|  |  |
| --- | --- |
| **Класс MotionEvent** |  |

|  |
| --- |
|  |

Объект класса **MotionEvent** используется для сообщения о событиях перемещения (мыши, пера, пальца, трекбола). Motion events (события перемещения) могут содержать либо абсолютные, либо относительные перемещения и другие данные, в зависимости от типа устройства.

[**Обзор класса MotionEvent**]

События перемещения описывают перемещения в виде кода действий (action code) и набора значений осей координат. Action code указывает изменение состояния, которое происходит, когда указатель опускается или поднимается. Значения координат осей описывают позицию и другие свойства перемещения.

Например, когда пользователь первый раз касается экрана, система передает событие касания (touch event) к соответствующему элементу визуального интерфейса View с кодом действия ACTION\_DOWN(это и есть action code), с набором координат X и Y точки касания, и с информацией о давлении, размере и ориентации области касания.

Некоторые устройства могут сообщать о сразу нескольких путях перемещения, происходящих одновременно. Экраны с поддержкой мультитача (Multi-touch screens, экраны с определением множественных касаний) выдает трассу перемещения для каждого пальца. Отдельные пальцы или другие объекты, которые генерируют трассы перемещения, называются указателями (pointers). События перемещения содержат информацию о всех указателях, которые активны в настоящий момент, даже если некоторые из них не переместились с момента последнего полученного события.

Количество указателей всегда изменяется только на 1, когда отдельные указатели опускаются и поднимаются, за исключением случая отмены жеста.

Каждый указатель имеет уникальный идентификатор, который назначается при первом опускании (показывается кодом ACTION\_DOWN или ACTION\_POINTER\_DOWN). Идентификатор указателя остается валидными, пока указатель не поднимется (показывается кодом ACTION\_UP илиACTION\_POINTER\_UP), или когда жест отменен (показывается кодом ACTION\_CANCEL).

Класс MotionEvent предоставляет несколько методов для опроса позиции и других свойств указателей, такие как getX(int), getY(int), getAxisValue(int), getPointerId(int), getToolType(int) и многие другие. Большинство этих методов принимают индекс указателя в качестве параметра, а не идентификатор указателя. Индекс каждого указателя находится в диапазоне от 0 доgetPointerCount()-1.

Заранее не определен порядок, в котором указатели появляются в событии перемещения. Таким образом индекс одного и того же указателя может поменяться от одного события к другому, но идентификатор указателя гарантированно останется неизменным, пока указатель остается активным. Используйте метод getPointerId(int), чтобы получить идентификатор указателя, и отслеживать его через последующие события перемещения в жесте. Тогда для последовательных событий перемещения используйте метод findPointerIndex(int), чтобы получить индекс указателя на предоставленный идентификатор указателя в этом событии перемещения.

Мышь и кнопки стилуса могут быть получены с использованием getButtonState(). Хорошая идея проверить состояние кнопки при обработки ACTION\_DOWN как части события касания. Приложение может выполнять разные действия при появлении события касания, если событие касания сопровождается вторым нажатием на кнопку, как например вызов контекстного меню.

[**Пакетная обработка (Batching)**]

Для эффективности события перемещения с ACTION\_MOVE могут группировать вместе несколько выборок перемещения в один объект (**batch**). Самые актуальные текущие координаты доступны через вызов getX(int) и getY(int). Предыдущие координаты в batch доступны через getHistoricalX(int, int) и getHistoricalY(int, int). Координаты будут "историческими" (historical) только потому, что они старше текущих координат в batch; однако они все равно более актуальные, чем другие координаты, которые были получены в других событиях перемещения. Чтобы обработать все координаты в batch с привязкой по времени, сначала рассмотрите historical-координаты и затем текущие координаты.

Пример, получение всех выборок для всех указателей в порядке по времени:

void printSamples(MotionEvent ev)

{

**final** int historySize = ev.getHistorySize();

**final** int pointerCount = ev.getPointerCount();

**for** (int h = 0; h < historySize; h++)

{

System.out.printf("At time %d:", ev.getHistoricalEventTime(h));

**for** (int p = 0; p < pointerCount; p++)

{

System.out.printf(" pointer %d: (%f,%f)",

ev.getPointerId(p), ev.getHistoricalX(p, h), ev.getHistoricalY(p, h));

}

}

System.out.printf("At time %d:", ev.getEventTime());

**for** (int p = 0; p < pointerCount; p++)

{

System.out.printf(" pointer %d: (%f,%f)",

ev.getPointerId(p), ev.getX(p), ev.getY(p));

}

}

[**Типы устройств**]

Интерпретация содержимого MotionEvent значительно варьируется в зависимости от источника класса устройства.

Для устройств указания с исходным классом SOURCE\_CLASS\_POINTER, таких как тачскрины, координаты указателя показывают абсолютные позиции, такие как координаты X/Y представления. Каждый полный жест представлен последовательностью событий перемещения с действиями, которые описывают переходы состояний указателя и перемещения. Жест начинается с события перемещения ACTION\_DOWN, которое предоставляет место первого опускания указателя. Для каждого дополнительного указателя, который опускается вниз или вверх, фреймворк будет генерировать событие перемещения с ACTION\_POINTER\_DOWN или ACTION\_POINTER\_UP соответственно. Перемещения указателя описаны в событиях перемещения с ACTION\_MOVE. Окончание жеста происходит либо когда поднимается последний указатель (по событию перемещения с ACTION\_UP), или когда жест отменен (с событием перемещения ACTION\_CANCEL).

Некоторые устройства указания, такие как мыши, могут поддерживать вертикальную и/или горизонтальную прокрутку (scrolling). Событие прокрутки сообщается как обычное событие перемещений с ACTION\_SCROLL, которое включает в себя смещение прокрутки в по осямAXIS\_VSCROLL и AXIS\_HSCROLL. См. описание getAxisValue(int) для дополнительной информации как получать эти дополнительные оси.

Устройства трекбол с исходным классом SOURCE\_CLASS\_TRACKBALL получают координаты указателя, которые задают относительное перемещение (изменение) по осям X/Y. Жест трекбола состоит из последовательности перемещений, описанных событиями перемещения с ACTION\_MOVE, перемежающимися со случающимися событиями ACTION\_DOWN или ACTION\_UP, когда кнопки трекбола нажимаются или отпускаются.

Устройства джойстика с исходным классом SOURCE\_CLASS\_JOYSTICK, получают координаты указателя с абсолютной позицией по осям джойстика. Значения осей джойстика нормализуются к диапазону от -1.0 до 1.0, где 0.0 соответствует центральному положению. Больше информации по набору доступных осей и диапазону перемещений можно получить через getMotionRange(int). Общие оси джойстика: AXIS\_X, AXIS\_Y, AXIS\_HAT\_X, AXIS\_HAT\_Y, AXIS\_Z и AXIS\_RZ.

Обратитесь к InputDevice для дополнительной информации о различных видах устройств ввода, источниках, предоставляющих координаты указателя.

[**Гарантии согласованности**]

События перемещения всегда передаются в представления (view) как последовательный поток событий. Что из себя представляет последовательный поток - зависит от типа устройства. Для событий касания последовательность включает в себя информацию по опусканию указателей с привязкой ко времени, групповым перемещением, и затем с поднятием по одному или с отменой жеста.

Хотя фреймворк пытается передать непрерывный поток событий перемещения к элементам интерфейса (View), нет полной гарантии о передаче всех событий. Некоторые события могут быть отброшены или модифицированы элементами интерфейса в приложении до того, как будут доставлены, в результате чего поток событий будет неполным. Элементы интерфейса (View) должны быть всегда готовы к обработке ACTION\_CANCEL и должны быть толерантны к аномальным ситуациям, таким как прием нового ACTION\_DOWN без предварительного получения ACTION\_UP для предыдущего жеста.

Полное описание констант и методов класса см. в [1].

[**Пример: определение места касания экрана**]

Для того, чтобы определить место касания экрана в программе Android, переопределите метод onTouchEvent для в классе Activity:

@Override

**public** boolean onTouchEvent(MotionEvent ev)

{

textDebug.append(Float.toString(ev.getX())

+ " "

+ Float.toString(ev.getY())

+"\n");

**return** **true**;

}

[**Пример: определение места касания на виджете ImageView**]

MotionEvent может использоваться для определения мест касания на изображении, чтобы по ним предпринимать различные действия. Простой пример - двухпозиционный выключатель, на левой картинке он в положении "OFF" (выключено), а на правой в положении "ON" (включено).

 

Идея состоит в следующем: при нажатии на верхнюю часть картинки выключатель должен "включаться" (т. е. должна показываться картинка, где выключатель в положении ON), а когда нажатие происходит на нижнюю часть картинки, то выключатель должен "выключаться" (должна показываться картинка, где выключатель в положении OFF). Далее процесс по шагам.

**1**. Создайте 2 картинки для состояний выключателя ON и OFF в виде файлов формата PNG, назовите эти файлы switch\_on.png и switch\_off.png соответственно. Положите их в папку res\drawable\ проекта. Сделайте Clean для проекта, чтобы в файле R.java сгенерировались идентификаторы для картинок.

**2**. Бросьте на форму программы виджет **ImageView**, дайте ему понятный идентификатор, например imageSwitch. В качестве источника картинки для ImageView укажите ресурс switch\_off (это наша картинка для выключенного состояния выключателя). В результате в файле activity\_main.xml появится определение наподобие следующего:

**< ImageView**

android:id="@+id/imageSwitch"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerHorizontal="true"

android:layout\_centerVertical="true"

android:src="@drawable/switch\_off" **/>**

**3**. В обработчике в класс Activity программы добавьте переменную ImageView, и в обработчике onCreate сделайте инициализацию этой переменной, привязав её к добавленному виджету imageSwitch. Добавьте также для этой переменной обработчик события OnTouchListener. Вот полный код для onCreate активности:

**public** **class** **MainActivity** **extends** Activity

{

ImageView imgSwitch;

@Override

**protected** void onCreate(Bundle savedInstanceState)

{

**super**.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

*//Привязка переменной ImageView к виджету imageSwitch.*

imgSwitch = (ImageView)findViewById(R.id.imageSwitch);

*//Установка обработчика для события касания на картинке.*

imgSwitch.setOnTouchListener(**new** View.OnTouchListener()

{

@Override

**public** boolean onTouch(View v, MotionEvent event)

{

int touchY = (int)event.getY();

**if** (touchY < v.getHeight()/2)

{

imgSwitch.setImageResource(R.drawable.switch\_on);

}

**else**

{

imgSwitch.setImageResource(R.drawable.switch\_off);

}

**return** **false**;

}

});

}

...

Немного пояснений к коду. Поскольку обработчик onTouch определен только для картинки imgSwitch, то он будет срабатывать только на касания картинки. Внутри обработчика вызов v.getHeight() даст высоту картинки, а вызов event.getY() даст координату касания по вертикали относительно верхнего края картинки. Т. е. касание до верхнего края картинки даст координату 0, а касание до нижнего края картинки даст координату v.getHeight()-1. Чтобы определить, к какой половине картинки было произведено касание, используется условный оператор if, который сравнивает координату касания с половиной от высоты картинки. Если меньше половины высоты, то касание было в верхней части "выключателя", и будет показана картинка для состояния ON. Если больше половины высоты, то значит касание в нижней части, и будет показана картинка для состояния OFF.

[**Ссылки**]

**1**. MotionEvent site:developer.android.com.  
**2**. [Получение доступа к ресурсам приложения Android](http://microsin.net/programming/android/accessing-resources.html).