Android-Tracker 原理

简介

Android-Tracker 是一个用户操作路径跟踪库。根据预先创建好的一系列路径，当用户的操作和设定一样时，通过回调的方式告知订阅者。

通过 Android-Tracker ， 可以实现 指定路径埋点。逻辑埋点，不侵入业务代码

这里 Tracker 库做的是属于：代码埋点，但是唯一区别就是不侵入业务代码，可以在一个类统一的配置埋点代码，好处就是管理方便，业务代码中不包含埋点代码。但是也需要对自己的业务非常熟悉。

Track 取自路径 的概念，想像一下，你的 app 有没有这种的埋点，同时有 activity A、B、C、D，需要分别统计 A->D （A 跳转到 D） ， B->D、C->D 的点，这种情况下，如果用普通实现可能就是在 D 页面通过 intent 取出来源是 A 或 B 或 C 来做埋点。这种两个页面以上组合起来称为一条路径 track. 还有更长的，如 A->B,然后页面 B 的某按钮点击，再进页面 C …… 等等，如果用 intent 传值方式需要把来源一路往下传，非常麻烦，而且让其他人在看代码的时候，往往不懂这个值是干嘛的，代码不清晰，所以 tracker 框架就是解决这类问题的。

如页面跳转表示为

```
Track.from(A.class).to(B.class).subscribe(new OnSubscribe<Intent>()
{
    @Override
    public void call(Intent intent) {
        Log.d(TAG, "A->B " + intent + " t=" + Thread.currentThread());
    }
});
```

当发生 startActivity...(A.this,B.class) 时，执行 call 回调。这样是不是很清晰。详细操作符见 api 文档。

一些术语：

1、操作点：是一个基础操作，如 activity 跳转，view 点击，activity 生命周期等

2、操作符：操作符是操作点在 tracker 框架对应的方法，如 操作点：view.setOnclickliener 对应操作符 viewClick(R.id.button) , startActivity 页面跳转操作 对应操作符为 to()，完整操作符见 api 文档

3、起始事件：一条路径的第一个事件。

4、终点事件：一条路径的最后一个事件 (subscribe 前的操作)

如：

```
Track.from(A.class).to(B.class).to(C.class).viewClick(R.id.button
2).subscribe(xx);
```

起始事件为： `Track.from(A.class).to(B.class)`

终点事件为：`.viewClick(R.id.button2)`

5、路径：一条路径由多个操作点组成，并且操作点是有序的，只有当第一个操作触发时，第二个操作才有可能被触发，以此类推。
6、点亮路径：当路径的第一个点被触发时，称为点亮的路径。会加入到点亮集中。后续的操作会先在点亮集中查找。当路径的点全部点亮时，回调给 订阅者。
7、熄灭路径：把该路径恢复当默认状态，也就是该路径已点亮的点都恢复。
8、取消订阅：不再订阅此路径。

一些规定：

1、当路径的所有操作都点亮时，才会触发回调给订阅者。
2、如果该路径已全部点亮，此时再触发终点事件，回调会再次的执行。如：

```
Track.from(A.class).to(B.class).to(C.class).viewClick(R.id.button2).subscribe(xx);
```

当 A->B、B->C 时，点击 button2，触发回调，再次点击，会再次触发。因为此时的点击依然是有效的。只有当 A->B 这个起始事件再次发生时，才会重新重头点亮该路径

3、上下文相关性：所有事件都是有上下文相关性的。每一个操作都依赖于前一个操作。
其中 to() 会把后面的操作改为 to() 里的上下文，而 activityFinish()、activityOnDestoryed() 会把后面的操作切回到上一个上下文。
to 如：

```
Track.from(A.class).to(B.class).to(C.class).viewClick(R.id.button2).subscribe(xx);
```

表示当 A->B，然后 B->C，然后 C 中的 R.id.button2 被点击，回调触发。并且在 B->C 的操作前，不关心在 B 页面做的任何操作，如点击、B 跳 D，D 返回 B。都不影响，只要最终是 B->C，则会点亮。
activityOnDestoryed 如：

```
Track.from(A.class).to(B.class)
        .viewClick(R.id.button2).activityOnDestroyed()
        .viewClick(R.id.button2).subscribe(xx);
```

表示 A->B，然后 B 页面的 R.id.button2 被点击，然后 B 页面关闭，然后 A 页面的 R.id.button2 被点击，则回调触发。

4、所有回调与操作都是在子线程。回调中的参数 为 事件发生时的参数，如 viewClick 的参数是被点击时的 view 。to 的参数是跳转时的 intent，你可以通过此参数获取一些额外数据。

实现原理：
想要实现无侵入式监听某些操作，必须要用到 AOP，AOP 原理是指在编译时，对所有 jar 进行处理，对需要侵入的代码进行切入点。这里 AOP 用到的 AspectJ，AspectJ 详细见其它。

1、首先先订阅需要的事件。每一个操作符，都转换为一个 Node 对象，添加到一个 List 集合中。然后向 TrackManager 注册，TM 管理所有路径。

2、然后用 AOP 对所有需要侵入的点 插入 track 的处理代码。见 AopAspect.java ，如点击事件，当 setOnClickListener 时，代理 OnClickLiener 对象，当点击事件当生时，切到 tracker 线程处理逻辑。首先对数据进行过滤，过滤无 ID 的 view，或者未向 track 注册的的 viewId，这些事件都直接忽略。所的操作符都类似，除了一些操作符自己比较的逻辑，核心点亮见 findTrack。