问题：

1. 我们有很多 ListView/ViewPager 是给 convertView 直接设置的监听器，所以 Hook onItemClick 方法没有作用，如果 hook onclick 方法，没法区分位置
2. 使用 AOP 针对点击事件，会多一次递归，效率较低；针对页面生命周期事件，AOP 和 BaseActivity 实现一样，但是在 Base 里更容易统计关联性(如统计页面停留时长，设计 resume 和 stop 两个过程，AOP 就很不好实现，但是 Base 可以很轻松实现)，所以目前使用了原声拦截的方式
3. viewId 确定 - - 所属页面/viewTree 路径+viewGroup 第几个子元素
4. Fragment 事件从 activity 中分离
5. 弹窗上的事件采集不到
6. ListView header、footer上的事件还没搞定
7. 把三个单独的，非继承自 BaseActionBarActivity 的 Activity 统一进行管理，采集生命周期事件

优化圈选功能，定接口，补充根据配置文件采集数据及本地存储、上传策略

上传事件格式：

```
2019-03-07 18:04:13.928 11050-11050/com.antourong.itouzi E/马东强:
viewClickEvent:UserBehaviorEvent{
    userHashId='6462129’,
 viewId='com.antourong.itouzi.MainActivity.DecorView[0].LinearLayout[0].FrameLayout[1].FitWindowsLinearLayout[0].ContentFrameLayout[1].RelativeLayout[2].DrawerLayout[0].FrameLayout[0].FrameLayout[0].RelativeLayout[0].LinearLayout[0].MaterialRefreshLayout[1].HomeScrollView[1].LinearLayout[0].FrameLayout[4].LinearLayout[0].LinearLayout[2].AppCompatImageView[0]’,
    view=android.support.v7.widget.AppCompatImageView{b1fe0f2 V.ED..... ........ 56,0-214,158 #7f1005ca app:id/image_discovery_common},
    pageName='MainActivity’,
    timeMillis=1551953053913,
    eventType=1
}

pageLifeCycleEvent:PageLifeCycleEvent{userHashId='6462129', timeMillis=1551953054317, eventType=4, pageName='com.antourong.itouzi.activity.MainActivity'}
```

流程定制：

1. 默认采集所有事件并全部上传
2. 通过开发者功能圈选，可以指定要采集事件的 View，没有圈选的 View 不采集
3. 如果通过圈选选定了 view，则通过接口加载圈选的 View id，只采集圈选的 View 事件

后续需要一个采集页面按钮 id 的功能
做完采集控件 id 的功能之后，就可以根据设置的配置文件，来管理采集哪些数据了

一、索引

目前成型的埋点方案有三种：

1. 代码埋点
2. 无埋点，也叫无痕埋点或全埋点
3. 可视化埋点

二、概述

一、代码埋点

所谓代码埋点，是指在每一处需要进行数据采集的业务逻辑点，专门插入数据采集代码，然后在指定事件发生时，记录事件，并按照一定策略调用数据上传接口上报数据。

优点：

1. 控制精准
2. 在收集个性化数据时也比较灵活

缺点：

1. 新增埋点依赖App发版，影响数据收集时机。
2. 新增埋点依赖开发，增加了开发测试的工作量，影响版本进度。
3. 业务侵入性强，埋点代码和业务代码耦合在一起，增加代码维护难度。
4. 如果埋点错误只能更新版本解决
5. 不能通过接口动态管理埋点位置

二、全埋点

所谓全埋点，是指将 APP 内产生的所有的满足某种条件的行为事件全部记录下来，并按照一定策略调用数据上传接口上报数据。

优点：

1. 自动记录满足埋点条件的所有记录，新增埋点不依赖开发
2. 全埋点一般采用 AOP 或者是方法替换的方式实现，埋点逻辑不侵入业务

缺点：

1. 上报数据量很大，里面有很多数据是没有价值的，也造成服务器压力很大
2. 不方便后续分析

三、可视化埋点

所谓可视化埋点，是指通过可视化工具配置要进行埋点的节点，在 APP 打开时，加载配置信息，根据配置文件采集指定节点产生的事件，并按照一定策略调用数据上传接口上报数据，GrowingIO 即采用的这种方式；可视化埋点是按需配置埋点方案。

优点：

1. 按需埋点
2. 可以通过圈选等操作动态管理埋点，非常灵活

缺点：

1. 开发量很大
2. 版本迭代、布局变更比较难确认唯一 ID

三、埋点技术方案整理

一、代码埋点

按需在指定位置添加埋点逻辑即可，将数据存储、上传等逻辑简单封装，在指定需要埋点处添加少量代码即可

二、全埋点

全埋点要对方法进行Hook，按照是否在运行时这个条件来区分，Android全埋点可以有下面两种方式：

1. 静态Hook：AspectJ实现AOP，编译期修改代码
2. 动态Hook：运行时替换View.OnClickListener等需要埋点的事件回调

这里的Hook其实就是一种AOP实现。

1. 使用 AspectJ 实现静态Hook：编译期织入代码，效率较高，可能的示例如下所示， 这段Aspect的代码定义了：在执行android.view.View.OnClickListener.onClick(android.view.View)方法原有的实现后面，需要插入AopUtil.sendTrackEventToSDK(joinPoint, "onViewOnClick”);
   AspectJ 在 Android 端的使用待进一步调研，在此不进一步体现。

```
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
@Aspect
public class ViewOnClickListenerAspectj {
    /**
      * android.view.View.OnClickListener.onClick(android.view.View)
      * @param joinPoint JoinPoint
      * @throws Throwable Exception
      */
    @After("execution(* android.view.View.OnClickListener.onClick(android.view.View))")
    public void onViewClickAOP(final JoinPoint joinPoint) throws Throwable {
        AopUtil.sendTrackEvent(joinPoint, "onViewOnClick");
    }
}
```

2. 使用动态代理实现动态Hook：运行时动态代理，伴随有大量反射调用，对性能影响较大；要代理所有 View 的 click 事件，需要遍历 View 树，也很消耗性能；
   通过动态代理主要代理的是 `mListenerInfo.mOnClickListener` `对象，本对象存储着 View 的点击事件`，可能的示例如下所示：
   待进一步调研

```
// ===== 首先定义代理监听器=====
// 点击监听器的代理类，具有上报点击行为的功能
class OnClickListenerWrapper implements View.OnClickListener {
    // 原始的点击监听器对象
    private View.OnClickListener onClickListener;
    public OnClickListenerWrapper(View.OnClickListener onClickListener) {
        this.onClickListener = onClickListener;
    }
    @Override
    public void onClick(View view) {
        // 让原来的点击监听器正常工作
        if(onClickListener != null){
            onClickListener.onClick(view);
        }
        // 点击事件上报，可以获取被点击view的一些属性
        track(APP_CLICK_EVENT_NAME, getSomeProperties(view));
    }
}



// ==== 其次反射获取一个View的mListenerInfo.mOnClickListener，替换成代理的点击监听器 ====
// 对一个View的点击监听器进行hook
public void hookView(View view) {
    // 1. 反射调用View的getListenerInfo方法（API>=14），获得mListenerInfo对象
    Class viewClazz = Class.forName("android.view.View");
    Method getListenerInfoMethod = viewClazz.getDeclaredMethod("getListenerInfo");
    if (!getListenerInfoMethod.isAccessible()) {
        getListenerInfoMethod.setAccessible(true);
    }
    Object mListenerInfo = listenerInfoMethod.invoke(view);
    // 2. 然后从mListenerInfo中反射获取mOnClickListener对象
    Class listenerInfoClazz = Class.forName("android.view.View$ListenerInfo");
    Field onClickListenerField = listenerInfoClazz.getDeclaredField("mOnClickListener");
    if (!onClickListenerField.isAccessible()) {
        onClickListenerField.setAccessible(true);
    }
    View.OnClickListener mOnClickListener = (View.OnClickListener)
onClickListenerField.get(mListenerInfo);
    // 3. 创建代理的点击监听器对象
    View.OnClickListener mOnClickListenerWrapper = new OnClickListenerWrapper(mOnClickListener);
    // 4. 把mListenerInfo的mOnClickListener设成新的onClickListenerWrapper
    onClickListenerField.set(mListenerInfo, mOnClickListenerWrapper);
    // 用这个似乎也可以：view.setOnClickListener(mOnClickListenerWrapper);
}

// ==== 最后，遍历当前页面 View 树，给所有的 View 添加代理
```

三、可视化埋点

可视化埋点分为两步：
首先，是通过可视化工具，配置要采集数据的 View
其次，APP 加载配置并解析，找到指定 View，Hook 它的事件并上报

其中 Hook View 部分技术方案和全埋点基本基本相同，下面简单说一下可视化采集配置信息的技术方案

可视化采集配置信息有两种方式：

1. 通过 WebSocket 使用 Web 后台完成圈选：和后台建立 WebSocket 连接之后，对手机屏幕截图，并从 DecorView 遍历当前页面 View 树，记录如下信息：

```
{
    "type": "snapshot_response",
    "payload": {
        "activities": [
            "activity": "com.sensorsdata.analytics.android.demo.MainActivity",
            "scale": 0.3809524,
            "serialized_objects": {
                "rootObject": 88528516,
                "objects": [
                    {
                        "hashCode": 88528516,
                        "id": -1,
                        "index": -1,
                        "sa_id_name": null,
                        "top": 0,
                        "left": 0,
                        "width": 1080,
                        "height": 1920,
                        "scrollX": 0,
                        "scrollY": 0,
                        "visibility": 0,
                        "translationX": 0,
                        "translationY": 0,
                        "classes": [
                            "com.android.internal.policy.DecorView",
                            "android.widget.FrameLayout",
                            "android.view.ViewGroup",
                            "android.view.View"
                        ],
                        "subviews": [
                            57495077,
                            150453242
                        ]
                    }
                ]
            },
            "image_hash": "785C4DC3B01B4AFA56BA0E3A56CE8657",
            "screenshot": “屏幕截图的base64编码"
        ],
        "snapshot_time_millis": 403
    }
}
```

将上面收集到数据发送到连接的WebSocket后台，由后台解析之后，可以把App界面的截图展示在Web页面。然后把可以监测的元素以方框的形式添加在界面上提示用户（web页面实现时，我推测只需要用到这个View的left、top、width、height属性在html上加一个div标签，然后设置一个有颜色的border属性即可）。用户可以在这个Web页面点击需要监测的元素，设置这个元素的事件名称（event_type和event_name），点击保存。保存一个需要监测的元素时，需要保存这个元素在当前Activity的ViewTree的路径path，以及这个View在父控件中的index等保持View唯一性的信息，以及要采集的事件信息等

2. 直接通过圈选工具，在移动端完成圈选(类似 GrowingIO)：可行的一种实现方案是获取圈选点当前坐标，然后遍历 View 树，看圈选点坐标在哪个 View 或 ViewGroup 范围内，然后记录该 View 的唯一性信息以及要采集的事件信息

四、View ID 的确定及版本迭代-参考笑璇整理的方案，思路基本一致

五、总结

埋点方案	优点	缺点	适用场景
代码埋点	1.使用灵活，精确控制发送时机 2.方便设置自定义业务相关的属性	1.埋点成本高，工作量大，必须是技术人员才能完成 2.更新成本高，一旦上线很难修改。只能通过热修复或者重新发版 3.对业务代码的侵入大	对业务上下文理解要求较高的业务数据，如电商购物这类可能经过多次页面跳转，埋点时还需要带上前面页面中的一些信息
全埋点	1.开发、维护成本低 2.可以追溯历史数据 3.对业务代码侵入小 4.可以收集到一些额外信息，例如界面的热力图	1.高额流量和计算成本 2.无法灵活收集属性 3.动态的Hook方式支持的控件有限、事件类型有限，大量事件监测时反射对App运行性能有影响 4.静态的Hook方式需要第三方编译器参与，打包时间增长	上下文相对独立的、通用的数据，如点击热力图，性能监控和日志
可视化埋点	1.可以按需埋点，灵活性好 2.对业务代码侵入小	1.界面的结构发生变化时，圈选的待监测元素可能会失效 2.支持的控件和事件类型有限 3.无法灵活地收集到上下文属性 4.开发量相对较大	上下文相对简单，依靠控件可以获得上下文信息，界面结构比较简单固定，如新闻阅读、游戏分享界面

参考：

1. 无埋点方案解析
2. View ID 生成规则
3. 神策SDK：可直接配置后台 URL，但是上传数据格式及后台怎么存储，需要去源码里面找
4. 如果需要 Gradle 插件，可以参考 沪江开源插件
5. 网易 HubbleData
6. 51CTO 方案
7. AspectJ 入门使用
8. 深入理解 Android AOP