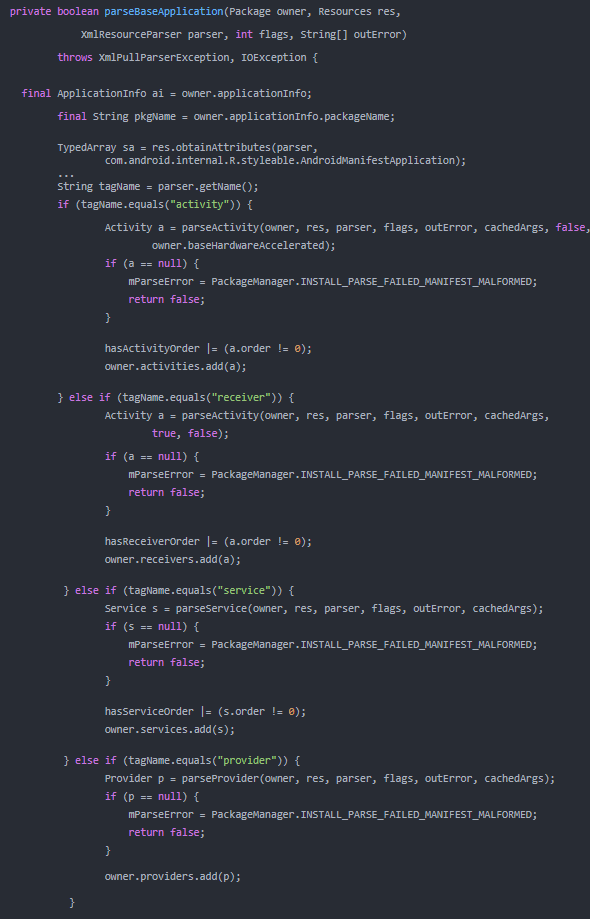
openXmlResourceParser()方法打开的就是我们的清单文件，也就是解析xml文件的节点信息，最后返回解析对象，然后把解析对象又传到了下面调用的parseBaseApk()方法里：

来看parseBaseApk方法的详情：

该方法是为了获取到app的版本号等信息(清单文件里定义的sdkVersion等)，最后返回parseBaseApkCommon()方法：

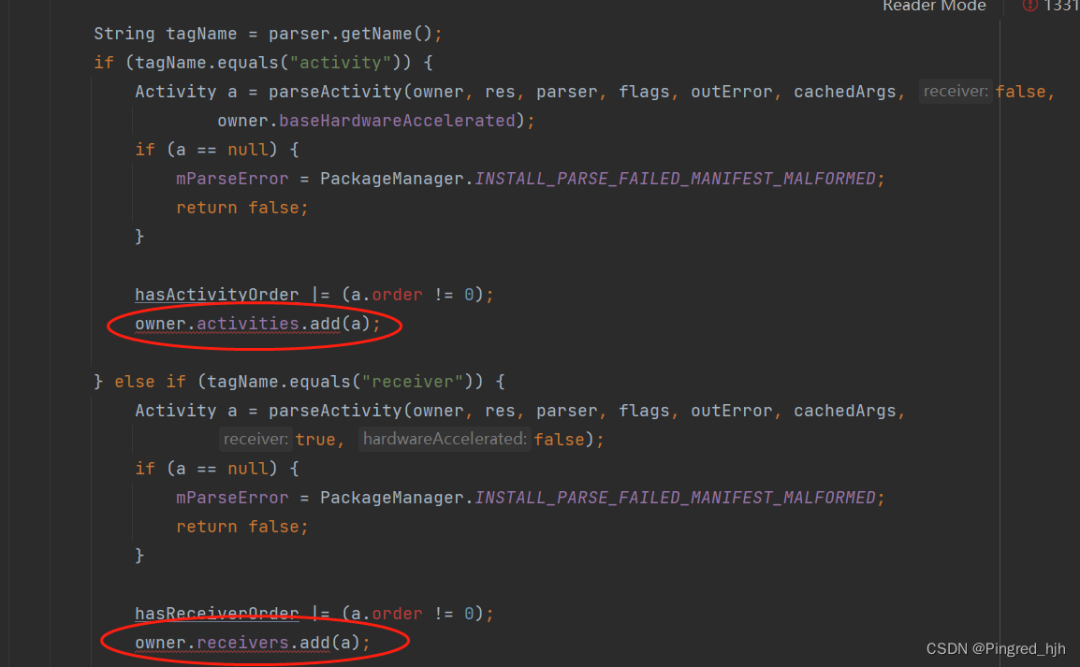


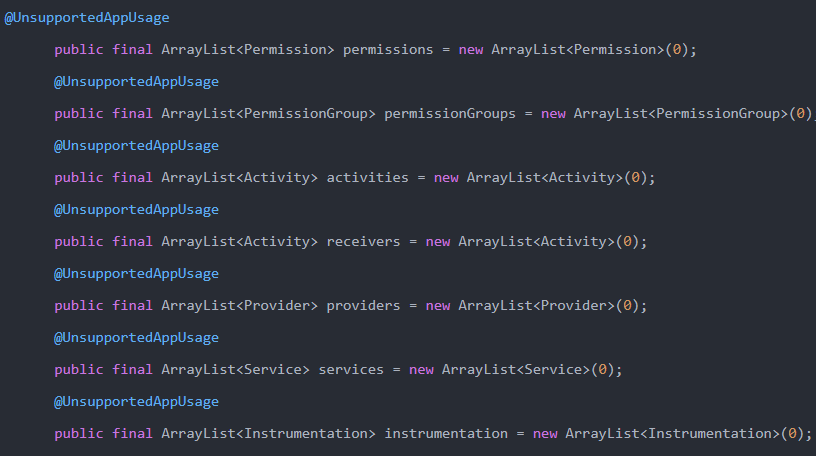
可以看到关键的代码，while循环里获取清单文件AndroidManifest.xml里每个节点，对不同的节点进行相应的解析方法，我们可以来看看其中一个方法，比如parseBaseApplication()：

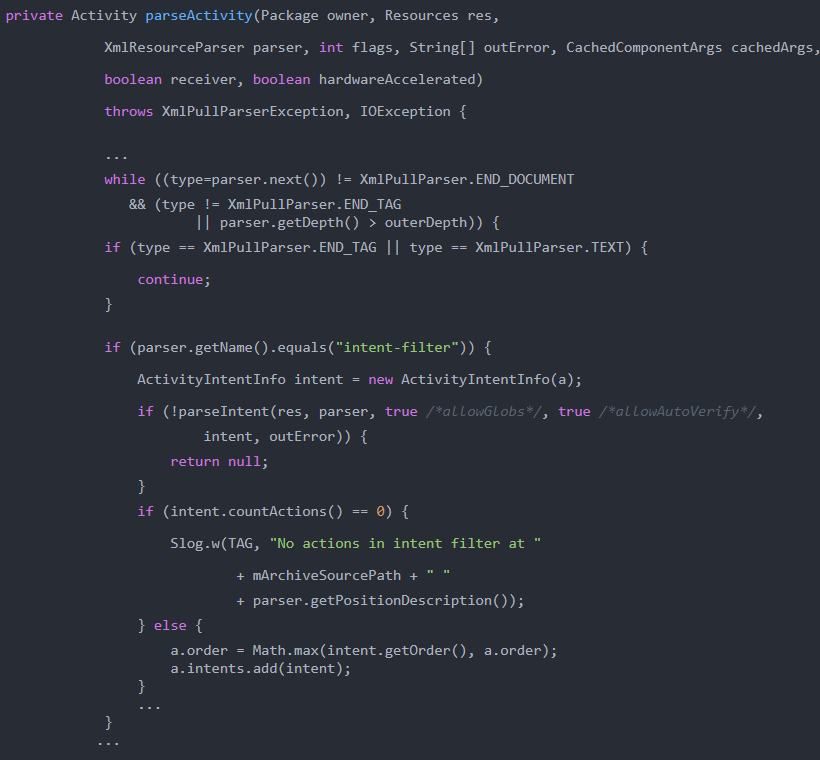


可以看到，分别对清单文件里application节点里的元素activity节点和service节点等进行解析与创建，创建的是Package它自己的Activity类和Service类，这些类跟我们平时见到的Activity类和Service类是不同的，是PMS的自定义的内部类，用来存储清单文件里我们设置的类的信息，用相应的集合来存储，最后供AMS创建真正的类时用的。

不过这里要注意的是，解析广播组件的时候，用的也是Activity来构造，那是因为广播和Activity在清单文件里设置的时候很像，因此这里也是处于方便的考虑，把广播也认作是Activity。

最后解析出来的Activity对象都被添加到Package里对应的集合里：也就是这些集合：

我们再来看看parseActivity()方法：



可以该方法里解析activity节点时，也会遍历activity节点里的子节点，即解析清单文件里面的比如activity节点里的intent-filter信息，把ActivityIntentInfo对象intent解析出来然后放进Activity的intents集合里，这个intents集合是Component类里的II集合(Activity继承Component)：

# **四、延伸**

经过上面的讲解后，大家都会对PMS有更深的理解了。在开发过程中，我们自己也可以通过PMS去做一些事情。

既然通过源码知道了：PMS是通过scanDirTracedLI()方法去扫描apk文件里的清单文件，然后把里面的类等信息解析成一个Package对象(通过PackageParser的parsePackage()方法所得)。

那我们也可以**「通过反射对应的方法去对一些特定的apk文件进行扫描和解析，然后根据Package对象获取到里面的类信息，去创建我们需要的四大组件类，从而达到我们的app和其他apk进行通信的目的了。」**