# BroadcastReceiver工作原理

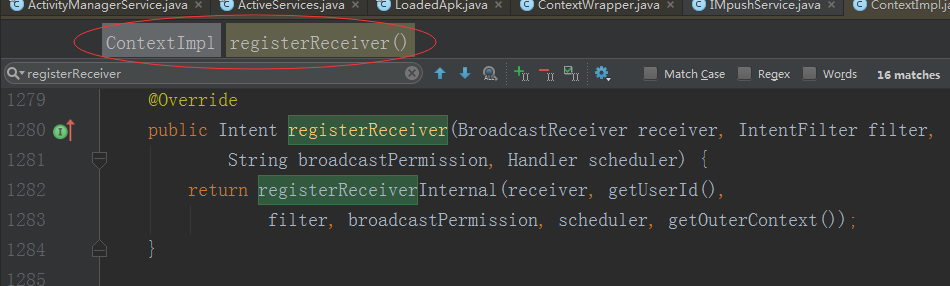
## 1、广播注册

// 注册广播  
registerReceiver(mReceiver, filter);

### 1.1本质调用ContextWrapper的registerReceiver方法

@Override  
public Intent registerReceiver(  
 BroadcastReceiver receiver, IntentFilter filter) {  
 return mBase.registerReceiver(receiver, filter);  
}

### 1.2调用ContextImpl的registerReceiver方法



### 1.3 调用AMS registerReceiver方法

private Intent registerReceiverInternal(BroadcastReceiver receiver, int userId,  
 IntentFilter filter, String broadcastPermission,  
 Handler scheduler, Context context) {  
 IIntentReceiver rd = null;  
 if (receiver != null) {  
 if (mPackageInfo != null && context != null) {  
 if (scheduler == null) {  
 scheduler = mMainThread.getHandler();  
 }  
 rd = mPackageInfo.getReceiverDispatcher(  
 receiver, context, scheduler,  
 mMainThread.getInstrumentation(), true);  
 } else {  
 if (scheduler == null) {  
 scheduler = mMainThread.getHandler();  
 }  
 rd = new LoadedApk.ReceiverDispatcher(  
 receiver, context, scheduler, null, true).getIIntentReceiver();  
 }  
 }  
 try {  
 final Intent intent = ActivityManagerNative.*getDefault*().registerReceiver(  
 mMainThread.getApplicationThread(), mBasePackageName,  
 rd, filter, broadcastPermission, userId);  
 if (intent != null) {  
 intent.setExtrasClassLoader(getClassLoader());  
 intent.prepareToEnterProcess();  
 }  
 return intent;  
 } catch (RemoteException e) {  
 throw e.rethrowFromSystemServer();  
 }  
}

ams的registReceiver方法

public Intent registerReceiver(IApplicationThread caller, String callerPackage,  
 IIntentReceiver receiver, IntentFilter filter, String permission, int userId) {  
 enforceNotIsolatedCaller("registerReceiver");  
 ArrayList<Intent> stickyIntents = null;  
 ProcessRecord callerApp = null;  
 int callingUid;  
 int callingPid;  
 synchronized(this) {  
 if (caller != null) {  
 callerApp = getRecordForAppLocked(caller);  
 if (callerApp == null) {  
 throw new SecurityException(  
 "Unable to find app for caller " + caller  
 + " (pid=" + Binder.*getCallingPid*()  
 + ") when registering receiver " + receiver);  
 }  
 if (callerApp.info.uid != Process.*SYSTEM\_UID* &&  
 !callerApp.pkgList.containsKey(callerPackage) &&  
 !"android".equals(callerPackage)) {  
 throw new SecurityException("Given caller package " + callerPackage  
 + " is not running in process " + callerApp);  
 }  
 callingUid = callerApp.info.uid;  
 callingPid = callerApp.pid;  
 } else {  
 callerPackage = null;  
 callingUid = Binder.*getCallingUid*();  
 callingPid = Binder.*getCallingPid*();  
 }  
  
 userId = mUserController.handleIncomingUser(callingPid, callingUid, userId, true,  
 *ALLOW\_FULL\_ONLY*, "registerReceiver", callerPackage);  
  
 Iterator<String> actions = filter.actionsIterator();  
 if (actions == null) {  
 ArrayList<String> noAction = new ArrayList<String>(1);  
 noAction.add(null);  
 actions = noAction.iterator();  
 }  
  
 // Collect stickies of users  
 int[] userIds = { UserHandle.USER\_ALL, UserHandle.getUserId(callingUid) };  
 while (actions.hasNext()) {  
 String action = actions.next();  
 for (int id : userIds) {  
 ArrayMap<String, ArrayList<Intent>> stickies = mStickyBroadcasts.get(id);  
 if (stickies != null) {  
 ArrayList<Intent> intents = stickies.get(action);  
 if (intents != null) {  
 if (stickyIntents == null) {  
 stickyIntents = new ArrayList<Intent>();  
 }  
 stickyIntents.addAll(intents);  
 }  
 }  
 }  
 }  
 }  
  
 ArrayList<Intent> allSticky = null;  
 if (stickyIntents != null) {  
 final ContentResolver resolver = mContext.getContentResolver();  
 // Look for any matching sticky broadcasts...  
 for (int i = 0, N = stickyIntents.size(); i < N; i++) {  
 Intent intent = stickyIntents.get(i);  
 // If intent has scheme "content", it will need to acccess  
 // provider that needs to lock mProviderMap in ActivityThread  
 // and also it may need to wait application response, so we  
 // cannot lock ActivityManagerService here.  
 if (filter.match(resolver, intent, true, *TAG*) >= 0) {  
 if (allSticky == null) {  
 allSticky = new ArrayList<Intent>();  
 }  
 allSticky.add(intent);  
 }  
 }  
 }  
  
 // The first sticky in the list is returned directly back to the client.  
 Intent sticky = allSticky != null ? allSticky.get(0) : null;  
 if (DEBUG\_BROADCAST) Slog.v(*TAG\_BROADCAST*, "Register receiver " + filter + ": " + sticky);  
 if (receiver == null) {  
 return sticky;  
 }  
  
 synchronized (this) {  
 if (callerApp != null && (callerApp.thread == null  
 || callerApp.thread.asBinder() != caller.asBinder())) {  
 // Original caller already died  
 return null;  
 }  
 ReceiverList rl = mRegisteredReceivers.get(receiver.asBinder());  
 if (rl == null) {  
 rl = new ReceiverList(this, callerApp, callingPid, callingUid,  
 userId, receiver);  
 if (rl.app != null) {  
 rl.app.receivers.add(rl);  
 } else {  
 try {  
 receiver.asBinder().linkToDeath(rl, 0);  
 } catch (RemoteException e) {  
 return sticky;  
 }  
 rl.linkedToDeath = true;  
 }  
 mRegisteredReceivers.put(receiver.asBinder(), rl);  
 } else if (rl.uid != callingUid) {  
 throw new IllegalArgumentException(  
 "Receiver requested to register for uid " + callingUid  
 + " was previously registered for uid " + rl.uid);  
 } else if (rl.pid != callingPid) {  
 throw new IllegalArgumentException(  
 "Receiver requested to register for pid " + callingPid  
 + " was previously registered for pid " + rl.pid);  
 } else if (rl.userId != userId) {  
 throw new IllegalArgumentException(  
 "Receiver requested to register for user " + userId  
 + " was previously registered for user " + rl.userId);  
 }  
 BroadcastFilter bf = new BroadcastFilter(filter, rl, callerPackage,  
 permission, callingUid, userId);  
 rl.add(bf);  
 if (!bf.debugCheck()) {  
 Slog.w(*TAG*, "==> For Dynamic broadcast");  
 }  
 mReceiverResolver.addFilter(bf);  
  
 // Enqueue broadcasts for all existing stickies that match  
 // this filter.  
 if (allSticky != null) {  
 ArrayList receivers = new ArrayList();  
 receivers.add(bf);  
  
 final int stickyCount = allSticky.size();  
 for (int i = 0; i < stickyCount; i++) {  
 Intent intent = allSticky.get(i);  
 BroadcastQueue queue = broadcastQueueForIntent(intent);  
 BroadcastRecord r = new BroadcastRecord(queue, intent, null,  
 null, -1, -1, null, null, AppOpsManager.OP\_NONE, null, receivers,  
 null, 0, null, null, false, true, true, -1);  
 queue.enqueueParallelBroadcastLocked(r);  
 queue.scheduleBroadcastsLocked();  
 }  
 }  
  
 return sticky;  
 }  
}

## 2、广播的发送

### 2.1调用ContextImpl 的sendBrocast

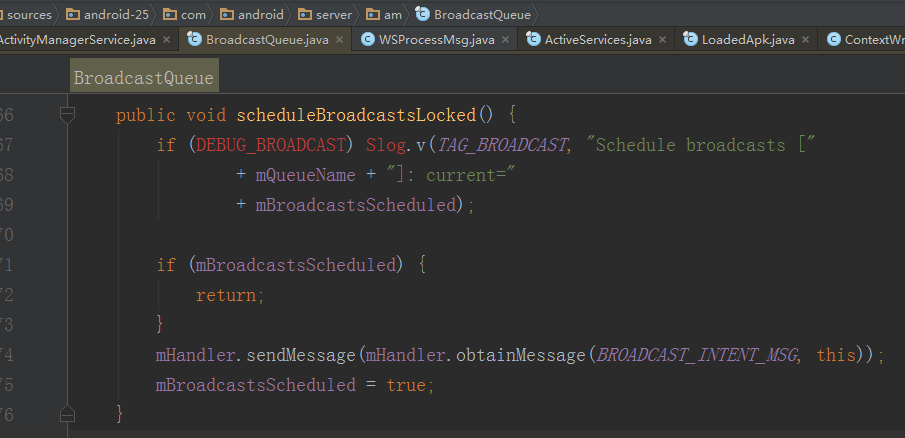
@Override  
public void sendBroadcast(Intent intent) {  
 warnIfCallingFromSystemProcess();  
 String resolvedType = intent.resolveTypeIfNeeded(getContentResolver());  
 try {  
 intent.prepareToLeaveProcess(this);  
  **ActivityManagerNative.*getDefault*().broadcastIntent**(  
 mMainThread.getApplicationThread(), intent, resolvedType, null,  
 Activity.*RESULT\_OK*, null, null, null, AppOpsManager.*OP\_NONE*, null, false, false,  
 getUserId());  
 } catch (RemoteException e) {  
 throw e.rethrowFromSystemServer();  
 }  
}

### 2.2调用AMS的broadcastIntent方法

public final int broadcastIntent(IApplicationThread caller,  
 Intent intent, String resolvedType, IIntentReceiver resultTo,  
 int resultCode, String resultData, Bundle resultExtras,  
 String[] requiredPermissions, int appOp, Bundle bOptions,  
 boolean serialized, boolean sticky, int userId) {  
 enforceNotIsolatedCaller("broadcastIntent");  
 synchronized(this) {  
 intent = verifyBroadcastLocked(intent);  
  
 final ProcessRecord callerApp = getRecordForAppLocked(caller);  
 final int callingPid = Binder.*getCallingPid*();  
 final int callingUid = Binder.*getCallingUid*();  
 final long origId = Binder.*clearCallingIdentity*();  
 int res = **broadcastIntentLocked**(callerApp,  
 callerApp != null ? callerApp.info.packageName : null,  
 intent, resolvedType, resultTo, resultCode, resultData, resultExtras,  
 requiredPermissions, appOp, bOptions, serialized, sticky,  
 callingPid, callingUid, userId);  
 Binder.*restoreCallingIdentity*(origId);  
 return res;  
 }  
}

### 2.3 broadcastIntentLocked方法中发送广播

if ((receivers != null && receivers.size() > 0)  
 || resultTo != null) {  
 BroadcastQueue queue = broadcastQueueForIntent(intent);  
 BroadcastRecord r = new BroadcastRecord(queue, intent, callerApp,  
 callerPackage, callingPid, callingUid, resolvedType,  
 requiredPermissions, appOp, brOptions, receivers, resultTo, resultCode,  
 resultData, resultExtras, ordered, sticky, false, userId);  
  
 if (DEBUG\_BROADCAST) Slog.v(*TAG\_BROADCAST*, "Enqueueing ordered broadcast " + r  
 + ": prev had " + queue.mOrderedBroadcasts.size());  
 if (DEBUG\_BROADCAST) Slog.i(*TAG\_BROADCAST*,  
 "Enqueueing broadcast " + r.intent.getAction());  
  
 boolean replaced = replacePending && queue.replaceOrderedBroadcastLocked(r);  
 if (!replaced) {  
 queue.enqueueOrderedBroadcastLocked(r);  
 queue.scheduleBroadcastsLocked();  
 }  
} else {



### 2.4 BroadcastQueue发送广播切换到主线程

final BroadcastHandler mHandler;  
  
private final class BroadcastHandler extends Handler {  
 public BroadcastHandler(Looper looper) {  
 super(looper, null, true);  
 }  
  
 @Override  
 public void handleMessage(Message msg) {  
 switch (msg.what) {  
 case *BROADCAST\_INTENT\_MSG*: {  
 if (DEBUG\_BROADCAST) Slog.v(  
 *TAG\_BROADCAST*, "Received BROADCAST\_INTENT\_MSG");  
 processNextBroadcast(true);  
 } break;  
 case *BROADCAST\_TIMEOUT\_MSG*: {  
 synchronized (mService) {  
 broadcastTimeoutLocked(true);  
 }  
 } break;  
 case *SCHEDULE\_TEMP\_WHITELIST\_MSG*: {  
 DeviceIdleController.LocalService dic = mService.mLocalDeviceIdleController;  
 if (dic != null) {  
 dic.addPowerSaveTempWhitelistAppDirect(UserHandle.getAppId(msg.arg1),  
 msg.arg2, true, (String)msg.obj);  
 }  
 } break;  
 }  
 }  
}