По следам бага и немного о событиях MotionEvent в Android

[Разработка под Android](https://habrahabr.ru/hub/android_dev/)\*, [Разработка мобильных приложений](https://habrahabr.ru/hub/mobile_dev/)\*, [Open source](https://habrahabr.ru/hub/open_source/" \o "Вы не подписаны на этот хаб)\*

Думаю, многие из нас писали код вида:

**@Override**

**public** **boolean** **onTouch**(View view, MotionEvent event) {

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

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

*// использование x и y...*

**return** **false**;

}

Но, думаю, не многие задумывались о том, какой путь проходит каждый объект MotionEvent прежде чем попасть в этот метод. В большинстве случае в этом нет необходимости, но все же случаются ситуации, когда незнание особенностей MotionEvent и обработки касаний приводит к печальным результатам.  
  
Год назад я с друзьями разрабатывал приложение, где очень многое упиралось в обработку касаний. Однажды, загрузив новые исходники из репозитория и собрав приложение, я обнаружил, что вертикальная координата касания определяется неверно. Просматривая последние коммиты команды, я наткнулся на интересную строку, где внезапно от y-координаты отнималось 100. То есть, что-то вроде «y -= 100;», причем, это число не было вынесено как константа и вообще было непонятно почему именно 100. На мой очевидный вопрос я получил ответ «Ну, мы опытным путем определили, что в этом месте y-координата всегда на 100 (пикселей) больше, чем должна быть». Здесь, конечно, стоило бы перечитать документацию по обработке касаний и, просмотрев код проекта, найти ошибку, но я решил пойти более интересным путем – проследить по исходникам Android за MotionEvent от его получения до утилизации.  
  
Если я смог кого-то заинтриговать историей в стиле «По следам полосатого бага» — добро пожаловать под кат.

Мораль

Для начала убедимся, что хранить MotionEvent, который пришел к нам с onTouch – плохо. Я использовал небольшое тестовое приложение со следующим кодом:

**package** com.alcsan.test;

*// imports…*

**public** **class** **MainActivity** **extends** **Activity** **implements** **OnTouchListener** {

**private** List<MotionEvent> mEventsHistory = **new** ArrayList<MotionEvent>();

**@Override**

**protected** **void** **onCreate**(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

View parentLayout = findViewById(R.id.parent\_layout);

parentLayout.setOnTouchListener(**this**);

}

**@Override**

**public** **boolean** **onTouch**(View view, MotionEvent event) {

logEventsHistory();

mEventsHistory.add(event);

**return** **false**;

}

**protected** **void** **logEventsHistory**() {

StringBuilder message = **new** StringBuilder();

**for** (MotionEvent event : mEventsHistory) {

message.append(event.getY());

message.append(" ");

}

Log.i("Events", message.toString());

}

}

Запускаем приложение, несколько раз тапаем в одну точку под ActionBar-ом и смотрим в логи. Лично я получил следующую картину: «32.0», «41.0 41.0», «39.0 39.0 39.0», «39.0 39.0 39.0 39.0». То есть, после первого вызова мы сохранили в истории объект с y=32, но уже после следующего нажатия y этого объекта равен 41, а в историю заносится объект с таким же y. На самом деле это все один и тот же объект, который был использован при первом вызове onTouch и повторно использован при втором его вызове. Поэтому мораль проста: не храните MotionEvent, полученный в onTouch! Используйте этот объект только в рамках метода onTouch, а для остальных нужд извлекайте из него координаты и храните их в PointF, например.

Исходники Android – пул MotionEvent

А теперь предлагаю заглянуть в кроличью нору исходников Android и определить почему MotionEvent ведет себя именно таким образом.  
  
Во-первых, уже по поведению тестового приложения понятно, что объекты MotionEvent не создаются при каждом касании, а повторно используются. Сделано это потому, что касаний может быть много за короткий промежуток времени и создание множества объектов ухудшило бы производительность. Как минимум за счет учащения сборки мусора. Представьте, сколько объектов создавалось бы за минуту игры в Fruit Ninja, ведь события – это не только DOWN, UP и CANCEL, но и MOVE.  
  
Логика работы с пулом объектов MotionEvent находится классе MotionEvent —[grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.2\_r1.1/android/view/MotionEvent.java](http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.2_r1.1/android/view/MotionEvent.java). С пулом здесь связаны статические методы и переменные. Максимальное количество одновременно хранимых объектов определяет константа MAX\_RECYCLED (и равна она 10), счетчик хранимых объектов – gRecyclerUsed, для синхронизации и обеспечения работы в асинхронном режиме используется gRecyclerLock. gRecyclerTop – голова списка объектов, оставленных на утилизацию. И еще есть не статическая переменная mNext, а также mRecycledLocation и mRecycled.  
  
Когда системе нужен объект, вызывается статический метод obtain(). Если пул пуст (gRecyclerTop == null), создается и возвращается новый объект. В противном же случае возвращается последний утилизированный объект (gRecyclerTop), а его место занимает предпоследний (gRecyclerTop = gRecyclerTop.mNext).  
  
Для утилизации вызывается recycle() на утилизируемом объекте. Он занимает место «последнего добавленного» (gRecyclerTop), а ссылка на текущий «последний» сохраняется в mNext (mNext = gRecyclerTop). Это все происходит после проверки на переполнение пула.

Исходники Android – обработка MotionEvent

Нырять слишком глубоко не будем и начнем с метода handleMessage(Message msg) —[grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.2\_r1.1/android/view/ViewRoot.java?av=f#1712](http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.2_r1.1/android/view/ViewRoot.java?av=f#1712) – класса ViewRoot. Сюда приходит уже готовый MotionEvent (полученный системой через MotionEvent.obtain()), обернутый в Message. Метод, кстати, служит для обработки не только касаний, но и других событий. Поэтому тело метода – большой switch, в котором нас интересуют строки с 1744 по 1847. Здесь происходит предварительная обработка события, затем mView.dispatchTouchEvent(event), затем же событие добавляется в пул: event.recycle(). Метод dispatchTouchEvent(…) вызывает событие слушателя, если таковой имеется, и пытается делегировать обработку события внутренним View.

Следы бага

И теперь вкратце о том, в чем заключался баг.  
  
Для начала немного о том, что конкретно делали с MotionEvent в том проекте. Получив объект, приложение сохраняло его в переменную, ждало некоторое количество миллисекунд и обрабатывало его. Связано такое поведение было с жестами: грубо говоря, если пользователь коснулся экрана и задержал палец на секунду – показать ему определенный диалог. Приложение получало событие ACTION\_DOWN и, не получив в течение секунды событий ACTION\_UP или ACTION\_CANCEL, реагировало. Причем, реагировало исходя из инициирующего MotionEvent. Таким образом, ссылка на него жила некоторое время, за которое могло произойти несколько других событий касания.  
  
Последовательно происходило следующее:  
1. Пользователь касался экрана.  
2. Система получала новый объект методом MotionEvent.obtain() и наполняла его данными о касании.  
3. Объект события попадал в handleMessage(…), там он предобрабатывался и, несколько методов спустя, попадал в метод onTouch() слушателя.  
4. Метод onTouch() сохранял ссылку на объект. Здесь же запускается таймер.  
5. В методе handleMessage(…) объект помещался в пул — event.recycle(). То есть, система теперь считает этот объект свободным для повторного использования.  
6. Пока таймер тикает, пользователь коснулся экрана еще несколько раз, при этом для обработки этих касаний использовался один и тот же объект.  
7. Таймер завершил отсчет, вызывается некий метод, который обращается по ссылке к объекту MotionEvent, полученному при первом касании. Объект тот же, а вот x и y уже успели поменяться.  
  
В тестовом же примере все тоже было просто:  
1. Первое касание. Запрашивается объект MotionEvent. Поскольку вызов первый – объект создается.  
2. Объект наполняется информацией о касании.  
3. Объект приходит в onTouch() и мы сохраняем ссылку на него в списке-истории.  
4. Объект утилизируется.  
5. Второе касание. Запрашивается объект MotionEvent. Поскольку в пуле уже есть один – он и возвращается.  
6. У полученного из пула объекта меняются координаты.  
7. Объект приходит в onTouch(), мы добавляем его в историю, но это тот же объект, что и уже есть в истории, а координаты первого касания утеряны – их заменили координаты второго касания.

Выводы

Да, проще и правильнее было бы почитать документацию и увидеть там, что хранить объекты MotionEvent таким образом нельзя. Быстрее было бы посмотреть решение проблемы на StackOverflow. Но, пройти путь MotionEvent по исходникам от создания до утилизации было интересно и познавательно.