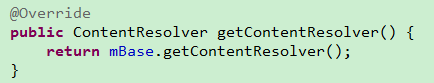
ContentProvider运行机制

ContentProvider是一种内容共享型组件，它通过Binder向其他组件和应用提供数据，当ContentProvider所在的组件启动时，ContentProvider会同时启动并被发布到AMS中。当ContentProvider未启动时，第一次访问他时就会触发ContentProvider的创建，同时启动ContentProvider所在的进程。ContentProvider的增删改查方法逻辑大致相同，此处以query方法为例进行分析。

**一、ContentResolver的获取**

访问ContentProvider需要ContentResolver，ContentResolve是一个抽象类，通常在一个Activity中通过Context.getContentResolver().query查询需要的数据，此处的context对象实际上是ContextWrapper，查看该类的query方法如下：



上面函数中的mBase对象实际上是ContextImpl，Activity创建过程中会通过attach方法将一个ContextImpl对象关联起来。定位到ContextImpl的getContentResolver方法，找到ContentResolver会在ContextImpl的构造函数中初始化：

捕获

ApplicationContentResolver为ContextImpl的内部类。它继承自ContentResolver，并实现其抽象方法。

1. **query方法中获取IContentProvider的逻辑过程**

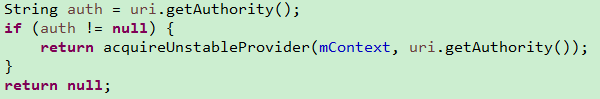
回到getContentResolver().query方法，该方法在ContentResolver中，它又调用了本类的query方法，在query方法中首先通过如下方法获取IContentProvider对象，IContentProvider是一个Binder类型的对象，下面的方法只有一个参数URI，通过不同的URI，便可以获得不同的ContentProvider。

捕获

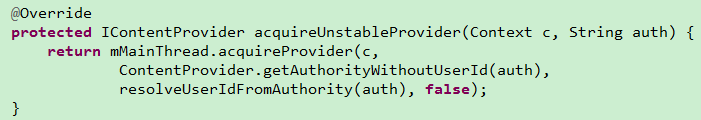
获得了IContentProvider对象以后，接下来在query方法中就会直接通过该对象调用query方法跨进程获取返回的Cursor数据集，具体的query方法的实现在具体的ContentProvider类中，

捕获

继续跟踪acquireUnstableProvider方法：



该方法判断authority不为空，接着调用acquireUnstableProvider方法，该方法是一个系统抽象方法，具体的实现在ApplicationContentResolver中，在该类中并没有其他的逻辑处理，而是直接调用了ActivityThread的acquireProvider方法：



ActivityThread的acquireProvider方法负责返回特定的Binder类型的IContentProvider对象，ActivityThread的acquireProvider方法的处理逻辑分三步：

1、首先从ActivityThread查找是否存在目标ContentProvider，如果存在就直接返回，这一步是通过acquireExistingProvider方法实现的，ActivityThread中会通过mProviderMap存储了已经启动的ContentProvider对象，该方法就是搜索mProviderMap来查找指定的ContentProvider的。

1. 如果目标ContentProvider不存在，那么就会发送一个进程间请求给AMS，让其启动目标ContentProvider，

捕获

上面的ActivityManagerNative.getDefault返回的是ActivityManagerProxy对象，ActivityManagerProxy是ActivityManagerService的代理类，安卓中所有对ActivityManagerService的访问都转换成对该代理类的访问，利于程序的解耦。

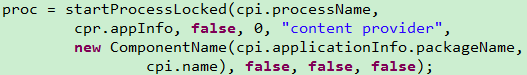
3、通过installProvider方法修改引用计数

这里假设在ActivityThread的acquireProvider方法中目标ContentProvider不存在，则程序会走到AMS的getContentProvider方法，最后调用到getContentProviderImpl，该方法逻辑如下

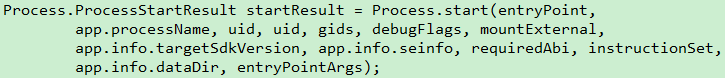
1. 首先通过如下方法查找目标的provider是否已经被发布到AMS，如果已经被发布到AMS，则返回目标provider

捕获

1. 如果目标provider没有被发布到AMS，则首先启动ContentProvider所在的进程，然后启动对应的ContentProvider



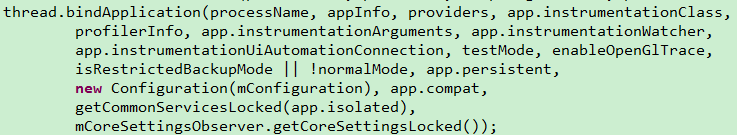
其内部主要是通过Process的start方法来完成一个新进程的启动的



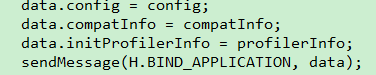
新进程启动以后其入口方法为ActivityThread的main方法。Main方法是一个静态方法，它的内部会首先创建ActivityThread实例，然后通过其attach方法进行一系列的初始化

捕获

Attach方法中会将ApplicationThread对象通过AMS的attachApplication跨进程传递给AMS，之后在AMS的attachApplication方法中又会调用attachApplicationLocked方法，attachApplicationLocked方法又调用了ApplicationThread的bindApplication方法，



而ApplicationThread的bindApplication方法中会发送一个BIND\_APPLICATION广播：



然后在handleMessage中又调用了handleBindApplication，该方法最终会完成Application的创建和provider的创建，handleBindApplication流程如下：

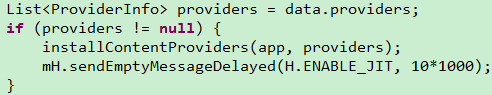
1. 创建ContextImpl

捕获

1. 创建Application对象

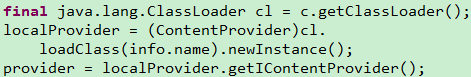
捕获

1. 启动ContentProvider，并调用其onCreate方法



installContentProviders会遍历当前进程的ProviderInfo列表并一一调用installProvider方法启动他们，并将已经启动的provider通过publishContentProviders发布到AMS，而AMS会将启动的provider存储在ProviderMap中，以便于以后其他应用直接可以从AMS中获取指定的ContentProvider。

installProvider方法中使用类加载器完成ContentProvider的创建



installProvider方法中创建了ContentProvider后还会调用ContentProvider的attachInfo方法启动ContentProvider，到此为止ContentProvider就启动完成了

捕获

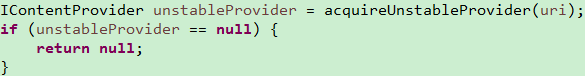
1. 调用Application的onCreate方法启动Application

捕获

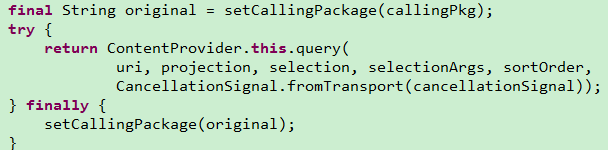
上面四步执行完毕以后，ContentProvider所在的进程就已经启动了，且ContentProvider也成功启动，其他应用可以直接通过AMS访问此ContentProvider了。

1. **query方法的具体实现**

通过上面的逻辑分析获取了对应的ContentProvider对象，然后返回ContentResolve的query方法中继续执行



通过返回值对象可以看出该方法返回的为IContentProvider，它是ContentProvider的Binder类型的对象，IContentProvider的具体实现在ContentProviderNative和ContentProvider.Transport中，其中ContentProvider.Transport继承自ContentProviderNative，所以其他的应用调用了IContentProvider的query方法时最终会以进程间通信的方式调用到ContentProvider.Transport的query方法中



可以看到ContentProvider.Transport的query方法最终调用到ContentProvider的query方法，query方法的执行结果也会通过Binder返回给调用者，这样就完成了整个功能的调用。