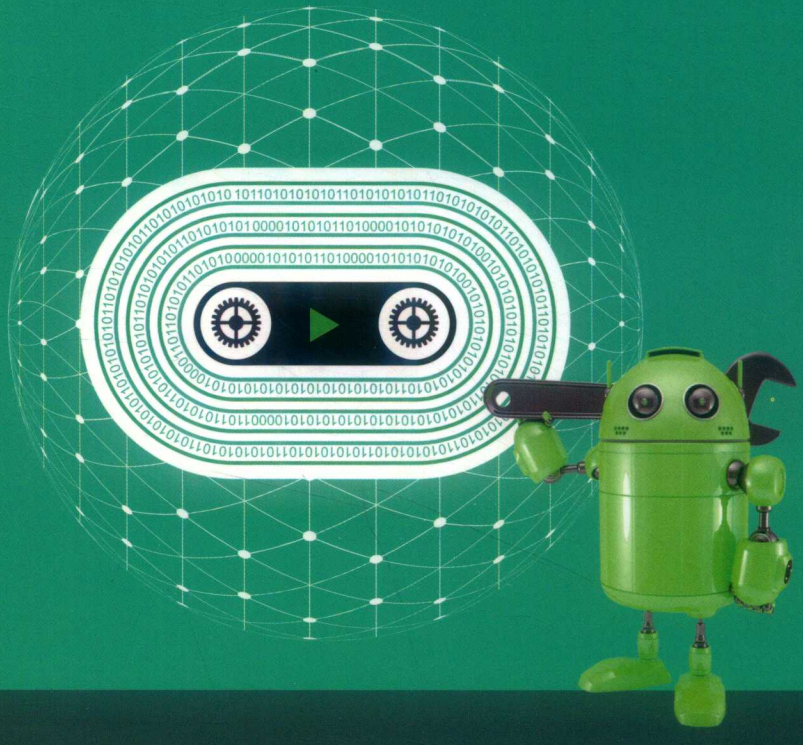
神策数据系列丛书

国内知名大数据公司神策数据出品

作者是神策数据合肥研发中心负责人，有近 10 年 Android 开发经验，开发和维护着知名商用开源 Android & iOS 数据埋点 SDK。



Android AutoTrack Solution

Android 全埋点解决方案

王灼洲 著

机械工业出版社
China Machine Press

神策数据系列丛书

Android AutoTrack Solution

Android 全埋点解决方案

王灼洲 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Android 全埋点解决方案 / 王灼洲著 . —北京：机械工业出版社，2019.3

ISBN 978-7-111-62149-2

I. A… II. 王… III. 移动终端 - 应用程序 - 程序设计 IV. TN929.53

中国版本图书馆 CIP 数据核字（2019）第 038462 号

Android 全埋点解决方案

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：张锡鹏

责任校对：李秋荣

印　　刷：北京诚信伟业印刷有限公司

版　　次：2019 年 3 月第 1 版第 1 次印刷

开　　本：186mm×240mm 1/16

印　　张：20.5

书　　号：ISBN 978-7-111-62149-2

定　　价：89.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88379426 88361066

投稿热线：（010）88379604

购书热线：（010）68326294

读者信箱：hzit@hzbook.com

版权所有 · 侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

华章科技
HZBOOKS | Science & Technology

Forward 推荐序

到目前为止，中国的信息化建设大致经历了两个阶段。2015 年之前，IT 系统的引入主要是为了提升业务运营的效率，形成一套人与 IT 组件构成的业务系统，在纯线上产品中，只有 IT 组件构成的业务系统。在 IT 化的过程中，产生了数据这一副产品，通过数据可以进行一些基础的统计和分析工作。2015 年之后，大数据的概念深入人心，大数据的场景逐步落地，之前的数据生成思路需要进行革新，不能只把数据当成副产品来看待，而是要考虑面向数据流的思路，IT 系统只是数据生成的载体。

这就要求我们在 IT 系统建设时，不能只是为了完成业务功能，还要考虑如何进行有效的数据采集，对工程师的技能要求发生了变化，不仅要会写代码实现功能，还要建立数据思维。

我 2007 年加入百度，2015 年离开，这八年的时间我主要做了一件事情，就是从零构建百度的用户行为数据平台，这其中走了不少弯路，也实现了不少价值。最深刻的一点体会就是：数据这件事情要做好，最重要的是数据源。只要数据源头解决好了，后面的分析处理都比较好办。那怎么才叫把数据源解决好呢？我也总结了四个字：大、全、细、时。大是指宏观上，当然也有物理层面的含义；全就是指要把多种数据源都采集下来，是全量而非抽样；细强调多维度，维度越多，越能精细化分析；时就是时效性，数据采集和查询分析都需要尽可能地实时。

为了实现对数据的采集，可以有三种方式：代码埋点、工具导入和全埋点。这三种方式都是手段，并且各有优缺点，选择时需要完全基于实际的业务需求和现状来设计，而不能一味地追求某一种方式，如果把全埋点当成必杀技，那就大错特错了。

灼洲作为神策数据的 iOS 和 Android SDK 开发负责人，这两年多来对相关的技术进行了深入的研究和大量实践。特别是得益于 Android 系统的开放性，使数据的自动收集更为容易。由于自动收集的本质是对所有操作进行拦截，相比于代码埋点只是采集的一部分必要操作，显然利用自动收集的方式收集的操作类型更全面，因此我们将它命名为全埋点，而不是无埋点。

此为试读，需要完整PDF请访问：www.ertongbook.com

当然，虽然这种方式是自动化的，但有一些精细化的维度，以及后端的数据，无法用这种方式来实现。但如果想要及时地看到一些产品的宏观指标，又不想要工程师做太多的配合，这是一种很好的方式。

神策数据志在推动国内企业数据化的建设进程。因此，我们将探索和实践的成果全部贡献出来，供各位开发者学习，期待更多的人能够认识到数据的重要性，以及学会数据采集的具体方法。

桑文锋

神策数据创始人 &CEO

Preface 前　言

为什么要写这本书?

转眼间，我从事 Android 研发工作已经有 9 个年头，作为国内第一批 Android 研发工作者，我见证了 Android 的发展历程，也开发和维护着国内第一个商用的开源 Android & iOS 数据埋点 SDK。

我目前就职于神策数据，担任神策数据合肥研发中心负责人。神策数据是一家以重构中国互联网数据根基为使命的公司，十分重视基础数据的采集与建模。随着大数据行业的快速发展，数据采集也变得越来越重要，数据基础夯实与否，取决于数据的采集方式。埋点方式多种多样，按照埋点位置不同，可以分为前端（客户端）埋点与后端（服务器端）埋点。其中全埋点（无埋点）是目前较为流行的前端埋点方式之一。

在服务数百家客户的过程中，我逐渐萌生出写此书的想法，原因有三：

第一，国内企业对全埋点技术需求迫切，但是图书市场仍处空白。

全埋点技术炙手可热，全埋点采用“全部采集，按需选取”的形式，对页面中所有交互元素的用户行为进行采集，通过界面配置来决定哪些数据需要进行分析，也被誉为“最全、最便捷、界面友好、技术门槛低”的数据采集方式。

第二，市面上存在对全埋点概念过度包装的情况，希望本书能够揭开全埋点的神秘面纱。

数据埋点技术在互联网（尤其是移动端）上使用非常普遍，一些数据分析服务厂商将全埋点概念经过包装后，作为核心技术来卖，给人神秘无比的感觉。

第三，给企业带来价值，推动开发者参与大数据行业的生态建设。

神策数据的采集技术一直在不断革新，神策 SDK 组件统称为 OpenSasdk，包括 C SDK、C++ SDK、CSharp SDK、Java SDK、Python SDK、PHP SDK、Ruby SDK、Golang SDK、Node SDK、APICloud SDK、Android SDK、iOS SDK 等，神策数据愿意将一些成熟的技术与国内外开发者交流与分享，并已于 2019 年 1 月正式成立供 IT 开发者的分享、使用与交流技术的开源社区——Sensors Data 开源社区，一方面能够更好地服务客户，推动企业的数字化转型；一方面借此造福同行，推动开发者参与数据行业生态建设。

我希望通过此书全面公开 Android 全埋点技术，从 0 到 1 进行详细介绍，尤其是控件

点击事件全埋点采集的 8 种方法，并都提供了完整的项目源码。

读者对象

本书适用于初级、中级、高级水平的 Android 开发工程师、技术经理、技术总监等。

如何阅读这本书

本书系统讲解了 Android 全埋点的解决方案，特别是控件点击事件的全埋点采集，总结并归纳了如下 8 种解决方案，并且都提供了完整的项目源码。

$AppStart、$AppEnd 全埋点方案

- $AppClick 全埋点方案 1：代理 View.OnClickListener
- $AppClick 全埋点方案 2：代理 Window.Callback
- $AppClick 全埋点方案 3：代理 View.AccessibilityDelegate
- $AppClick 全埋点方案 4：透明层
- $AppClick 全埋点方案 5：AspectJ
- $AppClick 全埋点方案 6：ASM
- $AppClick 全埋点方案 7：Javassist
- $AppClick 全埋点方案 8：AST

勘误和支持

由于作者的水平有限，编写时间仓促，以及技术不断地更新和迭代，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。为此，特意创建了一个网站：http://book.blendercn.org，读者可以将书中的错误发布在 Bug 勘误表页面中。同时，如果你遇到任何问题，也可以访问 Q&A 页面，我将尽量在线上为读者提供满意的解答。书中的全部源文件可以从上面这个网站下载，我会将相应的功能更新及时发布出来。如果你有更多的宝贵意见，也欢迎发送邮件至邮箱 congcong009@gmail.com，期待能够得到你们的真挚反馈。

致谢

感谢神策数据创始人团队桑文锋、曹犟、付力力、刘耀洲在工作中的指导和帮助。

感谢机械工业出版社华章公司的编辑杨福川老师，在这半年多的时间中始终支持我的写作，你的鼓励和帮助引导我能顺利完成全部书稿。

谨以此书献给大数据行业的关注者和建设者！

Contents 目　录

第 1 章 Chapter 1

全埋点概述

全埋点，也叫无埋点、无码埋点、无痕埋点、自动埋点。全埋点是指无须 Android 应用程序开发工程师写代码或者只写少量的代码，就能预先自动收集用户的所有行为数据，然后就可以根据实际的业务分析需求从中筛选出所需行为数据并进行分析。

全埋点采集的事件目前主要包括以下四种（事件名称前面的 $ 符号，是指该事件是预置事件，与之对应的是自定义事件）。

❑ $AppStart 事件

是指应用程序启动，同时包括冷启动和热启动场景。热启动也就是指应用程序从后台恢复的情况。

❑ $AppEnd 事件

是指应用程序退出，包括应用程序的正常退出、按 Home 键进入后台、应用程序被强杀、应用程序崩溃等场景。

❑ $AppViewScreen 事件

是指应用程序页面浏览，对于 Android 应用程序来说，就是指切换 Activity 或 Fragment。

❑ $AppClick 事件

是指应用程序控件点击，也即 View 被点击，比如点击 Button、ListView 等。

在采集的这四种事件当中，最重要并且采集难度最大的是 $AppClick 事件。所以，全埋点的解决方案基本上也都是围绕着如何采集 $AppClick 事件来进行的。

对于 $AppClick 事件的全埋点整体解决思路，归根结底，就是要自动找到那个被点击的控件处理逻辑（后文统称原处理逻辑），然后再利用一定的技术原理，对原处理逻辑进行“拦截”，或者在原处理逻辑的执行前面或执行者后面“插入”相应的埋点代码逻辑，从而达到自动埋点的效果。

此为试读，需要完整PDF请访问：www.ertongbook.com

至于如何做到自动“拦截”控件的原处理逻辑，一般都是参考 Android 系统的事件处理机制来进行的。关于 Android 系统的事件处理机制，本书由于篇幅有限，不再详述。

至于如何做到自动“插入”埋点代码逻辑，基本上都是参考编译器对 Java 代码的整体处理流程来进行的，即：

```
JavaCode --> .java --> .class --> .dex
```

选择在不同的处理阶段“插入”埋点代码，所采用的技术或者原理也不尽相同，所以全埋点的解决方案也是多种多样的。

面对这么多的全埋点方案，我们究竟该如何做选择呢？

在选择全埋点的解决方案时，我们需要从效率、兼容性、扩展性等方面进行综合考虑。

❑ **效率**

全埋点的基本原理，如上所述，其实就是利用某些技术对某些方法（控件被点击时的处理逻辑）进行拦截（或者叫代理）或者“插入”相关埋点代码。比如按钮 Button，如果要给它设置点击处理逻辑，需要设置 android.view.View.OnClickListener，并重写它的 onClick(android.view.View) 方法。如果要实现 $AppClick 事件的全埋点，我们就可以“拦截”onClick(android.view.View) 方法，或者在 onClick(android.view.View) 方法的前面或者后面“插入”相应的埋点逻辑代码。按照“在什么时候去代理或者插入代码”这个条件来区分的话，$AppClick 事件的全埋点技术可以大致分为如下两种方式。

❑ **静态代理**

所谓静态代理，就是指通过 Gradle Plugin 在应用程序编译期间“插入”代码或者修改代码（.class 文件）。比如 AspectJ、ASM、Javassist、AST 等方案均属于这种方式。这几种方案，我们在后面会一一进行介绍。

这几种方式处理的时机可以参考图 1-1。

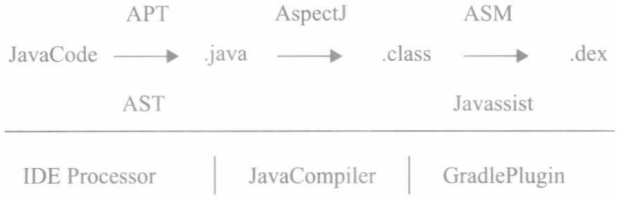


图 1-1 静态代理处理时机

❑ **动态代理**

所谓动态代理，就是指在代码运行的时候（Runtime）去进行代理。比如我们比较常见的代理 View.OnClickListener、Window.Callback、View.AccessibilityDelegate 等方案均属于这种方式。这几种方案，我们也会在后面一一进行介绍。

不同的方案，其处理能力和运行效率各不相同，同时对应用程序的侵入程度以及对应

用程序的整体性能的影响也各不相同。从总体上来说，静态代理明显优于动态代理，这是因为静态代理的“动作”是在应用程序的编译阶段处理的，不会对应用程序的整体性能有太大的影响，而动态代理的“动作”是在应用程序运行阶段发生的（也即 Runtime），所以会对应用程序的整体性能有一定的影响。

❑ 兼容性

随着 Android 生态系统的快速发展，不管是 Android 系统本身，还是与 Android 应用程序开发相关的组件和技术，都在飞速发展和快速迭代，从而也给我们研发全埋点方案带来一定的难度。比如不同的 Android 应用程序可以有不同的开发语言（Java、Kotlin）、不同的 Java 版本（Java7、Java8）、不同的开发 IDE（eclipse、Android Studio），更有不同的开发方式（原生开发、H5、混合开发），使用不同的第三方开发框架（React Native、APICloud、Weex）、不同的 Gradle 版本，以及 Lambda、D8、Instant Run、DataBinding、Fragment 等新技术的出现，都会给全埋点带来很多兼容性方面的问题。

❑ 扩展性

随着业务的快速发展和对数据分析需求的不断提高，对使用全埋点进行数据采集，也提出了更高的要求。一方面要求可以全部自动采集（采集的范围），同时又要求能有更精细化的采集控制粒度（采集可以自定义）。比如，如何给某个控件添加自定义属性？如果不想采集某个控件的点击事件应该如何控制？如果不想采集某种控件类型（ImageView）的点击事件又该如何处理？如果某个页面（Activity）上所有控件的点击事件都不想采集又该如何处理等。

任何一种全埋点的技术方案，都有优点和缺点，没有一种普适的完美解决方案。我们只需要针对不同的应用场景，选择最合适的数据采集方案即可。能满足实际数据采集需求的方案，才是最优的方案。

1.1　Android View 类型

在 Android 系统中，控件（View）的类型非常丰富。分类方式也是多种多样的。我们根据控件设置的监听器（listener）的不同，可以大致将控件分为如下几类。

❑ Button、CheckedTextView、TextView、ImageButton、ImageView 等

为这些控件设置的 listener 均是 android.view.View.OnClickListener。

下面以 Button 为例：

```
Button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //do something
    }
});
```

❑ SeekBar

SeekBar 设置的 listener 是 android.widget.SeekBar.OnSeekBarChangeListener，如：

```
SeekBar seekBar = findViewById(R.id.seekBar);
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
        // do something
        }

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
            // do something
        }

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
            // do something
        }
});
```

❑ TabHost

TabHost 设置的 listener 是 android.widget.TabHost.OnTabChangeListener，如：

```
TabHost tabHost = findViewById(R.id.tabhost);
tabHost.setOnTabChangedListener(new TabHost.OnTabChangeListener() {
    @Override
    public void onTabChanged(String tabName) {
        //do something
    }
});
```

❑ RatingBar

RatingBar 设置的 listerner 是 android.widget.RatingBar.OnRatingBarChangeListener，如：

```
RatingBar ratingBar = findViewById(R.id.ratingBar);
ratingBar.setOnRatingBarChangeListener(newRatingBar.OnRatingBarChangeListener() {
    @Override
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser) {
        //do something
    }
});
```

❑ CheckBox、SwitchCompat、RadioButton、ToggleButton、RadioGroup 等

这些 View 属于同一种类型，它们都是属于带有“状态”的按钮，它们设置的 listener 均是 CompoundButton.OnCheckedChangeListener。

下面以 CheckBox 为例：

```
CheckBox checkBox = findViewById(R.id.checkbox);
```

```
checkBox.setOnCheckedChangeListener(newCompoundButton.OnCheckedChangeListener(){
    @Override
    public void onCheckedChanged(CompoundButton compoundButton, boolean isChecked) {
        //do something
    }
});
```

❑ Spinner

Spinner 设置的 listener 是 android.widget.AdapterView.OnItemSelectedListener，如：

```
Spinner spinner = findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        //do something
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
    }
});
```

❑ MenuItem

主要是通过重写 Activity 的相关方法（onOptionsItemSelected、onContextItemSelected）来设置 listener，如：

```
//选项菜单
@Override
public boolean onOptionsItemSelected(android.view.MenuItem) {
    //do something
}

//上下文菜单
@Override
public boolean onContextItemSelected(android.view.MenuItem) {
    //do something
}
```

❑ ListView、GridView

ListView 和 GridView 都是 AdapterView 的子类，显示的内容都是一个“集合”。它们设置的 listener 均是 android.widget.AdapterView.OnItemClickListener，如：

```
ListView listView = findViewById(R.id.listView);
listView.setOnItemClickListener(new AdapterView.OnItemClickListener(){
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        //do something
    }
});
```

❑ ExpandableListView

ExpandableListView 也是 AdapterView 的子类，同时也是 ListView 的子类。它的点击分为 ChildClick 和 GroupClick 两种情况，所以，它设置的 listener 也是分为两种情况，即：android.widget.ExpandableListView.OnChildClickListener 和 android.widget.ExpandableListView.OnGroupClickListener，如：

```
ExpandableListView listview = findViewById(R.id.expandablelistview);
//ChildClick
listview.setOnChildClickListener(new ExpandableListView.OnChildClickListener() {
    @Override
    public boolean onChildClick(ExpandableListView expandableListView,
                        View view, int groupPosition, int childPosition, long id) {
        //do something
        return true;
    }
});

//GroupClick
listview.setOnGroupClickListener(new ExpandableListView.OnGroupClickListener() {
    @Override
    public boolean onGroupClick(ExpandableListView expandableListView,
                    View view, int childPosition, long id) {
        //do something
        return true;
    }
});
```

❑ Dialog

Dialog 设置的 listener 分为两种情况。对于常见的普通 Dialog，设置的 listener 是 android.content.DialogInterface.OnClickListener，如：

```
AlertDialog.Builder builder = new AlertDialog.Builder(context);
builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        //do something
    }
});
```

还有一种是显示列表的 Dialog，它设置的 listener 是 android.content.DialogInterface.OnMultiChoiceClickListener，如：

```
AlertDialog.Builder builder = new AlertDialog.Builder(context);
DialogInterface.OnMultiChoiceClickListener mutiListener =
                            new DialogInterface.OnMultiChoiceClickListener() {

    @Override
    public void onClick(DialogInterface dialogInterface, int which, boolean isChecked) {
```

```
            //do something
        }
    };
```

1.2　View 绑定 listener 方式

随着 Android 相关技术的不断更新迭代，给 View 绑定 listener 的方式也是多种多样的。下面以 Button 为例来介绍日常开发中比较常见的几种绑定 listener 的方式。

❑ 通过代码来设置 listener

```
Button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //do something
    }
});
```

这种方式是目前开发中最常用的方式，也是我们全埋点方案需要重点解决和重点支持的方式。

❑ 通过 android:onClick 属性绑定 listener

先在布局文件中声明 Button 的 android:onClick 属性，如：

```
<android.support.v7.widget.AppCompatButton
    android:id="@+id/xmlOnClick"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="xmlOnClick"
    android:text="android:onClick 绑定OnClickListener"/>
```

我们设置 android:onClick 的属性值为“xmlOnClick”，此时的“xmlOnClick”代表点击处理逻辑对应的方法名。然后在对应的 Activity 文件中声明 android:onClick 属性指定的方法 xmlOnClick：

```
public void xmlOnClick(View view) {
    //do something
}
```

注意：该方法必须有且仅有一个 View 类型的参数。

这种方式在一些新的项目中不是很常见，在一些比较老的 Android 项目中可能会有这样大量的使用方式。

❑ 通过注解绑定 listener

目前有很多第三方的库都提供了类似的功能，下面以 ButterKnife 为例：

```
@OnClick({R2.id.butterknife})
```

此为试读，需要完整PDF请访问：www.ertongbook.com