从显示Tap原理一探Android 12的Input 系统

小虾米君 郭霖 2022-08-04 08:00 发表于江苏

/ 今日科技快讯 /

近日，对于“网传B站HR称核心用户都是Loser”冲上热搜一事，B站回应表示，对该事件中的面试官言论非常气愤。经内部调查，该面试官已于2021年年底被劝退，会吸取教训加强管理，感谢监督。

/ 作者简介 /

本篇文章转自TechMerger的博客，文章主要分享了他对Android中Input系统的探索分析，相信会对大家有所帮助！

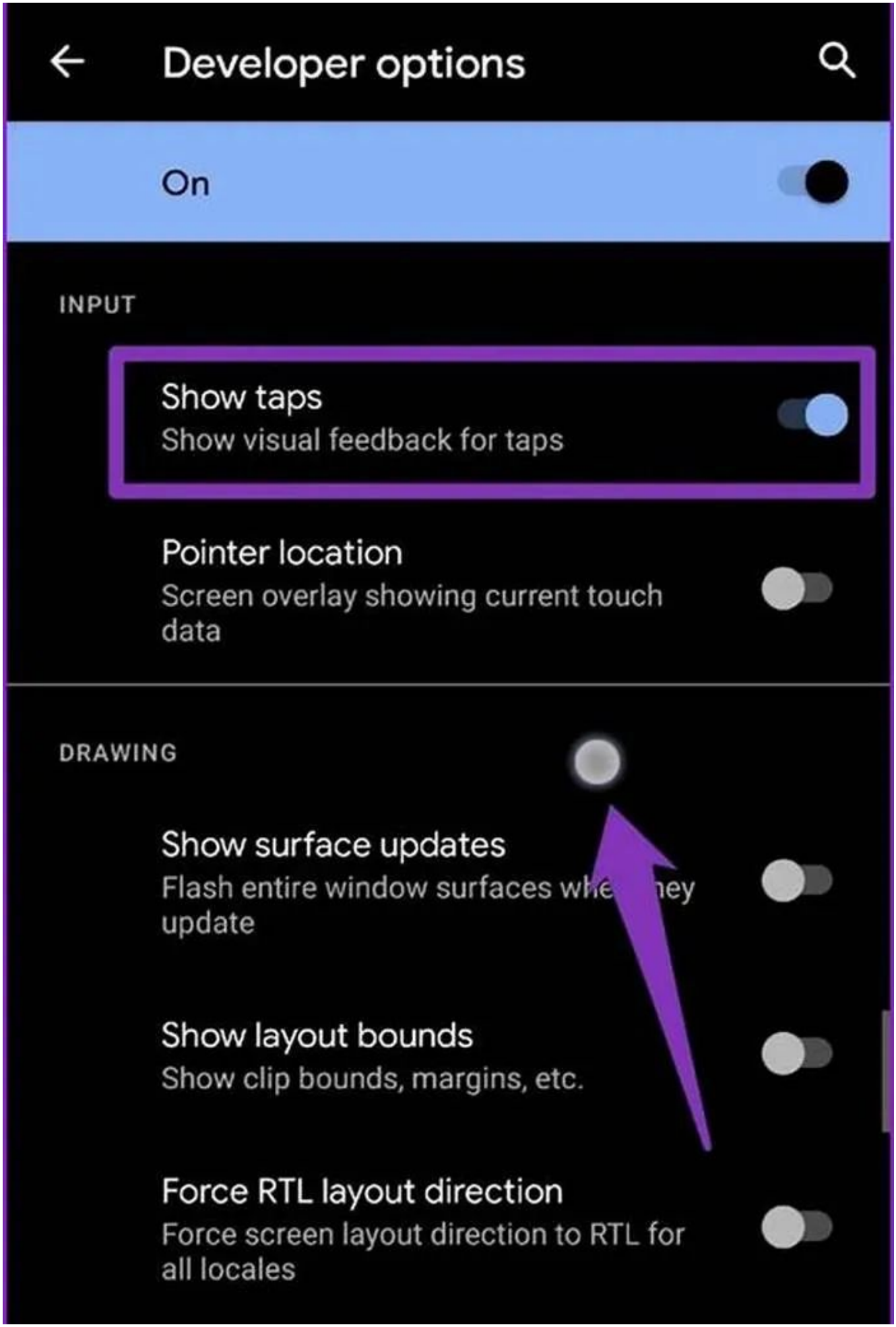
原文地址：

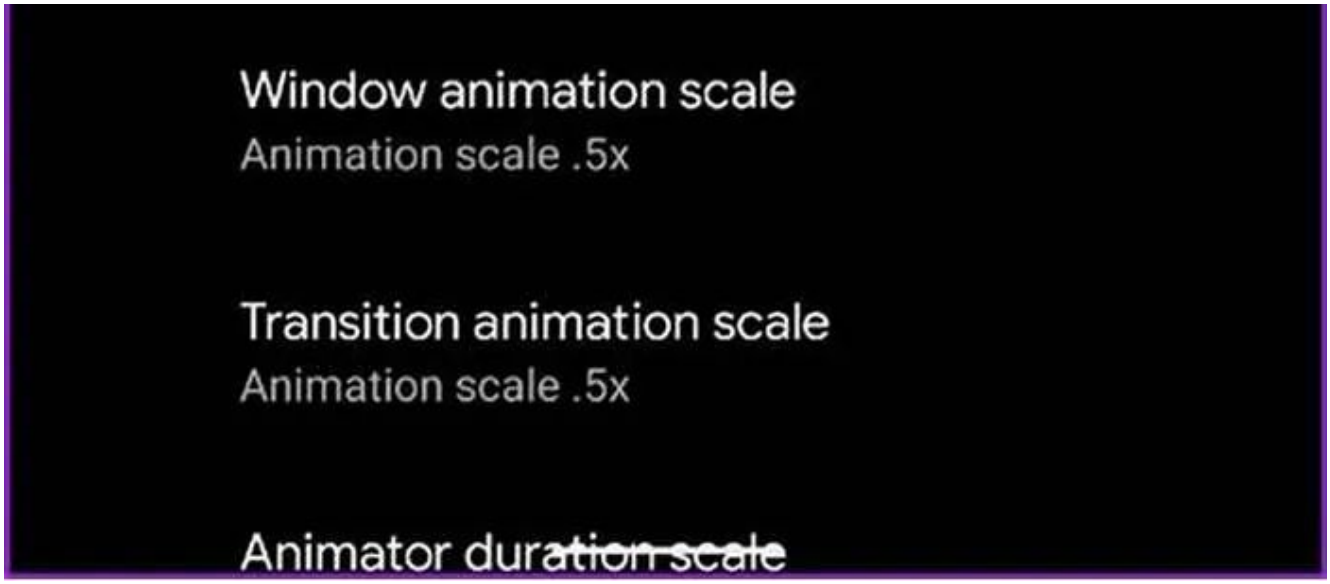
> https://juejin.cn/post/7102650378478419976

/ 前言 /

原以为显示 Tap 位置是 ViewRootImpl 里依据 Touch 位置显示的 PopupWindow ，实际不是、而且要复杂得多。

开发者选项画面里的 “Show taps” 选项，开发者一定不陌生。开启之后，截屏或录屏里可以直观地展示点击过的位置，非常方便。


Developer options
On
INPUT
Show taps
Show visual feedback for taps
Pointer location
Screen overlay showing current touch data
DRAWING
Show surface updates
Flash entire window surfaces whe ney update
Show layout bounds
Show clip bounds, margins, etc.
Force RTL layout direction
Force screen layout direction to RTL for all locales



类似的选项还有显示 Touch 参数的“Pointer location”，原理差不多。本次我们聚焦“Show taps”的功能，查阅 Android 12 的源码，将开启和显示流程分析清楚。

借此也窥探一下 Android 最重要的 Input 系统。

/ 正文 /

Settings 写入设置

首先是 Settings App 提供的开发者选项画面响应点击，将“Show taps”选项对应的设置 Key SHOW_TOUCHES 的 ON 值通过 android.provder.Settings 接口写入到保存系统设置数据的 SettingsProvier 中。

```
// packages/apps/Settings/src/com/android/settings/development/ShowTapsPreferenceController.ja
public class ShowTapsPreferenceController extends DeveloperOptionsPreferenceController ... {
  ...
  @Override
  public boolean onPreferenceChange(Preference preference, Object newValue) {
    final boolean isEnabled = (Boolean) newValue;
    Settings.System.putInt(mContext.getContentResolver(),
        Settings.System.SHOW_TOUCHES, isEnabled ? SETTING_VALUE_ON : SETTING_VALUE_OFF);
    return true;
  }
  ...
}
```

IMS 监听和反映设置

负责管理输入的系统服务 InputManagerService 在启动之际，会注册监听 SHOW_TOUCHES Key 的观察者，在设置产生变化的时候调用 JNI 开始反映设置。

```
// frameworks/base/services/core/java/com/android/server/input/InputManagerService.java
public class InputManagerService extends IInputManager.Stub... {
   ...
   public void start() {
      registerShowTouchesSettingObserver();
      ...
   }

   private void registerShowTouchesSettingObserver() {
      mContext.getContentResolver().registerContentObserver(
           Settings.System.getUriFor(Settings.System.SHOW_TOUCHES), true,
           new ContentObserver(mHandler) {
              @Override
              public void onChange(boolean selfChange) {
                 updateShowTouchesFromSettings();
              }
           }, UserHandle.USER_ALL);
   }

   private void updateShowTouchesFromSettings() {
      int setting = getShowTouchesSetting(0);
      nativeSetShowTouches(mPtr, setting != 0);
   }
   ...
}
```

JNI 端的 NativeInputManager 持有 InputFlinger，向其中负责读取事件的 InputReader 发出更新配置的请求，配置变更的 Type 为 CHANGE_SHOW_TOUCHES。

在此之前需要先更新管理配置信息的 mLocked 结构体中的 showTouches 成员，InputFlinger 在刷新配置的时候需要验证。

```
// frameworks/base/services/core/jni/com_android_server_input_InputManagerService.cpp
static void nativeSetShowTouches(JNIEnv* /* env */,
      jclass /* clazz */, jlong ptr, jboolean enabled) {
   NativeInputManager* im = reinterpret_cast<NativeInputManager*>(ptr);
```

```
    im->setShowTouches(enabled);
}

void NativeInputManager::setShowTouches(bool enabled) {
    { // acquire lock
        ...
        mLocked.showTouches = enabled;
    } // release lock

    mInputManager->getReader()->requestRefreshConfiguration(
            InputReaderConfiguration::CHANGE_SHOW_TOUCHES);
}
```

通过 InputReader 请求刷新配置

InputReader 接收到配置变化的 Type 之后，会根据记录待刷新配置的变量 mConfigurationChangesToRefresh 判断当前是否已经在刷新过程中。

如果尚未处于刷新中，则标记需要 wake 事件源头 EventHub，之后会将该变化添加该变量到中，最后就是通知 EventHub 唤醒。

```
// frameworks/native/services/inputflinger/reader/InputReader.cpp
void InputReader::requestRefreshConfiguration(uint32_t changes) {
    std::scoped_lock _l(mLock);

    if (changes) {
        bool needWake = !mConfigurationChangesToRefresh;
        mConfigurationChangesToRefresh |= changes;

        if (needWake) {
            mEventHub->wake();
        }
    }
}
```

EventHub 唤醒 InputReader 线程

IMS 过来的刷新请求最终需要 InputReader 线程来处理。

可是 InputReader 线程处在从 EventHub 中读取事件和没有事件时便调用 epoll_wait 进入等待状态的循环当中。

所以为了让其即刻处理配置变化，需要 EventHub 的手动唤醒。

```
// frameworks/native/services/inputflinger/reader/EventHub.cpp
void EventHub::wake() {
    ALOGV("wake() called");

    ssize_t nWrite;
    do {
        nWrite = write(mWakeWritePipeFd, "W", 1);
    } while (nWrite == -1 && errno == EINTR);

    if (nWrite != 1 && errno != EAGAIN) {
        ALOGW("Could not write wake signal: %s", strerror(errno));
    }
}

size_t EventHub::getEvents(int timeoutMillis, RawEvent* buffer, size_t bufferSize) {
    ...
    for (;;) {
        ...
        int pollResult = epoll_wait(mEpollFd, mPendingEventItems, EPOLL_MAX_EVENTS, timeoutMillis);
        ...
    }
    ...
}
```

InputReader 线程刷新配置

EventHub 唤醒后处于等待状态的 getEvents() 会结束，之后 InputReader 线程会进入下次循环即 loopOnce()。

其首先将检查是否存在待刷新的配置变化 changes，存在的话调用 refreshConfigurationLocked() 让 InputDevice 去重新配置这项变化。

```
void InputReader::loopOnce() {
    ...
    std::vector<InputDeviceInfo> inputDevices;
    { // acquire lock
        ...
```

```
      uint32_t changes = mConfigurationChangesToRefresh;
      if (changes) {
        mConfigurationChangesToRefresh = 0;
        timeoutMillis = 0;
        refreshConfigurationLocked(changes);
      } else if (mNextTimeout != LLONG_MAX) {
        nsecs_t now = systemTime(SYSTEM_TIME_MONOTONIC);
        timeoutMillis = toMillisecondTimeoutDelay(now, mNextTimeout);
      }
    } // release lock

    size_t count = mEventHub->getEvents(timeoutMillis, mEventBuffer, EVENT_BUFFER_SIZE);
    ...
}
```

需要留意，refreshConfigurationLocked() 在调用 InputDevice 进一步处理之前需要先从 JNI 获取配置（getReaderConfiguration()）的变化放入 mConfig 中。

```
void InputReader::refreshConfigurationLocked(uint32_t changes) {
    mPolicy->getReaderConfiguration(&mConfig);
    ...

    if (changes & InputReaderConfiguration::CHANGE_MUST_REOPEN) {
        mEventHub->requestReopenDevices();
    } else {
        for (auto& devicePair : mDevices) {
            std::shared_ptr<InputDevice>& device = devicePair.second;
            device->configure(now, &mConfig, changes);
        }
    }
    ...
}
```

InputDevice 配置变化

InputDevice 的 configure() 需要处理很多配置变化，比如键盘布局、麦克风等。对于 Show taps 的变化关注调用 InputMapper 的 congfigure() 即可。

```
// frameworks/native/services/inputflinger/reader/InputDevice.cpp
void InputDevice::configure(nsecs_t when, const InputReaderConfiguration* config,
                            uint32_t changes) {
    ...
    if (!isIgnored()) {
        ...
```

```
    for_each_mapper([this, when, config, changes](InputMapper& mapper) {
      mapper.configure(when, config, changes);
      mSources |= mapper.getSources();
    });
    ...
  }
}
```

TouchInputMapper 进一步处理

众多输入事件的物理数据需要对应的 InputMapper 来转化为上层能识别的事件类型。比如识别键盘输入的 KeyboardInputMapper、识别震动的 VibratorInputMapper 等等。

现在的触摸屏都支持多点触控，所以是 MultiTouchInputMapper 来处理的。可 MultiTouchInputMapper 没有复写 configure()，而是沿用由父类 TouchInputMapper 的共通处理。

```
// frameworks/native/services/inputflinger/reader/mapper/TouchInputMapper.cpp
void TouchInputMapper::configure(nsecs_t when, const InputReaderConfiguration* config,
                    uint32_t changes) {
  ...
  bool resetNeeded = false;
  if (!changes ||
    (changes &
     (InputReaderConfiguration::CHANGE_DISPLAY_INFO |
      InputReaderConfiguration::CHANGE_POINTER_CAPTURE |
      InputReaderConfiguration::CHANGE_POINTER_GESTURE_ENABLEMENT |
      InputReaderConfiguration::CHANGE_SHOW_TOUCHES |
      InputReaderConfiguration::CHANGE_EXTERNAL_STYLUS_PRESENCE))) {
    // Configure device sources, surface dimensions, orientation and
    // scaling factors.
    configureSurface(when, &resetNeeded);
  }
  ...
}
```

TouchInputMapper 会依据 changes 的类型进行对应处理，对于 SHOW_TOUCHES 的变化需要调用 configureSurface() 进一步处理。

创建和初始化 PointerController

configureSurface() 进行多个参数的测量和配置，其中和 “Show taps” 相关的是 PointerController 的创建，该类是 Mouse、Taps、Pointer location 等系统 Touch 显示的专用类。

需要留意的是创建之前需要验证从 JNI 里取得的 mConfig.showTouches 变量。

```
void TouchInputMapper::configureSurface(nsecs_t when, bool* outResetNeeded) {
    ...
    // Create pointer controller if needed, and keep it around if Pointer Capture is enabled to
    // preserve the cursor position.
    if (mDeviceMode == DeviceMode::POINTER ||
        (mDeviceMode == DeviceMode::DIRECT && mConfig.showTouches) ||
        (mParameters.deviceType == Parameters::DeviceType::POINTER && mConfig.pointerCapture)) {
        if (mPointerController == nullptr) {
            mPointerController = getContext()->getPointerController(getDeviceId());
        }
        if (mConfig.pointerCapture) {
            mPointerController->fade(PointerControllerInterface::Transition::IMMEDIATE);
        }
    } else {
        mPointerController.reset();
    }
    ...
}
```

getPointerController() 会回调到 InputReader 开启 PointerController 的创建和初始化。

```
std::shared_ptr<PointerControllerInterface> InputReader::getPointerControllerLocked(
        int32_t deviceId) {
    std::shared_ptr<PointerControllerInterface> controller = mPointerController.lock();
    if (controller == nullptr) {
        controller = mPolicy->obtainPointerController(deviceId);
        mPointerController = controller;
        updatePointerDisplayLocked();
    }
    return controller;
}
```

用于创建的 obtainPointerController() 的实现在 JNI 里，调用 PointerController 的静态方法 create() 开始构建实例。

```
std::shared_ptr<PointerControllerInterface> NativeInputManager::obtainPointerController(
    int32_t /* deviceId */) {
  ...
  std::shared_ptr<PointerController> controller = mLocked.pointerController.lock();
  if (controller == nullptr) {
    ensureSpriteControllerLocked();

    controller = PointerController::create(this, mLooper, mLocked.spriteController);
    mLocked.pointerController = controller;
    updateInactivityTimeoutLocked();
  }

  return controller;
}
```

PointerController 构建的同时需要构建持有的 MouseCursorController。

```
// frameworks/base/libs/input/PointerController.cpp
std::shared_ptr<PointerController> PointerController::create( ... ) {
  std::shared_ptr<PointerController> controller = std::shared_ptr<PointerController>(
      new PointerController(policy, looper, spriteController));
  ...
  return controller;
}

PointerController::PointerController( ... )
    : mContext(policy, looper, spriteController, *this), mCursorController(mContext) {
  std::scoped_lock lock(mLock);
  mLocked.presentation = Presentation::SPOT;
}
```

obtainPointerController() 执行完之后调用 updatePointerDisplayLocked() 执行 PointerController 的初始化。

初始化 PointerController

调用 PointerController 的 setDisplayViewport() 传入显示用的 DisplayViewPort。

```
void InputReader::updatePointerDisplayLocked() {
  ...
  std::optional<DisplayViewport> viewport =
      mConfig.getDisplayViewportById(mConfig.defaultPointerDisplayId);
  if (!viewport) {
```

```
    ...
    viewport = mConfig.getDisplayViewportById(ADISPLAY_ID_DEFAULT);
  }
  ...
  controller->setDisplayViewport(*viewport);
}
```

setDisplayViewport() 需要持有的 MouseCursorController 进一步初始化。

```
void PointerController::setDisplayViewport(const DisplayViewport& viewport) {
  ...
  mCursorController.setDisplayViewport(viewport, getAdditionalMouseResources);
}
```

MouseCursorController 需要获取 Display 相关的参数，并执行两个重要步骤：

1. loadResourcesLocked()
2. updatePointerLocked()

```
// frameworks/base/libs/input/MouseCursorController.cpp
void MouseCursorController::setDisplayViewport(const DisplayViewport& viewport,
                           bool getAdditionalMouseResources) {
  ...
  // Reset cursor position to center if size or display changed.
  if (oldViewport.displayId != viewport.displayId || oldDisplayWidth != newDisplayWidth ||
    oldDisplayHeight != newDisplayHeight) {
    float minX, minY, maxX, maxY;
    if (getBoundsLocked(&minX, &minY, &maxX, &maxY)) {
      mLocked.pointerX = (minX + maxX) * 0.5f;
      mLocked.pointerY = (minY + maxY) * 0.5f;
      // Reload icon resources for density may be changed.
      loadResourcesLocked(getAdditionalMouseResources);
    ...
    }
  } else if (oldViewport.orientation != viewport.orientation) {
    ...
  }

  updatePointerLocked();
}
```

加载 Pointer 相关资源

```
void MouseCursorController::loadResourcesLocked(bool getAdditionalMouseResources) REQUIRES(mLoc
    ...
    policy->loadPointerResources(&mResources, mLocked.viewport.displayId);
    policy->loadPointerIcon(&mLocked.pointerIcon, mLocked.viewport.displayId);
    ...
}
```

省略诸多细节，loadPointerResources() 将通过 IMS 的 JNI 端以及 PointerIcon 的 JNI 端创建 PointerIcon 实例，并读取显示 Taps 的资源。

getSystemIcon() 则是负责的函数，其将读取系统资源里名为 Pointer 的 Style，并读取 Taps 对应的资源 ID。

```
// frameworks/base/core/java/android/view/PointerIcon.java
    public static PointerIcon getSystemIcon(@NonNull Context context, int type) {
        ...
        int typeIndex = getSystemIconTypeIndex(type);
        if (typeIndex == 0) {
            typeIndex = getSystemIconTypeIndex(TYPE_DEFAULT);
        }

        int defStyle = sUseLargeIcons ?
                com.android.internal.R.style.LargePointer : com.android.internal.R.style.Pointer;
        TypedArray a = context.obtainStyledAttributes(null,
                com.android.internal.R.styleable.Pointer,
                0, defStyle);
        int resourceId = a.getResourceId(typeIndex, -1);
        ...
        icon = new PointerIcon(type);
        if ((resourceId & 0xff000000) == 0x01000000) {
            icon.mSystemIconResourceId = resourceId;
        } else {
            icon.loadResource(context, context.getResources(), resourceId);
        }
        systemIcons.append(type, icon);
        return icon;
    }

    private static int getSystemIconTypeIndex(int type) {
        switch (type) {
            ...
            case TYPE_SPOT_TOUCH:
                return com.android.internal.R.styleable.Pointer_pointerIconSpotTouch;
            ...
```

```
        default:
            return 0;
    }
}
```

资源 ID 为 pointer_spot_touch_icon。

```
<!-- frameworks/base/core/res/res/drawable/pointer_spot_touch_icon.xml -->
<?xml version="1.0" encoding="utf-8"?>
<pointer-icon xmlns:android="http://schemas.android.com/apk/res/android"
    android:bitmap="@drawable/pointer_spot_touch"
    android:hotSpotX="16dp"
    android:hotSpotY="16dp" />
```

其指向的图片就是如下熟悉的 Spot png：pointer_spot_touch.png。之后的 loadPointerIcon 阶段会将该图片解析成 Bitmap 并被管理在 SpriteIcon 中。

而 SpriteIcon 在 updatePointerLocked() 阶段会被存放到 SpriteController 中，等待显示的调度。

```
void MouseCursorController::updatePointerLocked() REQUIRES(mLock) {
    if (!mLocked.viewport.isValid()) {
        return;
    }
    sp<SpriteController> spriteController = mContext.getSpriteController();
    spriteController->openTransaction();

    ...
    if (mLocked.updatePointerIcon) {
        if (mLocked.requestedPointerType == mContext.getPolicy()->getDefaultPointerIconId()) {
            mLocked.pointerSprite->setIcon(mLocked.pointerIcon);
        ...
        }
        mLocked.updatePointerIcon = false;
    }

    spriteController->closeTransaction();
}
```

显示 tap

点击的时候 EventHub#getEvents() 会产生事件，InputReader#loopOnce() 会调用 processEventsLocked() 处理事件。

```
void InputReader::loopOnce() {
  ...
  size_t count = mEventHub->getEvents(timeoutMillis, mEventBuffer, EVENT_BUFFER_SIZE);

  { // acquire lock
    ...
    if (count) {
      processEventsLocked(mEventBuffer, count);
    }
    ....
  } // release lock
  ...
}
```

之后调用 InputMapper 开始加工事件，并在 TouchInputMapper#cookAndDispatch() 的时候调用 updateTouchSpots() 更新 PointerController 的一些参数。

```
void TouchInputMapper::updateTouchSpots() {
  ...
 mPointerController->setPresentation(PointerControllerInterface::Presentation::SPOT);
  mPointerController->fade(PointerControllerInterface::Transition::GRADUAL);

  mPointerController->setButtonState(mCurrentRawState.buttonState);
  setTouchSpots(mCurrentCookedState.cookedPointerData.pointerCoords,
          mCurrentCookedState.cookedPointerData.idToIndex,
          mCurrentCookedState.cookedPointerData.touchingIdBits, mViewport.displayId);
}
```

其中比较关键的 setTouchSpots() 是显示 Taps 的关键步骤，准备 x、y 坐标和压力值。

在 Reader 而不是 Dispatch、更不是 ViewRootImpl 的时候处理的原因在于：Read 到事件即显示可以更早地响，同时不用占用 App 进程。

```
void TouchInputMapper::setTouchSpots(const PointerCoords* spotCoords, const uint32_t* spotIdToIn
                    BitSet32 spotIdBits, int32_t displayId) {
  std::array<PointerCoords, MAX_POINTERS> outSpotCoords{};
```

```
    for (BitSet32 idBits(spotIdBits); !idBits.isEmpty();) {
        const uint32_t index = spotIdToIndex[idBits.clearFirstMarkedBit()];
        float x = spotCoords[index].getX();
        float y = spotCoords[index].getY();
        float pressure = spotCoords[index].getAxisValue(AMOTION_EVENT_AXIS_PRESSURE);
        ...
    }

    mPointerController->setSpots(outSpotCoords.data(), spotIdToIndex, spotIdBits, displayId);
}
```

其后 PointerController 会通过 TouchSpotController 创建 Spot 实例向其发送 updateSprite() 请求。最后回调 SpriteController 调用 setIcon 处理。

```
// frameworks/base/libs/input/TouchSpotController.cpp
void TouchSpotController::Spot::updateSprite(const SpriteIcon* icon, float x, float y,
                            int32_t displayId) {
    sprite->setLayer(Sprite::BASE_LAYER_SPOT + id);
    ...
    if (icon != mLastIcon) {
        mLastIcon = icon;
        if (icon) {
            sprite->setIcon(*icon);
            sprite->setVisible(true);
        } else {
            sprite->setVisible(false);
        }
    }
}


// frameworks/base/libs/input/SpriteController.cpp
void SpriteController::SpriteImpl::setIcon(const SpriteIcon& icon) {
    AutoMutex _l(mController->mLock);
    ...
    invalidateLocked(dirty);
}

void SpriteController::SpriteImpl::invalidateLocked(uint32_t dirty) {
    ...
    if (!wasDirty) {
        mController->invalidateSpriteLocked(this);
    }
}
```

```
void SpriteController::invalidateSpriteLocked(const sp<SpriteImpl>& sprite) {
    bool wasEmpty = mLocked.invalidatedSprites.isEmpty();
    mLocked.invalidatedSprites.push(sprite);
    if (wasEmpty) {
        if (mLocked.transactionNestingCount != 0) {
            mLocked.deferredSpriteUpdate = true;
        } else {
            mLooper->sendMessage(mHandler, Message(MSG_UPDATE_SPRITES));
        }
    }
}
```

MSG_UPDATE_SPRITES 经过 Handler 回调 doUpdateSprites()，将取出封装在 SpriteUpdate 中的 SpriteIcon 并执行 draw。

```
void SpriteController::doUpdateSprites() {
    ...
    for (size_t i = 0; i < numSprites; i++) {
        SpriteUpdate& update = updates.editItemAt(i);

        if ((update.state.dirty & DIRTY_BITMAP) && update.state.surfaceDrawn) {
            update.state.surfaceDrawn = false;
            update.surfaceChanged = surfaceChanged = true;
        }

        if (update.state.surfaceControl != NULL && !update.state.surfaceDrawn
                && update.state.wantSurfaceVisible()) {
            sp<Surface> surface = update.state.surfaceControl->getSurface();
            if (update.state.icon.draw(surface)) {
                update.state.surfaceDrawn = true;
                update.surfaceChanged = surfaceChanged = true;
            }
        }
    }
    ...
    updates.clear();
}
```

最后，SpriteIcon 将取出 Bitmap 描画到 Surface 的 Canvas 上去。

```
bool SpriteIcon::draw(sp<Surface> surface) const {
    ...
    graphics::Paint paint;
    paint.setBlendMode(ABLEND_MODE_SRC);

    graphics::Canvas canvas(outBuffer, (int32_t)surface->getBuffersDataSpace());
```

```
    canvas.drawBitmap(bitmap, 0, 0, &paint);
    ...
    status = surface->unlockAndPost();
    if (status) {
      ALOGE("Error %d unlocking and posting sprite surface after drawing.", status);
    }
    return !status;
}
```

/ 总体流程 /

通过一个框图简单回顾一下整个流程。

可以看到，简简单单的 Show taps 功能，从设置、配置、刷新再到显示，经历了多个进程、多个模块的协力。

通过这个过程的梳理，可以一探 InputManagerService 和 InputFinger 的大体原理。但篇幅有限省略了很多细节，有兴趣彻底贯通 Android Input 系统的话，没有捷径、只得一字一句地逐个啃！

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

Activity的五种启动模式

模仿Android微信小程序，实现小程序独立任务视图的效果

欢迎关注我的公众号

学习技术或投稿

▼

长按上图，识别图中二维码即可关注

阅读原文

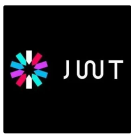
喜欢此内容的人还喜欢

WASM将引领下一代计算范式[译]

光谷码农

JWT：谁创造了我，我听命于谁

桑小榆呀



神作《凤凰架构：构建可靠的大型分布式系统》PDF 来了

码猿技术专栏