



Input 事件处理流程介绍

文档版本
发布日期

V1.0
2020-09-25

版权所有 © 紫光展锐（上海）科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐（上海）科技有限公司（以下简称紫光展锐）所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。

Unisoc Confidential For hiar

紫光展锐（上海）科技有限公司



前言

概述

本文档描述了 Android 系统中 Input 事件处理流程，包括 Input 事件读取、Input 事件派发、Input 事件分析。

读者对象

本文档主要适用于 Android 测试、开发人员。测试、开发人员必须具备以下经验和技能：


- 了解 Android 系统架构、各输入设备。

缩略语

缩略语	英文全名	中文解释
IMS	Input Manager Service	输入管理服务
WMS	Window Manager Service	窗口管理服务

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2020-09-25	第一次正式发布。

关键字

Input 搜索、Input 派发、Input 读取。

Unisoc Confidential For hiar

目 录

1 概述.....	1
1.1 IMS 简介.....	1
1.2 IMS 结构.....	2
1.3 IMS 初始化.....	4
2 Input 事件处理	5
2.1 Input 事件读取.....	5
2.2 Input 事件派发.....	7
3 Input 事件分析	9
3.1 Log 分析.....	9
3.1.1 Log 开启	9
3.1.2 Log 搜索	9
3.1.3 Log 分析	10
3.2 getevent 命令分析.....	11

Unisoc Confidential For hiar

图目录

图 1-1 IMS、WMS、ViewRootImpl 外部交互图	1
图 1-2 IMS、WMS、ViewRootImpl 内部交互图	2
图 1-3 IMS 结构图	3
图 2-1 InputReader 类处理过程	6
图 2-2 InputReader 消息结构变化流程图	7
图 2-3 InputDispatcher 类处理过程	8
图 3-1 搜索 Input 相关 Log 搜索项设置	10
图 3-2 \$ getevent -h 执行结果	11
图 3-3 \$ getevent 执行结果	12
图 3-4 \$ getevent -t /dev/input/event1 执行结果	12
图 3-5 \$ getevent -tlr /dev/input/event1 执行结果	13

Unisoc Confidential For hiar

1 概述

Android 系统中负责管理 Input 事件的主要是 IMS（Input Manager Service 输入管理服务）。

1.1 IMS 简介

IMS 的主要的任务是从设备中读取 Input 事件数据，然后将读取的事件发送到焦点窗口中去，同时还需要让系统有机会来处理一些系统按键。要完成这些工作，IMS 需要与其它模块交互，其中最主要的是 WMS（Window Manager Service 窗口管理服务）和 ViewRootImpl，各模块交互图详见图 1-1、图 1-2。

图1-1 IMS、WMS、ViewRootImpl 外部交互图

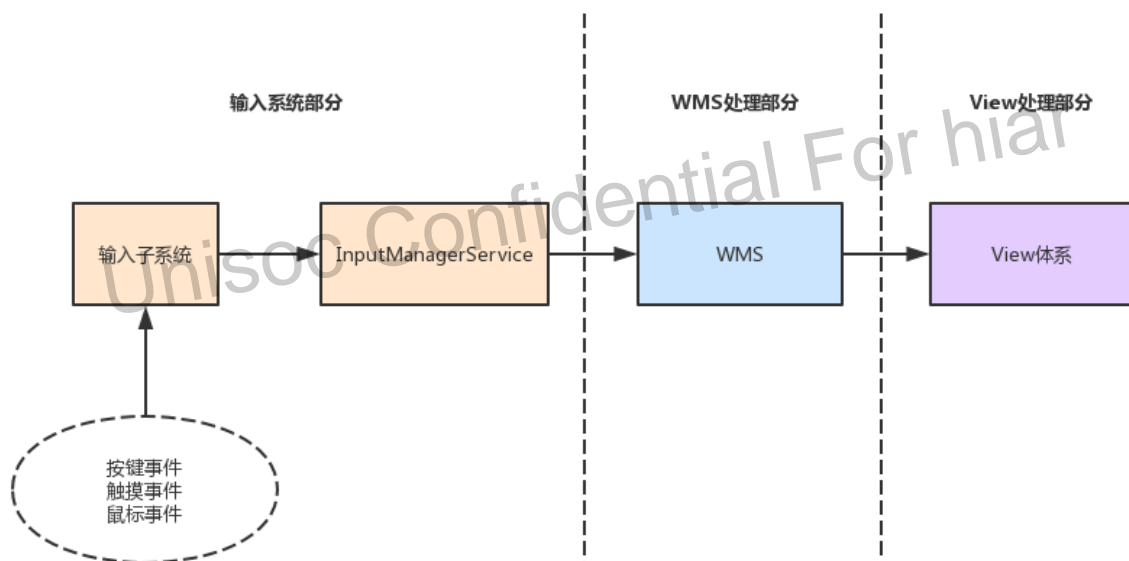
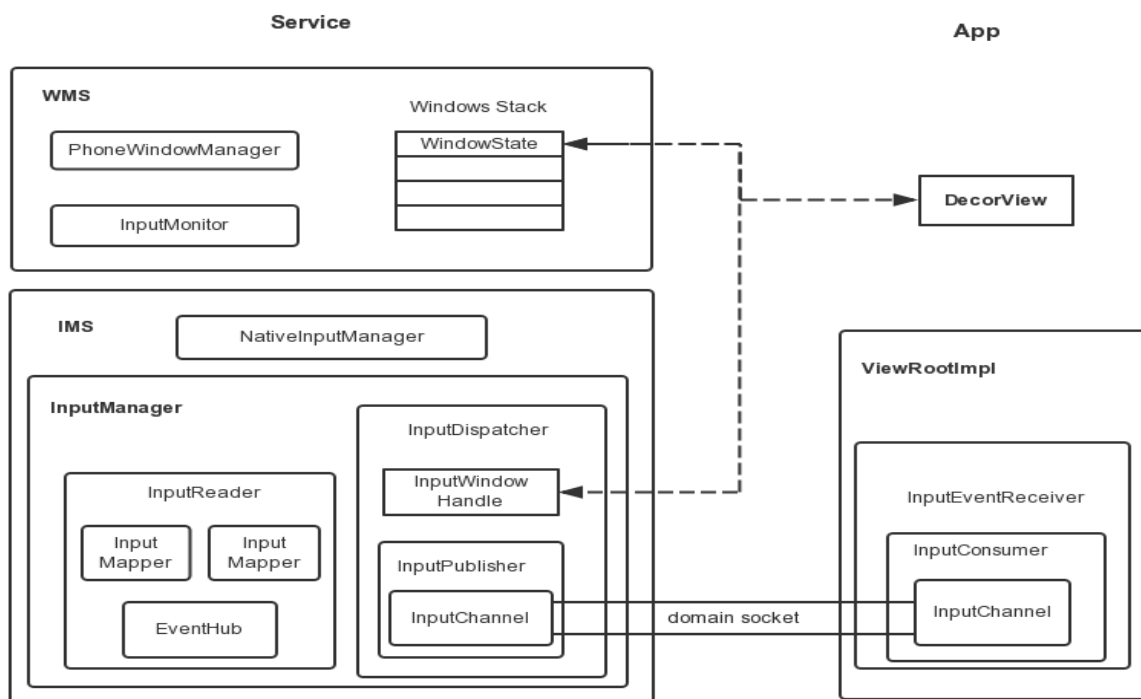


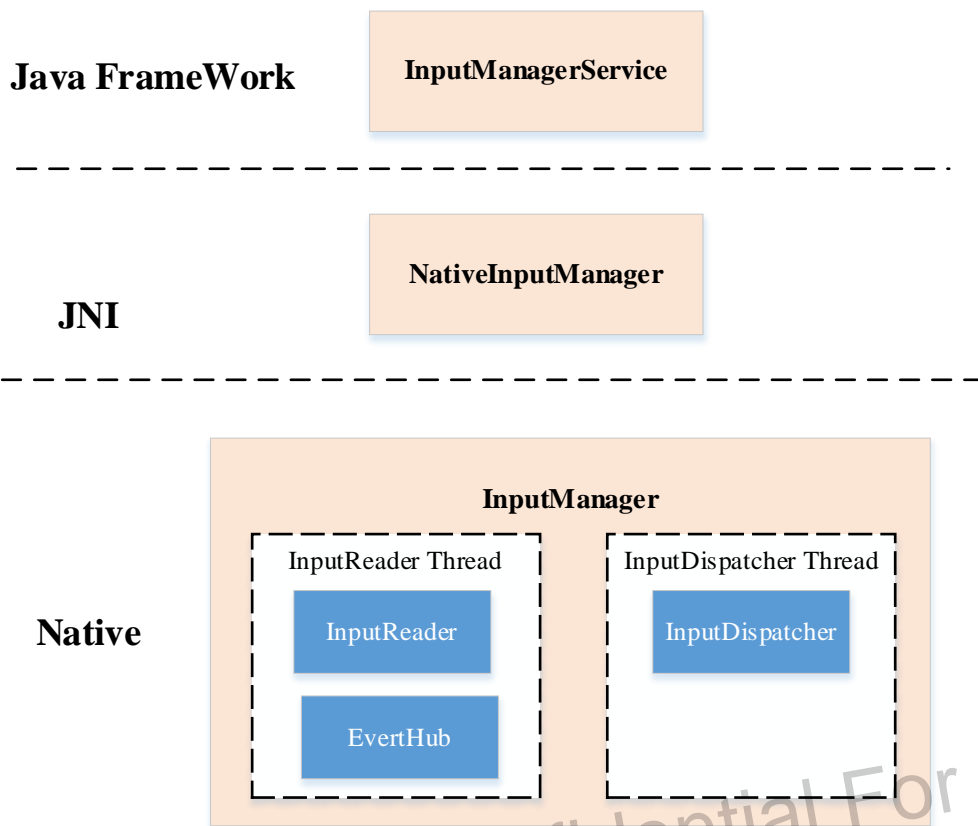
图1-2 IMS、WMS、ViewRootImpl 内部交互图



1.2 IMS 结构

IMS 结构详见图 1-3。

图1-3 IMS 结构图



InputManager 是输入控制中心，含有两个关键线程：

- **InputReaderThread**: 主要功能在 InputReader，用于从设备中读取事件，其中，EventHub 是输入设备的控制中心，直接与 Input 驱动交互，负责处理输入设备的增减、查询，输入事件的处理并向上层提供 getEvents()接口接收事件。
EventHub 的构造函数主要做三件事：
 - 创建 epoll 对象，之后就可以把多路输入设备的 fd（类似设备名称）挂到 epoll 上，等待输入事件。
 - 建立用于唤醒的 pipe，把读端挂到 epoll 上，如果有设备参数的变化需要处理，而 getEvents()又阻塞在设备上，调用 wake()在 pipe 的写端写入，可以让线程从等待中返回。
 - 利用 inotify 机制监听/dev/input 目录下的变更，如有则标识着设备的变化，需要处理。
- **InputDispatcherThread**: 主要功能在 InputDispatcher，用于将事件分发给目标窗口。

由于事件的处理是流水线模式，需要 InputReader 先读事件，然后 InputDispatcher 才能进一步处理和分发。

1.3 IMS 初始化

IMS 的构造函数通过调用 `nativeInit()` 来初始化，此时 IMS 的构造函数使用了 `DisplayThread` 的 `Handler`，意味着 IMS 中的消息队列处理都是在单独的 `DisplayThread` 中进行的。

`DisplayThread` 是系统中共享的单例前台线程，主要用作输入输出的处理，这样比较敏感的用户体验可以减少受其他工作的影响，减少延时。

IMS 初始化的顺序是 `NativerInputManager`、`InputManager`，`InputManager` 的初始化顺序是 `EventHub`、`InputDispatcher`、`InputReader`。

Unisoc Confidential For hiar

2

Input 事件处理

Input 事件处理分为两个重要的过程：Input 事件读取、Input 事件派发。

2.1 Input 事件读取

Input 事件通过 InputReader 读取，InputReader 功能是从 EventHub 中读取原始事件数据(RawEvent)，并由各个 InputMapper 处理之后输入对应的 InputListener。

InputReader 拥有一个 InputMapper 集合，InputMapper 的大部分工作在 InputReader 线程中完成。InputReader 可以接受任意线程的查询。

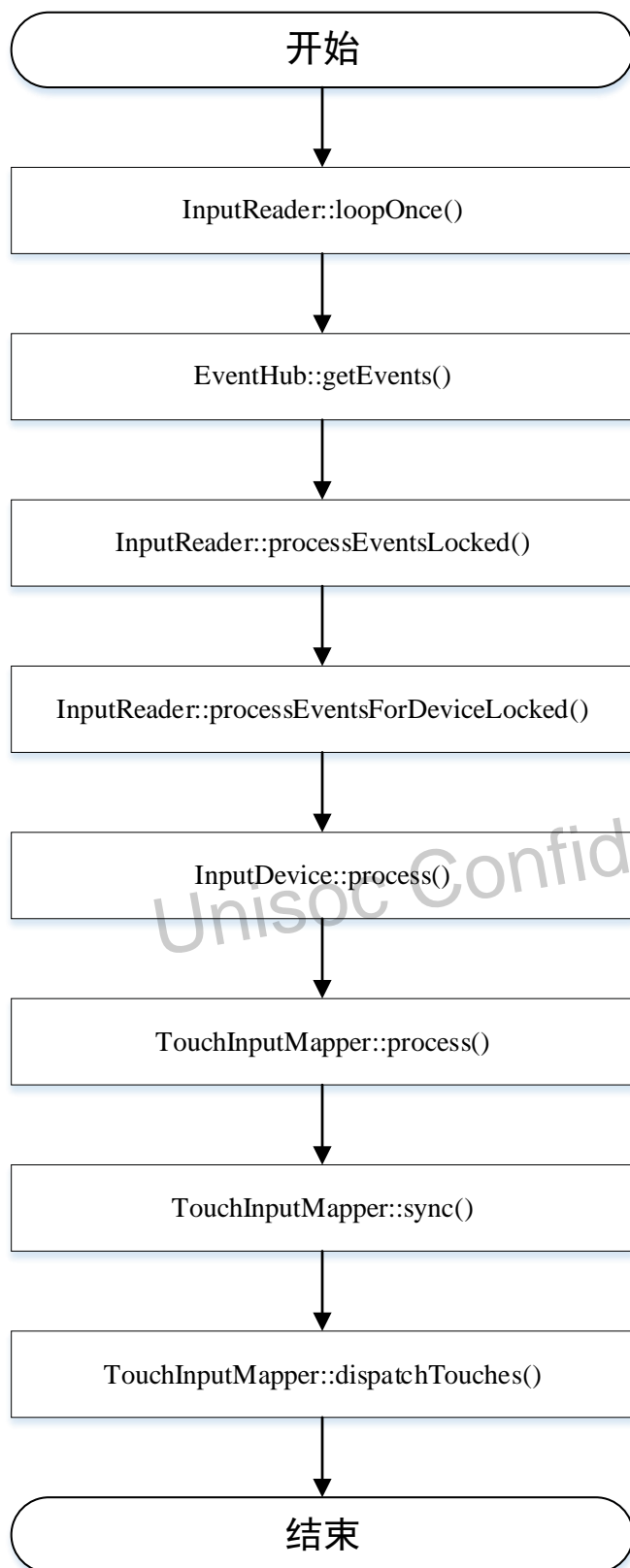
InputReader 的构造函数中，将第三个参数包装成 QueuedInputListener，其中，

- QueuedInputListener 中的成员变量 mArgsQueue 是一个缓冲队列，只有在 flush()时，才会一次性通知 InputDispatcher。
- QueuedInputListener 使用了 Command 模式，通过包装 InputDispatcher 实现 InputListenerInterface 接口，并将事件的处理请求封装成 NotifyArgs，使其有了缓冲执行的功能。

InputReader 类处理过程详见图 2-1。

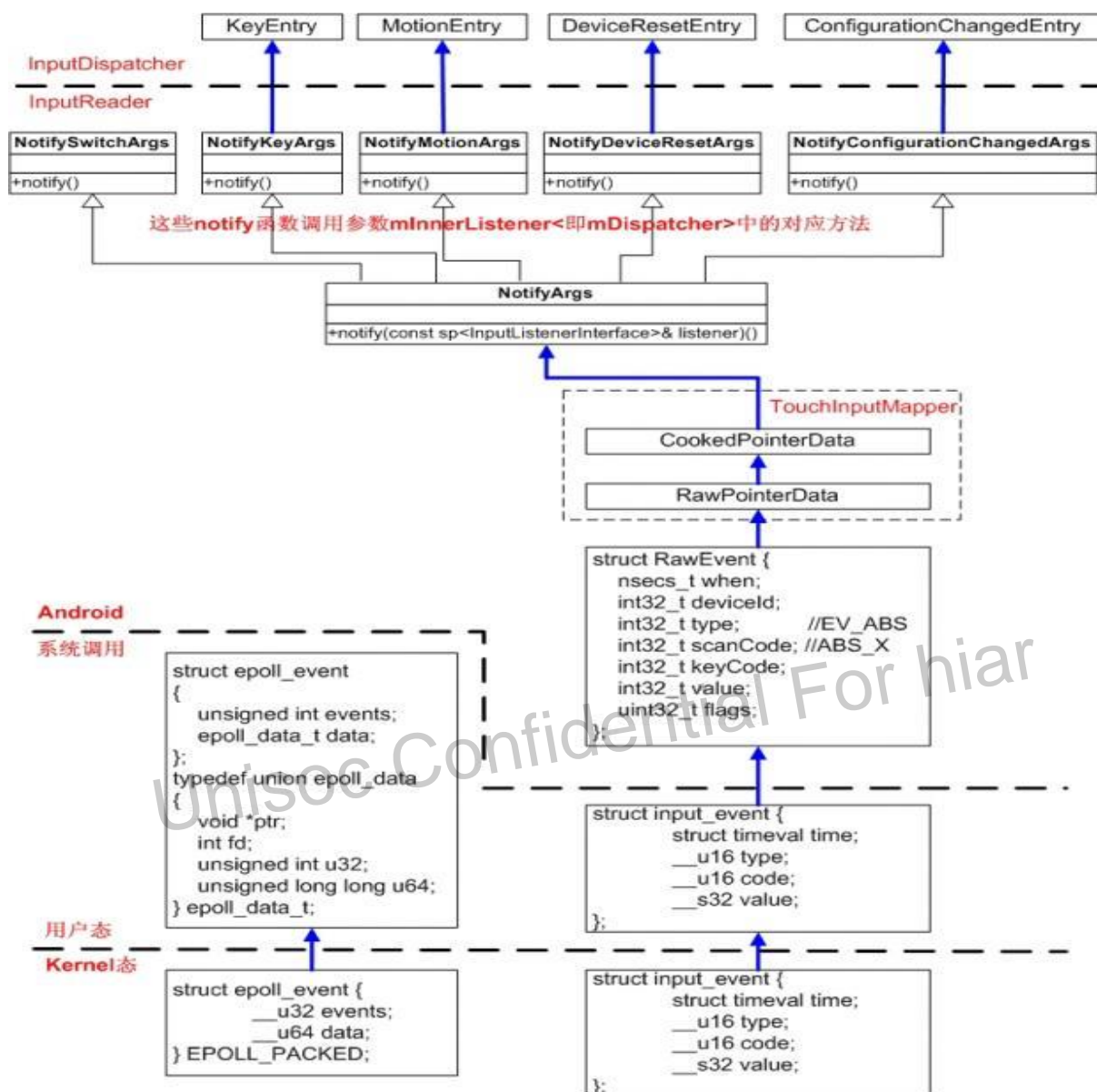
Unisoc Confidential For hiar

图2-1 InputReader 类处理过程



InputReader 消息结构变化的流程图如图 2-2。

图2-2 InputReader 消息结构变化流程图



2.2 Input 事件派发

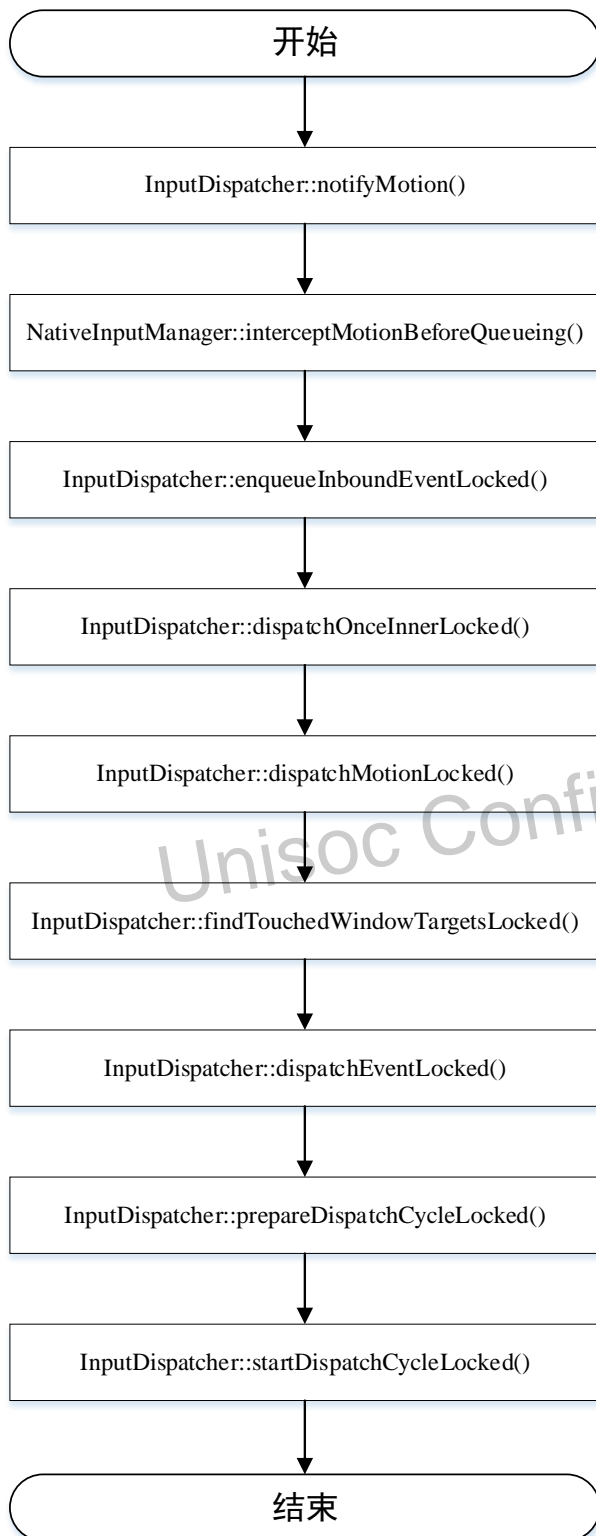
Input 事件通过 InputDispatcher 派发，InputDispatcher 的功能是把事件分发给输入目标，部分功能（如识别输入目标）由独立的 policy 对象控制。

InputDispatcher 需要监听 InputReader，使用 Listener 模式。

InputDispatcher 作为 InputReader 构造函数的第三个参数（InputDispatcher 构造一个 QueuedInputListener），实现 InputListenerInterface 接口。

InputDispatcher 类处理过程详见图 2-3。

图2-3 InputDispatcher 类处理过程



3

Input 事件分析

Input 事件分析方法有两种：

- 通过 log 分析
- 使用 getevent 命令分析

3.1 Log 分析

3.1.1 Log 开启

前提条件

手机是 debug 版本且已经 root。

开启方法

执行如下 adb 命令即可打开 Input 事件的 log。

```
adb shell setprop persist.sys.input.log true
adb shell getprop persist.sys.input.log (确认返回值是否为 true)
adb shell dumpsys input 或者 adb reboot
```

3.1.2 Log 搜索

搜索 Input 事件 log 步骤如下：

步骤 1 使用辅助工具 notepad++，打开保存在 PC 中 Log 文件。

步骤 2 按住快捷键 Ctrl + F，调出搜索框。

步骤 3 在查找搜索框中按需设置搜索项进行搜索。

例如查找所有 IMS 相关的 Log，搜索项设置如下：

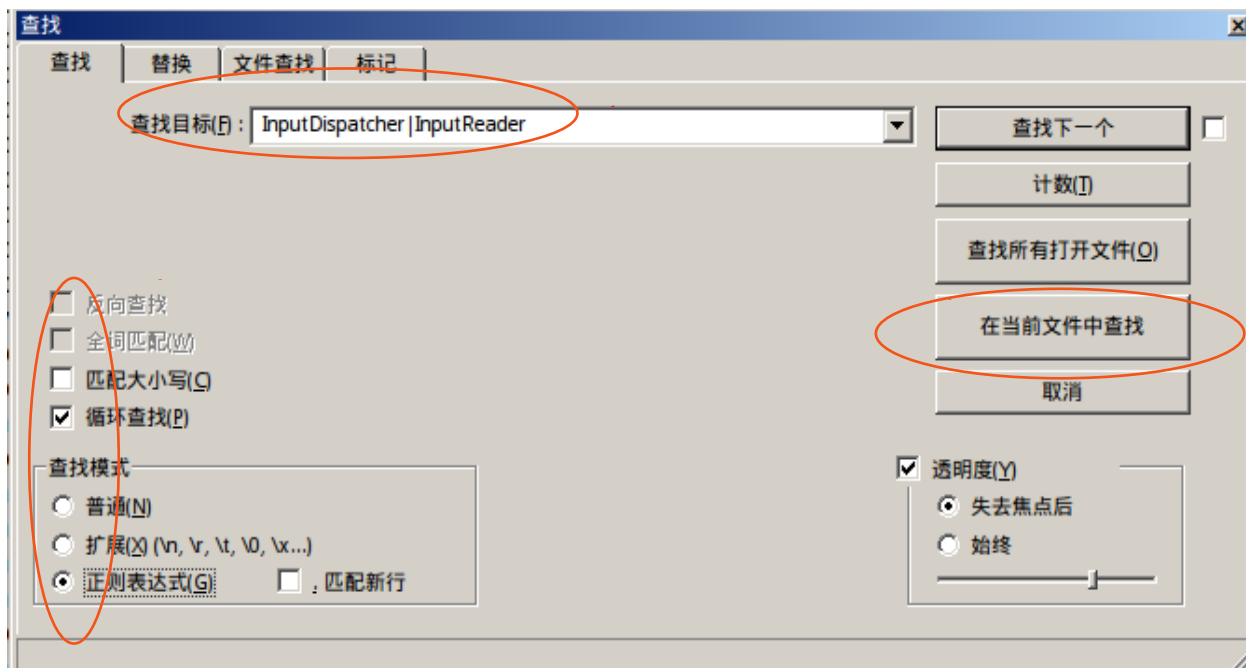
- 查找目标：输入 InputDispatcher|InputReader。

注意：InputDispatcher 和 InputReader 中间有竖线。

- 查找模式：勾选“正则表达式”。
- 查找范围：点击“在当前文件中查找”。

具体设置参见图 3-1。

图3-1 搜索 Input 相关 Log 搜索项设置



----结束

3.1.3 Log 分析

本节以手机中常用的 Touch 事件和 Key 事件为例，介绍 Log 分析方法。

InputReaderThread 不断的从驱动中读取 Event 并把它存放在 RawEvent 结构体中，RawEvent 结构体定义于 EventHub.h 中。当收到事件后，InputReaderThread 会构造一个 RawEvent 结构并传递给 InputReader processEventsLocked 方法处理。从 driver 读到的 RawEvent 结构如下：

```
struct RawEvent {
nsecs_t when;    //发生的时间
int32_t deviceId; //事件发生的deviceId，上层添加
int32_t type;     //事件类型：Key、touch等等
int32_t code;     //input事件code码
int32_t value;    //input事件值
};
```

Touch 事件 Log

InputReader: processEventsLocked: type=3 Count=3 code=57 value=-1 deviceId=6

其中：

- type=3 代表是触摸屏设备。
- value=-1 代表一次 Touch 事件完成。

Key 事件 Log

InputReader: processEventsLocked: type=1 Count=2 code=116 value=1 deviceId=7

InputReader: processEventsLocked: type=1 Count=2 code=116 value=0 deviceId=7

其中：

- type=1 时代表按键设备。
- value=1 时代表按键按下，value=0 时代表按键抬起。

processEventsLocked 中的事件上报为驱动事件上报的入口，可以从这个 Log 中分析是否接收到了驱动的事件上报。

3.2 getevent 命令分析

getevent 指令用于获取 input 输入事件，比如获取按键上报信息、获取触摸屏上报信息等。

常见功能及指令如下：

1. 查看帮助信息

```
$ getevent -h
```

图3-2 \$ getevent -h 执行结果

```
Usage: getevent [-t] [-n] [-s switchmask] [-S] [-v [mask]] [-d] [-p] [-i] [-l] [-q] [-c count] [-r] [device]
-t: show time stamps
-n: don't print newlines
-s: print switch states for given bits
-S: print all switch states
-v: verbosity mask (errs=1, dev=2, name=4, info=8, vers=16, pos. events=32, props=64)
-d: show HID descriptor, if available
-p: show possible events (errs, dev, name, pos. events)
-i: show all device info and possible events
-l: label event types and names in plain text
-q: quiet (clear verbosity mask)
-c: print given number of events then exit
-r: print rate events are received
```

2. 查看当前输入设备

```
$ getevent
```

执行完后，界面上会显示当前有哪些输入设备，数量与/dev/input 目录下相同。

图3-3 \$ getevent 执行结果

```
add device 1: /dev/input/event7
  name: "sf-keys"
add device 2: /dev/input/event6
  name: "compass"
add device 3: /dev/input/event5
  name: "alps_pxy"
add device 4: /dev/input/event4
  name: "accelerometer"
add device 5: /dev/input/event2
  name: "sprdphone Headset Jack"
add device 6: /dev/input/event1
  name: "adaptive_ts"
add device 7: /dev/input/event0
  name: "sprd-gpio-keys"
add device 8: /dev/input/event3
  name: "sprdphone Headset Keyboard"
```

- 查看触摸屏上报信息并显示时间戳，指令如下：

```
$ getevent -t /dev/input/event1 （然后点击或者滑动屏幕）
```

图3-4 \$ getevent -t /dev/input/event1 执行结果

```
[ 4628.619938] 0003 0039 0000005e
[ 4628.619938] 0003 0035 0000007a
[ 4628.619938] 0003 0036 00000390
[ 4628.619938] 0001 014a 00000001
[ 4628.619938] 0000 0000 00000000
[ 4628.641430] 0003 0035 0000007b
[ 4628.641430] 0003 0036 00000388
[ 4628.641430] 0000 0000 00000000
[ 4628.650711] 0003 0035 0000007c
[ 4628.650711] 0003 0036 00000374
[ 4628.650711] 0000 0000 00000000
[ 4628.661875] 0003 0035 00000076
[ 4628.661875] 0003 0036 0000034d
[ 4628.661875] 0000 0000 00000000
[ 4628.671132] 0003 0035 00000062
[ 4628.671132] 0003 0036 00000316
[ 4628.671132] 0000 0000 00000000
[ 4628.681008] 0003 0035 00000042
[ 4628.681008] 0003 0036 000002d3
[ 4628.681008] 0000 0000 00000000
[ 4628.690563] 0003 0039 ffffffff
[ 4628.690563] 0001 014a 00000000
[ 4628.690563] 0000 0000 00000000
```

- 一次性查看需要的触摸屏信息

使用组合参数，可一次性查看需要的触摸屏信息，命令如下：

```
$ getevent -tlr /dev/input/event1 （然后点击或滑动屏幕）
```

图3-5 \$ getevent -tlr /dev/input/event1 执行结果

[4781.796468]	EV_ABS	ABS_MT_TRACKING_ID	0000005f	
[4781.796468]	EV_ABS	ABS_MT_POSITION_X	00000153	
[4781.796468]	EV_ABS	ABS_MT_POSITION_Y	000003a1	
[4781.796468]	EV_KEY	BTN_TOUCH	DOWN	
[4781.796468]	EV_SYN	SYN_REPORT	00000000	
[4781.807527]	EV_ABS	ABS_MT_POSITION_Y	0000038c	报点率
[4781.807527]	EV_SYN	SYN_REPORT	00000000	rate 90
[4781.817799]	EV_ABS	ABS_MT_POSITION_X	00000158	
[4781.817799]	EV_ABS	ABS_MT_POSITION_Y	00000384	
[4781.817799]	EV_SYN	SYN_REPORT	00000000	rate 97
[4781.827596]	EV_ABS	ABS_MT_POSITION_X	00000167	
[4781.827596]	EV_ABS	ABS_MT_POSITION_Y	00000355	
[4781.827596]	EV_SYN	SYN_REPORT	00000000	rate 102
[4781.838003]	EV_ABS	ABS_MT_POSITION_X	00000187	
[4781.838003]	EV_ABS	ABS_MT_POSITION_Y	00000315	
[4781.838003]	EV_SYN	SYN_REPORT	00000000	rate 96
[4781.848314]	EV_ABS	ABS_MT_POSITION_X	000001bd	
[4781.848314]	EV_ABS	ABS_MT_POSITION_Y	000002d3	
[4781.848314]	EV_SYN	SYN_REPORT	00000000	rate 96
[4781.858672]	EV_ABS	ABS_MT_POSITION_X	0000020a	
[4781.858672]	EV_ABS	ABS_MT_POSITION_Y	00000296	
[4781.858672]	EV_SYN	SYN_REPORT	00000000	rate 96
[4781.868397]	EV_ABS	ABS_MT_POSITION_X	00000267	
[4781.868397]	EV_ABS	ABS_MT_POSITION_Y	0000025b	
[4781.868397]	EV_SYN	SYN_REPORT	00000000	rate 102
[4781.878024]	EV_ABS	ABS_MT_TRACKING_ID	ffffffff	
[4781.878024]	EV_KEY	BTN_TOUCH	UP	
[4781.878024]	EV_SYN	SYN_REPORT	00000000	rate 103

通过使用组合命令查看到有类似图 3-5 的信息输出，则表示有驱动事件上报。

Unisoc Confidential For hiar