## 加载按键映射流程分析

目录

[加载按键映射流程分析 1](#_Toc487012093)

[1. 流程分析 2](#_Toc487012094)

[2. 获取InputDeviceIdentifier的name 以及 Device的创建 2](#_Toc487012095)

[3. 加载idc文件 4](#_Toc487012096)

[4. 加载kl文件 8](#_Toc487012097)

[5. 使用kl文件，将扫描码，转换成按键码 17](#_Toc487012098)

[6. 总结 19](#_Toc487012099)

[6.1 getevent –v来查看输入设备信息 19](#_Toc487012100)

[6.2 logcat -s EventHub KeyLayoutMap分析kl文件加载过程 20](#_Toc487012101)

[6.3 实例 20](#_Toc487012102)

## 流程分析

这里以 amlogic安卓7.1sdk s905d 平台 为例，分析加载按键映射的流程。

先大致介绍一下整个流程，再做重点分析。输入事件流程一共涉及到下面这几个文件：

/frameworks/base/services/core/java/com/android/server/wm/WindowManagerService.java (按键事件定义) /frameworks/ base/core/java/android/hardware/input/InputManager.java (Java层输入管理)

/frameworks/native/services/inputflinger/InputManager.cpp (native层输入管理)

/frameworks/native/services/inputflinger/InputReader.cpp (事件读取线程)

/frameworks/native/services/inputflinger/InputDispatcher.cpp（事件分发线程）

/frameworks/native/services/inputflinger/EventHub.cpp (键码与键值转换)

其中，WindowManagerService.java和InputManager.java主要向Android为窗口系统提供服务，EventHub.cpp主要用来读取设备文件中的RawEvent，而InputReader.cpp和InputDispatcher.cpp算是它们之间的对接层。

它们的关系是：WindowManagerService通过InputManager提供的接口开启一个线程驱动InputReader不断地从/dev/input/目录下面的设备文件读取事件，然后通过InputDispatcher分发给连接到WindowManagerService服务的客户端。

InputReader从设备文件中读取的是RawEvent，在交给InputDispatcher进行分发之前，它需要先把RawEvent进行转化分类，拆分成KeyEvent、MotionEvent、TrackEvent各种类型等。这篇文章主要关注的就是这个RawEvent的拆分过程，所以我们的重点在EventHub.cpp中。并且，为了简单化分析过程，在这里我的分析只关注触摸屏事件。看它是如何从RawEvent被拆分成应用层用户事件MotionEvent的。

## 2. 获取InputDeviceIdentifier的name 以及 Device的创建

InputDeviceIdentifier的name 非常重要，后面寻找idc kl kcm文件都需要这个name。

我们看下面的调用流程EventHub::getEvents -> EventHub::scanDevicesLocked -> EventHub::scanDirLocked -> EventHub::openDeviceLocked

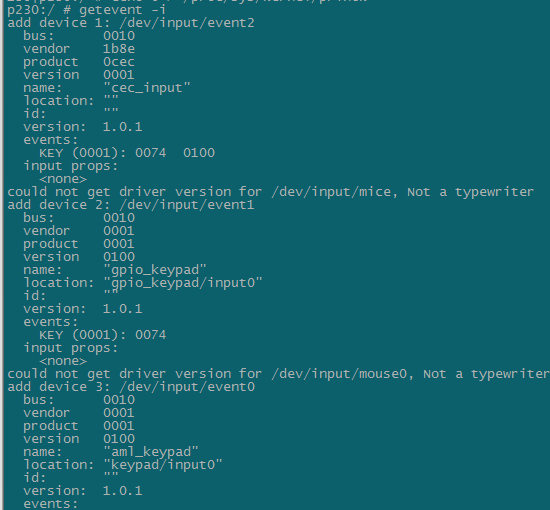
我们来看EventHub::openDeviceLocked函数，先是打开devicePath，然后利用ioctl获取InputDeviceIdentifier的name

1. status\_t EventHub::openDeviceLocked(const char \*devicePath) {
2. char buffer[80];
3. ALOGV("Opening device: %s", devicePath);
4. int fd = open(devicePath, O\_RDWR | O\_CLOEXEC);
5. if(fd < 0) {
6. ALOGE("could not open %s, %s\n", devicePath, strerror(errno));
7. return -1;
8. }
9. InputDeviceIdentifier identifier;
10. // Get device name.
11. if(ioctl(fd, EVIOCGNAME(sizeof(buffer) - 1), &buffer) < 1) {
12. //fprintf(stderr, "could not get device name for %s, %s\n", devicePath, strerror(errno));
13. } else {
14. buffer[sizeof(buffer) - 1] = '\0';
15. identifier.name.setTo(buffer);
16. }

下面创建了Device，各种打印。

1. Device\* device = **new** Device(fd, deviceId, String8(devicePath), identifier);
3. ALOGW("add device %d: %s\n", deviceId, devicePath);
4. ALOGW("  bus:        %04x\n"
5. "  vendor      %04x\n"
6. "  product     %04x\n"
7. "  version     %04x\n",
8. identifier.bus, identifier.vendor, identifier.product, identifier.version);
9. ALOGW("  name:       \"%s\"\n", identifier.name.string());
10. ALOGW("  location:   \"%s\"\n", identifier.location.string());
11. ALOGW("  unique id:  \"%s\"\n", identifier.uniqueId.string());
12. ALOGW("  descriptor: \"%s\"\n", identifier.descriptor.string());
13. ALOGW("  driver:     v%d.%d.%d\n",
14. driverVersion >> 16, (driverVersion >> 8) & 0xff, driverVersion & 0xff);
16. // Load the configuration file for the device.
17. loadConfigurationLocked(device);

先来看下打印



## 3. 加载idc文件

在我们的设备中，一般没有定义自己的idc文件，也就找不到。一般定义idc文件，是在这个文件中定义kl 和kcm文件。

我们再来分析下loadConfigurationLocked函数，调用getInputDeviceConfigurationFilePathByDeviceIdentifier函数，当configurationFile不为空的时候，调用PropertyMap::load加载idc文件，这部分代码是在system/libutil下面的，当有这个idc文件的时候，device->configuration就不为空。

1. **void** EventHub::loadConfigurationLocked(Device\* device) {
2. device->configurationFile = getInputDeviceConfigurationFilePathByDeviceIdentifier(
3. device->identifier, INPUT\_DEVICE\_CONFIGURATION\_FILE\_TYPE\_CONFIGURATION);
4. **if** (device->configurationFile.isEmpty()) {//configurationFile为空
5. ALOGD("No input device configuration file found for device '%s'.",
6. device->identifier.name.string());
7. } **else** {//如果有configurationFile文件，那我们就调用PropertyMap::load函数
8. ALOGD("input device configuration file name '%s'.",
9. device->configurationFile.string());
10. status\_t status = PropertyMap::load(device->configurationFile,
11. &device->configuration);
12. **if** (status) {
13. ALOGE("Error loading input device configuration file for device '%s'.  "
14. "Using default configuration.",
15. device->identifier.name.string());
16. }
17. }
18. }

调用了getInputDeviceConfigurationFilePathByDeviceIdentifier函数，其中type为0，代表是idc文件

1. String8 getInputDeviceConfigurationFilePathByDeviceIdentifier(
2. **const** InputDeviceIdentifier& deviceIdentifier,
3. InputDeviceConfigurationFileType type) {
4. **if** (deviceIdentifier.vendor !=0 && deviceIdentifier.product != 0) {//不进入这个分支
5. **if** (deviceIdentifier.version != 0) {
6. // Try vendor product version.
7. String8 versionPath(getInputDeviceConfigurationFilePathByName(
8. String8::format("Vendor\_%04x\_Product\_%04x\_Version\_%04x",
9. deviceIdentifier.vendor, deviceIdentifier.product,
10. deviceIdentifier.version),
11. type));
12. **if** (!versionPath.isEmpty()) {
13. **return** versionPath;
14. }
15. }
17. // Try vendor product.
18. String8 productPath(getInputDeviceConfigurationFilePathByName(
19. String8::format("Vendor\_%04x\_Product\_%04x",
20. deviceIdentifier.vendor, deviceIdentifier.product),
21. type));
22. **if** (!productPath.isEmpty()) {
23. **return** productPath;
24. }
25. }
27. // Try device name.
28. **return** getInputDeviceConfigurationFilePathByName(deviceIdentifier.name, type);
29. }

于是我们再来看getInputDeviceConfigurationFilePathByName函数：

1. String8 getInputDeviceConfigurationFilePathByName(
2. **const** String8& name, InputDeviceConfigurationFileType type) {
3. // Search system repository.
4. String8 path;
5. path.setTo(getenv("ANDROID\_ROOT"));
6. path.append("/usr/");
7. appendInputDeviceConfigurationFileRelativePath(path, name, type);
8. ALOGD("Probing for system provided input device configuration file: path='%s'", path.string());
9. **if** (!access(path.string(), R\_OK)) {
10. ALOGD("Found");
11. **return** path;
12. }
14. // Search user repository.
15. // TODO Should only look here if not in safe mode.
16. path.setTo(getenv("ANDROID\_DATA"));
17. path.append("/system/devices/");
18. appendInputDeviceConfigurationFileRelativePath(path, name, type);
19. ALOGD("Probing for system user input device configuration file: path='%s'", path.string());
20. **if** (!access(path.string(), R\_OK)) {
21. ALOGD("Found");
22. **return** path;
23. }
25. // Not found.
26. ALOGD("Probe failed to find input device configuration file: name='%s', type=%d",
27. name.string(), type);
28. **return** String8();
29. }

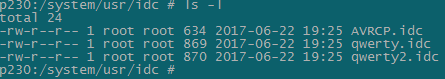
这个函数就是寻找各种匹配的idc文件，最后没找到就返回一个空的String。我们来看下appendInputDeviceConfigurationFileRelativePath函数

1. **static** **void** appendInputDeviceConfigurationFileRelativePath(String8& path,
2. **const** String8& name, InputDeviceConfigurationFileType type) {
3. path.append(CONFIGURATION\_FILE\_DIR[type]);
4. **for** (**size\_t** i = 0; i < name.length(); i++) {
5. **char** ch = name[i];
6. **if** (!isValidNameChar(ch)) {
7. ch = '\_';
8. }
9. path.append(&ch, 1);
10. }
11. path.append(CONFIGURATION\_FILE\_EXTENSION[type]);
12. }
13. **static** **const** **char**\* CONFIGURATION\_FILE\_DIR[] = {
14. "idc/",
15. "keylayout/",
16. "keychars/",
17. };
19. **static** **const** **char**\* CONFIGURATION\_FILE\_EXTENSION[] = {
20. ".idc",
21. ".kl",
22. ".kcm",
23. };

这个函数就是用传进来的路径和名字，组成一个idc文件。然后在getInputDeviceConfigurationFilePathByName文件中看用appendInputDeviceConfigurationFileRelativePath文件组成的idc文件是否有这个文件，有那就找到了返回idc的文件路径，如果没有最后返回一个空的string。我们看我们的这段log。

1. InputDevice: Probing for system provided input device configuration file: path='/system/usr/idc/comip\_snd\_soc\_Headset.idc'
2. InputDevice: Probing for system user input device configuration file: path='/data/system/devices/idc/comip\_snd\_soc\_Headset.idc'
3. InputDevice: Probe failed to find input device configuration file: name='comip\_snd\_soc Headset', type=0
4. EventHub: No input device configuration file found for device 'comip\_snd\_soc Headset'.

这段log说明没有这样的idc文件。



我们看我们机顶盒的idc目录，都是原生的，也就是framework/base/data下面的文件，都是原生的也就肯定找不到匹配的idc文件。idc文件中保存这kl kcm文件的名字。

下面是qwerty.idc文件，下面是它的内容，keyboard.layout代表kl的文件名，keyboard.characterMap代表kcm的文件名。

1. touch.deviceType = touchScreen
2. touch.orientationAware = 1
3. keyboard.layout = qwerty
4. keyboard.characterMap = qwerty
5. keyboard.orientationAware = 1
6. keyboard.builtIn = 1
7. cursor.mode = navigation
8. cursor.orientationAware = 1

## 4. 加载kl文件

下面重点来说下加载kl 和 kcm文件的过程，它是在openDeviceLocked函数中调用loadKeyMapLocked函数完成的。

那我们继续分析openDeviceLocked函数，关于加载kl文件的那部分代码：

1. status\_t EventHub::openDeviceLocked(const char \*devicePath) {
2. ...
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. }

其中loadKeyMapLocked函数如下

1. status\_t EventHub::loadKeyMapLocked(Device\* device) {
2. **return** device->keyMap.load(device->identifier, device->configuration);
3. }

我们再来看load函数，前面我们的idc文件没有找到匹配的，因此第一个分支可以直接跳过，可以直接看probeKeyMap函数。

1. status\_t KeyMap::load(**const** InputDeviceIdentifier& deviceIdenfifier,
2. **const** PropertyMap\* deviceConfiguration) {
3. // Use the configured key layout if available.
4. **if** (deviceConfiguration) {
5. String8 keyLayoutName;
6. **if** (deviceConfiguration->tryGetProperty(String8("keyboard.layout"),
7. keyLayoutName)) {
8. status\_t status = loadKeyLayout(deviceIdenfifier, keyLayoutName);
9. **if** (status == NAME\_NOT\_FOUND) {
10. ALOGE("Configuration for keyboard device '%s' requested keyboard layout '%s' but "
11. "it was not found.",
12. deviceIdenfifier.name.string(), keyLayoutName.string());
13. }
14. }
16. String8 keyCharacterMapName;
17. **if** (deviceConfiguration->tryGetProperty(String8("keyboard.characterMap"),
18. keyCharacterMapName)) {
19. status\_t status = loadKeyCharacterMap(deviceIdenfifier, keyCharacterMapName);
20. **if** (status == NAME\_NOT\_FOUND) {
21. ALOGE("Configuration for keyboard device '%s' requested keyboard character "
22. "map '%s' but it was not found.",
23. deviceIdenfifier.name.string(), keyLayoutName.string());
24. }
25. }
27. **if** (isComplete()) {
28. **return** OK;
29. }
30. }
32. // Try searching by device identifier.
33. **if** (probeKeyMap(deviceIdenfifier, String8::empty())) {
34. **return** OK;
35. }
37. // Fall back on the Generic key map.
38. // TODO Apply some additional heuristics here to figure out what kind of
39. //      generic key map to use (US English, etc.) for typical external keyboards.
40. **if** (probeKeyMap(deviceIdenfifier, String8("Generic"))) {
41. **return** OK;
42. }
44. // Try the Virtual key map as a last resort.
45. **if** (probeKeyMap(deviceIdenfifier, String8("Virtual"))) {
46. **return** OK;
47. }
49. // Give up!
50. ALOGE("Could not determine key map for device '%s' and no default key maps were found!",
51. deviceIdenfifier.name.string());
52. **return** NAME\_NOT\_FOUND;
53. }

**没有找到匹配InputDeviceIdentifier的name的kl文件 使用原生的Generic.kl文件**

第一种情况是没有找到匹配InputDeviceIdentifier的name的kl文件，这个时候我们一般就用Generic.kl文件代替。

下面我们直接看probeKeyMap函数：

1. **bool** KeyMap::probeKeyMap(**const** InputDeviceIdentifier& deviceIdentifier,
2. **const** String8& keyMapName) {
3. **if** (!haveKeyLayout()) {//是否有kl文件
4. loadKeyLayout(deviceIdentifier, keyMapName);
5. }
6. **if** (!haveKeyCharacterMap()) {//是否有kcm文件
7. loadKeyCharacterMap(deviceIdentifier, keyMapName);
8. }
9. **return** isComplete();
10. }

先来看下isComplete函数，kl文件和kcm文件都有了才返回true，看load函数，当isComplete返回true，就直接return了，因为kl 和 kcm文件都找到了。

1. **inline** **bool** isComplete() **const** {
2. **return** haveKeyLayout() && haveKeyCharacterMap();
3. }

下面我们看下加载kl文件的过程，kcm文件的加载过程和kl类似我们就不看了。

1. status\_t KeyMap::loadKeyLayout(**const** InputDeviceIdentifier& deviceIdentifier,
2. **const** String8& name) {
3. String8 path(getPath(deviceIdentifier, name,
4. INPUT\_DEVICE\_CONFIGURATION\_FILE\_TYPE\_KEY\_LAYOUT));
5. **if** (path.isEmpty()) {
6. **return** NAME\_NOT\_FOUND;
7. }
8. ALOGE("loadKeyLayout path '%s'.",
9. path.string());
11. status\_t status = KeyLayoutMap::load(path, &keyLayoutMap);
12. **if** (status) {
13. **return** status;
14. }
16. keyLayoutFile.setTo(path);
17. **return** OK;
18. }

先看下getPath函数，第一个在load函数中调用loadKeyLayout的name是空的，所以这里就是用了getInputDeviceConfigurationFilePathByDeviceIdentifier函数。

1. String8 KeyMap::getPath(**const** InputDeviceIdentifier& deviceIdentifier,
2. **const** String8& name, InputDeviceConfigurationFileType type) {
3. **return** name.isEmpty()
4. ? getInputDeviceConfigurationFilePathByDeviceIdentifier(deviceIdentifier, type)
5. : getInputDeviceConfigurationFilePathByName(name, type);
6. }

看下getInputDeviceConfigurationFilePathByDeviceIdentifier函数，第一部分就是各种Vendor之类的kl，我们没有走进这个分支。所以说，如果有在机顶盒的/system/usr/keylayout/下有Vendor之类的kl文件的时候 ，会率先加载Vendor之类的kl的kl文件，而Vendor之类的kl文件的命名规则上面提到了，为Vendor\_%04x\_Product\_%04x.kl 其中Vendor和Product号可以通过 # getevent -v 来查看到，如我们的机顶盒 vendor 0001， product 0001，其我们可以在/system/usr/keylayout/添加一个Vendor\_0001\_Product\_0001.kl来按自己的规则来加载按键映射。

1. String8 getInputDeviceConfigurationFilePathByDeviceIdentifier(
2. **const** InputDeviceIdentifier& deviceIdentifier,
3. InputDeviceConfigurationFileType type) {
4. **if** (deviceIdentifier.vendor !=0 && deviceIdentifier.product != 0) {
5. **if** (deviceIdentifier.version != 0) {
6. // Try vendor product version.
7. String8 versionPath(getInputDeviceConfigurationFilePathByName(
8. String8::format("Vendor\_%04x\_Product\_%04x\_Version\_%04x",
9. deviceIdentifier.vendor, deviceIdentifier.product,
10. deviceIdentifier.version),
11. type));
12. **if** (!versionPath.isEmpty()) {
13. **return** versionPath;
14. }
15. }
17. // Try vendor product.
18. String8 productPath(getInputDeviceConfigurationFilePathByName(
19. String8::format("Vendor\_%04x\_Product\_%04x",
20. deviceIdentifier.vendor, deviceIdentifier.product),
21. type));
22. **if** (!productPath.isEmpty()) {
23. **return** productPath;
24. }
25. }
27. // Try device name.
28. **return** getInputDeviceConfigurationFilePathByName(deviceIdentifier.name, type);
29. }

因此我们直接看下getInputDeviceConfigurationFilePathByName函数，和之前找idc那个函数一样，只是这里是用来找kl文件了

1. String8 getInputDeviceConfigurationFilePathByName(
2. **const** String8& name, InputDeviceConfigurationFileType type) {
3. // Search system repository.
4. String8 path;
5. path.setTo(getenv("ANDROID\_ROOT"));
6. path.append("/usr/");
7. appendInputDeviceConfigurationFileRelativePath(path, name, type);
8. ALOGD("Probing for system provided input device configuration file: path='%s'", path.string());
9. **if** (!access(path.string(), R\_OK)) {
10. ALOGD("Found");
11. **return** path;
12. }
14. // Search user repository.
15. // TODO Should only look here if not in safe mode.
16. path.setTo(getenv("ANDROID\_DATA"));
17. path.append("/system/devices/");
18. appendInputDeviceConfigurationFileRelativePath(path, name, type);
19. ALOGD("Probing for system user input device configuration file: path='%s'", path.string());
20. **if** (!access(path.string(), R\_OK)) {
21. ALOGD("Found");
22. **return** path;
23. }
25. // Not found.
26. ALOGD("Probe failed to find input device configuration file: name='%s', type=%d",
27. name.string(), type);
28. **return** String8();
29. }

我们回过头在来看load函数调用的第二个probeKeyMap函数，是传入了Generic参数，

1. **if** (probeKeyMap(deviceIdenfifier, String8("Generic"))) {
2. **return** OK;
3. }

我们再来看看probeKeyMap函数，还是调用了loadKeyLayout函数

1. **bool** KeyMap::probeKeyMap(**const** InputDeviceIdentifier& deviceIdentifier,
2. **const** String8& keyMapName) {
3. **if** (!haveKeyLayout()) {
4. loadKeyLayout(deviceIdentifier, keyMapName);
5. }
6. **if** (!haveKeyCharacterMap()) {
7. loadKeyCharacterMap(deviceIdentifier, keyMapName);
8. }
9. **return** isComplete();
10. }
12. status\_t KeyMap::loadKeyLayout(**const** InputDeviceIdentifier& deviceIdentifier,
13. **const** String8& name) {
14. String8 path(getPath(deviceIdentifier, name,
15. INPUT\_DEVICE\_CONFIGURATION\_FILE\_TYPE\_KEY\_LAYOUT));
16. **if** (path.isEmpty()) {
17. **return** NAME\_NOT\_FOUND;
18. }
19. ALOGE("kangchen  loadKeyLayout path '%s'.",
20. path.string());
22. status\_t status = KeyLayoutMap::load(path, &keyLayoutMap);
23. **if** (status) {
24. **return** status;
25. }
27. keyLayoutFile.setTo(path);
28. **return** OK;
29. }

同样我们来看getPath函数，这个时候name不是空了，就调用getInputDeviceConfigurationFilePathByName函数

1. String8 KeyMap::getPath(**const** InputDeviceIdentifier& deviceIdentifier,
2. **const** String8& name, InputDeviceConfigurationFileType type) {
3. **return** name.isEmpty()
4. ? getInputDeviceConfigurationFilePathByDeviceIdentifier(deviceIdentifier, type)
5. : getInputDeviceConfigurationFilePathByName(name, type);
6. }

getInputDeviceConfigurationFilePathByName函数，最后就在这个函数中找到了Generic.kl文件。

1. String8 getInputDeviceConfigurationFilePathByName(
2. **const** String8& name, InputDeviceConfigurationFileType type) {
3. // Search system repository.
4. String8 path;
5. path.setTo(getenv("ANDROID\_ROOT"));
6. path.append("/usr/");
7. appendInputDeviceConfigurationFileRelativePath(path, name, type);
8. ALOGD("Probing for system provided input device configuration file: path='%s'", path.string());
9. **if** (!access(path.string(), R\_OK)) {
10. ALOGD("Found");
11. **return** path;
12. }
14. // Search user repository.
15. // TODO Should only look here if not in safe mode.
16. path.setTo(getenv("ANDROID\_DATA"));
17. path.append("/system/devices/");
18. appendInputDeviceConfigurationFileRelativePath(path, name, type);
19. ALOGD("Probing for system user input device configuration file: path='%s'", path.string());
20. **if** (!access(path.string(), R\_OK)) {
21. ALOGD("Found");
22. **return** path;
23. }
25. // Not found.
26. ALOGD("Probe failed to find input device configuration file: name='%s', type=%d",
27. name.string(), type);
28. **return** String8();
29. }

找到kl文件后，我们会对这个文件在load函数中进行解析，这个我们就不分析了，就是把扫描码转换成按键码之类。

1. status\_t KeyLayoutMap::load(**const** String8& filename, sp<KeyLayoutMap>\* outMap) {
2. outMap->clear();
4. Tokenizer\* tokenizer;
5. status\_t status = Tokenizer::open(filename, &tokenizer);
6. **if** (status) {
7. ALOGE("Error %d opening key layout map file %s.", status, filename.string());
8. } **else** {
9. sp<KeyLayoutMap> map = **new** KeyLayoutMap();
10. **if** (!map.get()) {
11. ALOGE("Error allocating key layout map.");
12. status = NO\_MEMORY;
13. } **else** {
14. #if DEBUG\_PARSER\_PERFORMANCE
15. nsecs\_t startTime = systemTime(SYSTEM\_TIME\_MONOTONIC);
16. #endif
17. Parser parser(map.get(), tokenizer);
18. status = parser.parse();
19. #if DEBUG\_PARSER\_PERFORMANCE
20. nsecs\_t elapsedTime = systemTime(SYSTEM\_TIME\_MONOTONIC) - startTime;
21. ALOGD("Parsed key layout map file '%s' %d lines in %0.3fms.",
22. tokenizer->getFilename().string(), tokenizer->getLineNumber(),
23. elapsedTime / 1000000.0);
24. #endif
25. **if** (!status) {
26. \*outMap = map;
27. }
28. }
29. **delete** tokenizer;
30. }
31. **return** status;
32. }

下面的数字代表扫描码，而旁边的代码键值码。

1. key 113   VOLUME\_MUTE
2. key 114   VOLUME\_DOWN
3. key 115   VOLUME\_UP
4. key 116   POWER

这样就完成了kl文件的加载解析。

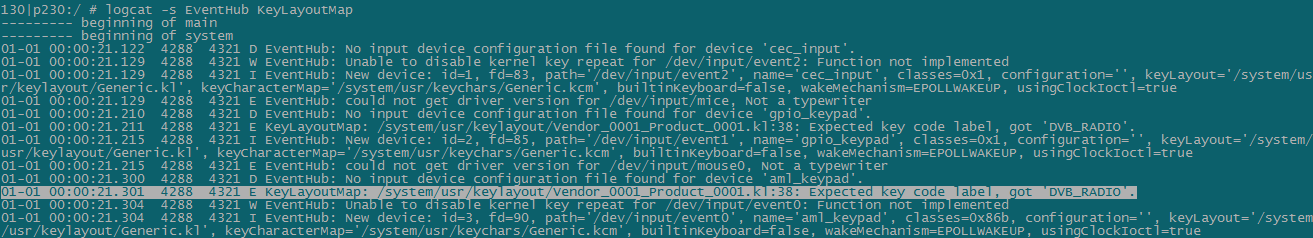
分析kl文件的加载 log，我们可以通过以下命令加抓取

#logcat -s EventHub KeyLayoutMap



而如果我的修改的r

下面的这个例子是在有Vendor\_0001\_Product\_0001.kl的文件下加载了Generic.kl文件，是因为在Vendor\_0001\_Product\_0001.kl中DVB\_RADIO的映射关系没有找到造成的。帮去掉DVB\_RADIO映射的这一行。或才在frameworks中注册DVB\_RADIO按键。



## 5. 使用kl文件，将扫描码，转换成按键码

之前我们在按键流程（一），已经讲解了按键最后到各个InputMapper中的process函数中处理，下面我们看这个函数，我们调用EventHub的mapKey来将扫描码转换成按键码。

1. **void** KeyboardInputMapper::process(**const** RawEvent\* rawEvent) {
2. **switch** (rawEvent->type) {
4. **case** EV\_KEY: {
5. int32\_t scanCode = rawEvent->code;
6. int32\_t usageCode = mCurrentHidUsage;
7. mCurrentHidUsage = 0;
9. **if** (isKeyboardOrGamepadKey(scanCode)) {
10. int32\_t keyCode;
11. uint32\_t flags;
12. **if** (getEventHub()->mapKey(getDeviceId(), scanCode, usageCode, &keyCode, &flags)) {//扫描码对应成按键码
13. keyCode = AKEYCODE\_UNKNOWN;
14. flags = 0;
15. }
16. processKey(rawEvent->when, rawEvent->value != 0, keyCode, scanCode, flags);
17. }
18. **break**;
19. }
20. **case** EV\_MSC: {
21. **if** (rawEvent->code == MSC\_SCAN) {
22. mCurrentHidUsage = rawEvent->value;
23. }
24. **break**;
25. }
26. **case** EV\_SYN: {
27. **if** (rawEvent->code == SYN\_REPORT) {
28. mCurrentHidUsage = 0;
29. }
30. }
31. }
32. }

最后在processKey函数中，将发送到InputDispatch中做后续处理。这个我们在之前的博客中也分析过了。

我们再来看看mapKey函数，先处理的kcm，再处理kl的。

1. status\_t EventHub::mapKey(int32\_t deviceId,
2. int32\_t scanCode, int32\_t usageCode, int32\_t metaState,
3. int32\_t\* outKeycode, int32\_t\* outMetaState, uint32\_t\* outFlags) **const** {
4. AutoMutex \_l(mLock);
5. Device\* device = getDeviceLocked(deviceId);
6. status\_t status = NAME\_NOT\_FOUND;
8. **if** (device) {
9. // Check the key character map first.
10. sp<KeyCharacterMap> kcm = device->getKeyCharacterMap();
11. **if** (kcm != NULL) {
12. **if** (!kcm->mapKey(scanCode, usageCode, outKeycode)) {
13. \*outFlags = 0;
14. status = NO\_ERROR;
15. }
16. }
18. // Check the key layout next.
19. **if** (status != NO\_ERROR && device->keyMap.haveKeyLayout()) {
20. **if** (!device->keyMap.keyLayoutMap->mapKey(
21. scanCode, usageCode, outKeycode, outFlags)) {
22. status = NO\_ERROR;
23. }
24. }
26. **if** (status == NO\_ERROR) {
27. **if** (kcm != NULL) {
28. kcm->tryRemapKey(\*outKeycode, metaState, outKeycode, outMetaState);
29. } **else** {
30. \*outMetaState = metaState;
31. }
32. }
33. }
35. **if** (status != NO\_ERROR) {
36. \*outKeycode = 0;
37. \*outFlags = 0;
38. \*outMetaState = metaState;
39. }
41. **return** status;
42. }

至于详细分析kcm的mapKey和kl的mapKey放在以后分析了

## 6. 总结

### 6.1 getevent –v来查看输入设备信息



上面可以看到，我们机顶盒的红外输入设置的Vendor 为0001，Product为0001，故添加的kl映射文件命名为：Vendor\_0001\_Product\_0001.kl

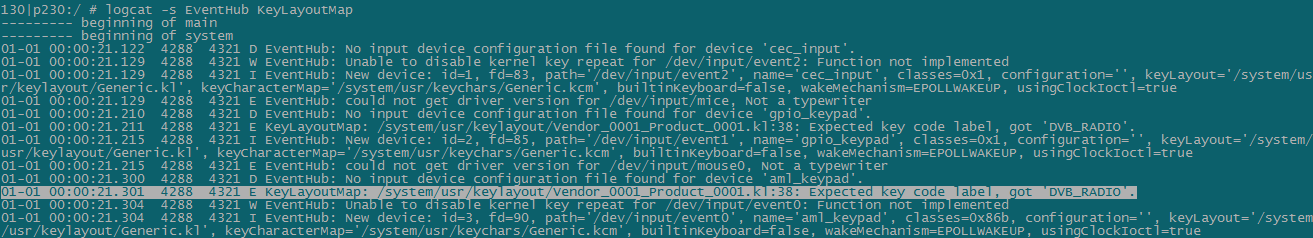
分析kl文件的加载 log，我们可以通过以下命令加抓取

### 6.2 logcat -s EventHub KeyLayoutMap分析kl文件加载过程

如下图所示，是正确加载Vendor\_0001\_Product\_0001.kl文件的log



而下面的这个例子是在有Vendor\_0001\_Product\_0001.kl的文件下加载了Generic.kl文件，从打印中可以看到这是因为在Vendor\_0001\_Product\_0001.kl中DVB\_RADIO的映射关系没有找到造成的。帮去掉DVB\_RADIO映射的这一行。或才在frameworks中注册DVB\_RADIO按键。



### 6.3 实例

问题：S905D DT5501-T2 V6.8.4版本中在APSettings中设置Wi-Fi热点不能弹出软键盘，而S905X平台上是好的。

分析 ：

s905x平台上remote.conf使用Generic.kl文件去加载按键映射的，而s905d平台调用的是Vendor\_0001\_Product\_0001.kl文件去加载按键映射的。而Vendor\_0001\_Product\_0001.kl与Generic.kl有一些按键键值不一样，故出现以上问题。

        其中Generic.kl是sdk官方原版通用的按键映射文件，我们一般不要去改动，以免造成市场上的一些遥控器各别按键适配出错等问题。

        因此，我们以后统一使用Vendor\_0001\_Product\_0001.kl来增加或修改按键映射，而不要去修改Generic.kl。以前s905x平台上在Generic.kl的修改麻烦移到Vendor\_0001\_Product\_0001.kl。

        至于为什么s905x平台在Vendor\_0001\_Product\_0001.kl文件存在的情况下却使用了Generic.kl去加载映射，原因如下：  
        通过红外遥控按键kl文件加载过程代码分析，知道如果Vendor\_0001\_Product\_0001.kl文件存在，则优先使用Vendor\_0001\_Product\_0001.kl，如不存在，则使用通用的Generic.kl文件去加载按键。至于s905x上没有使用Vendor\_0001\_Product\_0001.kl是因为我们使用的Vendor\_0001\_Product\_0001.kl中包含的DVB相关按键的映射值，而OTT平台我们是没有加DVB相关按键的映射值，因此在加载DVB相关的映射关系时出错，而重新使用通用的Generic.kl文件去加载映射。因此针对s905x平台在Vendor\_0001\_Product\_0001.kl把DVB的相关码值映射给屏蔽掉就能正常使用Vendor\_0001\_Product\_0001.kl去加载按键映射。

        另外，我们以前使用的Vendor\_0001\_Product\_0001.kl和Generic.kl按键键值不一样的码值做记录，把Vendor\_0001\_Product\_0001.kl以下三个键的映射修改为和Generic.kl一样的即可。

        OK键    ----  28  ---- "DPAD\_CENTER"

        HOME键  ----  172 ---- "HOME"

        MENU键  ----  139 ---- "MENU"