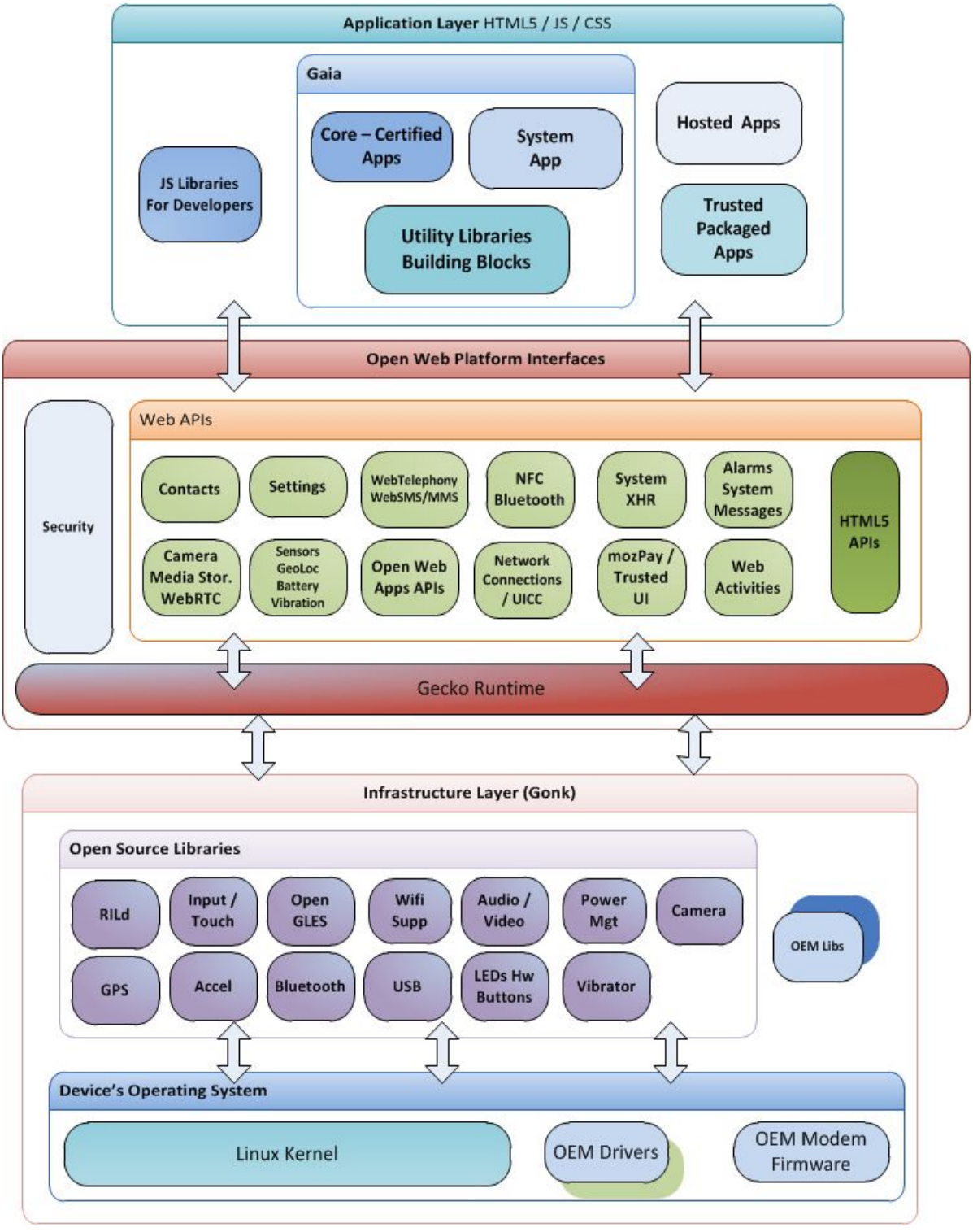
Firefox OS 输入系统分析

吴炜炜　2015.4.13

Firefox OS 是基于 Android 内核和现有的设备驱动程序，底层驱动部分与 android 一致，因此输入系统(input/touch)借用 android 的部分实现，在 gecko 层对获取的输入消息进行处理、分发。



1. 相关模块的介绍：

1.1 Gonk：

Gonk 在 Firefox OS stack 中可看作是 kernel 层级的组件，在 Gecko 和 底层硬件中间充当接口的作用。Gonk 对底层的硬件进行控制，并且将硬件信息及操控接口暴露给 Gecko 中的 Web APIs。Gonk 可以看作是一个 黑盒， 在屏幕后面做了所有复杂细节的工作用于在硬件层级上对 mobile device 进行控制。Gonk 是一个设备接口层，可看作是硬件和 Gecko 之间的适配器。

Gonk 是 Firefox OS 平台更低层的系统，包括了 Linux kernel （基于 AOSP）和用户空间硬件抽象层(HAL)。内核和一些用户空间库都是公共的开源项目：linux, libusb, bluez 等，其他的一些硬件抽象层部分是与 android 项目共享的：GPS, camera 等，这种设计方式对 OEM 将软件组件从其他 Android 实现上移植过来是非常方便的。你可以认为 Gonk 是一个非常简单的 Linux 版本。Gonk 是 Gecko 层的端口目标，也就是说 Gecko 层有到 Gonk 的端口，就像 Gecko 到 Mac OS X, Windows, 和 Android 一样。因为 Firefox OS 对 Gonk 拥有完全的控制权，相比其他操作系统，我们可以释放更多的接口到 Gecko。例如，Gecko 拥有到 Gonk 电话栈和帧缓冲区的直接入口，但在其他操作系统却没有。

1.2 gecko/widget:

针对每个平台实现的跨平台 API， 用来处理操作系统/环境组件，例如创建和处理窗口，弹出框以及其他本地组件相关的代码，以及将与绘制和事件相关的系统消息转化为用于 Mozilla 其他地方的消息（例如，view/ 和 content/，后者会转化许多消息到另外的 API，DOM 事件 API）。

1.3 b2g/chrome/content/shell.js：

shell.js 是在 Gaia system app 中装载的第一个脚本文件。

shell.js 导入了所有需要的模块，注册键值监听，定义了 sendCustomEvent 和 sendChromeEvent 与 Gaia 通信，并且提供了 webapp 的安装助手： indexedDB quota, RemoteDebugger, keyboard helper, 和 screenshot 工具。

shell.js 最重要的功能就是启动了 Gaia system app，之后又将整个系统相关的管理工作移交给了 Gaia system app。

2. 输入消息处理流程

2.1 android 处理流程

输入事件（input/touch）是要经过驱动层注册为输入设备，然后上报到 kernel/drivers/input/input.c 中，这里有相关函数的定义。然后通过 sys 上报到 frameworks/services/input/EventHub.cpp 中，在这里会对设备进行扫描并且判断是哪种设备，然后在 InputReader.cpp 中对原始数据进行读取。在 framewoks/services/input/InputDispatcher.cpp 中实现数据的派发。在 framework/base/core/jni/Android_view_KeyEvent.cpp 中实现通过 JNI 机制向上层的 KeyEvent.java 提供数据，并且在 frameworks/base/core/java/android/view/KeyEvent.java 中向上层的 APP 开发人员提供接口。

2.2 Firefox OS 处理流程

由于 Firefox Os 底层驱动部分与 android 一致，输入事件经 kernel 上报，在 gecko/widget/gonk 部分完成对事件的监听、处理及分发；然后 Mozilla 在 gecko/widget/gonk 部分整合 android 中 frameworks 部分对消息的监听、捕获、处理部分，并通过 nsIAppShell 进行封装。

这些事件是通过 nsIAppShell 的 Gonk implementation 传入的，这个接口用来表示 Gecko 应用的最初入口点。也就是说， 输入设备会调用 nspAppShell 对象的方法，来表示 Gecko 子系统想要发送事件到用户界面。

例如:

```
void GeckoInputDispatcher::notifyKey(nsecs_t eventTime,
                                     int32_t deviceId,
                                     int32_t source,
                                     uint32_t policyFlags,
                                     int32_t action,
                                     int32_t flags,
                                     int32_t keyCode,
                                     int32_t scanCode,
                                     int32_t metaState,
                                     nsecs_t downTime) {
  UserInputData data;
  data.timeMs = nanosecsToMillisecs(eventTime);
  data.type = UserInputData::KEY_DATA;
  data.action = action;
  data.flags = flags;
  data.metaState = metaState;
  data.key.keyCode = keyCode;
  data.key.scanCode = scanCode;
  {
    MutexAutoLock lock(mQueueLock);
    mEventQueue.push(data);
  }
  gAppShell->NotifyNativeEvent();
}
```

这些事件都来自于标准的 Linux input_event 系统。Firefox OS 使用了 light abstraction layer 来覆写它；它也提供了较好的特性如事件过滤机制。 您可以在 widget/gonk/libui/EventHub.cpp 文件的 EventHub::getEvents() 的方法中找到创建输入事件的代码。

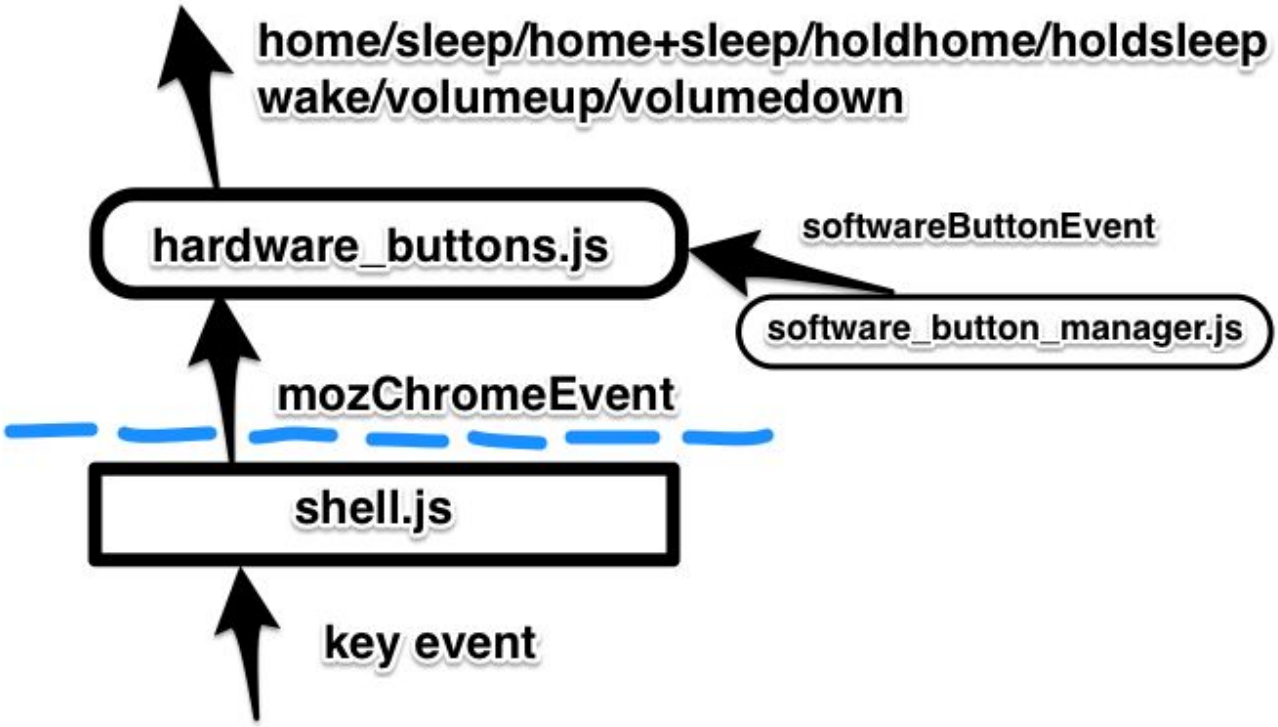
当 Gecko 层接收到事件时，就会通过 nsAppShell 分发到 DOM 中。

```
static nsEventStatus sendKeyEventWithMsg(uint32_t keyCode,
                                         uint32_t msg,
                                         uint64_t timeMs,
                                         uint32_t flags) {
    nsKeyEvent event(true, msg, NULL);
    event.keyCode = keyCode;
    event.location = nsIDOMKeyEvent::DOM_KEY_LOCATION_MOBILE;
    event.time = timeMs;
    event.flags |= flags;
    return nsWindow::DispatchInputEvent(event);
}
```

此后，事件就会由 Gecko 本身处理或作为 DOM events 分发到 web 应用中进一步处理。

Gecko 到 Gaia 的传递：

shell.js 过滤所有的硬件按键事件，并将其打包在 mozChromeEvent，然后发送到 gaia 的 syatem 应用。在 system 中，hardware_buttons.js 监听这些低层 mozChromeEvent，对其进行处理，并产生更高层次的事件处理，其他 system 的模块监听通过该模块所产生的高级别按键事件。

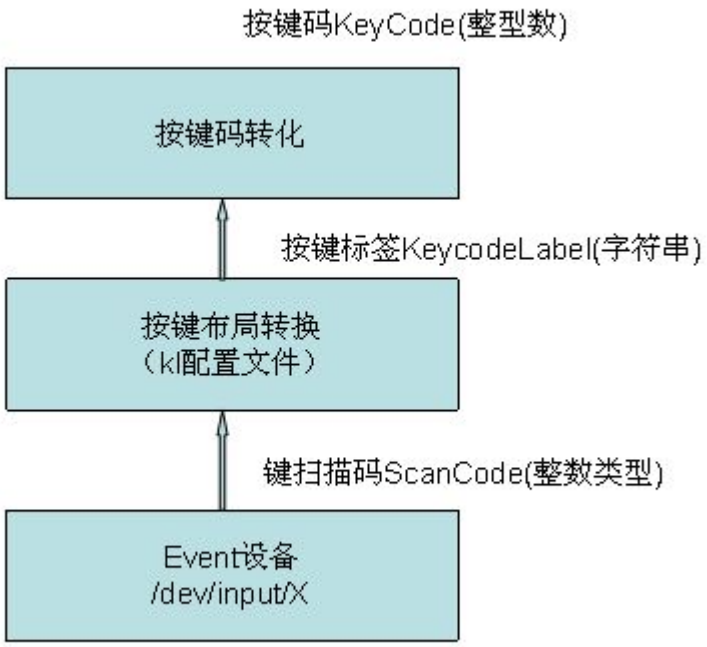


3. 新增按键的方法

3.1 输入事件的键值转换

android： ScanCode -> KeyCodeLabel -> KeyCode -> Keyevent

firefox os： ScanCode -> KeyCodeLabel -> KeyCode-> DomKeyCode -> DomKeyevent



1） 键扫描码（ScanCode）是由 linux 的 Input 驱动框架定义的整数类型，可参考 input.h 头文件（./external/kernel-headers/original/linux/input.h）。

2） 按键码（KeyCode）是一个整数，在上层的 JAVA 程序中主要通过这个值来判断系统的实现。

3） Dom 按键码（DomKeyCode）是 2 级 DOM 定义事件对应的按键码值，这个值 firefox os 中供 gaia 使用，此值遵循 W3C 标准定义，保证所有浏览器的事件键值有统一标准（具体参见：http://www.w3.org/TR/1999/WD-DOM-Level-2-19990923/events.html#Events-KeyEvent）。

实现过程：

1） ScanCode -> KeyCodeLabel

键盘布局文件(*.kl) 把 ScanCode 转换为 KeyCodeLabe , *.kl 文件在 ./frameworks/base/data/中定义，不同项目可以定义自己的 kl 文件，没有定义系统会默认使用 Generic.kl。

2） KeyCodeLabel -> KeyCode。

通过查找 KEYCODES[]数组，得到 KeyCodeLabel 字符串对应的 KeyCode 值。KEYCODES[]
在./gecko/widget/gonk/libui/KeycodeLabels.h 中定义，
在./gecko/widget/gonk/libui/android_keycodes.h 中定义 KeyCode 的枚举值,KeyCode 即为 android 使用的键值。

3） KeyCode-> DomKeyCode

由于 W3C 为浏览器定义的事件键值与 android 定义的不同，因此需要完成此步转换，通过 kKeyMapping[]完成，该数组在./gecko/widget/gonk/GonkKeyMapping.h 定义；完成转换后，通过 nsAppShell 分发到 DOM 中。

3.2 新增按键实例：

下面以 YES 键为例（FireE 2.2 项目），完成新增按键的添加。

1. Kernel 部分修改

由于使用虚拟按键模拟 GC 的一系列按键，是通过在./kernel/drivers/input/touchscreen/gt9xx 修改完

成的，如果是实体按键，需要在./kernel/drivers/input/keyboard 做修改。YES 键原本使用 enter 的键值，可能会与其他操作冲突，现使用 linux 输入中未被使用的值 KEY_OK（#define KEY_OK 0x160）替代。

2. *.kl 文件的修改

在./frameworks/base/data/keyboards/ Generic.kl 中添加 KeyCodeLabe 的转换定义：key 352 OK。

3. Gecko 部分修改

1）KeyCodeLabe 转换 KeyCode

由于 android 中也无 KEY_OK 对应键值定义，需要在./widget/gonk/libui/KeycodeLabe ls.h 中添加：{ "OK", 260 }, ./widget/gonk/libui/android_keycodes.h 中做对应添加： AKEYCODE_OK = 260。由于系统编译时会对 kl 文件做验证，使用的是 frameworks 中原有 android 的头文件，因此需要在原有 android 的 KeycodeLabels.h 和 keycodes.h 做同样修改，否则会导致编译不过。

2）KeyCode 转换 DomKeyCode

在./gecko/widget/gonk/GonkKeyMapping.h 添加 NS_VK_YES，Mozilla 只做了 android 0～164 键值的转换，我们需要扩充至 260 与 android 对应，中间用不到的值赋为 0 即可。NS_VK_YES 的定义通过宏定义字符串拼接完成，与 DOM_VK_YES 的值相同，在 dom/events/VirtualKeyCodeList.h 中添加：DEFINE_VK_INTERNAL(_YES) ，在 dom/webidl/KeyEvent.webidl 添加 DOM_VK_YES 的定义（使用 W3C 未使用的值）： const unsigned long DOM_VK_YES = 0xB1; dom/interfaces/events/nsIDOMKeyEvent.idl 需要做同样添加。

完成以上步骤，我们就可以在上层监听到 YES 键的事件，对应的 event.KeyCode 是 177，我们可能还需要 event.code 和 event.key。

event.code 需要在./widget/NativeKeyToDOMCodeName.h 中添加转换映射：CODE_MAP_ANDROID(Yes, 0x0160) ，在./dom/events/PhysicalKeyCodeNameList.h 添加 Yes 定义：DEFINE_PHYSICAL_KEY_CODE_NAME_WITH_SAME_NAME(Yes)；event.key 需要在./widget/NativeKeyToDOMKeyName.h 添加转换：KEY_MAP_ANDROID (Yes, AKEYCODE_OK)，在 dom/events/KeyNameList.h 中添加相应的定义：DEFINE_KEYNAME_WITH_SAME_NAME(Yes)。