Android Input Architecture

Last edited by **caoquanli** 1 month ago

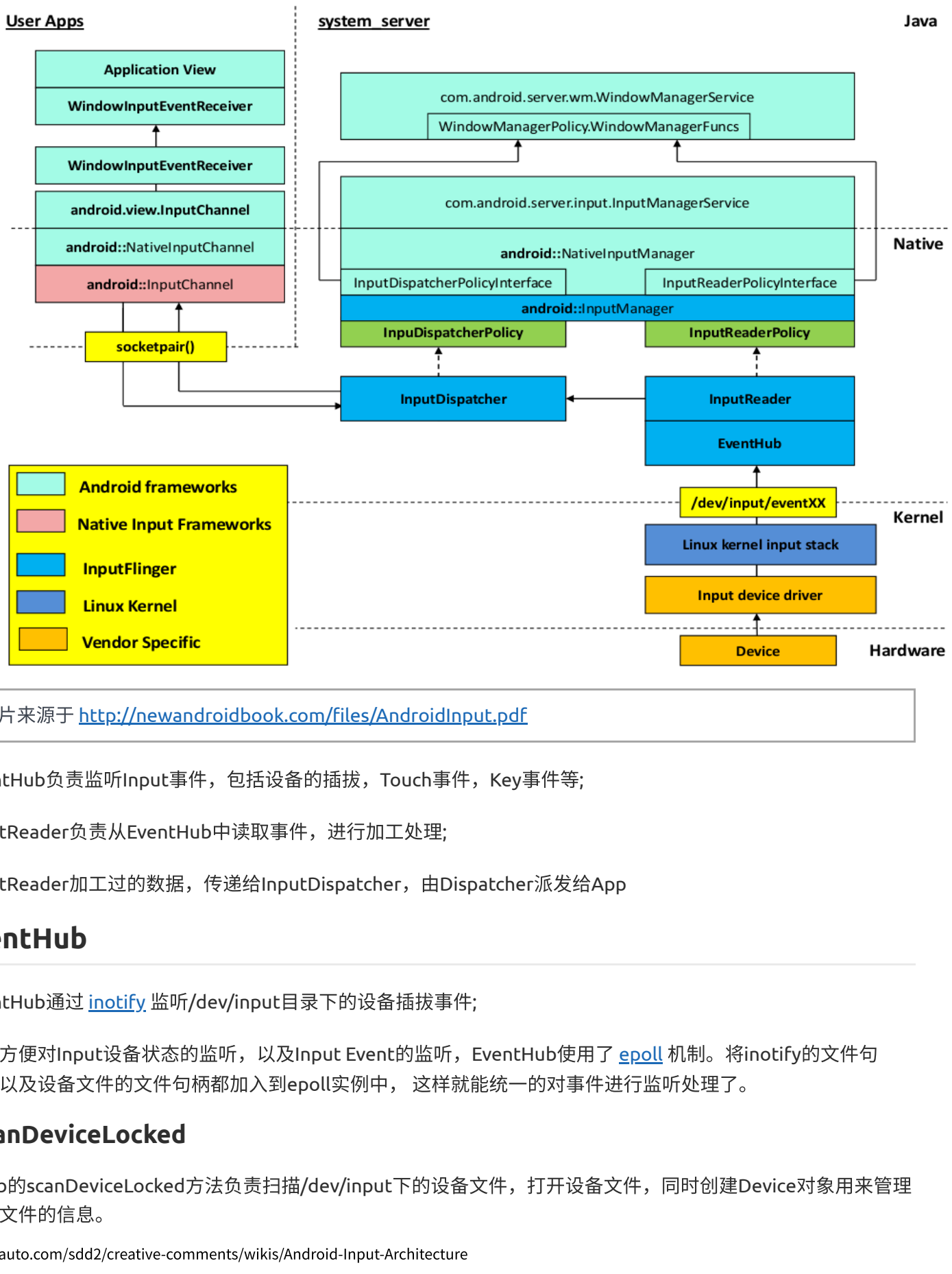
Android Input Architecture

Table of Contents

这篇文章用来记载Input模块的知识点。

1. 模块图一览

Input模块负责从输入设备，比如Touch屏幕，键盘等设备读取输入事件，经过处理之后传递给android系统或者是android app进行相应的处理。 以下是Input模块的整体方框图：



> 此图片来源于 http://newandroidbook.com/files/AndroidInput.pdf

- EventHub负责监听Input事件，包括设备的插拔，Touch事件，Key事件等;
- InputReader负责从EventHub中读取事件，进行加工处理;
- InputReader加工过的数据，传递给InputDispatcher，由Dispatcher派发给App

2. EventHub

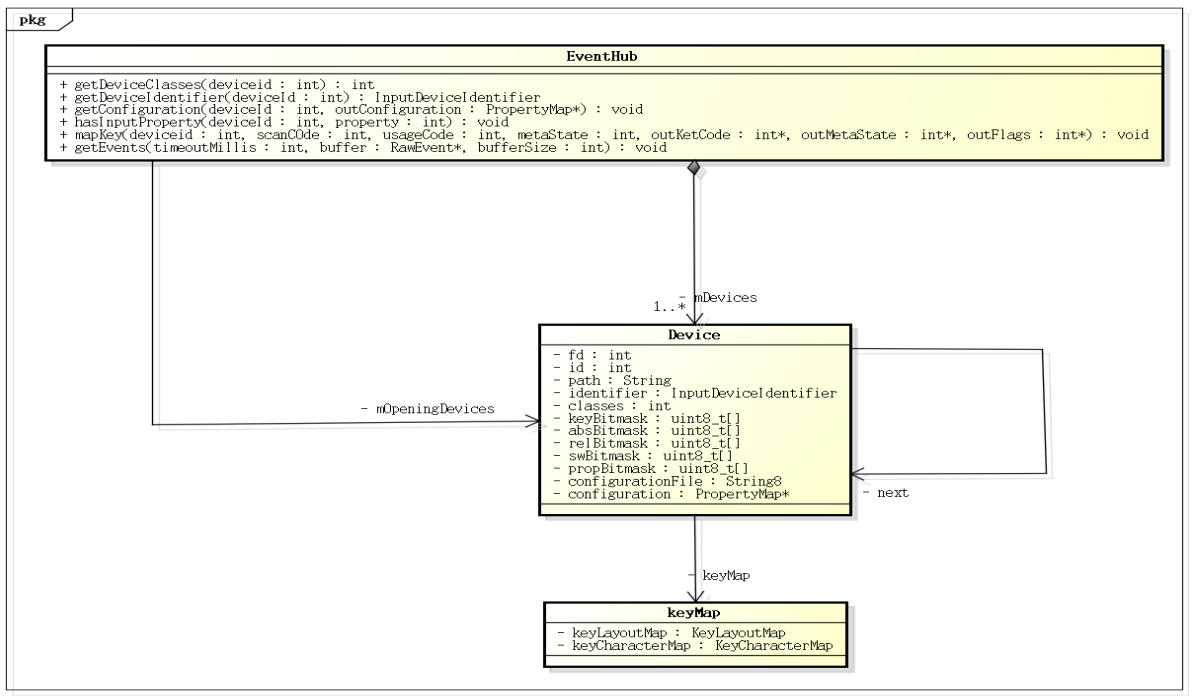
1. EventHub通过 inotify 监听/dev/input目录下的设备插拔事件;
2. 为了方便对Input设备状态的监听，以及Input Event的监听，EventHub使用了 epoll 机制。将inotify的文件句柄，以及设备文件的文件句柄都加入到epoll实例中， 这样就能统一的对事件进行监听处理了。

2.1. scanDeviceLocked

EventHub的scanDeviceLocked方法负责扫描/dev/input下的设备文件，打开设备文件，同时创建Device对象用来管理这些设备文件的信息。

- EventHub的openDeviceLOcked方法用来打开设备文件，并获取到设备的描述符以及相关的其他信息（版本，支持的按键类型等）。在此方法中会加载设备的配置文件：
    - loadConfigurationLocked方法用来匹配以及解析设备的配置文件。Input Device的配置文件名字以USB vendor，product id来合成，或者是设备的名字，配置文件 以 key = value 的格式书写。关于配置文件的详细信息可以参考： Input Device Configuration Files .
    - loadKeyMapLocked用来加载解析以下两个文件
        - Key Layout Files. 这个文件映射linux key code到android key code
        - Key Character Map Files 这个文件用来完成字符映射功能
- registerDeviceForEpollLocked将打开的设备文件句柄注册到epoll中，开始监听设备的事件

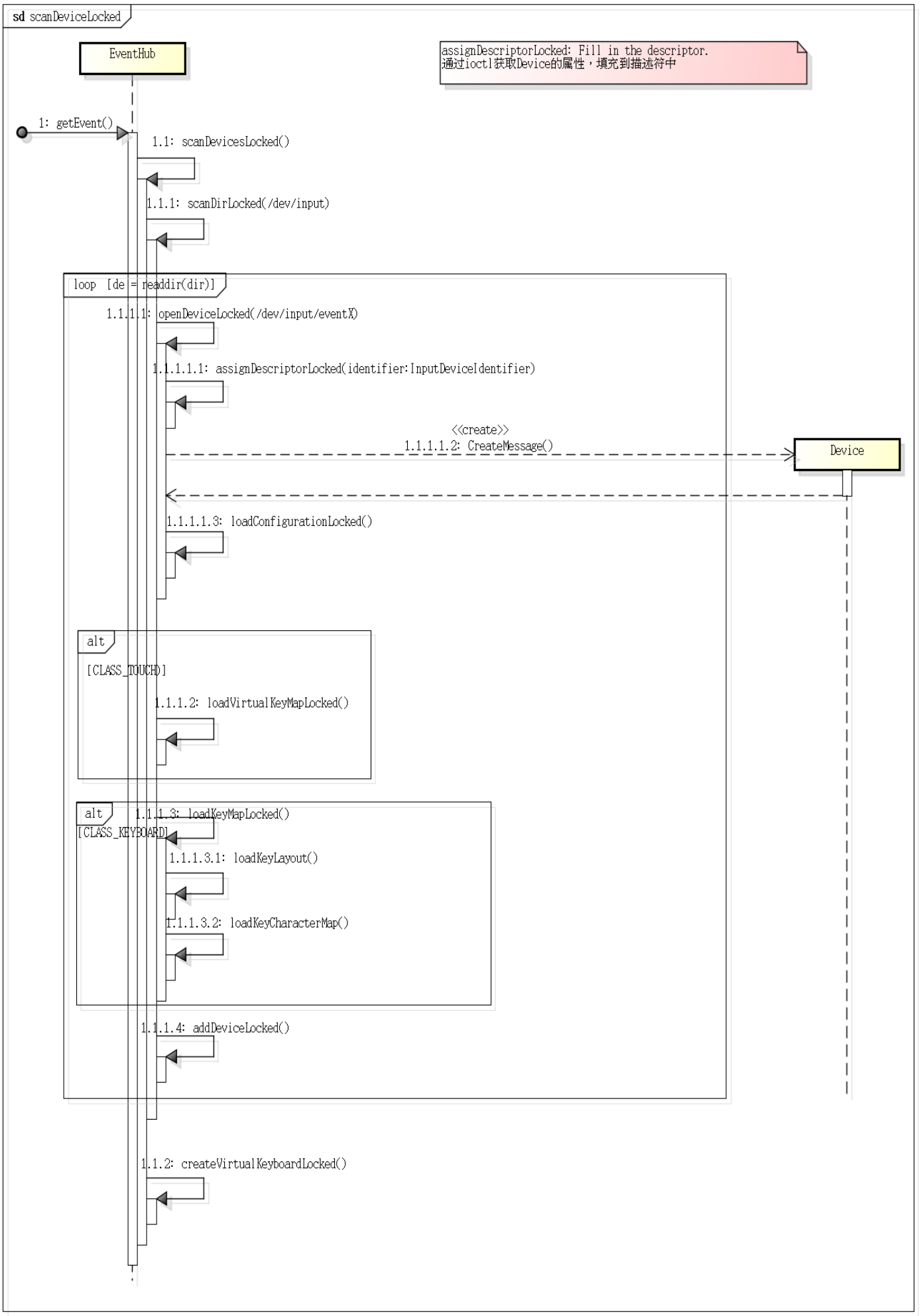
EventHub为每一个扫描到的设备生成对象Device，使用链表的方式管理Device对象。



关于Device对象中的几个关键成员变量的解释：

1. InputDeviceIdentifier：通过ioctl的方式获取device的属性，保存于此
2. configurationFile 存放配置文件的路径名，比如：/system/usr/idc/DEVICE_NAME.idc
3. configuration：解析完毕配置文件，将key，value保存于此
4. KeyMap对象用来保存key layout文件路径以及解析结果，key charactor 映射文件以及解析结果

scanDeviceLocked的时序如下如：



2.2. getEvent

在getEvent方法中，EventHub处于阻塞状态，直到超时或者dev文件中有了event数据。

```
int pollResult = epoll_wait(mEpollFd, mPendingEventItems, EPOLL_MAX_EVENTS, timeoutMillis)
```

epoll_wait监听文件的状态。当有文件状态可读的时候，EventHub调用read方法读取数据。dev的原始数据为：

```
struct input_event {
        struct timeval time;
        __u16 type;
        __u16 code;
        __s32 value;
};
```

在getEvent方法中，会将input_event转换巍RawEvent

```
struct RawEvent {
    nsecs_t when;
    int32_t deviceId;
    int32_t type;
    int32_t code;
```

```
    int32_t value;
};
```

input_event和RawEvent的对应关系如下图：

3. InputReader

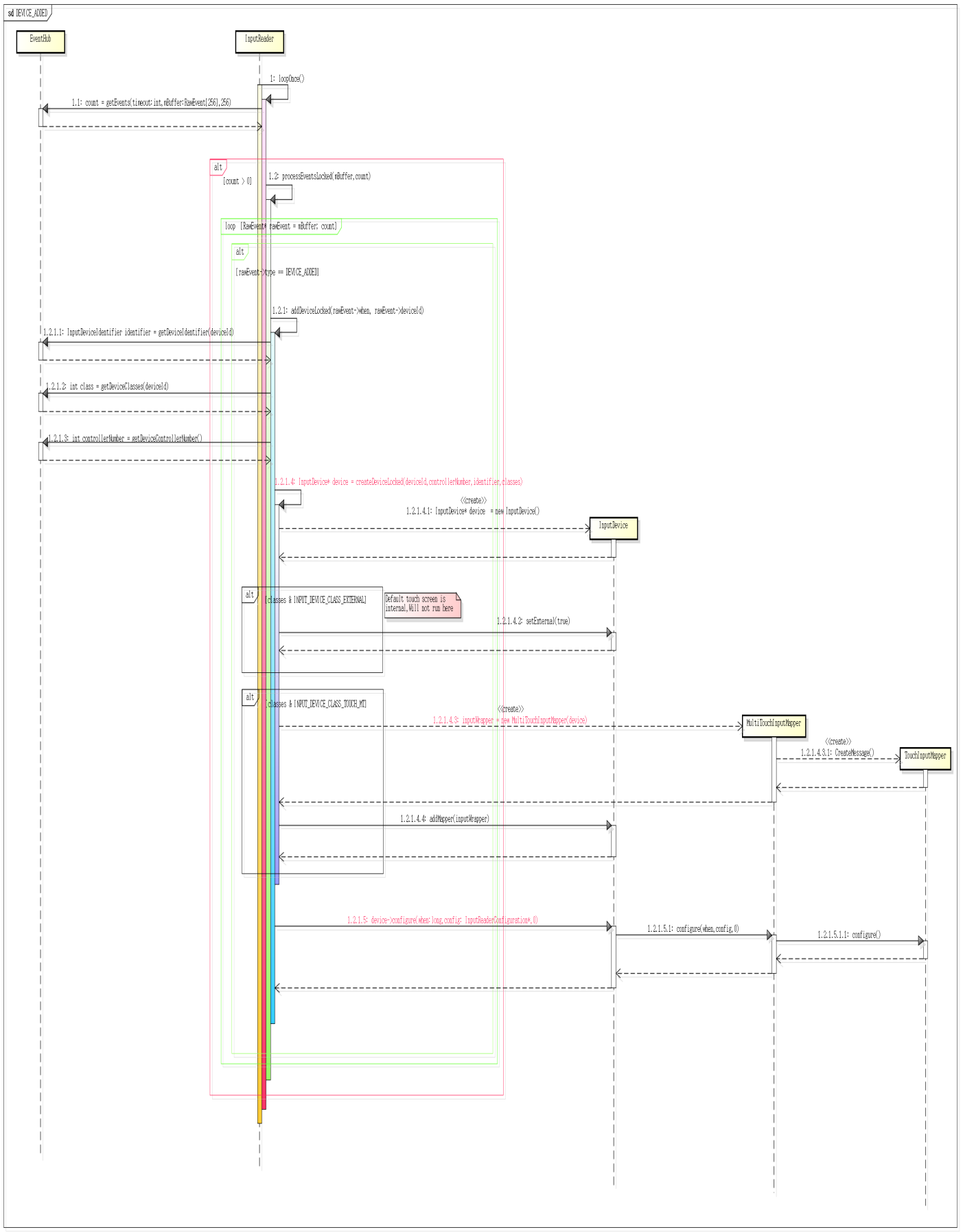
InputReaderThread线程在其threadLoop方法中调用InputReader的loopOnce方法，此方法循环的从EventHub中获取event事件。在processEventsLocked中对EventHub获取的RawEvent进行处理。 event的类型有：

- Input event，包括touch事件，key事件等
- DEVICE_ADDED,发现新的设备
- DEVICE_REMOVED,设备移除
- FINISHED_DEVICE_SCAN，设备扫描结束

这里以DEVICE_ADDED和Input event为例子，梳理下InputReader的流程

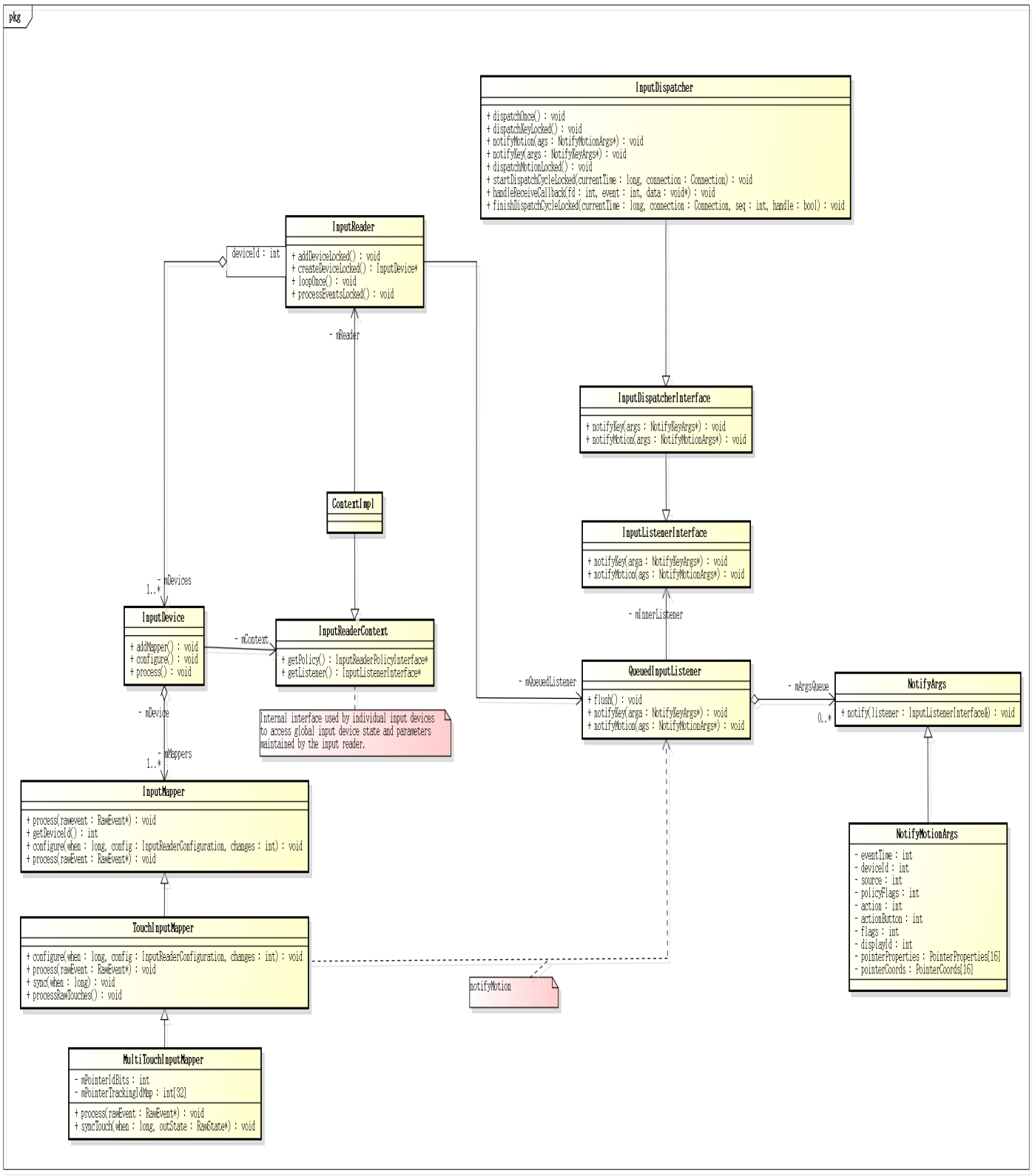
3.1. DEVICE_ADDED

在processEventsLocked方法中，对于RawEvent Type为DEVICE_ADDED的event，调用 void InputReader::addDeviceLocked(nsecs_t when, int32_t deviceId) 处理。 这个流程的时序图如下：



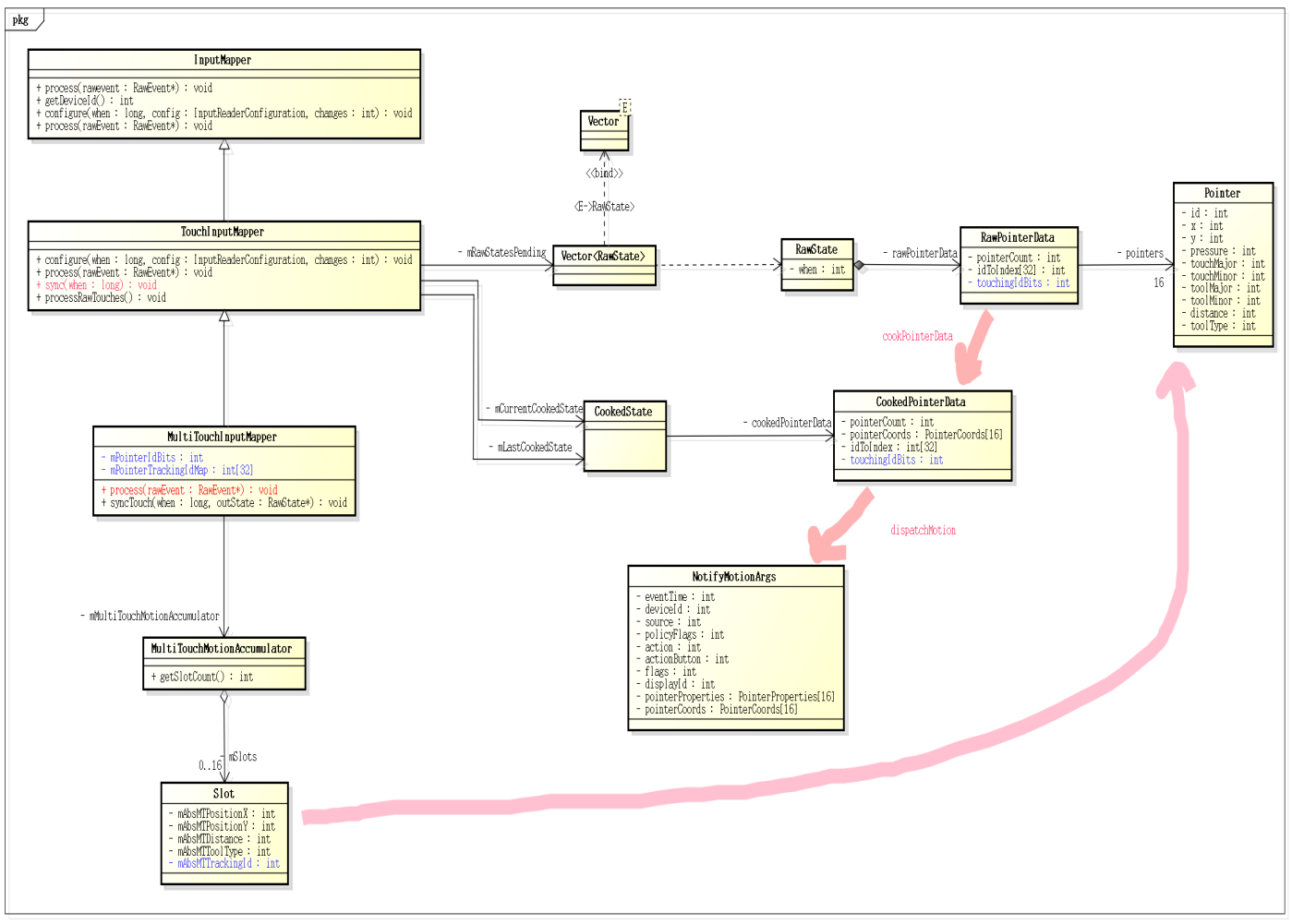
- InputReader会为每一个input device生成一个InputDevice对象
- 每一个InputDevice内部会含有一个或者多个InputMapper，这些Mapper用来解析input raw数据，合成一个后续InputDispatcher能处理的数据结构
- InputReader模块负责读取raw数据，然后根据device id找到对应的InputDevice，把raw数据送给InputDevice进行处理。处理完毕的数据封装成NotifyArgs，交给QueuedInputListener通知给InputDispatcher.

InputReader与InputDevice之间的关系，InputDevice与InputMapper之间的关系见下图：



3.2. Touch事件的加工

EventHub从device中读取出来的数据是Raw数据，之后将这些Raw数据传递给InputDevice进行处理，经过数据的分析和合成，最终生成NotifyArgs发送给InputDisoatcher。 下面是这个流程的block图：



- 每次Touch事件会存在多个event上报，每次touch事件的“分隔符”为SYN_REPORT。下面是touch事件上报数据的一个例子，从数据中可以数据包采取SYN_REPORT分隔，而每次数据包有多个event。

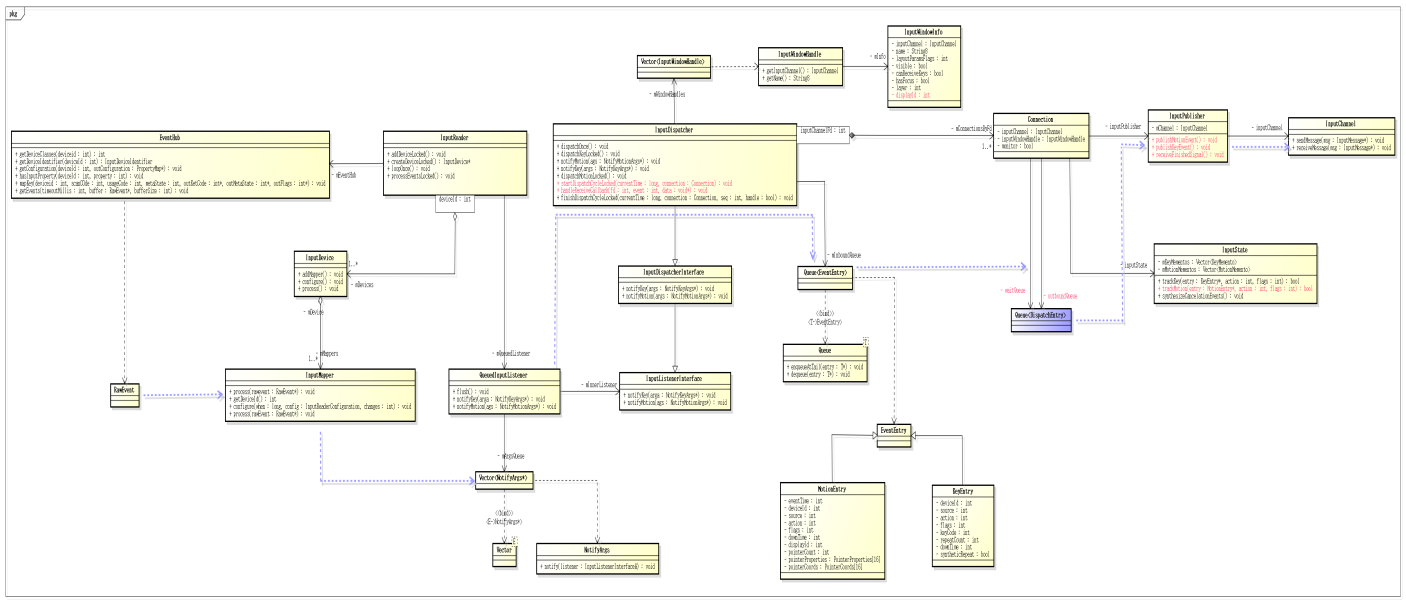
```
/dev/input/event0: EV_SYN       0004                000053f4
/dev/input/event0: EV_SYN       0005                32a4eab6
/dev/input/event0: EV_ABS       ABS_MT_TRACKING_ID  00000014
/dev/input/event0: EV_ABS       ABS_MT_POSITION_X   000002ed
/dev/input/event0: EV_ABS       ABS_MT_POSITION_Y   000003fb
/dev/input/event0: EV_ABS       ABS_MT_PRESSURE     00000038
/dev/input/event0: EV_ABS       ABS_MT_TOUCH_MAJOR  00000009
/dev/input/event0: EV_ABS       ABS_MT_TOUCH_MINOR  00000009
/dev/input/event0: EV_ABS       ABS_MT_SLOT         00000001
/dev/input/event0: EV_ABS       ABS_MT_TRACKING_ID  00000015
/dev/input/event0: EV_ABS       ABS_MT_POSITION_X   00000136
/dev/input/event0: EV_ABS       ABS_MT_POSITION_Y   00000486
/dev/input/event0: EV_ABS       ABS_MT_PRESSURE     00000035
/dev/input/event0: EV_ABS       ABS_MT_TOUCH_MINOR  00000007
/dev/input/event0: EV_SYN       SYN_REPORT          00000000

/dev/input/event0: EV_SYN       0004                000053f4
/dev/input/event0: EV_SYN       0005                371d30e1
/dev/input/event0: EV_ABS       ABS_MT_SLOT         00000000
/dev/input/event0: EV_ABS       ABS_MT_PRESSURE     0000003c
/dev/input/event0: EV_ABS       ABS_MT_SLOT         00000001
/dev/input/event0: EV_ABS       ABS_MT_POSITION_X   00000134
/dev/input/event0: EV_ABS       ABS_MT_PRESSURE     0000003c
/dev/input/event0: EV_ABS       ABS_MT_TOUCH_MAJOR  00000009
/dev/input/event0: EV_ABS       ABS_MT_TOUCH_MINOR  00000008
/dev/input/event0: EV_SYN       SYN_REPORT          00000000

/dev/input/event0: EV_SYN       0004                000053f4
/dev/input/event0: EV_SYN       0005                37983943
/dev/input/event0: EV_ABS       ABS_MT_POSITION_X   00000132
/dev/input/event0: EV_SYN       SYN_REPORT          00000000

/dev/input/event0: EV_SYN       0004                000053f4
/dev/input/event0: EV_SYN       0005                381b38a0
/dev/input/event0: EV_ABS       ABS_MT_POSITION_X   00000130
/dev/input/event0: EV_SYN       SYN_REPORT          00000000
/dev/input/event0: EV_SYN       0004                000053f4
/dev/input/event0: EV_SYN       0005                3895c0d5
/dev/input/event0: EV_ABS       ABS_MT_POSITION_X   0000012e
/dev/input/event0: EV_SYN       SYN_REPORT          00000000

/dev/input/event0: EV_SYN       0004                000053f4
/dev/input/event0: EV_SYN       0005                391794b7
/dev/input/event0: EV_ABS       ABS_MT_SLOT         00000000
/dev/input/event0: EV_ABS       ABS_MT_PRESSURE     0000003d
/dev/input/event0: EV_ABS       ABS_MT_SLOT         00000001
/dev/input/event0: EV_ABS       ABS_MT_POSITION_X   0000012c
/dev/input/event0: EV_ABS       ABS_MT_PRESSURE     0000003d
/dev/input/event0: EV_SYN       SYN_REPORT          00000000
```

- 对于支持Slot协议的Touch驱动，采取的是增量数据上报，也就是只传递发生了改变的数据。
- 每根手指驱动会分配一个trackid和一个slot号。MultiTouchInputMapper就是根据此slot号来保存每根手指的数据。比如第一根手指touch的时候，slot为0;第二根手指touch的时候slot为1，android最多支持16根手指。如果ABS_MT_SLOT没有发生变化，那么下一个数据包可以不发送ABS_MT_SLOT事件，这样可以节约空间。
- 如何判断手指是Down还是Up或者是Move？这个逻辑是在TouchInputMapper::dispatchTouches中完成。android使用bit位来保存手指的状态，对比bit位可以知道手指的Down或者Up。而move事件的判断需要对比坐标值是否发生了改变

4. InputDispatcher

- InputDispatcher维护当前的window handle信息,当接收到event事件的时候，从window handle列表中获取到可以处理event的窗口句柄，然后将event派发给这些window
- InputDispatcher负责从InputReader接收event信息，dispatcher内部维护了三个event对列: mInboundQueue维护接收到的event;Connection的outboundQueue维护等待派发的event队列;Connection的waitQueue队列维护已经派发出去但是尚未接收到App finish应答消息的event队列

InputDispatcher的整个模块图以及event事件流如下图所示：



上图中，蓝色虚箭头表示event事件流的处理流程。

5. Input ANR

InputDispatcher在派发Key事件(dispatchKeyLocked)和Motion事件(android-8.1.0_r28)的时候,都会选找到合适的窗口，以便把input事件派发到这些窗口。在这个过程中，会在 `checkWindowReadyForMoreInputLocked` 检查窗口的事件处理情况，如果发现key或者touch事件有挤压，就会走到 `handleTargetsNotReadyLocked` 这个流程。在 `handleTargetsNotReadyLocked` 方法中，如果还没来得及处理的事件超时了5S，就会调用`onANRLocked`出发anr的流程。

这里以Key事件的派发流程解释Anr发生的详细过程：

- 在 `void InputDispatcher::dispatchOnceInnerLocked(nsecs_t* nextWakeupTime)` 中，首先会给 `mPendingEvent` 变量赋值，也就是从当前的input队列中获取最开头的事件;
- 调用 `resetANRTimeoutsLocked` 方法，清除 `mInputTargetWaitCause` 和 `mInputTargetWaitApplicationHandle` 变量
- 调用 `dispatchKeyLocked` 开始进入派发事件阶段
    - 调用 `findFocusedWindowTargetsLocked` 找到当前的focus窗口
        - 调用 `checkWindowReadyForMoreInputLocked` 检查当前窗口是否有事件积压
        - 如果有事件积压，调用 `handleTargetsNotReadyLocked` ,在此方法中会给 `mInputTargetWaitStartTime` 和 `mInputTargetWaitTimeoutTime` 变量赋值，同时设置下次looper唤醒的时间为 `mInputTargetWaitTimeoutTime` ， `handleTargetsNotReadyLocked` 返回 `INPUT_EVENT_INJECTION_PENDING`
        - 由于check流程的失败，所以 `findFocusedWindowTargetsLocked` 没有成功，返回 `INPUT_EVENT_INJECTION_PENDING` 。这就导致 `dispatchKeyLocked` 不会进入真正的派发流程
- 由于 `dispatchKeyLocked` 的失败，导致 `dispatchOnceInnerLocked` 中不会清除 `mPendingEvent` 变量,清除代码如下：

```
    if (done) {
        if (dropReason != DROP_REASON_NOT_DROPPED) {
            dropInboundEventLocked(mPendingEvent, dropReason);
        }
        mLastDropReason = dropReason;

        releasePendingEventLocked();
        *nextWakeupTime = LONG_LONG_MIN;  // force next poll to wake up immediately
    }
```

- 当 `mInputTargetWaitTimeoutTime` 超时事件到之后， `InputDispatcher::dispatchOnceInnerLocked` 再次被调用。由于 `mPendingEvent` 变量此时不为空，所以不会再去input队列中重新获取事件，也不会运行 `resetANRTimeoutsLocked` 方法
- `dispatchKeyLocked` 方法再次运行。当再次运行到 `handleTargetsNotReadyLocked` 方法的时候，由于当前的事件超过了 `mInputTargetWaitTimeoutTime` ，所以进入 `onANRLocked` 处理流程