

原

Android Input流程分析（二）：EventHub

2017年09月05日 22:15:53

Invoker123

阅读数：564

现在，InputReader线程已经开始运行,在InputReaderThread::threadLoop开始读取EventHub送上来的事件。mReader指向一个InputReader对象，调用InputReader::loopOnce函数。

*/frameworks/native/services/inputflinger/InputReader.cpp*

```
1 bool InputReaderThread::threadLoop() {
2     mReader->loopOnce();
3     return true;
4 }
```

先忽略其他细节，InputReader是在getEvents中捞取事件的。mEventBuffer是一个RawEvent结构体数组，mEventHub指向一个EventHub对象。

*/frameworks/native/services/inputflinger/InputReader.cpp*

```
1 void InputReader::loopOnce() {
2     int32_t oldGeneration;
3     int32_t timeoutMillis;
4     bool inputDevicesChanged = false;
5     Vector<InputDeviceInfo> inputDevices;
6     { // acquire lock
7         AutoMutex _l(mLock);
8
9         oldGeneration = mGeneration;
10        timeoutMillis = -1;
11
12        uint32_t changes = mConfigurationChangesToRefresh;
13        if (changes) {
14            mConfigurationChangesToRefresh = 0;
15            timeoutMillis = 0;
16            refreshConfigurationLocked(changes);
17        } else if (mNextTimeout != LLONG_MAX) {
18            nsecs_t now = systemTime(SYSTEM_TIME_MONOTONIC);
19            timeoutMillis = toMillisecondTimeoutDelay(now, mNextTimeout);
20        }
21    } // release lock
22
23    size_t count = mEventHub->getEvents(timeoutMillis, mEventBuffer, EVENT_BUFFER_SIZE);
24
25    { // acquire lock
26        AutoMutex _l(mLock);
27        mReaderIsAliveCondition.broadcast();
28
29        if (count) {
30            processEventsLocked(mEventBuffer, count);
31        }
32    }
33
34    if (mNextTimeout != LLONG_MAX) {
35        nsecs_t now = systemTime(SYSTEM_TIME_MONOTONIC);
36        if (now >= mNextTimeout) {
37            #if DEBUG_RAW_EVENTS
38                ALOGD("Timeout expired, latency=%0.3fms", (now - mNextTimeout) * 0.000001f);
39            #endif
40
41            mNextTimeout = LLONG_MAX;
42            timeoutExpiredLocked(now);
43        }
44    }
45
46    if (oldGeneration != mGeneration) {
47        inputDevicesChanged = true;
48        getInputDevicesLocked(inputDevices);
49    } // release lock
50
51    // Send out a message that the describes the changed input devices.
52    if (inputDevicesChanged) {
53        mPolicy->notifyInputDevicesChanged(inputDevices);
54    }
```

👍

1

💬

🔖

📱

<

>

```
57 // This must happen outside of the lock because the listener could potentially call
58 // back into the InputReader's methods, such as getScanCodeState, or become blocked
59 // on another thread similarly waiting to acquire the InputReader lock thereby
60 // resulting in a deadlock. This situation is actually quite plausible because the
61 // listener is actually the input dispatcher, which calls into the window manager,
62 // which occasionally calls into the input reader.
63 mQueuedListener->flush();
}
```

1

EventHub在什么时候创建的呢？在NativeInputManager的构造函数中。看看EventHub的构造函数。首先创建了epoll文件描述符mEpollFd，用于通知用户空间程序文件系统变化的机制。inotify\_init用于创建一个inotify的fd,inotify\_add\_watch中DEVICE\_PATH为 “/dev/input” ,inotify\_add\_watch作用是监控/dev/input目录下文件的变化。之后，将mEpollFd加入到epoll监控队列中，并将EPOLL\_ID\_INOTIFY保存到eventItem的union成员epoll\_data中。然后，创建一个管道，将读端加入到epoll监控队列中，并将EPOLL\_ID\_WAKE保存到eventItem的union成员epoll\_data中。

#### /frameworks/native/services/inputflinger/EventHub.cpp

```
1  EventHub::EventHub(void) :
2      mBuiltInKeyboardId(NO_BUILT_IN_KEYBOARD), mNextDeviceId(1), mControllerNumbers(),
3      mOpeningDevices(0), mClosingDevices(0),
4      mNeedToSendFinishedDeviceScan(false),
5      mNeedToReopenDevices(false), mNeedToScanDevices(true),
6      mPendingEventCount(0), mPendingEventIndex(0), mPendingINotify(false) {
7      acquire_wake_lock(PARTIAL_WAKE_LOCK, WAKE_LOCK_ID);
8
9      mEpollFd = epoll_create(EPOLL_SIZE_HINT);
10     LOG_ALWAYS_FATAL_IF(mEpollFd < 0, "Could not create epoll instance. errno=%d", errno);
11
12     mINotifyFd = inotify_init();
13     int result = inotify_add_watch(mINotifyFd, DEVICE_PATH, IN_DELETE | IN_CREATE);
14     LOG_ALWAYS_FATAL_IF(result < 0, "Could not register INotify for %s. errno=%d",
15         DEVICE_PATH, errno);
16
17     struct epoll_event eventItem;
18     memset(&eventItem, 0, sizeof(eventItem));
19     eventItem.events = EPOLLIN;
20     eventItem.data.u32 = EPOLL_ID_INOTIFY;
21     result = epoll_ctl(mEpollFd, EPOLL_CTL_ADD, mINotifyFd, &eventItem);
22     LOG_ALWAYS_FATAL_IF(result != 0, "Could not add INotify to epoll instance. errno=%d", errno);
23
24     int wakeFds[2];
25     result = pipe(wakeFds);
26     LOG_ALWAYS_FATAL_IF(result != 0, "Could not create wake pipe. errno=%d", errno);
27
28     mWakeReadPipeFd = wakeFds[0];
29     mWakeWritePipeFd = wakeFds[1];
30
31     result = fcntl(mWakeReadPipeFd, F_SETFL, O_NONBLOCK);
32     LOG_ALWAYS_FATAL_IF(result != 0, "Could not make wake read pipe non-blocking. errno=%d",
33         errno);
34
35     result = fcntl(mWakeWritePipeFd, F_SETFL, O_NONBLOCK);
36     LOG_ALWAYS_FATAL_IF(result != 0, "Could not make wake write pipe non-blocking. errno=%d",
37         errno);
38
39     eventItem.data.u32 = EPOLL_ID_WAKE;
40     result = epoll_ctl(mEpollFd, EPOLL_CTL_ADD, mWakeReadPipeFd, &eventItem);
41     LOG_ALWAYS_FATAL_IF(result != 0, "Could not add wake read pipe to epoll instance. errno=%d",
42         errno);
43
44     int major, minor;
45     getLinuxRelease(&major, &minor);
46     // EPOLLWAKEUP was introduced in kernel 3.5
47     mUsingEpollWakeup = major > 3 || (major == 3 && minor >= 5);
48 }
```

回到EventHub::getEvents函数。readBuffer是内核input\_event数组，event指向RawEvent数组下个可用的地址，capacity表示input\_event数组的剩余容量，awoken表示是否通过写入管道唤醒阻塞的epoll\_wait。随后进入一个死循环。

#### /frameworks/native/services/inputflinger/EventHub.cpp

```
1  size_t EventHub::getEvents(int timeoutMillis, RawEvent* buffer, size_t bufferSize) {
2      ALOG_ASSERT(bufferSize >= 1);
3
4      AutoMutex _l(mLock);
5      if (mUsingEpollWakeup) {
```

```
8     RawEvent* event = buffer;
9     size_t capacity = bufferSize;
10    bool awoken = false;
11    ...
```

1

这部分是重启设备处理流程。当Input的一些设置被改变时，mNeedToReopenDevices会变true,表示要重启设备。调用closeAllDevicesLocked关闭所有打开的设备，将mNeedToScanDevices设为true，表示下次进入getEvents函数时要扫描打开设备。之后，使用break退出主循环，等待loopOnce再次调用getEvents函数进行扫描设备的操作。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1  for (;;) {
2      nsecs_t now = systemTime(SYSTEM_TIME_MONOTONIC);
3
4      // Reopen input devices if needed.
5      if (mNeedToReopenDevices) {
6          mNeedToReopenDevices = false;
7
8          ALOGI("Reopening all input devices due to a configuration change.");
9
10         closeAllDevicesLocked();
11         mNeedToScanDevices = true;
12         break; // return to the caller before we actually rescan
13     }
14     ...
15 }
```

mDevices类型为KeyedVector< int32\_t, Device\*>，第一个模板参数为Device的id。第二个模板参数是对应的Device结构体指针，KeyedVector内部使用SortedVector实现，里面元素按key值的大小排列。closeAllDevicesLocked就是逐一对mDevices里的Device执行closeDeviceLocked函数进行关闭。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1  void EventHub::closeAllDevicesLocked() {
2      while (mDevices.size() > 0) {
3          closeDeviceLocked(mDevices.valueAt(mDevices.size() - 1));
4      }
5  }
```

mBuiltInKeyboardId在EventHub构造函数中被初始化为NO\_BUILT\_IN\_KEYBOARD（-2）。在打开设备时，若设备的类别为键盘，游戏手柄及虚拟键盘时，mBuiltInKeyboardId会被设为该设备的id。当要关闭的设备类型是上述类型时，将 mBuiltInKeyboardId重新置成NO\_BUILT\_IN\_KEYBOARD。isVirtual函数判断Device的fd是否小于0，若大于或等于0，将这个fd从mEpollFd监控队列中移除。releaseControllerNumberLocked将Device的controllerNumber置0，并清除掉mControllerNumbers相应的标志位。然后将参数对应的Device从mDevices中移除，并将对应的fd关闭。

mOpeningDevices对应一个打开的Device组成的链表的头节点。从这个头节点开始遍历，找到要关闭的Device,将其从链表中删除。若找不到，将这个要关闭的Device插入到到mClosingDevices链表的头部，mClosingDevices仍指向这个链表的头部。mClosingDevices对应一个关闭的Device组成的链表的头节点。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1  void EventHub::closeDeviceLocked(Device* device) {
2      ALOGI("Removed device: path=%s name=%s id=%d fd=%d classes=0x%x\n",
3          device->path.string(), device->identifier.name.string(), device->id,
4          device->fd, device->classes);
5
6      if (device->id == mBuiltInKeyboardId) {
7          ALOGW("built-in keyboard device %s (id=%d) is closing! the apps will not like this",
8              device->path.string(), mBuiltInKeyboardId);
9          mBuiltInKeyboardId = NO_BUILT_IN_KEYBOARD;
10     }
11
12     if (!device->isVirtual()) {
13         if (epoll_ctl(mEpollFd, EPOLL_CTL_DEL, device->fd, NULL)) {
14             ALOGW("Could not remove device fd from epoll instance. errno=%d", errno);
15         }
16     }
17
18     releaseControllerNumberLocked(device);
19
20     mDevices.removeItem(device->id);
21     device->close();
22
23     // Unlink for opening devices list if it is present.
24     Device* pred = NULL;
25     bool found = false;
26     for (Device* entry = mOpeningDevices; entry != NULL; ) {
27         if (entry == device) {
```

```
31     pred = entry;
32     entry = entry->next;
33 }
34 if (found) {
35     // Unlink the device from the opening devices list then delete it.
36     // We don't need to tell the client that the device was closed because
37     // it does not even know it was opened in the first place.
38     ALOGI("Device %s was immediately closed after opening.", device->path.string());
39     if (pred) {
40         pred->next = device->next;
41     } else {
42         mOpeningDevices = device->next;
43     }
44     delete device;
45 } else {
46     // Link into closing devices list.
47     // The device will be deleted later after we have informed the client.
48     device->next = mClosingDevices;
49     mClosingDevices = device;
50 }
51 }
```

1

这里要提一下Device结构体。

*/frameworks/native/services/inputflinger/EventHub.h*

```
1 struct Device {
2     Device* next; //Device是以链表的形式组织的
3
4     int fd; // 文件描述符
5     const int32_t id; //唯一id
6     const String8 path; //设备路径
7     const InputDeviceIdentifier identifier; //厂商信息
8
9     uint32_t classes; //类别
10    //掩码数组
11    uint8_t keyBitmask[(KEY_MAX + 1) / 8];
12    uint8_t absBitmask[(ABS_MAX + 1) / 8];
13    uint8_t relBitmask[(REL_MAX + 1) / 8];
14    uint8_t swBitmask[(SW_MAX + 1) / 8];
15    uint8_t ledBitmask[(LED_MAX + 1) / 8];
16    uint8_t ffBitmask[(FF_MAX + 1) / 8];
17    uint8_t propBitmask[(INPUT_PROP_MAX + 1) / 8];
18    //配置信息
19    String8 configurationFile;
20    PropertyMap* configuration;
21    //键盘映射表
22    VirtualKeyMap* virtualKeyMap;
23    KeyMap keyMap;
24
25    sp<KeyCharacterMap> overlayKeyMap;
26    sp<KeyCharacterMap> combinedKeyMap;
27    //力反馈相关
28    bool ffEffectPlaying;
29    int16_t ffEffectId; // initially -1
30
31
32    int32_t controllerNumber;
33
34    int32_t timestampOverrideSec;
35    int32_t timestampOverrideUsec;
36    ...
}
```

遍历mClosingDevices链表，每次得到一个Device，填充好一个RawEvent，标记为DEVICE\_REMOVED类型，event指针加1表示使用掉了一个RawEvent,然后delete掉这个Device，将mNeedToSendFinishedDeviceScan标记为true,表示待会要发送扫描完成的事件。capacity减1，若等于0表示nput\_event数组已无剩余空间，退出遍历。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 while (mClosingDevices) {
2     Device* device = mClosingDevices;
3     ALOGV("Reporting device closed: id=%d, name=%s\n",
4         device->id, device->path.string());
5     mClosingDevices = device->next;
6     event->when = now;
7     event->deviceId = device->id == mBuiltInKeyboardId ? BUILT_IN_KEYBOARD_ID : device->id;
}
```

```
11         delete device;
12         mNeedToSendFinishedDeviceScan = true;
13         if (--capacity == 0) {
14             break;
15         }
16     }
```

1

InputReader第一次调用getEvents时，上面步骤都是不执行的。mNeedToScanDevices在构造函数中被初始化为true,所以会从这里开始。关键代码是调用scanDevicesLocked。最后也将mNeedToSendFinishedDeviceScan 设为true,待会会发送设备扫描完成事件。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1  if (mNeedToScanDevices) {
2      mNeedToScanDevices = false;
3      scanDevicesLocked();
4      mNeedToSendFinishedDeviceScan = true;
5  }
```

scanDirLocked扫描/dev/input下的文件，并以这些文件的完整路径名为入参调用openDeviceLocked函数。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1  void EventHub::scanDevicesLocked() {
2      status_t res = scanDirLocked(DEVICE_PATH);//DEVICE_PATH为"/dev/input"
3      if(res < 0) {
4          ALOGE("scan dir failed for %s\n", DEVICE_PATH);
5      }
6      if (mDevices.indexOfKey(VIRTUAL_KEYBOARD_ID) < 0) {
7          createVirtualKeyboardLocked();
8      }
9  }
```

openDeviceLocked函数实现比较繁琐。选择重点的来讲。第一部分，使用从fd获得的信息填充identifier的成员，包括name,bus,product,vendor,version，location和uniqueId。assignDescriptorLocked用来设置identifier的descriptor和nonce成员。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1  status_t EventHub::openDeviceLocked(const char *devicePath) {
2      char buffer[80];
3
4      ALOGV("Opening device: %s", devicePath);
5
6      int fd = open(devicePath, O_RDWR | O_CLOEXEC);
7      if(fd < 0) {
8          ALOGE("could not open %s, %s\n", devicePath, strerror(errno));
9          return -1;
10     }
11
12     InputDeviceIdentifier identifier;
13
14     // Get device name.
15     if(ioctl(fd, EVIOCGNAME(sizeof(buffer) - 1), &buffer) < 1) {
16         //fprintf(stderr, "could not get device name for %s, %s\n", devicePath, strerror(errno));
17     } else {
18         buffer[sizeof(buffer) - 1] = '\0';
19         identifier.name.setTo(buffer);
20     }
21
22     // Check to see if the device is on our excluded list
23     for (size_t i = 0; i < mExcludedDevices.size(); i++) {
24         const String8& item = mExcludedDevices.itemAt(i);
25         if (identifier.name == item) {
26             ALOGI("ignoring event id %s driver %s\n", devicePath, item.string());
27             close(fd);
28             return -1;
29         }
30     }
31
32     // Get device driver version.
33     int driverVersion;
34     if(ioctl(fd, EVIOCGVERSION, &driverVersion)) {
35         ALOGE("could not get driver version for %s, %s\n", devicePath, strerror(errno));
36         close(fd);
37         return -1;
38     }
39 }
```



```
42 struct input_id inputId;
43 if(ioctl(fd, EVIOCGID, &inputId)) {
44     ALOGE("could not get device input id for %s, %s\n", devicePath, strerror(errno));
45     close(fd);
46     return -1;
47 }
48 identifier.bus = inputId.bustype;
49 identifier.product = inputId.product;
50 identifier.vendor = inputId.vendor;
51 identifier.version = inputId.version;
52
53 // Get device physical location.
54 if(ioctl(fd, EVIOCGPHYS(sizeof(buffer) - 1), &buffer) < 1) {
55     //fprintf(stderr, "could not get location for %s, %s\n", devicePath, strerror(errno));
56 } else {
57     buffer[sizeof(buffer) - 1] = '\0';
58     identifier.location.setTo(buffer);
59 }
60
61 // Get device unique id.
62 if(ioctl(fd, EVIOCGUNIQ(sizeof(buffer) - 1), &buffer) < 1) {
63     //fprintf(stderr, "could not get idstring for %s, %s\n", devicePath, strerror(errno));
64 } else {
65     buffer[sizeof(buffer) - 1] = '\0';
66     identifier.uniqueId.setTo(buffer);
67 }
68
69 // Fill in the descriptor.
70 assignDescriptorLocked(identifier);
71
72 // Make file descriptor non-blocking for use with poll().
73 if (fcntl(fd, F_SETFL, O_NONBLOCK)) {
74     ALOGE("Error %d making device file descriptor non-blocking.", errno);
75     close(fd);
76     return -1;
77 }
78 ...
79 }
```

1

看看assignDescriptorLocked函数。uniqueId用来描述identifier的唯一性，当uniqueId为空时，就使用nonce来确保唯一性。descriptor是用identifier各个成员的值按一定格式组合起来再使用SHA1加密得到的值，generateDescriptor函数的作用就是产生这个SHA1值赋给descriptor。如果uniqueId为空，检查mDevices中各个Device对应的descriptor是否等于当前得到的descriptor，若相等，将nonce加1，重新使用generateDescriptor生成新的descriptor，直到这个新的descriptor是唯一的。

***/frameworks/native/services/inputflinger/EventHub.cpp***

```
1 void EventHub::assignDescriptorLocked(InputDeviceIdentifier& identifier) {
2     // Compute a device descriptor that uniquely identifies the device.
3     // The descriptor is assumed to be a stable identifier. Its value should not
4     // change between reboots, reconnections, firmware updates or new releases
5     // of Android. In practice we sometimes get devices that cannot be uniquely
6     // identified. In this case we enforce uniqueness between connected devices.
7     // Ideally, we also want the descriptor to be short and relatively opaque.
8
9     identifier.nonce = 0;
10    String8 rawDescriptor = generateDescriptor(identifier);
11    if (identifier.uniqueId.isEmpty()) {
12        // If it didn't have a unique id check for conflicts and enforce
13        // uniqueness if necessary.
14        while(getDeviceByDescriptorLocked(identifier.descriptor) != NULL) {
15            identifier.nonce++;
16            rawDescriptor = generateDescriptor(identifier);
17        }
18    }
19    ALOGV("Created descriptor: raw=%s, cooked=%s", rawDescriptor.string(),
20        identifier.descriptor.string());
21 }
22 }
```

***/frameworks/native/services/inputflinger/EventHub.cpp***

```
1 static String8 generateDescriptor(InputDeviceIdentifier& identifier) {
2     String8 rawDescriptor;
3     rawDescriptor.appendFormat(":%04x:%04x:", identifier.vendor,
4         identifier.product);
5 }
```

```

8     rawDescriptor.append("uniqueId:");
9     rawDescriptor.append(identifier.uniqueId);
10 } else if (identifier.nonce != 0) {
11     rawDescriptor.appendFormat("nonce:%04x", identifier.nonce);
12 }
13
14 if (identifier.vendor == 0 && identifier.product == 0) {
15     // If we don't know the vendor and product id, then the device is probably
16     // built-in so we need to rely on other information to uniquely identify
17     // the input device. Usually we try to avoid relying on the device name or
18     // location but for built-in input device, they are unlikely to ever change.
19     if (!identifier.name.isEmpty()) {
20         rawDescriptor.append("name:");
21         rawDescriptor.append(identifier.name);
22     } else if (!identifier.location.isEmpty()) {
23         rawDescriptor.append("location:");
24         rawDescriptor.append(identifier.location);
25     }
26 }
27 }
28 identifier.descriptor = sha1(rawDescriptor);
29 return rawDescriptor;
}

```

1

回到openDeviceLocked函数中。在EventHub构造函数中，mNextDeviceId初始化为1。这样，新创建的Device的id会从1起逐渐递增。之后是一大串的Log。

#### ***/frameworks/native/services/inputflinger/EventHub.cpp***

```

1  int32_t deviceId = mNextDeviceId++;
2  Device* device = new Device(fd, deviceId, String8(devicePath), identifier);
3
4  ALOGV("add device %d: %s\n", deviceId, devicePath);
5  ALOGV(" bus:      %04x\n"
6        " vendor    %04x\n"
7        " product   %04x\n"
8        " version    %04x\n",
9        identifier.bus, identifier.vendor, identifier.product, identifier.version);
10 ALOGV(" name:      \"%s\"\n", identifier.name.string());
11 ALOGV(" location:  \"%s\"\n", identifier.location.string());
12 ALOGV(" unique id: \"%s\"\n", identifier.uniqueId.string());
13 ALOGV(" descriptor: \"%s\"\n", identifier.descriptor.string());
14 ALOGV(" driver:    v%d.%d.%d\n",
15        driverVersion >> 16, (driverVersion >> 8) & 0xff, driverVersion & 0xff);
16

```

loadConfigurationLocked用于加载按键文件，详见我另一篇博客：《Android加载按键文件流程》。之后的流程统一按处理按键事件展开。

#### ***/frameworks/native/services/inputflinger/EventHub.cpp***

```

1  ...
2  // Load the configuration file for the device.
3  loadConfigurationLocked(device);
4  ...

```

EV\_KEY表示按键类型的事件。能够上报这类事件的设备有键盘，鼠标，手柄，手写板等一切拥有按钮的设备（包括手机上的实体按键）。在Device结构体中，对应的事件位掩码keyBitmask描述了可以产生的事件的集合。按键事件的全集包括字符按键，方向键，控制键，鼠标键，游戏按键等。

#### ***/frameworks/native/services/inputflinger/EventHub.cpp***

```

1  ioctl(fd, EVIOCGBIT(EV_KEY, sizeof(device->keyBitmask)), device->keyBitmask);
2  ...
3  // See if this is a keyboard. Ignore everything in the button range except for
4  // joystick and gamepad buttons which are handled like keyboards for the most part.
5  bool haveKeyboardKeys = containsNonZeroByte(device->keyBitmask, 0, sizeof_bit_array(BTN_MISC))
6      || containsNonZeroByte(device->keyBitmask, sizeof_bit_array(KEY_OK),
7      sizeof_bit_array(KEY_MAX + 1));
8  bool haveGamepadButtons = containsNonZeroByte(device->keyBitmask, sizeof_bit_array(BTN_MISC),
9      sizeof_bit_array(BTN_MOUSE))
10     || containsNonZeroByte(device->keyBitmask, sizeof_bit_array(BTN_JOYSTICK),
11     sizeof_bit_array(BTN_DIGI));
12 if (haveKeyboardKeys || haveGamepadButtons) {
13     device->classes |= INPUT_DEVICE_CLASS_KEYBOARD;
14 }
15

```

如果Device类型为keyboard或joystick,加载解析.kl和.kcm文件。类型为keyboard的情况还会把mBuiltInKeyboardId设置为该Device的id。根据传上来的按键集合，可以将keyboard类型分为几类：

INPUT\_DEVICE\_CLASS\_ALPHAKY表示字符按键设备 INPUT\_DEVICE\_CLASS\_DPAD表示拥有方向键的设备 INPUT\_DEVICE\_CLASS\_GAMEPAD表示游戏手柄设备

```
1 // Load the key map.
2 // We need to do this for joysticks too because the key layout may specify axes.
3 status_t keyMapStatus = NAME_NOT_FOUND;
4 if (device->classes & (INPUT_DEVICE_CLASS_KEYBOARD | INPUT_DEVICE_CLASS_JOYSTICK)) {
5     // Load the keymap for the device.
6     keyMapStatus = loadKeyMapLocked(device);
7 }
8
9 // Configure the keyboard, gamepad or virtual keyboard.
10 if (device->classes & INPUT_DEVICE_CLASS_KEYBOARD) {
11     // Register the keyboard as a built-in keyboard if it is eligible.
12     if (!keyMapStatus
13         && mBuiltInKeyboardId == NO_BUILT_IN_KEYBOARD
14         && isEligibleBuiltInKeyboard(device->identifier,
15                                     device->configuration, &device->keyMap)) {
16         mBuiltInKeyboardId = device->id;
17     }
18     // 'Q' key support = cheap test of whether this is an alpha-capable kbd
19     if (hasKeycodeLocked(device, AKEYCODE_Q)) {
20         device->classes |= INPUT_DEVICE_CLASS_ALPHAKEY;
21     }
22
23     // See if this device has a DPAD.
24     if (hasKeycodeLocked(device, AKEYCODE_DPAD_UP) &&
25         hasKeycodeLocked(device, AKEYCODE_DPAD_DOWN) &&
26         hasKeycodeLocked(device, AKEYCODE_DPAD_LEFT) &&
27         hasKeycodeLocked(device, AKEYCODE_DPAD_RIGHT) &&
28         hasKeycodeLocked(device, AKEYCODE_DPAD_CENTER)) {
29         device->classes |= INPUT_DEVICE_CLASS_DPAD;
30     }
31
32     // See if this device has a gamepad.
33     for (size_t i = 0; i < sizeof(GAMEPAD_KEYCODES)/sizeof(GAMEPAD_KEYCODES[0]); i++) {
34         if (hasKeycodeLocked(device, GAMEPAD_KEYCODES[i])) {
35             device->classes |= INPUT_DEVICE_CLASS_GAMEPAD;
36             break;
37         }
38     }
```

1

将Device对应的fd添加到epoll监控队列中，监听事件为EPOLLIN。在3.5版本以上的Linux中，增设监听事件EPOLLWAKEUP。

```
1 // Register with epoll.
2 struct epoll_event eventItem;
3 memset(&eventItem, 0, sizeof(eventItem));
4 eventItem.events = EPOLLIN;
5 if (mUsingEpollWakeup) {
6     eventItem.events |= EPOLLWAKEUP;
7 }
8 eventItem.data.u32 = deviceId;
9 if (epoll_ctl(mEpollFd, EPOLL_CTL_ADD, fd, &eventItem)) {
10     ALOGE("Could not add device fd to epoll instance. errno=%d", errno);
11     delete device;
12     return -1;
13 }
14 }
```

最后，往mDevices添加id-Device\*键值对，往mOpeningDevices表示的链表头部插入该Device,mOpeningDevices继续指向该链表的头部。

```
1 ...
2 addDeviceLocked(device);
3 return 0;
4 }
```

```
1 void EventHub::addDeviceLocked(Device* device) {
2     mDevices.add(device->id, device);
3     device->next = mOpeningDevices;
4     mOpeningDevices = device;
5 }
```



值得注意的是，在scanDevicesLocked函数中，还将创建VirtualKeyboard，此处不再详述。至此，/dev/input/下的所有Device信息都已经被初始化，mOpeningDevices链表已经就绪。现在返回到getEvents函数中。

遍历mOpeningDevices链表，每个Device对应生成一个RawEvent。RawEvent的时间戳设为当前时间，deviceId设为mBuiltInKeyboardId;Device的id,事件类型记为DEVICE\_ADDED。mNeedToSendFinishedDeviceScan设为true，表示要发送设备扫描完成事件，最后将capacity减1，因为产生了- 1 out\_event事件，input\_event数组剩余容量减1。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 ...
2 while (mOpeningDevices != NULL) {
3     Device* device = mOpeningDevices;
4     ALOGV("Reporting device opened: id=%d, name=%s\n",
5         device->id, device->path.string());
6     mOpeningDevices = device->next;
7     event->when = now;
8     event->deviceId = device->id == mBuiltInKeyboardId ? 0 : device->id;
9     event->type = DEVICE_ADDED;
10    event += 1;
11    mNeedToSendFinishedDeviceScan = true;
12    if (--capacity == 0) {
13        break;
14    }
15 }
```

每次重启，卸载，开启设备时，都要将mNeedToSendFinishedDeviceScan设为true，这会对应一个RawEvent和input\_event。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 if (mNeedToSendFinishedDeviceScan) {
2     mNeedToSendFinishedDeviceScan = false;
3     event->when = now;
4     event->type = FINISHED_DEVICE_SCAN;
5     event += 1;
6     if (--capacity == 0) {
7         break;
8     }
9 }
```

InputReader在线程循环中不断调用loopOnce,也就意味着getEvents函数会被循环调用。先看看第一次调用的情况。前面提到，第一次进入getEvents函数，入口点在scanDevicesLocked函数以用来扫描打开的设备。在EventsHub的构造函数中，mPendingEventIndex和mPendingEventCount都被初始化为0，所以第一次进入getEvents函数时并不会进入以下循环：

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 ...
2 while (mPendingEventIndex < mPendingEventCount) {
3     ...
```

设备的添加导致产生RawEvent事件，使得event指针不等于RawEvents数组首地址。所以接下来会退出getEvents函数的主循环，直接返回产生的RawEvent数量。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 if (event != buffer || awoken) {
2     break;
3 }
4 ...
5 return event - buffer;
6
```

第二次进入getEvents时，由于mNeedToScanDevices已被置为false，所以不会再去扫描设备。此时event的值与buffer相同，这次会进入到epoll\_wait中。epoll监听了以下fd:Inotify的fd,第一次扫描的设备fd，唤醒管道读端的fd。epoll\_wait在等待timeoutMillis的时间内，返回的pollResult为发生的目标事件数量，同时赋给mPendingEventCount，发生的事件保存在mPendingEventItems中。下一步，重新进入getEvent函数主循环。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 ...
2 int pollResult = epoll_wait(mEpollFd, mPendingEventItems, EPOLL_MAX_EVENTS, timeoutMillis);
3 ...
4 else {
5     // Some events occurred.
6     mPendingEventCount = size_t(pollResult);
7 }
8
```

mPendingEventCount已经为正，终于可以进入while (mPendingEventIndex < mPendingEventCount)循环了。  
遍历mPendingEventItems中的epoll\_event,如果是Inotify的EPOLLIN事件，将mPendingINotify设为true。之后contine这个循环。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 while (mPendingEventIndex < mPendingEventCount) {
2     const struct epoll_event& eventItem = mPendingEventItems[mPendingEventIndex++];
3     if (eventItem.data.u32 == EPOLL_ID_INOTIFY) {
4         if (eventItem.events & EPOLLIN) {
5             mPendingINotify = true;
6         } else {
7             ALOGW("Received unexpected epoll event 0x%08x for INotify.", eventItem.events);
8         }
9         continue;
10    }
11 }
```

1

如果是管道读端的EPOLLIN事件，把awoken设置成true，读走管道读端的数据后，continue这个循环。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 if (eventItem.data.u32 == EPOLL_ID_WAKE) {
2     if (eventItem.events & EPOLLIN) {
3         ALOGV("awoken after wake()");
4         awoken = true;
5         char buffer[16];
6         ssize_t nRead;
7         do {
8             nRead = read(mWakeReadPipeFd, buffer, sizeof(buffer));
9         } while ((nRead == -1 && errno == EINTR) || nRead == sizeof(buffer));
10    } else {
11        ALOGW("Received unexpected epoll event 0x%08x for wake read pipe.",
12            eventItem.events);
13    }
14    continue;
15 }
```

根据eventItem找到发生EPOLLIN事件的Device后，从该Device对应的fd读取数据到readBuffer中，readBuffer是一个input\_event结构体数组。如果数取的数据为空或者出错返回ENODEV，说明对应的Device已经被移除，deviceChanged设为true,调用closeDeviceLocked处理该设备移除事件。如果正常返回，count得到在Device上发生的input\_event数量。遍历readBuffer上的每个input\_event，接下来用input\_event的成员去初始化RawEvent，readBuffer剩余容量减1。若剩余容量为0，则退出当前循环。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1     ssize_t deviceIndex = mDevices.indexOfKey(eventItem.data.u32);
2     ...
3     Device* device = mDevices.valueAt(deviceIndex);
4     if (eventItem.events & EPOLLIN) {
5         int32_t readSize = read(device->fd, readBuffer,
6             sizeof(struct input_event) * capacity);
7         if (readSize == 0 || (readSize < 0 && errno == ENODEV)) {
8             // Device was removed before INotify noticed.
9             ALOGW("could not get event, removed? (fd: %d size: %" PRIu32
10                " bufferSize: %zu capacity: %zu errno: %d)\n",
11                device->fd, readSize, bufferSize, capacity, errno);
12             deviceChanged = true;
13             closeDeviceLocked(device);
14         } else if (readSize < 0) {
15             if (errno != EAGAIN && errno != EINTR) {
16                 ALOGW("could not get event (errno=%d)", errno);
17             }
18         } else if ((readSize % sizeof(struct input_event)) != 0) {
19             ALOGE("could not get event (wrong size: %d)", readSize);
20         } else {
21             int32_t deviceId = device->id == mBuiltInKeyboardId ? 0 : device->id;
22
23             size_t count = size_t(readSize) / sizeof(struct input_event);
24             for (size_t i = 0; i < count; i++) {
25                 struct input_event& iev = readBuffer[i];
26                 ALOGV("%s got: time=%d.%06d, type=%d, code=%d, value=%d",
27                     device->path.string(),
28                     (int) iev.time.tv_sec, (int) iev.time.tv_usec,
29                     iev.type, iev.code, iev.value);
30
31                 // Some innut devices mav have a better concent of the time
32             }
```

```
35 // This is a custom Android extension of the input protocol
36 // mainly intended for use with uinput based device drivers.
37 if (iev.type == EV_MSC) {
38     if (iev.code == MSC_ANDROID_TIME_SEC) {
39         device->timestampOverrideSec = iev.value;
40         continue;
41     } else if (iev.code == MSC_ANDROID_TIME_USEC) {
42         device->timestampOverrideUsec = iev.value;
43         continue;
44     }
45 }
46 ...
47 event->deviceId = deviceId;
48 event->type = iev.type;
49 event->code = iev.code;
50 event->value = iev.value;
51 event += 1;
52 capacity -= 1;
53 }
54 if (capacity == 0) {
55     // The result buffer is full. Reset the pending event index
56     // so we will try to read the device again on the next iteration.
57     mPendingEventIndex -= 1;
58     break;
59 }
60 }
61 }
62 } else if (eventItem.events & EPOLLHUP) {
63     ALOGI("Removing device %s due to epoll hang-up event.",
64         device->identifier.name.string());
65     deviceChanged = true;
66     closeDeviceLocked(device);
67 } else {
68     ALOGW("Received unexpected epoll event 0x%08x for device %s.",
69         eventItem.events, device->identifier.name.string());
70 }
```

1

处理完mPendingEventItems的事件后，mPendingEventIndex的值与mPendingEventCount的值相等。如果之前处理的mPendingEventItem中的事件有发生在Inotify的（mPendingINotify为true），说明发生了设备增删事件,此时会调用readNotifyLocked处理设备增删事件，将deviceChanged设为true,mPendingINotify设为false。

***/frameworks/native/services/inputflinger/EventHub.cpp***

```
1 // readNotify() will modify the list of devices so this must be done after
2 // processing all other events to ensure that we read all remaining events
3 // before closing the devices.
4 if (mPendingINotify && mPendingEventIndex >= mPendingEventCount) {
5     mPendingINotify = false;
6     readNotifyLocked();
7     deviceChanged = true;
8 }
9 }
```

EventHub::readNotifyLocked函数就是遍历所有的Inotify事件，根据inotify\_event的mask值判断设备是新增还是卸载进行处理。如果是设备新增事件，调用openDeviceLocked进行处理。如果是设备卸载事件，调用closeDeviceByPathLocked进行处理，closeDeviceByPathLocked内部调用了closeDeviceLocked。

***/frameworks/native/services/inputflinger/EventHub.cpp***

```
1 status_t EventHub::readNotifyLocked() {
2     int res;
3     char devname[PATH_MAX];
4     char *filename;
5     char event_buf[512];
6     int event_size;
7     int event_pos = 0;
8     struct inotify_event *event;
9
10    ALOGV("EventHub::readNotify nfd: %d\n", mINotifyFd);
11    res = read(mINotifyFd, event_buf, sizeof(event_buf));
12    if(res < (int)sizeof(*event)) {
13        if(errno == EINTR)
14            return 0;
15        ALOGW("could not get event, %s\n", strerror(errno));
16        return -1;
17    }
18 }
```

```
21 strcpy(devname, DEVICE_PATH);
22 filename = devname + strlen(devname);
23 *filename++ = '/';
24
25 while(res >= (int)sizeof(*event)) {
26     event = (struct inotify_event *)(event_buf + event_pos);
27     //printf("%d: %08x | %s| \n", event->wd, event->mask, event->len ? event->name : "");
28     if(event->len) {
29         strcpy(filename, event->name);
30         if(event->mask & IN_CREATE) {
31             openDeviceLocked(devname);
32         } else {
33             ALOGI("Removing device '%s' due to inotify event\n", devname);
34             closeDeviceByPathLocked(devname);
35         }
36     }
37     event_size = sizeof(*event) + event->len;
38     res -= event_size;
39     event_pos += event_size;
40 }
41 return 0;
42 }
```

1

deviceChanged被设置为true的地方有三处：1.读取Device到readBuffer中的内容为空或者产生ENODEV的错误时；2.Device事件为EPOLLHUP（被挂起）时；3.发生Inotify事件，处理完mClosingDevices，mOpeningDevices，mDevices等数据结构的变化及Device的信息初始化操作后。continue表示重新进入getEvents函数主循环，根据设备变化事件生成相应的RawEvent。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 ...
2 if (deviceChanged) {
3     continue;
4 }
```

第二次进入getEvents函数，可能由于没有生成RawEvent，导致event的值和buffer的值相等,所以不会进入break阶段。生成RawEvent的地方有：1.进入getEvents函数时mNeedToReopenDevices为true，表示需要重启设备；2.第一次进入getEvents函数，需要扫描/dev/input/下面的设备；3.mNeedToSendFinishedDeviceScan为true,表示生成扫描完成事件，发生在重启设备，首次扫描设备，设备新增和卸载阶段。也就是说，第二次进入getEvents函数时，只要不发生上述事件，event的值和buffer的值就会相等。将mPendingEventIndex置0，是为了能使下次进入getEvents函数能进入while (mPendingEventIndex < mPendingEventCount)循环，之后流程会走到epoll\_wait阶段。epoll\_wait监控了Inotify的fd,管道读端的fd和设备的fd。epoll\_wait可能会发生阻塞，因为第四个参数值可能为-1，而设备那边真的风平浪静没有任何动作发生。这样，epoll\_wait函数便不能返回。Android设置了EventHub::wake往管道写端写入一个 ‘w’ 以唤醒epoll\_wait，使epoll\_wait函数得以返回，以重新进入getEvents函数主循环，重新进入主循环后会在” if (event != buffer || awoken)” 处返回。发生break退出getEvents函数主循环的地方有以下几处：1.input\_event结构体数组readBuffer被塞满；2.重启设备；3.生成了RawEvent事件；4.管道唤醒操作；5.epoll\_wait返回值为0。epoll\_wait之所以设置在” if (event != buffer || awoken)” 之后是因为生成了RawEvent就不用再监听了，直接返回。没有生成就继续监听，以期待进入主循环生成RawEvent。getEvents函数最终返回生成的RawEvent数量。

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 if (event != buffer || awoken) {
2     break;
3 }
4 ...
5 mPendingEventIndex = 0;
6
7 mLock.unlock(); // release lock before poll, must be before release_wake_lock
8 release_wake_lock(WAKE_LOCK_ID);
9
10 int pollResult = epoll_wait(mEpollFd, mPendingEventItems, EPOLL_MAX_EVENTS, timeoutMillis);
11
12 acquire_wake_lock(PARTIAL_WAKE_LOCK, WAKE_LOCK_ID);
13 mLock.lock(); // reacquire lock after poll, must be after acquire_wake_lock
14
15 if (pollResult == 0) {
16     // Timed out.
17     mPendingEventCount = 0;
18     break;
19 }
20
21 if (pollResult < 0) {
22     // An error occurred.
23     mPendingEventCount = 0;
24 }
```

```
27         if (errno != EINTR) {
28             ALOGW("poll failed (errno=%d)\n", errno);
29             usleep(100000);
30         }
31     } else {
32         // Some events occurred.
33         mPendingEventCount = size_t(pollResult);
34     }
35 }
36
37 // All done, return the number of events we read.
38 return event - buffer;
39 }
```

*/frameworks/native/services/inputflinger/EventHub.cpp*

```
1 void EventHub::wake() {
2     ALOGV("wake() called");
3
4     ssize_t nWrite;
5     do {
6         nWrite = write(mWakeWritePipeFd, "W", 1);
7     } while (nWrite == -1 && errno == EINTR);
8
9     if (nWrite != 1 && errno != EAGAIN) {
10         ALOGW("Could not write wake signal, errno=%d", errno);
11     }
12 }
```

## 总结

首次进入getEvents函数的流程是扫描/dev/input下的设备，并添加到epoll监控队列中。第二次进入getEvents函数时，启用epoll\_wait进行监听,之后进入主循环。首次循环中，根据epoll\_wait返回的结果，在循环中生成对应的RawEvent，若有RawEvent，getEvents函数会返回等待第三次进入。若没有RawEvent，会进入二次循环，此时epoll\_wait可能又带来了新的事件让程序去生成RawEvent。。。 （这里忽略了一些个别情况）。



想对作者说点什么

### Android6.0输入系统之EventHub源码分析

阅读数 579

上篇文章《Android6.0输入系统之InputManagerService构成分析》完成了IMS的创建，接着就沿着输入系统这条路...

博文

来自： [我是一个Xiao菜鸟...](#)

### Android 输入系统（二）EventHub

阅读数 1万+

接着上一篇的InputManagerService，这里主要介绍一下EventHub。EventHub主要是访问/dev/input下端所有设...

博文

来自： [奋斗的菜鸟ing](#)

### Android输入事件流程中的EventHub分析及源码演示

阅读数 2万+

Android2.3的输入事件流程与以前版本有了较大的不同，这里做一下详细的分析，最后我把自己分析时用的演示代码...

博文

来自： [农场老马的专栏](#)

### EventHub

阅读数 646

在EventHub的构造函数中：mEpollFd=epoll\_create(Epoll\_SIZE\_HINT);mINotifyFd=inotify\_init();//DEVICE\_PA...

博文

来自： [woailuojp的博客](#)

### EventHub分析(二)

阅读数 1722

2,EventHub一台移动设备中能产生输入消息的部件很多,比如键盘,触摸屏以及按键等等,EventHub主要是将这些设备...

博文

来自： [Jack的博客](#)

### input子系统三 input系统启动和EventHub数据读取

阅读数 520

一、框架介绍由下图可以看出，在系统服务启动时会通过InputManager启动InputReader和InputDispatcher，创建...

博文

来自： [frank\\_zyp的博客](#)

### Vue2.0 事件发射与接收

阅读数 2万+

由于vue2.0移除了1.0中的\$dispatch和\$broadcast这两个组件之间通信传递数据的方法,官方的给出的最简单的升级...

博文

来自： [Riptide](#)

### Android Framework Input 机制分析

阅读数 827

App进程的Java层的ViewRoot对象，请求与底层建立通信，通过Binder机制调用WindowManagerService|进而转...

博文

来自： [Ron的专栏](#)

### Android Input流程分析(三):InputReader - Invoker123...\_CSDN博客

11-9

### Android input子系统之InputReader获取输入事件详细分...\_CSDN博客

11-9

Android Input流程分析(二):EventHub

现在,InputReader线程已经开始运行 来自: Invoker123的博客

Android 输入系统之EventHub篇 - 愿景的博客(android系统定...

登录

注册

×



<b>input之上层（二）</b>	阅读数 393	
在上篇blog中以InputReaderThread为入口，主要了解了一下getEvents这个函数，接下来还有很多函数在等着我们... 博文 来自： <a href="#">s_jason的博客</a>		
<b>Android Input System分析（一）--基本架构</b>	阅读数 603	1
说明：本文中诸多图片均来源于网络，如有冒犯，请谅解。开始之前，我们先从整个android大的架构来俯视一遍inp... 博文 来自： <a href="#">u013543848的博客</a>		
<b>EventHub分析(二) - Jack的博客 - CSDN博客</b>		10-25
Android Input流程分析(二):EventHub 09-05 351 Android Input流程分析(二):EventHub 现在,InputReader线程已经开始运行 来自: Invoker123的博客 vi		
<b>Android输入事件流程中的EventHub分析及源码演示 - 农..._CSDN博客</b>		12-6
Android Input流程分析(二):EventHub 现在,InputReader线程已经开始运行 来自: Invoker123的博客 Android6.0输入系统之EventHub源码分析 07-10 25 ..		



**new\_abc**  
581篇文章  
[关注](#) 排名:871



**刘望舒**  
271篇文章  
[关注](#) 排名:716



**东月之神**  
197篇文章  
[关注](#) 排名:2000+



**小码哥\_WS**  
135篇文章  
[关注](#) 排名:千里之外

<b>Android输入系统(一)</b>	阅读数 204	
Android输入系统(一)首先我们明白，在PC或者手机上我们都支持热插拔，比如现在有个键盘，现在键盘插入USB接... 博文 来自： <a href="#">我叫王菜鸟</a>		
<b>Android 输入系统(二)EventHub - 奋斗的菜鸟ing - CSDN博客</b>		11-2
Android Input流程分析(二):EventHub 09-05 355 Android Input流程分析(二):EventHub 现在,InputReader线程已经开始运行 来自: Invoker123的博客 vue ...		
<b>EventHub - woailuojp的博客 - CSDN博客</b>		10-22
Android Input流程分析(二):EventHub 09-05 344 Android Input流程分析(二):EventHub 现在,InputReader线程已经开始运行 来自: Invoker123的博客 vue ...		

<b>Android层面上对sensor及event事件的处理</b>	阅读数 9273	
可能这篇总结写得会有一点凌乱，一会讲sensor，一会又讲event的。但是把两者摆在一起也是有原因的，sensor的... 博文 来自： <a href="#">hbk320的专栏</a>		

<b>【Android休眠】之Android对PowerKey事件的处理(2)EventHub</b>	阅读数 1986	
EventHub是Android中Input事件的处理中心，完成kernel上报事件的读取、初步处理、传递。(1)读取：Input设备... 博文 来自： <a href="#">u013686019的专栏</a>		

<b>Android Input Framework(二)---EventHub - qq_1733415..._CSDN博客</b>		12-2
<b>Android 输入事件系统之 EventHub 和 Input Lib(事件解..._CSDN博客</b>		10-22
Android Input流程分析(二):EventHub 09-05 344 Android Input流程分析(二):EventHub 现在,InputReader线程已经开始运行 来自: Invoker123的博客 Andro...		

<b>Input调用流程(好文)</b>	阅读数 647	
原址 先介绍一下每个模块的工作职责：EventHub,InputReader,InputManager...1模块功能1.1EventHub 它是... 博文 来自： <a href="#">unbroken</a>		

<b>Android触摸事件的传递(三)--输入系统EventHub - weixi..._CSDN博客</b>		2-18
Android Input流程分析(二):EventHub 09-05 阅读数 457 AndroidInput流程分析...EventHub 现在,InputReader线程已经开始运行 博文 来自: Invoker123的博客 ...		

<b>Android 4.0 事件输入(Event Input)系统</b>	阅读数 1170	
1.TouchScreen功能在Android4.0下不工作 原来在Android2.3.5下能正常工作的TouchScreen功能，移植到And... 博文		

<b>在 Android 4.1上，分析 input -- android framework 部分 2</b>	阅读数 4687	
Android4.0 input touch解析前言：在网上看到好多关于android input device流程分析，但是都不全，有的只是从l... 博文 来自： <a href="#">快乐&amp;&amp;平凡</a>		

<b>Android系统源码阅读（11）：Android的InputManagerService的工作过程</b>	阅读数 1529	
Android系统源码阅读（11）：Android应用的InputManagerService的工作过程请对照AOSP版本：6.0.1_r50。1... 博文 来自： <a href="#">天天吃吃</a>		

<b>Android6.0 按键流程（二）KeyboardInputMapper扫描码转成键盘码</b>	阅读数 3636	
在上一篇博客中，我们详细分析了InputReader中读取设备事件，到processEventsLocked函数处理事件（包括设备... 博文 来自： <a href="#">kc58236582的博客</a>		

<b>Android6.0 按键kl文件加载过程分析</b>	阅读数 7622	
在之前按键过程分析的几篇博客中，我分析过关于按键kl文件的加载，但是讲的不是非常详细，这篇博客主要把kl文... 博文 来自： <a href="#">kc58236582的博客</a>		

<b>Android 4.2 Input Event事件处理流程&lt;一&gt;事情派发</b>	阅读数 6287	
事件的开始是从eventhub开始的，我们先来看下流程图 博文 来自： <a href="#">new_abc的专栏</a>		

<b>Android OTA升级原理和流程分析（四）---Android系统Recovery模式的工作原理</b>	阅读数 5484	
AndroidOTA升级原理和流程分析（四）---Android系统Recovery模式的工作原理 在使用update.zip包升级时怎... 博文 来自： <a href="#">ylyuanlu的专栏</a>		

识别和匹配idc配置文件

voidEventHub::loadConfigurationLocked(Device\*device){device->configurationFile=getInputDeviceConfigu...

博文 来自： [coldsnow33的专栏](#)

vuejs组件通信的eventHub(巴士)

eventHub以我理解，他有点像vuexeventHub是什么？但是又是缩小版的，可以说是乞丐版，因为他所操作的东西...

博文 来自： [dexing07的博客](#)

Android 输入系统之EventHub篇

做Android系统定制两年多了，受到别人启发，将自己的学习工作经历整理成博客，供以后重温，好了废话不多说，...

博文 来自： [愿景的博客](#)

Android 输入事件系统之 EventHub 和 Input Lib(事件解析库)

从 Android事件输入系统整体框架 一文可知InputLibs是一个事件解析库，完成事件解析、keycode转换，设备配置...

博文 来自： [学无止境，静下心...](#)

Android input子系统之InputReader获取输入事件详细分析 - - EventHub->getevents

此文章只分析EventHub获取输入事件的getevents函数的具体实现首先在EventHub的构造函数中,将以下变量进行初...

博文 来自： [qq\\_30025621的博客](#)

vue data: { eventHub: new Vue() }

由于vue2.0移除了1.0中的\$dispatch和\$broadcast这两个组件之间通信传递数据的方法,官方的给出的最简单的升级...

博文 来自： [人民群众小学生](#)

Android6.0 按键流程（一）读取设备事件

之前我们整理过一篇按键流程，但是不是太详细有的细节的地方有遗漏，今天在Android6.0上重新总结下，先从读取...

博文 来自： [kc58236582的博客](#)

对 “Android输入事件流程中的EventHub分析及源码演示” 的补充

前面在工作涉及到Android的事件输入流程时，曾经以单点触摸为例，写过一篇文章介绍EventHub是如何从输入设...

博文 来自： [农场老马的专栏](#)

Android 4.1 Input设备流程分析

Android 4.1 Input设备流程分析包含触摸流程的详细分析图

03-25  
下载

InputManagerService之事件的初始化与分发

该篇文章接总章，来详细谈论说明InputManagerService体系，从初始化到事件获取跟分发。咱们在进行前，先明确...

博文 来自： [李云轩的专栏](#)

向系统注入两点触摸事件

之前遇到的问题：向系统发送一个长按触摸事件，再发送其他触摸事件时之前的长按事件消失。不多说，直接上代码...

博文 来自： [沉淀之路—Android](#)

Android OTA升级原理和流程分析（七）---Recovery服务的核心install\_package函数

Android系统Recovery工作原理之使用update.zip升级过程分析（七）---Recovery服务的核心install\_package函数...

博文 来自： [ylyuanlu的专栏](#)

Android5.0 按键kl文件加载过程分析

在之前按键过程分析的几篇博客中，我分析过关于按键kl文件的加载，但是讲的不是非常详细，这篇博客主要把kl文...

博文 来自： [100度多0点01度](#)

android学习笔记6 eventhub

Eventhub是在Native的inputmanager创建的时候被创建的，inputreadthread就通过eventhub的getevent方法来...

博文 来自： [shadow\\_dance的...](#)

EventHub演示程序及源码

这个资源里面的源码及可执行文件是用于这里这篇文档的: <http://blog.csdn.net/a345017062/archive/2011/05/13/6417929.aspx> 文档中对A...

05-13  
下载

eventhub

<http://blog.csdn.net/myarrow/article/details/7091061>frameworks/base/services/input/EventHub.cpp如何获...

博文 来自： [u013039170的专栏](#)

Android目录Frameworks下base/libs/ui/EventHub.cpp中加键盘布局文件名称的注释

文件frameworks/base/libs/ui/EventHub.cpp的intEventHub::openDevice(constchar\*deviceName)函数约775行...

博文 来自： [如是观的专栏](#)

EventHub与设备、Input事件的交互

关于EventHub的学习有一点前置知识很重要，就是epoll和inotify机制，可以参考这篇博文：[www.cheelok.com/\\_i...](http://www.cheelok.com/_i...)

博文 来自： [weixin\\_34261739...](#)

input子系统整体流程全面分析（触摸屏驱动为例）

input输入子系统整体流程 input子系统在内核中的实现，包括输入子系统（InputCore），事件处理层（Ev...

博文 来自： [u012497906的专栏](#)

Android6.0 按键流程 KeyboardInputMapper扫描码转成键盘码 (二)

我们详细分析了InputReader中读取设备事件，到processEventsLocked函数处理事件（包括设备事件，设备添加、...

博文 来自： [dk\\_work的博客](#)

Android系统input按键处理流程（从驱动到framework）

（暂时列出提纲，后续添加具体内容）涉及到的几个文件：1.out/target/product/rkpx2/system/usr/keylayout/q...

博文 来自： [jwq2011的专栏](#)

Android 事件驱动流程

Android input event Android 事件驱动流程

04-01

下载

1

Android的Input流程分析(好文)

阅读数 336

原址参考其实Android5.0中事件输入子系统的框架和流程没有本质变化。Service端的实现在/frameworks/native/s...

博文 来自： unbroken

Android Input流程分析（三）：InputReader

阅读数 765

回到InputReader的loopOnce函数。 现在getEvents捞上来的RawEvent均保存在mEventBuffer中。/native/se...

博文 来自： Invoker123的博客

Android Input Framework(二)---EventHub

阅读数 401

Android Input Framework(二)---EventHub (2012-09-17 10:20:39)转载▼ 标签： 杂谈 分类： android 1 EventHub ...

博文 来自： qq\_17334155的专栏

Android系统启动流程（二）解析Zygote进程启动过程

阅读数 5659

上一篇文章我们分析了init进程，init进程中主要做了三件事，其中一件就是创建了Zygote进程，那么Zygote进程是...

博文 来自： 刘望舒的专栏

android 是如何找到触摸屏设备节点

阅读数 3294

1.触摸屏属于input设备，住、主设备号是固定是13，但是次设备号，应该不是固定。2.android会遍历/dev/input,进...

博文 来自： xiayu98020214的...



这款传奇超刺激，十倍爆率上线送，一刀一怪随便点！

AOSP源码分析：Android Input事件的产生、读取和分发

阅读数 861

大家好，今天为大家推荐来自MIUI的Cheeeelok同学的AOSP源码分析系列文章，本文依然从源码的角度带大家理解...

博文 来自： 技术视界

应用层分发事件分析(七)

阅读数 1027

1概念从硬件的角度看,事件主要分为以下几类:1,按键事件(KeyEvent)2,触摸事件(TouchEvent)3,鼠标事件(MouseEv...

博文 来自： Jack的博客

基于PyTorch的深度学习入门教程（六）——数据并行化

阅读数 4720

前言本文参考PyTorch官网的教程，分为五个基本模块来介绍PyTorch。为了避免文章过长，这五个模块分别在五篇...

博文 来自： 雁回晴空的博客专栏

Android实现QQ分享及注意事项

阅读数 5547

一、获取APPID和帮助文档可以参看新手引导和接入说明：http://wiki.open.qq.com/wiki/移动应用接入wiki索引分...

博文 来自： 水寒

CNN笔记：通俗理解卷积神经网络

阅读数 21万+

通俗理解卷积神经网络（cs231n与5月dl班课程笔记） 1 前言

2012年我在北京组织过8期machine l...

博文 来自： 结构之法 算法之道

Android开发本地及网络Mp3音乐播放器(十二)创建NetMusicListAdapter、SearchResult显示...

阅读数 9291

实现功能： 实现NetMusicListAdapter（网络音乐列表适配器） 实现SearchResult（搜索音乐对象） 使用Jsoup组...

博文 来自： iwanghang(一个播...

【小程序】微信小程序开发实践

阅读数 25万+

帐号相关流程注册范围 企业 政府 媒体 其他组织换句话说讲就是不让个人开发者注册。:)填写企业信息不能使用和之前...

博文 来自： 小雨同学的技术博客

HttpClient使用详解

阅读数 80万+

Http协议的重要性相信不用我多说了，HttpClient相比传统JDK自带的URLConnection，增加了易用性和灵活性（具...

博文 来自： 鹏霄万里展雄飞

android客户端与服务器端交互 如何保持session

阅读数 4万+

最近在开发项目的过程中，遇到android与web服务器要在同一session下通信的问题。 在解决问题前先回顾下Sessi...

博文 来自： charming的专栏

三菱FX系列PLC与PC通讯的实现之专有协议（计算机联接）的程序设计之一

阅读数 1万+

阅读内容为：FX系列微型可编程控制器用户手册（通讯篇）中计算机链接功能章节。采用本方法通信，pc端的实现...

博文 来自： pengjc2001的博客

如何在ArcGIS Online中构建自己的应用程序模板初级篇-显示地图

阅读数 4万+

开发ArcGIS Online应用程序模板之前，需要了解怎么使用ArcGIS API for JavaScript。 在ArcGIS Online当中如...

博文 来自： ArcGIS产品与技术...

再谈iOS 7的手势滑动返回功能

阅读数 8万+

之前随手写过一篇《使用UIScreenEdgePanGestureRecognizer实现swipe to pop效果》，挺粗糙的。现在使用默...

博文 来自： JasonLee的专栏

[ASP.NET]二维码的创建

阅读数 5439

又好一段时间没有写东西了，继续回归原来的模式，多做记录，最近要实现个unity的二维码方面的功能，首先就要...

博文 来自： 学无止境的专栏

jquery/js实现一个网页同时调用多个倒计时(最新的)

阅读数 44万+

jquery/js实现一个网页同时调用多个倒计时(最新的) 最近需要网页添加多个倒计时. 查阅网络,基本上都是千篇一律的...

博文 来自： Websites

将Excel文件导入数据库（POI+Excel+MySQL+jsp页面导入）第一次优化

阅读数 3万+










[关于我们](#) [招聘](#) [广告服务](#) [网站地图](#)

 百度提供站内搜索 京ICP备19004658号  
©1999-2019 北京创新乐知网络技术有限公司

[网络110报警服务](#) [经营性网站备案信息](#)  
[北京互联网违法和不良信息举报中心](#)  
[中国互联网举报中心](#) [家长监护](#)

1