Овладение жестами в Android

Что же происходит в системе, когда пользователь касается экрана? И самое главное — как с этим обращаться? Пора разобраться с этим раз и навсегда. Вашему вниманию представляется заметка об обретенном понимании и опыте использования Android Touch System.

1. [Внешность обманчива](https://stfalcon.com/ru/blog/post/learning-android-gestures#itsnoteasy)
2. [Что же под капотом?](https://stfalcon.com/ru/blog/post/learning-android-gestures#whatsinside)
3. [Системные детекторы жестов и касаний](https://stfalcon.com/ru/blog/post/learning-android-gestures#systemgesturedetectors)
4. [Свой детектор жестов](https://stfalcon.com/ru/blog/post/learning-android-gestures#writegesturedetector)
5. [Перехватывай и делегируй!](https://stfalcon.com/ru/blog/post/learning-android-gestures#interceptgestures)
6. [Напоследок](https://stfalcon.com/ru/blog/post/learning-android-gestures#conclusion)

1. Внешность обманчива

Недавно передо мной встала задача разработать [FrescoImageViewer](https://github.com/stfalcon-studio/FrescoImageViewer" \t "_blank) — библиотеку для просмотра фотографий, загружаемых при помощи [Fresco](http://frescolib.org/" \t "_blank). Помимо этого необходимо было реализовать «pinch to zoom», переключение посредством ViewPager, а так же некое подобие «swipe to dismiss» — возможность закрыть изображение вертикальным свайпом. Собрав вместе основные компоненты я столкнулся с основной проблемой — **конфликтом жестов**.

Поскольку у меня был довольно скромный опыт в этой сфере, то первое решение, которое пришло мне в голову было таким: анализировать события в onTouchEvent() внутри моего CustomView и в нужный момент передавать управление. Но на деле поведение оказалось не таким очевидным, как я того ожидал.

В документации указано, что onTouchEvent() должен возвращать *true*, если событие было обработано, и *false* в противном случае. Однако, по каким-то причинам, там не указано, что **если вернуть *true*, а потом изменить значение обратно на *false* — поведение не изменится до окончания жеста**. Т.е. сообщив системе из onTouchEvent(), что элемент заинтересован в происходящем — это решение неизменно. Именно этот нюанс заставил меня потрепать себе нервы, а после открыть Google и погрузиться в изучение, как оказалось, целого фреймворка управления жестами.

2. Что же под капотом?

Итак, для понимания происходящего, предлагаю шаг за шагом разобрать, что происходит внутри этого нехитрого механизма на примере Activity с ViewGroup и дочерним View внутри, на который мы только что опустили палец:

1. Событие ввода оборачивается системой в объект MotionEvent, в котором находится вся полезная информация (тип действия, текущие и предыдущие координаты касания, время события, количество пальцев на экране и их порядок и т. д.).
2. Сформированный объект попадает в Activity.dispatchTouchEvent() (который всегда вызывается первым). Если активность не возвращает *true* (не заинтересована в обработке события на своем уровне), то событие отправляется корневому View.
3. Корневой элемент вызывает dispatchTouchEvent() у всех причастных дочерних элементов в обратном порядке их добавления. А те, в свою очередь, делают то же со своими дочерними элементами, таким образом «пропуская» событие вниз по вложенности до тех пор, пока на него кто-то не отреагирует (вернет *true*).\*
4. Дойдя до dispatchTouchEvent() самого нижнего View, цепочка идет в обратную сторону путем метода onTouchEvent(), который так же возвращает результат своей заинтересованности.
5. Если же никто не заинтересован — событие вернется в Activity.onTouchView().

Так же во ViewGroup и View перед вызовом onTouchEvent() проверяется наличие OnTouchListener. Если он был задан — вызовется OnTouchListener.onTouch(), в противном же случае — onTouchEvent().

*\*Здесь есть одна оговорка — у ViewGroup после dispatchTouchEvent() дополнительно вызывается onInterceptTouchEvent(), давая возможность перехватить событие, не оповещая вложенные элементы, тем самым изменив поведение ViewGroup на идентичное View:*

* *Если при перехвате жеста ViewGroup сообщит о своей заинтересованности — все дочерние элементы получат ACTION\_CANCEL.*
* *В случае, когда внутри View нужно избежать перехватывания родительским контейнером и его предками, необходимо вызвать requestDisallowInterceptTouchEvent(true) у ViewGroup.*

3. Системные детекторы жестов и касаний

Благо, при работе с жестами не нужно изобретать велосипеды и обрабатывать все вручную. Во многом помогает зашитый в SDK GestureDetector. Он включает в себя интерфейсы OnGestureListener, OnDoubleTapListener и OnContextClickListener для уведомлений о произошедшем событии и его типе. Вот как они выглядят:

**public** **interface** OnGestureListener {

**boolean** onDown(MotionEvent e);

**void** onShowPress(MotionEvent e);

**boolean** onSingleTapUp(MotionEvent e);

**boolean** onScroll(MotionEvent e1, MotionEvent e2, **float** distanceX, **float** distanceY);

**void** onLongPress(MotionEvent e);

**boolean** onFling(MotionEvent e1, MotionEvent e2, **float** velocityX, **float** velocityY);

}

**public** **interface** OnDoubleTapListener {

**boolean** onSingleTapConfirmed(MotionEvent e);

**boolean** onDoubleTap(MotionEvent e);

**boolean** onDoubleTapEvent(MotionEvent e);

}

**public** **interface** OnContextClickListener {

**boolean** onContextClick(MotionEvent e);

}

Как видно из названий методов, с помощью GestureDetector мы можем распознать singleTap, doubleTap, longPress, scroll и fling (подробное описание каждого из методов можно найти в Javadoc или [официальной документации Android](https://developer.android.com/reference/android/view/GestureDetector.html)).

Но этого мало! Есть еще ScaleGestureDetector и у него всего лишь один listener:

**public** **interface** OnScaleGestureListener {

**boolean** onScale(ScaleGestureDetector detector);

**boolean** onScaleBegin(ScaleGestureDetector detector);

**void** onScaleEnd(ScaleGestureDetector detector);

}

Он распознает жест «щепок» (или «pinch») и оповещает о его начале, конце и продолжительности. Кроме слушателя присутствуют вспомогательные методы для получения всей необходимой информации (см. [документацию](https://developer.android.com/reference/android/view/ScaleGestureDetector.html)).

Со встроенными классами мы теперь знакомы, но как же их использовать? Да очень просто! Необходимо просто создать экземпляр нужного нам детектора:

scaleDetector = **new** ScaleGestureDetector(context, listener());

...и передавать в него полученный MotionEvent. К примеру, в onTouchEvent():

@Override

**public** **boolean** onTouchEvent(MotionEvent event) {

scaleDetector.onTouchEvent(event);

**return** **super**.onTouchEvent(event);

}

Готово! Все распознаваемые жесты будут попадать в переданный listener.

4. Свой детектор жестов

К сожалению, стандартными средствами мы можем узнать только о касании и о движении указателя (MotionEvent.ACTION\_DOWN и MotionEvent.ACTION\_MOVE), но иногда (хотя, мне кажется, намного чаще), при обработке жестов, необходимо знать их **направление**. В этом нам не в силах помочь даже стандартные детекторы, а потому придется написать собственный.

Назовем его SwipeDirectionDetector. Его задача единоразово оповестить слушателя о выявлении направления свайпа. Логика простая: запоминаем координаты события на ACTION\_DOWN, а после измеряем длину до точки на ACTION\_MOVE. Как только дистанция достаточная для определения направления — вычисляем угол и на его основании получаем направление.

Для начала определим метод onTouchEvent(), принимающий MotionEvent, и опишем в нем логику вычислений:

**public** **boolean** onTouchEvent(MotionEvent event) {

**switch** (event.getAction()) {

**case** MotionEvent.ACTION\_DOWN:

startX = event.getX();

startY = event.getY();

**break**;

**case** MotionEvent.ACTION\_CANCEL:

**case** MotionEvent.ACTION\_UP:

startX = startY = 0.0f;

**break**;

**case** MotionEvent.ACTION\_MOVE:

**if** (getDistance(event) > touchSlop) {

**float** x = event.getX();

**float** y = event.getY();

Direction direction = Direction.get(getAngle(startX, startY, x, y));

onDirectionDetected(direction);

}

**break**;

}

**return** **false**;

}

Объект Direction мы определим как enum и добавим туда методы get() для определения направления в зависимости от угла и inRange() для проверки его попадания в заданный диапазон.

**public** **enum** Direction {

UP,

DOWN,

LEFT,

RIGHT;

**public** **static** Direction get(**double** angle) {

**if** (inRange(angle, 45, 135)) {

**return** Direction.UP;

} **else** **if** (inRange(angle, 0, 45) || inRange(angle, 315, 360)) {

**return** Direction.RIGHT;

} **else** **if** (inRange(angle, 225, 315)) {

**return** Direction.DOWN;

} **else** {

**return** Direction.LEFT;

}

}

**private** **static** **boolean** inRange(**double** angle, **float** init, **float** end) {

**return** (angle >= init) && (angle < end);

}

}

Теперь дело за малым — создать экземпляр детектора и, как было показано в примере выше, передавать в него полученные MotionEvent:

directionDetector = **new** SwipeDirectionDetector(getContext()) {

@Override

**public** **void** onDirectionDetected(Direction direction) {

**this**.direction = direction;

}

};

...

@Override

**public** **boolean** onTouchEvent(MotionEvent event) {

directionDetector.onTouchEvent(event);

**return** **super**.onTouchEvent(event);

}

5. Перехватывай и делегируй!

Теперь давайте рассмотрим простой кейс — custom view с ViewPager и контейнером для «swipe to dismiss» внутри. Если просто совместить компоненты — жест будет обрабатываться одновременно, что не есть хорошо с точки зрения UX.

Для решения проблемы необходимо переопределить dispatchTouchEvent() и в нем оповещать написанный нами детектор. Возвращать метод должен *true*, так как нам нужно перехватить управление на себя. Как только направление определено, мы можем передавать события нужному виджету и делать это нужно **только посредством dispatchTouchEvent()**.

@Override

**public** **boolean** dispatchTouchEvent(MotionEvent event) {

directionDetector.onTouchEvent(event);

*//passing UP action to widgets and reseting the direction*

**if** (event.getAction() == MotionEvent.ACTION\_UP) {

direction = **null**;

pager.dispatchTouchEvent(event);

swipeDismissListener.onTouch(dismissContainer, event);

}

*//passing initial action to widgets*

**if** (event.getAction() == MotionEvent.ACTION\_DOWN) {

swipeDismissListener.onTouch(dismissContainer, event);

pager.dispatchTouchEvent(event);

}

**if** (direction != **null**) {

**switch** (direction) {

**case** UP:

**case** DOWN:

**return** swipeDismissListener.onTouch(dismissContainer, event);

**case** LEFT:

**case** RIGHT:

**return** pager.dispatchTouchEvent(event);

}

}

**return** **true**;

}

Теперь конфликт жестов решен, а самое главное — чисто и красиво :)

6. Напоследок

Для большего понимания и примеров советую посмотреть [исходный код](https://github.com/stfalcon-studio/FrescoImageViewer)упомянутой библиотеки. Надеюсь, что кому-то эта заметка даст больше понимания системы жестов и сэкономит драгоценное время и нервы :)

P. S. А вот [отличнейшая лекция](https://newcircle.com/s/post/1567/mastering_the_android_touch_syste) на английском языке от Дейва Смита (Dave Smith), с помощью которой мне удалось разобраться в этой непростой головоломке.

*Нужен MVP, разработка под iOS, Android или прототип приложения? Ознакомьтесь с*[*нашим портфолио*](https://stfalcon.com/ru/portfolio)*и*[*сделайте заказ*](https://stfalcon.com/ru/contacts)*уже сегодня!*