**1) 按键处理主要的功臣是WindowManagerService. mInputManager**

*@WindowManagerService.java*

**[java]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. mInputManager = **new** InputManager(context, **this**);

*@InputManager.java*

**[java]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. **public** InputManager(Context context, WindowManagerService windowManagerService) {
2. **this**.mContext = context;
3. **this**.mWindowManagerService = windowManagerService;
4. **this**.mCallbacks = **new** Callbacks();
6. Looper looper = windowManagerService.mH.getLooper();
8. Slog.i(TAG, "Initializing input manager");
9. nativeInit(mContext, mCallbacks, looper.getQueue());
11. // Add ourself to the Watchdog monitors.
12. Watchdog.getInstance().addMonitor(**this**);
13. }

*@com\_android\_server\_InputManager.cpp*

**[cpp]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. **static** **void** android\_server\_InputManager\_nativeInit(JNIEnv\* env, jclass clazz,
2. jobject contextObj, jobject callbacksObj, jobject messageQueueObj) {
3. **if** (gNativeInputManager == NULL) {
4. sp<Looper> looper = android\_os\_MessageQueue\_getLooper(env, messageQueueObj);
5. gNativeInputManager = **new** NativeInputManager(contextObj, callbacksObj, looper);
6. } **else** {
7. LOGE("Input manager already initialized.");
8. jniThrowRuntimeException(env, "Input manager already initialized.");
9. }
10. }

13. NativeInputManager::NativeInputManager(jobject contextObj,
14. jobject callbacksObj, **const** sp<Looper>& looper) :
15. mLooper(looper) {
16. JNIEnv\* env = jniEnv();
18. mContextObj = env->NewGlobalRef(contextObj);
19. mCallbacksObj = env->NewGlobalRef(callbacksObj);
21. {
22. AutoMutex \_l(mLock);
23. mLocked.displayWidth = -1;
24. mLocked.displayHeight = -1;
25. mLocked.displayExternalWidth = -1;
26. mLocked.displayExternalHeight = -1;
27. mLocked.displayOrientation = DISPLAY\_ORIENTATION\_0;
29. mLocked.systemUiVisibility = ASYSTEM\_UI\_VISIBILITY\_STATUS\_BAR\_VISIBLE;
30. mLocked.pointerSpeed = 0;
31. mLocked.pointerGesturesEnabled = **true**;
32. mLocked.showTouches = **false**;
33. }
35. sp<EventHub> eventHub = **new** EventHub();
36. mInputManager = **new** InputManager(eventHub, **this**, **this**);
37. }

*@inputManager.cpp*

**[cpp]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. InputManager::InputManager(
2. **const** sp<EventHubInterface>& eventHub,
3. **const** sp<InputReaderPolicyInterface>& readerPolicy,
4. **const** sp<InputDispatcherPolicyInterface>& dispatcherPolicy) {
5. mDispatcher = **new** InputDispatcher(dispatcherPolicy);
6. mReader = **new** InputReader(eventHub, readerPolicy, mDispatcher);
7. initialize();
8. }

**2) 接下来，重点来了，inputDispatcher负责事件（key、touch）的分发，而事件处理延时的ANR也在它这里。InputReader主要负责读取底层传上来的事件，这里就不介绍了。**

*@**inputDispatcher.cpp*

**[cpp]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. // If the currently focused window is still working on previous events then keep waiting.
2. **if** (! isWindowFinishedWithPreviousInputLocked(mFocusedWindowHandle)) {
3. #if DEBUG\_FOCUS
4. LOGD("Waiting because focused window still processing previous input.");
5. #endif
6. injectionResult = handleTargetsNotReadyLocked(currentTime, entry,
7. mFocusedApplicationHandle, mFocusedWindowHandle, nextWakeupTime);
8. **goto** Unresponsive;
9. }
11. ...
13. int32\_t InputDispatcher::handleTargetsNotReadyLocked(nsecs\_t currentTime,
14. **const** EventEntry\* entry,
15. **const** sp<InputApplicationHandle>& applicationHandle,
16. **const** sp<InputWindowHandle>& windowHandle,
17. nsecs\_t\* nextWakeupTime) {
19. ...
21. **if** (currentTime >= mInputTargetWaitTimeoutTime) {
22. onANRLocked(currentTime, applicationHandle, windowHandle,
23. entry->eventTime, mInputTargetWaitStartTime);
25. // Force poll loop to wake up immediately on next iteration once we get the
26. // ANR response back from the policy.
27. \*nextWakeupTime = LONG\_LONG\_MIN;
28. **return** INPUT\_EVENT\_INJECTION\_PENDING;
29. } **else** {
30. // Force poll loop to wake up when timeout is due.
31. **if** (mInputTargetWaitTimeoutTime < \*nextWakeupTime) {
32. \*nextWakeupTime = mInputTargetWaitTimeoutTime;
33. }
34. **return** INPUT\_EVENT\_INJECTION\_PENDING;
35. }
36. }
38. **void** InputDispatcher::onANRLocked(
39. nsecs\_t currentTime, **const** sp<InputApplicationHandle>& applicationHandle,
40. **const** sp<InputWindowHandle>& windowHandle,
41. nsecs\_t eventTime, nsecs\_t waitStartTime) {
42. LOGI("Application is not responding: %s.  "
43. "%01.1fms since event, %01.1fms since wait started",
44. getApplicationWindowLabelLocked(applicationHandle, windowHandle).string(),
45. (currentTime - eventTime) / 1000000.0,
46. (currentTime - waitStartTime) / 1000000.0);
48. CommandEntry\* commandEntry = postCommandLocked(
49. & InputDispatcher::doNotifyANRLockedInterruptible);
50. commandEntry->inputApplicationHandle = applicationHandle;
51. commandEntry->inputWindowHandle = windowHandle;
52. }
54. **void** InputDispatcher::doNotifyANRLockedInterruptible(
55. CommandEntry\* commandEntry) {
56. mLock.unlock();
58. nsecs\_t newTimeout = mPolicy->notifyANR(
59. commandEntry->inputApplicationHandle, commandEntry->inputWindowHandle);
61. mLock.lock();
63. resumeAfterTargetsNotReadyTimeoutLocked(newTimeout,
64. commandEntry->inputWindowHandle != NULL
65. ? commandEntry->inputWindowHandle->getInputChannel() : NULL);
66. }

这里的mPlicy是在构造的时候初始化的。追溯前面的inputManager.cpp,com\_android\_server\_InputManager.cpp,很快

发现mPlicy其实就是NativeInputManager。

**[cpp]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. InputDispatcher::InputDispatcher(**const** sp<InputDispatcherPolicyInterface>& policy) :
2. mPolicy(policy),

*@ com\_android\_server\_InputManager.cpp*

**[cpp]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. nsecs\_t NativeInputManager::notifyANR(**const** sp<InputApplicationHandle>& inputApplicationHandle,
2. **const** sp<InputWindowHandle>& inputWindowHandle) {
3. #if DEBUG\_INPUT\_DISPATCHER\_POLICY
4. LOGD("notifyANR");
5. #endif
7. JNIEnv\* env = jniEnv();
9. jobject inputApplicationHandleObj =
10. getInputApplicationHandleObjLocalRef(env, inputApplicationHandle);
11. jobject inputWindowHandleObj =
12. getInputWindowHandleObjLocalRef(env, inputWindowHandle);
14. jlong newTimeout = env->CallLongMethod(mCallbacksObj,
15. gCallbacksClassInfo.notifyANR, inputApplicationHandleObj, inputWindowHandleObj);
16. **if** (checkAndClearExceptionFromCallback(env, "notifyANR")) {
17. newTimeout = 0; // abort dispatch
18. } **else** {
19. assert(newTimeout >= 0);
20. }
22. env->DeleteLocalRef(inputWindowHandleObj);
23. env->DeleteLocalRef(inputApplicationHandleObj);
24. **return** newTimeout;
25. }

**3) 仔细回忆下前面从java层到native层的inputManger和inputDispatcher的构造，不难发现mPlicy的notifyANR（）最终回到inputManager.java中对应Callbacks的notifyANR（）。（JNI：不仅java层访问本地方法，同样在本地方法中也可以访问java层方法）**

*@inputManager.java*

**[java]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. **public** **long** notifyANR(InputApplicationHandle inputApplicationHandle,
2. InputWindowHandle inputWindowHandle) {
3. **return** mWindowManagerService.mInputMonitor.notifyANR(
4. inputApplicationHandle, inputWindowHandle);
5. }

*@WindowManagerService.java*

**[java]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. **final** InputMonitor mInputMonitor = **new** InputMonitor(**this**);

*@InputMonitor.java*

**[java]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. **public** **long** notifyANR(InputApplicationHandle inputApplicationHandle,
2. InputWindowHandle inputWindowHandle) {
3. AppWindowToken appWindowToken = **null**;
4. **if** (inputWindowHandle != **null**) {
5. **synchronized** (mService.mWindowMap) {
6. WindowState windowState = (WindowState) inputWindowHandle.windowState;
7. **if** (windowState != **null**) {
8. Slog.i(WindowManagerService.TAG, "Input event dispatching timed out sending to "
9. + windowState.mAttrs.getTitle());
10. appWindowToken = windowState.mAppToken;
11. }
12. }
13. }
15. **if** (appWindowToken == **null** && inputApplicationHandle != **null**) {
16. appWindowToken = inputApplicationHandle.appWindowToken;
17. **if** (appWindowToken != **null**) {
18. Slog.i(WindowManagerService.TAG,
19. "Input event dispatching timed out sending to application "
20. + appWindowToken.stringName);
21. }
22. }
24. **if** (appWindowToken != **null** && appWindowToken.appToken != **null**) {
25. **try** {
26. // Notify the activity manager about the timeout and let it decide whether
27. // to abort dispatching or keep waiting.
28. **boolean** abort = appWindowToken.appToken.keyDispatchingTimedOut();
29. **if** (! abort) {
30. // The activity manager declined to abort dispatching.
31. // Wait a bit longer and timeout again later.
32. **return** appWindowToken.inputDispatchingTimeoutNanos;
33. }
34. } **catch** (RemoteException ex) {
35. }
36. }
37. **return** 0; // abort dispatching
38. }
40. **class** AppWindowToken **extends** WindowToken {
41. // Non-null only for application tokens.
42. **final** IApplicationToken appToken;

其实这里的appToken实质上是一个ActivityRecord的“代理”。（需了解binder）

*@ActivityRecord.java*

**[java]** [view plaincopy](http://blog.csdn.net/wuhengde/article/details/8007448)

1. **final** **class** ActivityRecord {
2. **final** ActivityManagerService service; // owner
3. **final** ActivityStack stack; // owner
4. **final** IApplicationToken.Stub appToken; // window manager token
6. **public** **boolean** keyDispatchingTimedOut() {
7. ActivityRecord r;
8. ProcessRecord anrApp = **null**;
9. **synchronized**(service) {
10. r = getWaitingHistoryRecordLocked();
11. **if** (r != **null** && r.app != **null**) {
12. **if** (r.app.debugging) {
13. **return** **false**;
14. }
16. **if** (service.mDidDexOpt) {
17. // Give more time since we were dexopting.
18. service.mDidDexOpt = **false**;
19. **return** **false**;
20. }
22. **if** (r.app.instrumentationClass == **null**) {
23. anrApp = r.app;
24. } **else** {
25. Bundle info = **new** Bundle();
26. info.putString("shortMsg", "keyDispatchingTimedOut");
27. info.putString("longMsg", "Timed out while dispatching key event");
28. service.finishInstrumentationLocked(
29. r.app, Activity.RESULT\_CANCELED, info);
30. }
31. }
32. }
34. **if** (anrApp != **null**) {
35. service.appNotResponding(anrApp, r, **this**,
36. "keyDispatchingTimedOut");
37. }
39. **return** **true**;
40. }

接下去就是ActivityManagerService的appNotResponding，我们平常所看到的ANR对话框正出自这里。