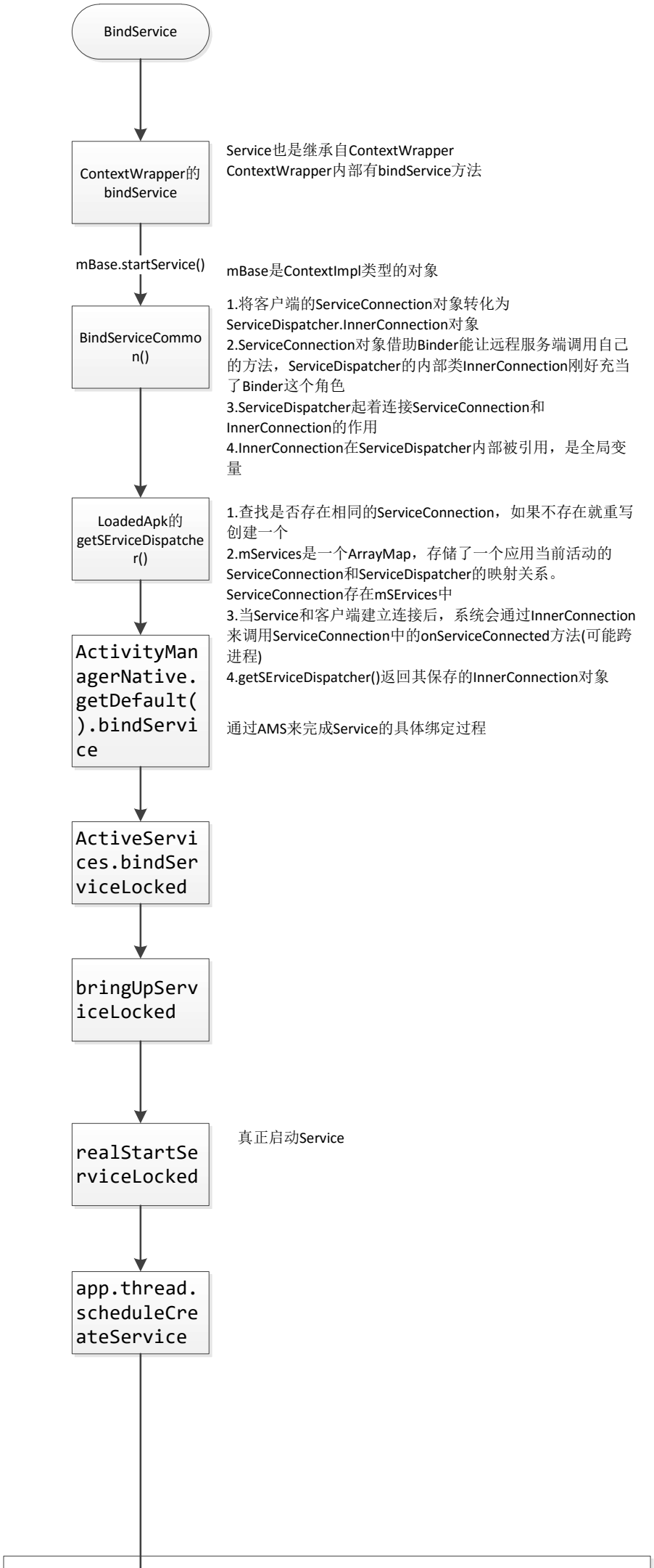
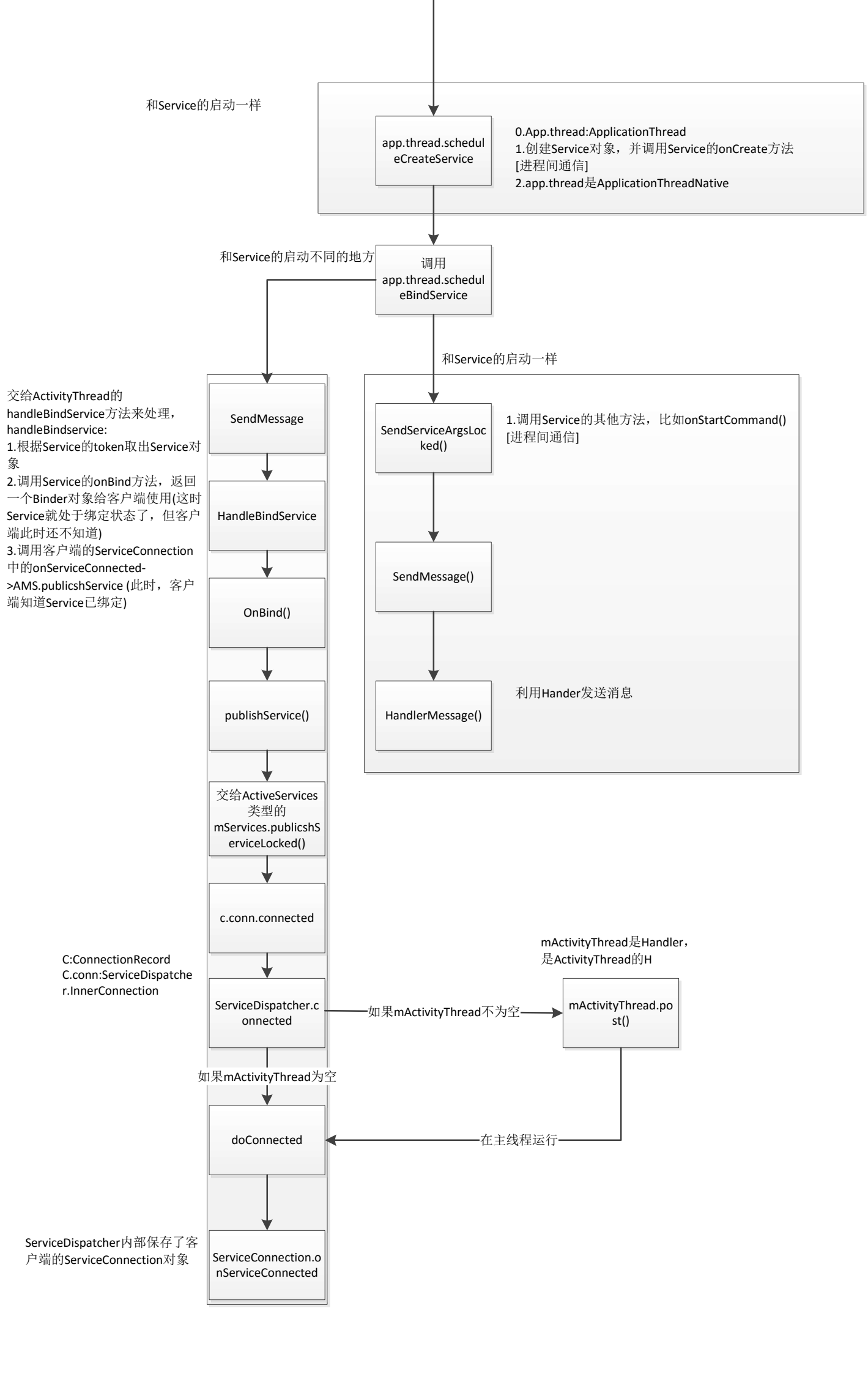

BindService
ContextWrapper的bindService
Service也是继承自ContextWrapper
ContextWrapper内部有bindService方法
mBase.startService()
mBase是ContextImpl类型的对象
BindServiceCommon()
1.将客户端的ServiceConnection对象转化为ServiceDispatcher.InnerConnection对象
2.ServiceConnection对象借助Binder能让远程服务端调用自己的方法，ServiceDispatcher的内部类InnerConnection刚好充当了Binder这个角色
3.ServiceDispatcher起着连接ServiceConnection和InnerConnection的作用
4.InnerConnection在ServiceDispatcher内部被引用，是全局变量
LoadedApk的getSErviceDispatcher()
1.查找是否存在相同的ServiceConnection，如果不存在就重写创建一个
2.mServices是一个ArrayMap，存储了一个应用当前活动的ServiceConnection和ServiceDispatcher的映射关系。ServiceConnection存在mSErvices中
3.当Service和客户端建立连接后，系统会通过InnerConnection来调用ServiceConnection中的onServiceConnected方法(可能跨进程)
4.getSErviceDispatcher()返回其保存的InnerConnection对象
ActivityManagerNative.getDefault().bindService
通过AMS来完成Service的具体绑定过程
ActiveServices.bindServiceLocked
bringUpServiceLocked
realStartServiceLocked
真正启动Service
app.thread.scheduleCreateService


和Service的启动一样
app.thread.scheduleCreateService
0.App.thread:ApplicationThread
1.创建Service对象，并调用Service的onCreate方法
[进程间通信]
2.app.thread是ApplicationThreadNative
和Service的启动不同的地方
调用
app.thread.scheduleBindService
和Service的启动一样
交给ActivityThread的
handleBindService方法来处理，
handleBindservice:
1.根据Service的token取出Service对象
2.调用Service的onBind方法，返回一个Binder对象给客户端使用(这时Service就处于绑定状态了，但客户端此时还不知道)
3.调用客户端的ServiceConnection中的onServiceConnected->AMS.publicshService (此时，客户端知道Service已绑定)
SendMessage
HandleBindService
OnBind()
publishService()
交给ActiveServices类型的
mServices.publicshServiceLocked()
c.conn.connected
SendServiceArgsLocked()
1.调用Service的其他方法，比如onStartCommand()
[进程间通信]
SendMessage()
HandlerMessage()
利用Hander发送消息
mActivityThread是Handler，
是ActivityThread的H
C:ConnectionRecord
C.conn:ServiceDispatcher.InnerConnection
ServiceDispatcher.connected
如果mActivityThread不为空
mActivityThread.post()
如果mActivityThread为空
doConnected
在主线程运行
ServiceDispatcher内部保存了客户端的ServiceConnection对象
ServiceConnection.onServiceConnected