# Service启动流程分析

# ========StartService启动==============



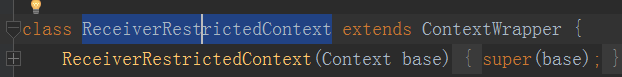
## 1、调用ContextWrapper的startService方法

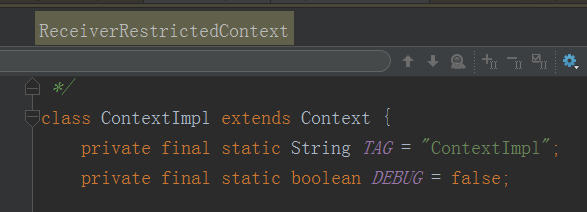
@Override  
public ComponentName startService(Intent service) {  
 return mBase.startService(service);  
}

## 2、类结构说明



ContextImpl 是ReceiverRestrictedContext的内部类





## 3、调用内部类ContextImpl 的startService方法

@Override  
public ComponentName startService(Intent service) {  
 warnIfCallingFromSystemProcess();  
 return startServiceCommon(service, mUser);  
}

## 4、调用AMS的startService方法

**ActivityManagerNative 类似IPC中的stub所有方法运行在远程进程的binder线程池中**

private ComponentName startServiceCommon(Intent service, UserHandle user) {  
 try {  
 validateServiceIntent(service);  
 service.prepareToLeaveProcess(this);  
 ComponentName cn = **ActivityManagerNative.*getDefault*().startService**(  
 mMainThread.getApplicationThread(), service, service.resolveTypeIfNeeded(  
 getContentResolver()), getOpPackageName(), user.getIdentifier());  
 if (cn != null) {  
 if (cn.getPackageName().equals("!")) {  
 throw new SecurityException(  
 "Not allowed to start service " + service  
 + " without permission " + cn.getClassName());  
 } else if (cn.getPackageName().equals("!!")) {  
 throw new SecurityException(  
 "Unable to start service " + service  
 + ": " + cn.getClassName());  
 }  
 }  
 return cn;  
 } catch (RemoteException e) {  
 throw e.rethrowFromSystemServer();  
 }  
}

## 5、AMS的startService方法调用ActiveService类的startServiceLocked方法（ActiveService类中）

@Override  
public ComponentName startService(IApplicationThread caller, Intent service,  
 String resolvedType, String callingPackage, int userId)  
 throws TransactionTooLargeException {  
 enforceNotIsolatedCaller("startService");  
 // Refuse possible leaked file descriptors  
 if (service != null && service.hasFileDescriptors() == true) {  
 throw new IllegalArgumentException("File descriptors passed in Intent");  
 }  
  
 if (callingPackage == null) {  
 throw new IllegalArgumentException("callingPackage cannot be null");  
 }  
  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*,  
 "startService: " + service + " type=" + resolvedType);  
 synchronized(this) {  
 final int callingPid = Binder.*getCallingPid*();  
 final int callingUid = Binder.*getCallingUid*();  
 final long origId = Binder.*clearCallingIdentity*();  
 ComponentName res = **mServices.startServiceLocked**(caller, service,  
 resolvedType, callingPid, callingUid, callingPackage, userId);  
 Binder.*restoreCallingIdentity*(origId);  
 return res;  
 }  
}

****

### 5.1 ActiveService的startServiceLocked方法

ComponentName startServiceLocked(IApplicationThread caller, Intent service, String resolvedType,  
 int callingPid, int callingUid, String callingPackage, final int userId)  
 throws TransactionTooLargeException {  
 if (*DEBUG\_DELAYED\_STARTS*) Slog.v(*TAG\_SERVICE*, "startService: " + service  
 + " type=" + resolvedType + " args=" + service.getExtras());  
  
 final boolean callerFg;  
 if (caller != null) {  
 final ProcessRecord callerApp = mAm.getRecordForAppLocked(caller);  
 if (callerApp == null) {  
 throw new SecurityException(  
 "Unable to find app for caller " + caller  
 + " (pid=" + Binder.*getCallingPid*()  
 + ") when starting service " + service);  
 }  
 callerFg = callerApp.setSchedGroup != ProcessList.*SCHED\_GROUP\_BACKGROUND*;  
 } else {  
 callerFg = true;  
 }  
  
  
 ServiceLookupResult res =  
 retrieveServiceLocked(service, resolvedType, callingPackage,  
 callingPid, callingUid, userId, true, callerFg, false);  
 if (res == null) {  
 return null;  
 }  
 if (res.record == null) {  
 return new ComponentName("!", res.permission != null  
 ? res.permission : "private to package");  
 }  
  
 ServiceRecord r = res.record;  
  
 if (!mAm.mUserController.exists(r.userId)) {  
 Slog.w(*TAG*, "Trying to start service with non-existent user! " + r.userId);  
 return null;  
 }  
  
 if (!r.startRequested) {  
 final long token = Binder.*clearCallingIdentity*();  
 try {  
 // Before going further -- if this app is not allowed to run in the  
 // background, then at this point we aren't going to let it period.  
 final int allowed = mAm.checkAllowBackgroundLocked(  
 r.appInfo.uid, r.packageName, callingPid, true);  
 if (allowed != ActivityManager.APP\_START\_MODE\_NORMAL) {  
 Slog.w(*TAG*, "Background start not allowed: service "  
 + service + " to " + r.name.flattenToShortString()  
 + " from pid=" + callingPid + " uid=" + callingUid  
 + " pkg=" + callingPackage);  
 return null;  
 }  
 } finally {  
 Binder.*restoreCallingIdentity*(token);  
 }  
 }  
  
 NeededUriGrants neededGrants = mAm.checkGrantUriPermissionFromIntentLocked(  
 callingUid, r.packageName, service, service.getFlags(), null, r.userId);  
  
 // If permissions need a review before any of the app components can run,  
 // we do not start the service and launch a review activity if the calling app  
 // is in the foreground passing it a pending intent to start the service when  
 // review is completed.  
 if (Build.PERMISSIONS\_REVIEW\_REQUIRED) {  
 if (!requestStartTargetPermissionsReviewIfNeededLocked(r, callingPackage,  
 callingUid, service, callerFg, userId)) {  
 return null;  
 }  
 }  
  
 if (unscheduleServiceRestartLocked(r, callingUid, false)) {  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "START SERVICE WHILE RESTART PENDING: " + r);  
 }  
 r.lastActivity = SystemClock.*uptimeMillis*();  
 r.startRequested = true;  
 r.delayedStop = false;  
 r.pendingStarts.add(new ServiceRecord.StartItem(r, false, r.makeNextStartId(),  
 service, neededGrants));  
  
 final ServiceMap smap = getServiceMap(r.userId);  
 boolean addToStarting = false;  
 if (!callerFg && r.app == null  
 && mAm.mUserController.hasStartedUserState(r.userId)) {  
 ProcessRecord proc = mAm.getProcessRecordLocked(r.processName, r.appInfo.uid, false);  
 if (proc == null || proc.curProcState > ActivityManager.PROCESS\_STATE\_RECEIVER) {  
 // If this is not coming from a foreground caller, then we may want  
 // to delay the start if there are already other background services  
 // that are starting. This is to avoid process start spam when lots  
 // of applications are all handling things like connectivity broadcasts.  
 // We only do this for cached processes, because otherwise an application  
 // can have assumptions about calling startService() for a service to run  
 // in its own process, and for that process to not be killed before the  
 // service is started. This is especially the case for receivers, which  
 // may start a service in onReceive() to do some additional work and have  
 // initialized some global state as part of that.  
 if (*DEBUG\_DELAYED\_SERVICE*) Slog.v(*TAG\_SERVICE*, "Potential start delay of "  
 + r + " in " + proc);  
 if (r.delayed) {  
 // This service is already scheduled for a delayed start; just leave  
 // it still waiting.  
 if (*DEBUG\_DELAYED\_STARTS*) Slog.v(*TAG\_SERVICE*, "Continuing to delay: " + r);  
 return r.name;  
 }  
 if (smap.mStartingBackground.size() >= mMaxStartingBackground) {  
 // Something else is starting, delay!  
 Slog.i(*TAG\_SERVICE*, "Delaying start of: " + r);  
 smap.mDelayedStartList.add(r);  
 r.delayed = true;  
 return r.name;  
 }  
 if (*DEBUG\_DELAYED\_STARTS*) Slog.v(*TAG\_SERVICE*, "Not delaying: " + r);  
 addToStarting = true;  
 } else if (proc.curProcState >= ActivityManager.PROCESS\_STATE\_SERVICE) {  
 // We slightly loosen when we will enqueue this new service as a background  
 // starting service we are waiting for, to also include processes that are  
 // currently running other services or receivers.  
 addToStarting = true;  
 if (*DEBUG\_DELAYED\_STARTS*) Slog.v(*TAG\_SERVICE*,  
 "Not delaying, but counting as bg: " + r);  
 } else if (*DEBUG\_DELAYED\_STARTS*) {  
 StringBuilder sb = new StringBuilder(128);  
 sb.append("Not potential delay (state=").append(proc.curProcState)  
 .append(' ').append(proc.adjType);  
 String reason = proc.makeAdjReason();  
 if (reason != null) {  
 sb.append(' ');  
 sb.append(reason);  
 }  
 sb.append("): ");  
 sb.append(r.toString());  
 Slog.v(*TAG\_SERVICE*, sb.toString());  
 }  
 } else if (*DEBUG\_DELAYED\_STARTS*) {  
 if (callerFg) {  
 Slog.v(*TAG\_SERVICE*, "Not potential delay (callerFg=" + callerFg + " uid="  
 + callingUid + " pid=" + callingPid + "): " + r);  
 } else if (r.app != null) {  
 Slog.v(*TAG\_SERVICE*, "Not potential delay (cur app=" + r.app + "): " + r);  
 } else {  
 Slog.v(*TAG\_SERVICE*,  
 "Not potential delay (user " + r.userId + " not started): " + r);  
 }  
 }  
  
 return **startServiceInnerLocked**(smap, service, r, callerFg, addToStarting);  
}

### 5.2 startServiceInnerLocked 调用bringUpServiceLocked

ComponentName startServiceInnerLocked(ServiceMap smap, Intent service, ServiceRecord r,  
 boolean callerFg, boolean addToStarting) throws TransactionTooLargeException {  
 ServiceState stracker = r.getTracker();  
 if (stracker != null) {  
 stracker.setStarted(true, mAm.mProcessStats.getMemFactorLocked(), r.lastActivity);  
 }  
 r.callStart = false;  
 synchronized (r.stats.getBatteryStats()) {  
 r.stats.startRunningLocked();  
 }  
 String error = **bringUpServiceLocked**(r, service.getFlags(), callerFg, false, false);  
 if (error != null) {  
 return new ComponentName("!!", error);  
 }  
  
 if (r.startRequested && addToStarting) {  
 boolean first = smap.mStartingBackground.size() == 0;  
 smap.mStartingBackground.add(r);  
 r.startingBgTimeout = SystemClock.*uptimeMillis*() + *BG\_START\_TIMEOUT*;  
 if (*DEBUG\_DELAYED\_SERVICE*) {  
 RuntimeException here = new RuntimeException("here");  
 here.fillInStackTrace();  
 Slog.v(*TAG\_SERVICE*, "Starting background (first=" + first + "): " + r, here);  
 } else if (*DEBUG\_DELAYED\_STARTS*) {  
 Slog.v(*TAG\_SERVICE*, "Starting background (first=" + first + "): " + r);  
 }  
 if (first) {  
 smap.rescheduleDelayedStarts();  
 }  
 } else if (callerFg) {  
 smap.ensureNotStartingBackground(r);  
 }  
  
 return r.name;  
}

### 5.3 bringUpServiceLocked调用realStartServiceLocked

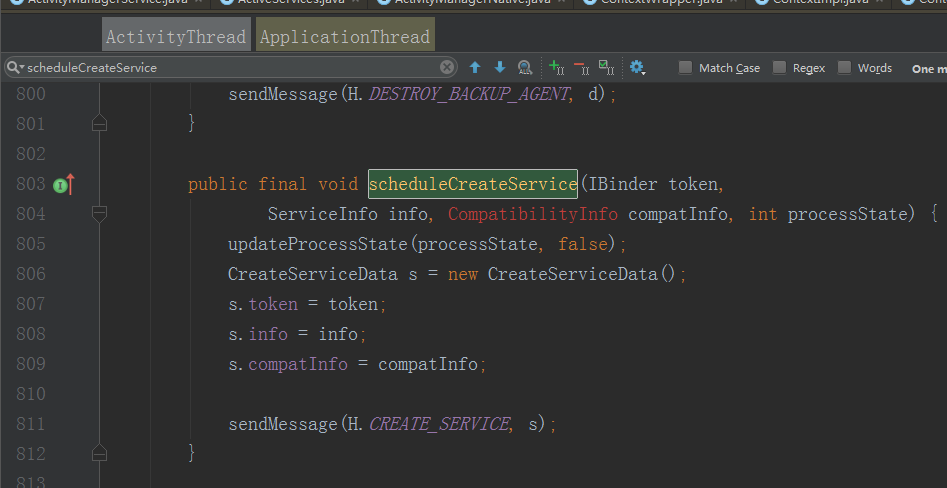
private String bringUpServiceLocked(ServiceRecord r, int intentFlags, boolean execInFg,  
 boolean whileRestarting, boolean permissionsReviewRequired)  
 throws TransactionTooLargeException {  
 //Slog.i(TAG, "Bring up service:");  
 //r.dump(" ");  
  
 if (r.app != null && r.app.thread != null) {  
 sendServiceArgsLocked(r, execInFg, false);  
 return null;  
 }  
  
 if (!whileRestarting && r.restartDelay > 0) {  
 // If waiting for a restart, then do nothing.  
 return null;  
 }  
  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "Bringing up " + r + " " + r.intent);  
  
 // We are now bringing the service up, so no longer in the  
 // restarting state.  
 if (mRestartingServices.remove(r)) {  
 r.resetRestartCounter();  
 clearRestartingIfNeededLocked(r);  
 }  
  
 // Make sure this service is no longer considered delayed, we are starting it now.  
 if (r.delayed) {  
 if (*DEBUG\_DELAYED\_STARTS*) Slog.v(*TAG\_SERVICE*, "REM FR DELAY LIST (bring up): " + r);  
 getServiceMap(r.userId).mDelayedStartList.remove(r);  
 r.delayed = false;  
 }  
  
 // Make sure that the user who owns this service is started. If not,  
 // we don't want to allow it to run.  
 if (!mAm.mUserController.hasStartedUserState(r.userId)) {  
 String msg = "Unable to launch app "  
 + r.appInfo.packageName + "/"  
 + r.appInfo.uid + " for service "  
 + r.intent.getIntent() + ": user " + r.userId + " is stopped";  
 Slog.w(*TAG*, msg);  
 bringDownServiceLocked(r);  
 return msg;  
 }  
  
 // Service is now being launched, its package can't be stopped.  
 try {  
 AppGlobals.getPackageManager().setPackageStoppedState(  
 r.packageName, false, r.userId);  
 } catch (RemoteException e) {  
 } catch (IllegalArgumentException e) {  
 Slog.w(*TAG*, "Failed trying to unstop package "  
 + r.packageName + ": " + e);  
 }  
  
 final boolean isolated = (r.serviceInfo.flags&ServiceInfo.*FLAG\_ISOLATED\_PROCESS*) != 0;  
 final String procName = r.processName;  
 ProcessRecord app;  
  
 if (!isolated) {  
 app = mAm.getProcessRecordLocked(procName, r.appInfo.uid, false);  
 if (DEBUG\_MU) Slog.v(*TAG\_MU*, "bringUpServiceLocked: appInfo.uid=" + r.appInfo.uid  
 + " app=" + app);  
 if (app != null && app.thread != null) {  
 try {  
 app.addPackage(r.appInfo.packageName, r.appInfo.versionCode, mAm.mProcessStats);  
 realStartServiceLocked(r, app, execInFg);  
 return null;  
 } catch (TransactionTooLargeException e) {  
 throw e;  
 } catch (RemoteException e) {  
 Slog.w(*TAG*, "Exception when starting service " + r.shortName, e);  
 }  
  
 // If a dead object exception was thrown -- fall through to  
 // restart the application.  
 }  
 } else {  
 // If this service runs in an isolated process, then each time  
 // we call startProcessLocked() we will get a new isolated  
 // process, starting another process if we are currently waiting  
 // for a previous process to come up. To deal with this, we store  
 // in the service any current isolated process it is running in or  
 // waiting to have come up.  
 app = r.isolatedProc;  
 }  
  
 // Not running -- get it started, and enqueue this service record  
 // to be executed when the app comes up.  
 if (app == null && !permissionsReviewRequired) {  
 if ((app=mAm.startProcessLocked(procName, r.appInfo, true, intentFlags,  
 "service", r.name, false, isolated, false)) == null) {  
 String msg = "Unable to launch app "  
 + r.appInfo.packageName + "/"  
 + r.appInfo.uid + " for service "  
 + r.intent.getIntent() + ": process is bad";  
 Slog.w(*TAG*, msg);  
 bringDownServiceLocked(r);  
 return msg;  
 }  
 if (isolated) {  
 r.isolatedProc = app;  
 }  
 }  
  
 if (!mPendingServices.contains(r)) {  
 mPendingServices.add(r);  
 }  
  
 if (r.delayedStop) {  
 // Oh and hey we've already been asked to stop!  
 r.delayedStop = false;  
 if (r.startRequested) {  
 if (*DEBUG\_DELAYED\_STARTS*) Slog.v(*TAG\_SERVICE*,  
 "Applying delayed stop (in bring up): " + r);  
 stopServiceLocked(r);  
 }  
 }  
  
 return null;  
}

### 5.4 realStartServiceLocked调用app.thread.scheduleCreateService

private final void realStartServiceLocked(ServiceRecord r,  
 ProcessRecord app, boolean execInFg) throws RemoteException {  
 if (app.thread == null) {  
 throw new RemoteException();  
 }  
 if (DEBUG\_MU)  
 Slog.v(*TAG\_MU*, "realStartServiceLocked, ServiceRecord.uid = " + r.appInfo.uid  
 + ", ProcessRecord.uid = " + app.uid);  
 r.app = app;  
 r.restartTime = r.lastActivity = SystemClock.*uptimeMillis*();  
  
 final boolean newService = app.services.add(r);  
 bumpServiceExecutingLocked(r, execInFg, "create");  
 mAm.updateLruProcessLocked(app, false, null);  
 mAm.updateOomAdjLocked();  
  
 boolean created = false;  
 try {  
 if (*LOG\_SERVICE\_START\_STOP*) {  
 String nameTerm;  
 int lastPeriod = r.shortName.lastIndexOf('.');  
 nameTerm = lastPeriod >= 0 ? r.shortName.substring(lastPeriod) : r.shortName;  
 EventLogTags.writeAmCreateService(  
 r.userId, System.*identityHashCode*(r), nameTerm, r.app.uid, r.app.pid);  
 }  
 synchronized (r.stats.getBatteryStats()) {  
 r.stats.startLaunchedLocked();  
 }  
 mAm.notifyPackageUse(r.serviceInfo.packageName,  
 PackageManager.NOTIFY\_PACKAGE\_USE\_SERVICE);  
 app.forceProcessStateUpTo(ActivityManager.PROCESS\_STATE\_SERVICE);  
  **app.thread.scheduleCreateService**(r, r.serviceInfo,  
 mAm.compatibilityInfoForPackageLocked(r.serviceInfo.applicationInfo),  
 app.repProcState);  
 r.postNotification();  
 created = true;  
 } catch (DeadObjectException e) {  
 Slog.w(*TAG*, "Application dead when creating service " + r);  
 mAm.appDiedLocked(app);  
 throw e;  
 } finally {  
 if (!created) {  
 // Keep the executeNesting count accurate.  
 final boolean inDestroying = mDestroyingServices.contains(r);  
 serviceDoneExecutingLocked(r, inDestroying, inDestroying);  
  
 // Cleanup.  
 if (newService) {  
 app.services.remove(r);  
 r.app = null;  
 }  
  
 // Retry.  
 if (!inDestroying) {  
 scheduleServiceRestartLocked(r, false);  
 }  
 }  
 }  
  
 if (r.whitelistManager) {  
 app.whitelistManager = true;  
 }  
  
 requestServiceBindingsLocked(r, execInFg);  
  
 updateServiceClientActivitiesLocked(app, null, true);  
  
 // If the service is in the started state, and there are no  
 // pending arguments, then fake up one so its onStartCommand() will  
 // be called.  
 if (r.startRequested && r.callStart && r.pendingStarts.size() == 0) {  
 r.pendingStarts.add(new ServiceRecord.StartItem(r, false, r.makeNextStartId(),  
 null, null));  
 }  
  
 sendServiceArgsLocked(r, execInFg, true);  
  
 if (r.delayed) {  
 if (*DEBUG\_DELAYED\_STARTS*) Slog.v(*TAG\_SERVICE*, "REM FR DELAY LIST (new proc): " + r);  
 getServiceMap(r.userId).mDelayedStartList.remove(r);  
 r.delayed = false;  
 }  
  
 if (r.delayedStop) {  
 // Oh and hey we've already been asked to stop!  
 r.delayedStop = false;  
 if (r.startRequested) {  
 if (*DEBUG\_DELAYED\_STARTS*) Slog.v(*TAG\_SERVICE*,  
 "Applying delayed stop (from start): " + r);  
 stopServiceLocked(r);  
 }  
 }  
}

## 6、调用ActivityThread内部类 ApplicationThread的scheduleCreateService方法

ApplicationThread的方法均运行在binder线程池中，需要切换。



## 7、调用Handler H类

case *CREATE\_SERVICE*:  
 Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, ("serviceCreate: " + String.*valueOf*(msg.obj)));  
 handleCreateService((CreateServiceData)msg.obj);  
 Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);  
 break;



### 7.1创建service并启动

private void handleCreateService(CreateServiceData data) {  
 // If we are getting ready to gc after going to the background, well  
 // we are back active so skip it.  
 unscheduleGcIdler();  
  
 LoadedApk packageInfo = getPackageInfoNoCheck(  
 data.info.applicationInfo, data.compatInfo);  
 Service service = null;  
 try {  
 java.lang.ClassLoader cl = packageInfo.getClassLoader();  
 service = (Service) cl.loadClass(data.info.name).newInstance();  
 } catch (Exception e) {  
 if (!mInstrumentation.onException(service, e)) {  
 throw new RuntimeException(  
 "Unable to instantiate service " + data.info.name  
 + ": " + e.toString(), e);  
 }  
 }  
  
 try {  
 if (*localLOGV*) Slog.v(*TAG*, "Creating service " + data.info.name);  
  
 ContextImpl context = ContextImpl.*createAppContext*(this, packageInfo);  
 context.setOuterContext(service);  
  
 Application app = packageInfo.makeApplication(false, mInstrumentation);  
 service.attach(context, this, data.info.name, data.token, app,  
 ActivityManagerNative.*getDefault*());  
 service.onCreate();  
 mServices.put(data.token, service);  
 try {  
 ActivityManagerNative.*getDefault*().serviceDoneExecuting(  
 data.token, *SERVICE\_DONE\_EXECUTING\_ANON*, 0, 0);  
 } catch (RemoteException e) {  
 throw e.rethrowFromSystemServer();  
 }  
 } catch (Exception e) {  
 if (!mInstrumentation.onException(service, e)) {  
 throw new RuntimeException(  
 "Unable to create service " + data.info.name  
 + ": " + e.toString(), e);  
 }  
 }  
}

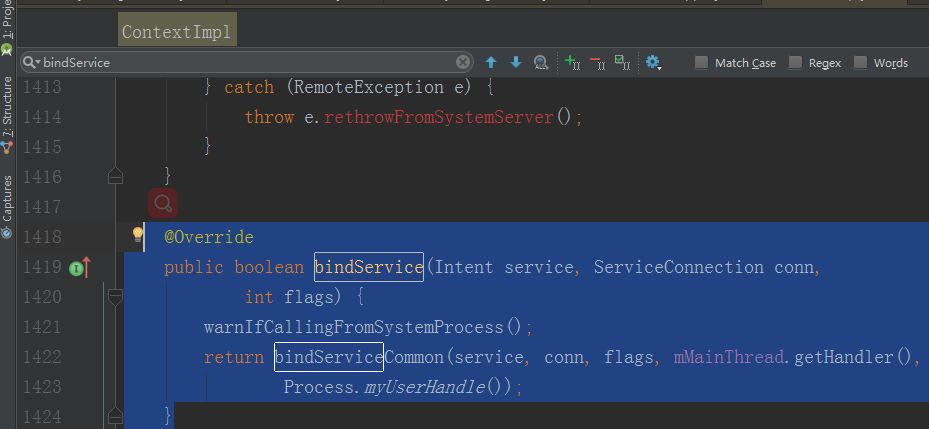
**ActivityManagerNative.*getDefault*()=AMS**

# ========BindService启动==============

## 1、调用ContextWrapper的bindService方法

@Override  
public boolean bindService(Intent service, ServiceConnection conn,  
 int flags) {  
 return mBase.bindService(service, conn, flags);  
}

## 2、调用内部类ContextImpl 的bindService方法



private boolean bindServiceCommon(Intent service, ServiceConnection conn, int flags, Handler  
 handler, UserHandle user) {  
 IServiceConnection sd;  
 if (conn == null) {  
 throw new IllegalArgumentException("connection is null");  
 }  
 if (mPackageInfo != null) {  
 sd = mPackageInfo.getServiceDispatcher(conn, getOuterContext(), handler, flags);  
 } else {  
 throw new RuntimeException("Not supported in system context");  
 }  
 validateServiceIntent(service);  
 try {  
 IBinder token = getActivityToken();  
 if (token == null && (flags&*BIND\_AUTO\_CREATE*) == 0 && mPackageInfo != null  
 && mPackageInfo.getApplicationInfo().targetSdkVersion  
 < android.os.Build.VERSION\_CODES.*ICE\_CREAM\_SANDWICH*) {  
 flags |= *BIND\_WAIVE\_PRIORITY*;  
 }  
 service.prepareToLeaveProcess(this);  
 int res = **ActivityManagerNative.*getDefault*().****bindService**(  
 mMainThread.getApplicationThread(), getActivityToken(), service,  
 service.resolveTypeIfNeeded(getContentResolver()),  
 sd, flags, getOpPackageName(), user.getIdentifier());  
 if (res < 0) {  
 throw new SecurityException(  
 "Not allowed to bind to service " + service);  
 }  
 return res != 0;  
 } catch (RemoteException e) {  
 throw e.rethrowFromSystemServer();  
 }  
}

## 3、AMS的bindService方法调用ActiveService类的bindServiceLocked方法（ActiveService类中）

public int bindService(IApplicationThread caller, IBinder token, Intent service,  
 String resolvedType, IServiceConnection connection, int flags, String callingPackage,  
 int userId) throws TransactionTooLargeException {  
 enforceNotIsolatedCaller("bindService");  
  
 // Refuse possible leaked file descriptors  
 if (service != null && service.hasFileDescriptors() == true) {  
 throw new IllegalArgumentException("File descriptors passed in Intent");  
 }  
  
 if (callingPackage == null) {  
 throw new IllegalArgumentException("callingPackage cannot be null");  
 }  
  
 synchronized(this) {  
 return mServices.**bindServiceLocked**(caller, token, service,  
 resolvedType, connection, flags, callingPackage, userId);  
 }  
}

### 3.1 bindServiceLocked方法调用bringUpServiceLocked

int bindServiceLocked(IApplicationThread caller, IBinder token, Intent service,  
 String resolvedType, final IServiceConnection connection, int flags,  
 String callingPackage, final int userId) throws TransactionTooLargeException {  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "bindService: " + service  
 + " type=" + resolvedType + " conn=" + connection.asBinder()  
 + " flags=0x" + Integer.*toHexString*(flags));  
 final ProcessRecord callerApp = mAm.getRecordForAppLocked(caller);  
 if (callerApp == null) {  
 throw new SecurityException(  
 "Unable to find app for caller " + caller  
 + " (pid=" + Binder.*getCallingPid*()  
 + ") when binding service " + service);  
 }  
  
 ActivityRecord activity = null;  
 if (token != null) {  
 activity = ActivityRecord.*isInStackLocked*(token);  
 if (activity == null) {  
 Slog.w(*TAG*, "Binding with unknown activity: " + token);  
 return 0;  
 }  
 }  
  
 int clientLabel = 0;  
 PendingIntent clientIntent = null;  
 final boolean isCallerSystem = callerApp.info.uid == Process.*SYSTEM\_UID*;  
  
 if (isCallerSystem) {  
 // Hacky kind of thing -- allow system stuff to tell us  
 // what they are, so we can report this elsewhere for  
 // others to know why certain services are running.  
 service.setDefusable(true);  
 clientIntent = service.getParcelableExtra(Intent.EXTRA\_CLIENT\_INTENT);  
 if (clientIntent != null) {  
 clientLabel = service.getIntExtra(Intent.EXTRA\_CLIENT\_LABEL, 0);  
 if (clientLabel != 0) {  
 // There are no useful extras in the intent, trash them.  
 // System code calling with this stuff just needs to know  
 // this will happen.  
 service = service.cloneFilter();  
 }  
 }  
 }  
  
 if ((flags&Context.BIND\_TREAT\_LIKE\_ACTIVITY) != 0) {  
 mAm.enforceCallingPermission(android.Manifest.permission.MANAGE\_ACTIVITY\_STACKS,  
 "BIND\_TREAT\_LIKE\_ACTIVITY");  
 }  
  
 if ((flags & Context.BIND\_ALLOW\_WHITELIST\_MANAGEMENT) != 0 && !isCallerSystem) {  
 throw new SecurityException(  
 "Non-system caller " + caller + " (pid=" + Binder.*getCallingPid*()  
 + ") set BIND\_ALLOW\_WHITELIST\_MANAGEMENT when binding service " + service);  
 }  
  
 final boolean callerFg = callerApp.setSchedGroup != ProcessList.*SCHED\_GROUP\_BACKGROUND*;  
 final boolean isBindExternal = (flags & Context.*BIND\_EXTERNAL\_SERVICE*) != 0;  
  
 ServiceLookupResult res =  
 retrieveServiceLocked(service, resolvedType, callingPackage, Binder.*getCallingPid*(),  
 Binder.*getCallingUid*(), userId, true, callerFg, isBindExternal);  
 if (res == null) {  
 return 0;  
 }  
 if (res.record == null) {  
 return -1;  
 }  
 ServiceRecord s = res.record;  
  
 boolean permissionsReviewRequired = false;  
  
 // If permissions need a review before any of the app components can run,  
 // we schedule binding to the service but do not start its process, then  
 // we launch a review activity to which is passed a callback to invoke  
 // when done to start the bound service's process to completing the binding.  
 if (Build.PERMISSIONS\_REVIEW\_REQUIRED) {  
 if (mAm.getPackageManagerInternalLocked().isPermissionsReviewRequired(  
 s.packageName, s.userId)) {  
  
 permissionsReviewRequired = true;  
  
 // Show a permission review UI only for binding from a foreground app  
 if (!callerFg) {  
 Slog.w(*TAG*, "u" + s.userId + " Binding to a service in package"  
 + s.packageName + " requires a permissions review");  
 return 0;  
 }  
  
 final ServiceRecord serviceRecord = s;  
 final Intent serviceIntent = service;  
  
 RemoteCallback callback = new RemoteCallback(  
 new RemoteCallback.OnResultListener() {  
 @Override  
 public void onResult(Bundle result) {  
 synchronized(mAm) {  
 final long identity = Binder.*clearCallingIdentity*();  
 try {  
 if (!mPendingServices.contains(serviceRecord)) {  
 return;  
 }  
 // If there is still a pending record, then the service  
 // binding request is still valid, so hook them up. We  
 // proceed only if the caller cleared the review requirement  
 // otherwise we unbind because the user didn't approve.  
 if (!mAm.getPackageManagerInternalLocked()  
 .isPermissionsReviewRequired(  
 serviceRecord.packageName,  
 serviceRecord.userId)) {  
 try {  
 **bringUpServiceLocked**(serviceRecord,  
 serviceIntent.getFlags(),  
 callerFg, false, false);  
 } catch (RemoteException e) {  
 /\* ignore - local call \*/  
 }  
 } else {  
 unbindServiceLocked(connection);  
 }  
 } finally {  
 Binder.*restoreCallingIdentity*(identity);  
 }  
 }  
 }  
 });  
  
 final Intent intent = new Intent(Intent.ACTION\_REVIEW\_PERMISSIONS);  
 intent.addFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK* | Intent.*FLAG\_ACTIVITY\_EXCLUDE\_FROM\_RECENTS*);  
 intent.putExtra(Intent.*EXTRA\_PACKAGE\_NAME*, s.packageName);  
 intent.putExtra(Intent.EXTRA\_REMOTE\_CALLBACK, callback);  
  
 if (DEBUG\_PERMISSIONS\_REVIEW) {  
 Slog.i(*TAG*, "u" + s.userId + " Launching permission review for package "  
 + s.packageName);  
 }  
  
 mAm.mHandler.post(new Runnable() {  
 @Override  
 public void run() {  
 mAm.mContext.startActivityAsUser(intent, new UserHandle(userId));  
 }  
 });  
 }  
 }  
  
 final long origId = Binder.*clearCallingIdentity*();  
  
 try {  
 if (unscheduleServiceRestartLocked(s, callerApp.info.uid, false)) {  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "BIND SERVICE WHILE RESTART PENDING: "  
 + s);  
 }  
  
 if ((flags&Context.*BIND\_AUTO\_CREATE*) != 0) {  
 s.lastActivity = SystemClock.*uptimeMillis*();  
 if (!s.hasAutoCreateConnections()) {  
 // This is the first binding, let the tracker know.  
 ServiceState stracker = s.getTracker();  
 if (stracker != null) {  
 stracker.setBound(true, mAm.mProcessStats.getMemFactorLocked(),  
 s.lastActivity);  
 }  
 }  
 }  
  
 mAm.startAssociationLocked(callerApp.uid, callerApp.processName, callerApp.curProcState,  
 s.appInfo.uid, s.name, s.processName);  
  
 AppBindRecord b = s.retrieveAppBindingLocked(service, callerApp);  
 ConnectionRecord c = new ConnectionRecord(b, activity,  
 connection, flags, clientLabel, clientIntent);  
  
 IBinder binder = connection.asBinder();  
 ArrayList<ConnectionRecord> clist = s.connections.get(binder);  
 if (clist == null) {  
 clist = new ArrayList<ConnectionRecord>();  
 s.connections.put(binder, clist);  
 }  
 clist.add(c);  
 b.connections.add(c);  
 if (activity != null) {  
 if (activity.connections == null) {  
 activity.connections = new HashSet<ConnectionRecord>();  
 }  
 activity.connections.add(c);  
 }  
 b.client.connections.add(c);  
 if ((c.flags&Context.*BIND\_ABOVE\_CLIENT*) != 0) {  
 b.client.hasAboveClient = true;  
 }  
 if ((c.flags&Context.BIND\_ALLOW\_WHITELIST\_MANAGEMENT) != 0) {  
 s.whitelistManager = true;  
 }  
 if (s.app != null) {  
 updateServiceClientActivitiesLocked(s.app, c, true);  
 }  
 clist = mServiceConnections.get(binder);  
 if (clist == null) {  
 clist = new ArrayList<ConnectionRecord>();  
 mServiceConnections.put(binder, clist);  
 }  
 clist.add(c);  
  
 if ((flags&Context.*BIND\_AUTO\_CREATE*) != 0) {  
 s.lastActivity = SystemClock.*uptimeMillis*();  
 if (bringUpServiceLocked(s, service.getFlags(), callerFg, false,  
 permissionsReviewRequired) != null) {  
 return 0;  
 }  
 }  
  
 if (s.app != null) {  
 if ((flags&Context.BIND\_TREAT\_LIKE\_ACTIVITY) != 0) {  
 s.app.treatLikeActivity = true;  
 }  
 if (s.whitelistManager) {  
 s.app.whitelistManager = true;  
 }  
 // This could have made the service more important.  
 mAm.updateLruProcessLocked(s.app, s.app.hasClientActivities  
 || s.app.treatLikeActivity, b.client);  
 mAm.updateOomAdjLocked(s.app);  
 }  
  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "Bind " + s + " with " + b  
 + ": received=" + b.intent.received  
 + " apps=" + b.intent.apps.size()  
 + " doRebind=" + b.intent.doRebind);  
  
 if (s.app != null && b.intent.received) {  
 // Service is already running, so we can immediately  
 // publish the connection.  
 try {  
 c.conn.connected(s.name, b.intent.binder);  
 } catch (Exception e) {  
 Slog.w(*TAG*, "Failure sending service " + s.shortName  
 + " to connection " + c.conn.asBinder()  
 + " (in " + c.binding.client.processName + ")", e);  
 }  
  
 // If this is the first app connected back to this binding,  
 // and the service had previously asked to be told when  
 // rebound, then do so.  
 if (b.intent.apps.size() == 1 && b.intent.doRebind) {  
  **requestServiceBindingLocked**(s, b.intent, callerFg, true);  
 }  
 } else if (!b.intent.requested) {  
 requestServiceBindingLocked(s, b.intent, callerFg, false);  
 }  
  
 getServiceMap(s.userId).ensureNotStartingBackground(s);  
  
 } finally {  
 Binder.*restoreCallingIdentity*(origId);  
 }  
  
 return 1;  
}

### 3.2bringUpServiceLocked调用requestServiceBindingLocked

private final boolean requestServiceBindingLocked(ServiceRecord r, IntentBindRecord i,  
 boolean execInFg, boolean rebind) throws TransactionTooLargeException {  
 if (r.app == null || r.app.thread == null) {  
 // If service is not currently running, can't yet bind.  
 return false;  
 }  
 if ((!i.requested || rebind) && i.apps.size() > 0) {  
 try {  
 bumpServiceExecutingLocked(r, execInFg, "bind");  
 r.app.forceProcessStateUpTo(ActivityManager.PROCESS\_STATE\_SERVICE);  
  **r.app.thread.scheduleBindService**(r, i.intent.getIntent(), rebind,  
 r.app.repProcState);  
 if (!rebind) {  
 i.requested = true;  
 }  
 i.hasBound = true;  
 i.doRebind = false;  
 } catch (TransactionTooLargeException e) {  
 // Keep the executeNesting count accurate.  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "Crashed while binding " + r, e);  
 final boolean inDestroying = mDestroyingServices.contains(r);  
 serviceDoneExecutingLocked(r, inDestroying, inDestroying);  
 throw e;  
 } catch (RemoteException e) {  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "Crashed while binding " + r);  
 // Keep the executeNesting count accurate.  
 final boolean inDestroying = mDestroyingServices.contains(r);  
 serviceDoneExecutingLocked(r, inDestroying, inDestroying);  
 return false;  
 }  
 }  
 return true;  
}

### 3.3 requestServiceBindingLocked ---> r.app.thread.scheduleBindService

### 3.4 ActivityThread的scheduleBindService

public final void scheduleBindService(IBinder token, Intent intent,  
 boolean rebind, int processState) {  
 updateProcessState(processState, false);  
 BindServiceData s = new BindServiceData();  
 s.token = token;  
 s.intent = intent;  
 s.rebind = rebind;  
  
 if (*DEBUG\_SERVICE*)  
 Slog.v(*TAG*, "scheduleBindService token=" + token + " intent=" + intent + " uid="  
 + Binder.*getCallingUid*() + " pid=" + Binder.*getCallingPid*());  
 sendMessage(H.*BIND\_SERVICE*, s);  
}

### 3.5 handleBindService((BindServiceData)msg.obj);

private void handleBindService(BindServiceData data) {  
 Service s = mServices.get(data.token);  
 if (*DEBUG\_SERVICE*)  
 Slog.v(*TAG*, "handleBindService s=" + s + " rebind=" + data.rebind);  
 if (s != null) {  
 try {  
 data.intent.setExtrasClassLoader(s.getClassLoader());  
 data.intent.prepareToEnterProcess();  
 try {  
 if (!data.rebind) {  
 IBinder binder = s.onBind(data.intent);  
 ActivityManagerNative.*getDefault*().publishService(  
 data.token, data.intent, binder);  
 } else {  
 s.onRebind(data.intent);  
 ActivityManagerNative.*getDefault*().serviceDoneExecuting(  
 data.token, *SERVICE\_DONE\_EXECUTING\_ANON*, 0, 0);  
 }  
 ensureJitEnabled();  
 } catch (RemoteException ex) {  
 throw ex.rethrowFromSystemServer();  
 }  
 } catch (Exception e) {  
 if (!mInstrumentation.onException(s, e)) {  
 throw new RuntimeException(  
 "Unable to bind to service " + s  
 + " with " + data.intent + ": " + e.toString(), e);  
 }  
 }  
 }  
}

### 3.6调用AMS的pushService方法，pushService调用ActiveService的pushServiceLocked方法。

public void publishService(IBinder token, Intent intent, IBinder service) {  
 // Refuse possible leaked file descriptors  
 if (intent != null && intent.hasFileDescriptors() == true) {  
 throw new IllegalArgumentException("File descriptors passed in Intent");  
 }  
  
 synchronized(this) {  
 if (!(token instanceof ServiceRecord)) {  
 throw new IllegalArgumentException("Invalid service token");  
 }  
 mServices.**publishServiceLocked**((ServiceRecord)token, intent, service);  
 }  
}

### 3.7 ActiveService的publishServiceLocked 调用c.conn.connected方法

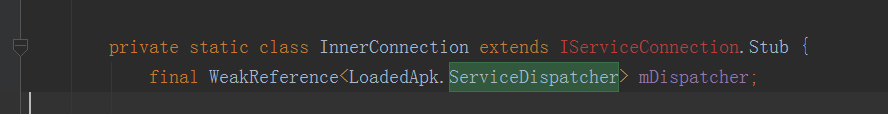
void publishServiceLocked(ServiceRecord r, Intent intent, IBinder service) {  
 final long origId = Binder.*clearCallingIdentity*();  
 try {  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "PUBLISHING " + r  
 + " " + intent + ": " + service);  
 if (r != null) {  
 Intent.FilterComparison filter  
 = new Intent.FilterComparison(intent);  
 IntentBindRecord b = r.bindings.get(filter);  
 if (b != null && !b.received) {  
 b.binder = service;  
 b.requested = true;  
 b.received = true;  
 for (int conni=r.connections.size()-1; conni>=0; conni--) {  
 ArrayList<ConnectionRecord> clist = r.connections.valueAt(conni);  
 for (int i=0; i<clist.size(); i++) {  
 ConnectionRecord c = clist.get(i);  
 if (!filter.equals(c.binding.intent.intent)) {  
 if (DEBUG\_SERVICE) Slog.v(  
 *TAG\_SERVICE*, "Not publishing to: " + c);  
 if (DEBUG\_SERVICE) Slog.v(  
 *TAG\_SERVICE*, "Bound intent: " + c.binding.intent.intent);  
 if (DEBUG\_SERVICE) Slog.v(  
 *TAG\_SERVICE*, "Published intent: " + intent);  
 continue;  
 }  
 if (DEBUG\_SERVICE) Slog.v(*TAG\_SERVICE*, "Publishing to: " + c);  
 try {  
  **c.conn.connected**(r.name, service);  
 } catch (Exception e) {  
 Slog.w(*TAG*, "Failure sending service " + r.name +  
 " to connection " + c.conn.asBinder() +  
 " (in " + c.binding.client.processName + ")", e);  
 }  
 }  
 }  
 }  
  
 serviceDoneExecutingLocked(r, mDestroyingServices.contains(r), false);  
 }  
 } finally {  
 Binder.*restoreCallingIdentity*(origId);  
 }  
}

### 3.8 c.conn类结构说明



LoadAPK中的内部类static final class ServiceDispatcher {

ServiceDispatcher中的内部类InnerConnection 实现该IServiceConnection.Stub aidl接口。

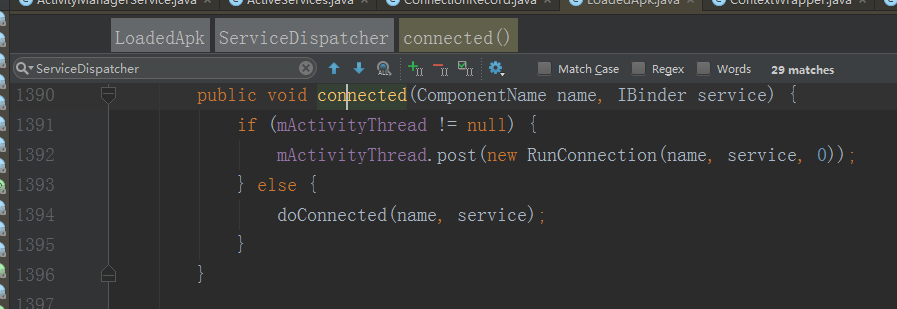


### 3.9 InnerConnection 实现该c.conn.connected(r.name, service);

### 连接

private static class InnerConnection extends IServiceConnection.Stub {  
 final WeakReference<LoadedApk.ServiceDispatcher> mDispatcher;  
  
 InnerConnection(LoadedApk.ServiceDispatcher sd) {  
 mDispatcher = new WeakReference<LoadedApk.ServiceDispatcher>(sd);  
 }  
  
 public void **connected**(ComponentName name, IBinder service) throws RemoteException {  
 LoadedApk.ServiceDispatcher sd = mDispatcher.get();  
 if (sd != null) {  
 sd.**connected**(name, service);  
 }  
 }  
}

### 3.10 调用ServiceDispatcher的connected方法



private final class RunConnection implements Runnable {  
 RunConnection(ComponentName name, IBinder service, int command) {  
 mName = name;  
 mService = service;  
 mCommand = command;  
 }  
  
 public void run() {  
 if (mCommand == 0) {  
 doConnected(mName, mService);  
 } else if (mCommand == 1) {  
 doDeath(mName, mService);  
 }  
 }  
  
 final ComponentName mName;  
 final IBinder mService;  
 final int mCommand;  
}

public void doConnected(ComponentName name, IBinder service) {  
 ServiceDispatcher.ConnectionInfo old;  
 ServiceDispatcher.ConnectionInfo info;  
  
 synchronized (this) {  
 if (mForgotten) {  
 // We unbound before receiving the connection; ignore  
 // any connection received.  
 return;  
 }  
 old = mActiveConnections.get(name);  
 if (old != null && old.binder == service) {  
 // Huh, already have this one. Oh well!  
 return;  
 }  
  
 if (service != null) {  
 // A new service is being connected... set it all up.  
 info = new ConnectionInfo();  
 info.binder = service;  
 info.deathMonitor = new DeathMonitor(name, service);  
 try {  
 service.linkToDeath(info.deathMonitor, 0);  
 mActiveConnections.put(name, info);  
 } catch (RemoteException e) {  
 // This service was dead before we got it... just  
 // don't do anything with it.  
 mActiveConnections.remove(name);  
 return;  
 }  
  
 } else {  
 // The named service is being disconnected... clean up.  
 mActiveConnections.remove(name);  
 }  
  
 if (old != null) {  
 old.binder.unlinkToDeath(old.deathMonitor, 0);  
 }  
 }  
  
 // If there was an old service, it is now disconnected.  
 if (old != null) {  
 mConnection.onServiceDisconnected(name);  
 }  
 // If there is a new service, it is now connected.  
 if (service != null) {  
 mConnection.onServiceConnected(name, service);  
 }  
}  
  
public void doDeath(ComponentName name, IBinder service) {  
 synchronized (this) {  
 ConnectionInfo old = mActiveConnections.get(name);  
 if (old == null || old.binder != service) {  
 // Death for someone different than who we last  
 // reported... just ignore it.  
 return;  
 }  
 mActiveConnections.remove(name);  
 old.binder.unlinkToDeath(old.deathMonitor, 0);  
 }  
  
 mConnection.onServiceDisconnected(name);  
}

public void doConnected(ComponentName name, IBinder service) {  
 ServiceDispatcher.ConnectionInfo old;  
 ServiceDispatcher.ConnectionInfo info;  
  
 synchronized (this) {  
 if (mForgotten) {  
 // We unbound before receiving the connection; ignore  
 // any connection received.  
 return;  
 }  
 old = mActiveConnections.get(name);  
 if (old != null && old.binder == service) {  
 // Huh, already have this one. Oh well!  
 return;  
 }  
  
 if (service != null) {  
 // A new service is being connected... set it all up.  
 info = new ConnectionInfo();  
 info.binder = service;  
 info.deathMonitor = new DeathMonitor(name, service);  
 try {  
 service.linkToDeath(info.deathMonitor, 0);  
 mActiveConnections.put(name, info);  
 } catch (RemoteException e) {  
 // This service was dead before we got it... just  
 // don't do anything with it.  
 mActiveConnections.remove(name);  
 return;  
 }  
  
 } else {  
 // The named service is being disconnected... clean up.  
 mActiveConnections.remove(name);  
 }  
  
 if (old != null) {  
 old.binder.unlinkToDeath(old.deathMonitor, 0);  
 }  
 }  
  
 // If there was an old service, it is now disconnected.  
 if (old != null) {  
 mConnection.onServiceDisconnected(name);  
 }  
 // If there is a new service, it is now connected.  
 if (service != null) {  
 mConnection.onServiceConnected(name, service);  
 }  
}