

# 客户端监控体系建设

美团 冯天锡





冯天锡

2016年加入美团

先后负责日志、信令、脚本引擎等相关基础设施的开发

# 目录

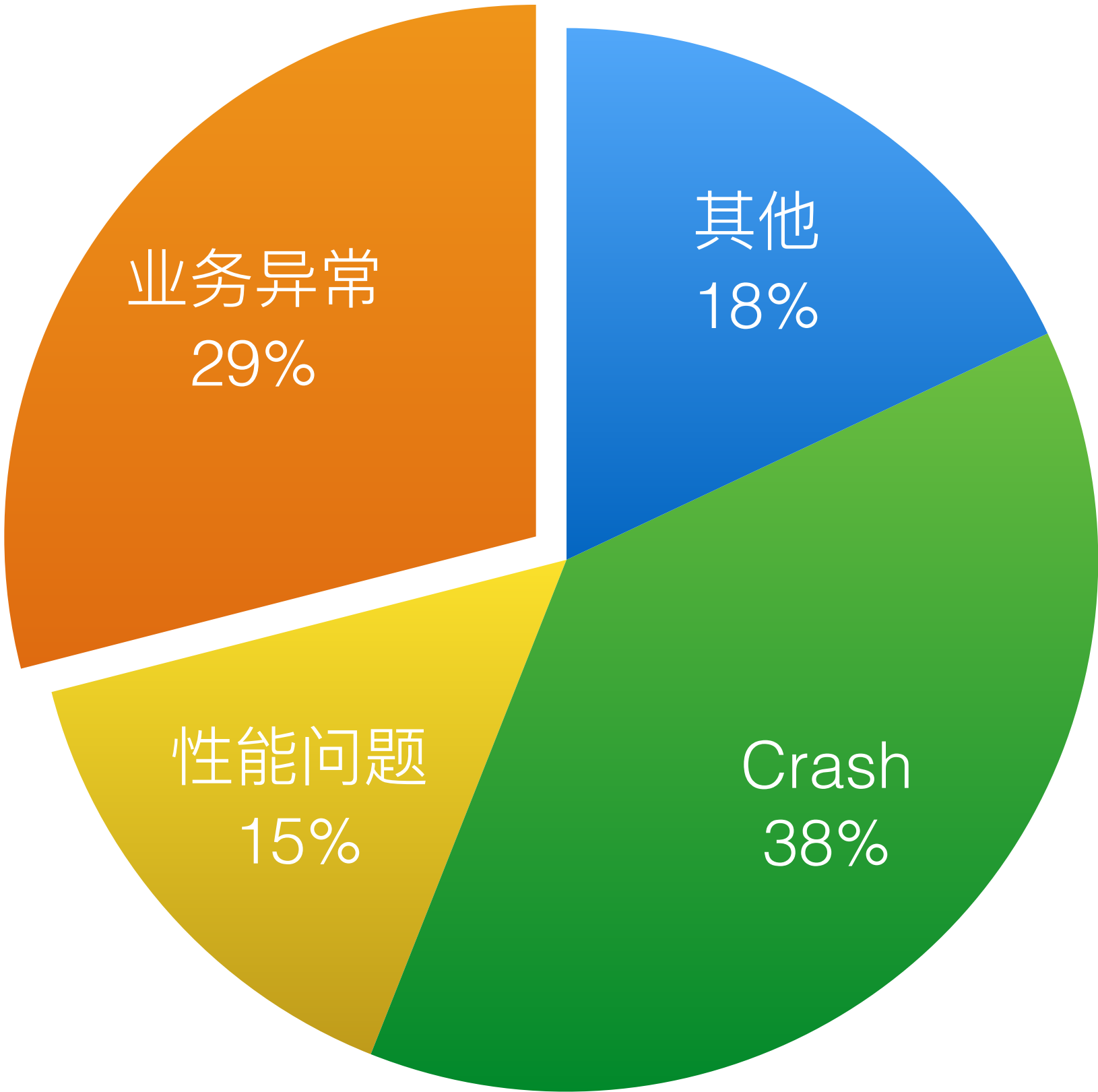
01 背景

02 解决方案

03 总结

综述

线上问题



综述

# 业务异常

阻断用户正常操作流程，进而影响用户无法完成业务流程的问题

偶发性

难定位

痛点  
来源被动

客服反馈



QA/RD自测



领导



问题缺乏有效的回流路径

痛点

辅助信息匮乏

Crash监控



性能监控

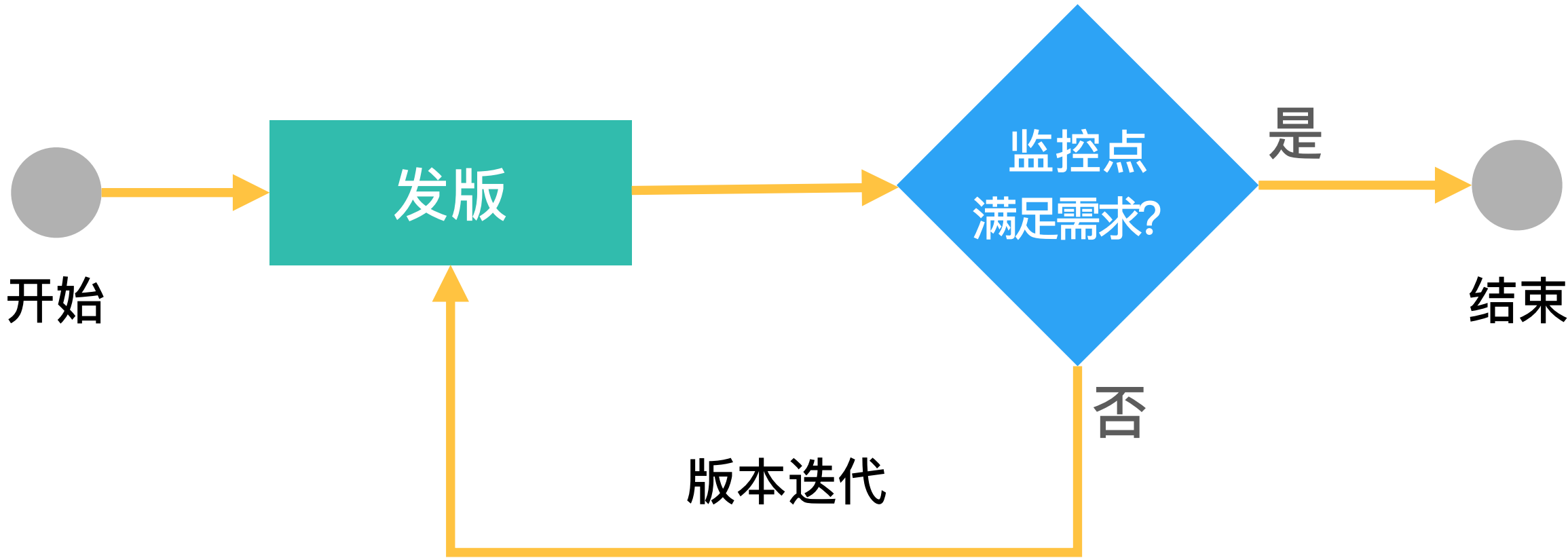


业务异常监控



问题缺乏关键的辅助信息

痛点  
灵活性缺失



问题处理依赖版本周期



# 目录

01 背景

02 解决方案

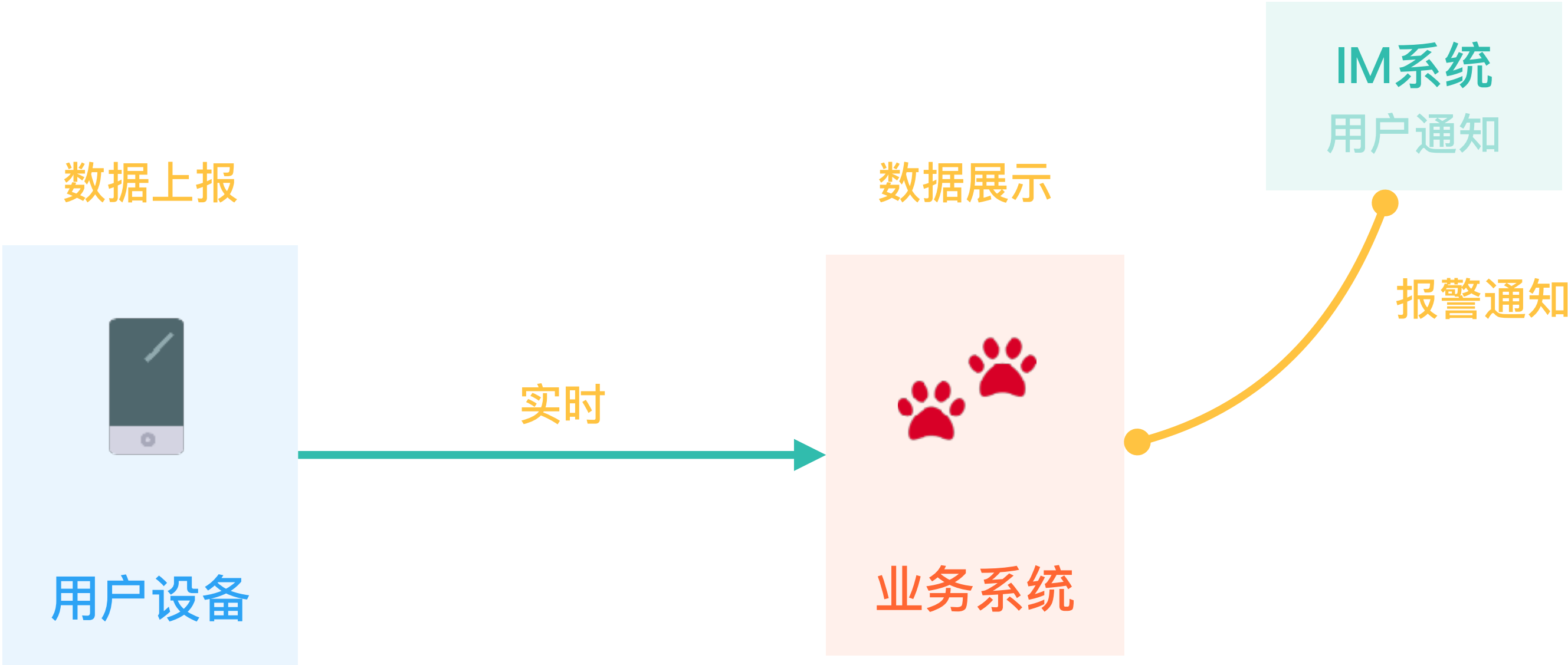
03 总结

解决方案  
归纳



解决方案

主动



解决方案  
主动

# 构建埋点工具集合



## 自动埋点

自动无干扰记录所有方法执行路径



## 半自动埋点

通过注解解析和下发监控配置的方式，记录各种自定义的监控行为

主动  
桩

利用字节码工具在不修改源码的情况下给程序动态统一添加功能

插桩前



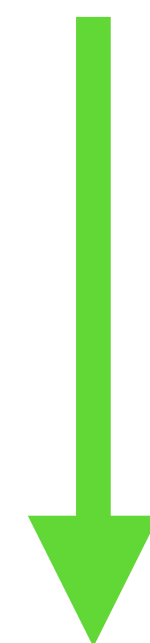
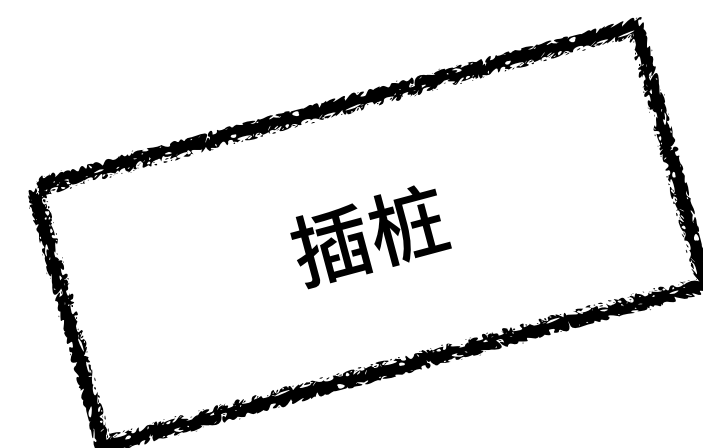
插桩后



主动

## 自动埋点

```
public String concat(String one) {  
    if (Sniffer.isSupport(.....)) {  
        // do nothing  
    }  
    return "hello" + one;  
}
```



### 构建时自动插桩

- 全面覆盖
- 不干扰原有代码

### 运行时异步DB存储

- 存储本地，减少流量消耗
- 自定义空间大小，循环使用

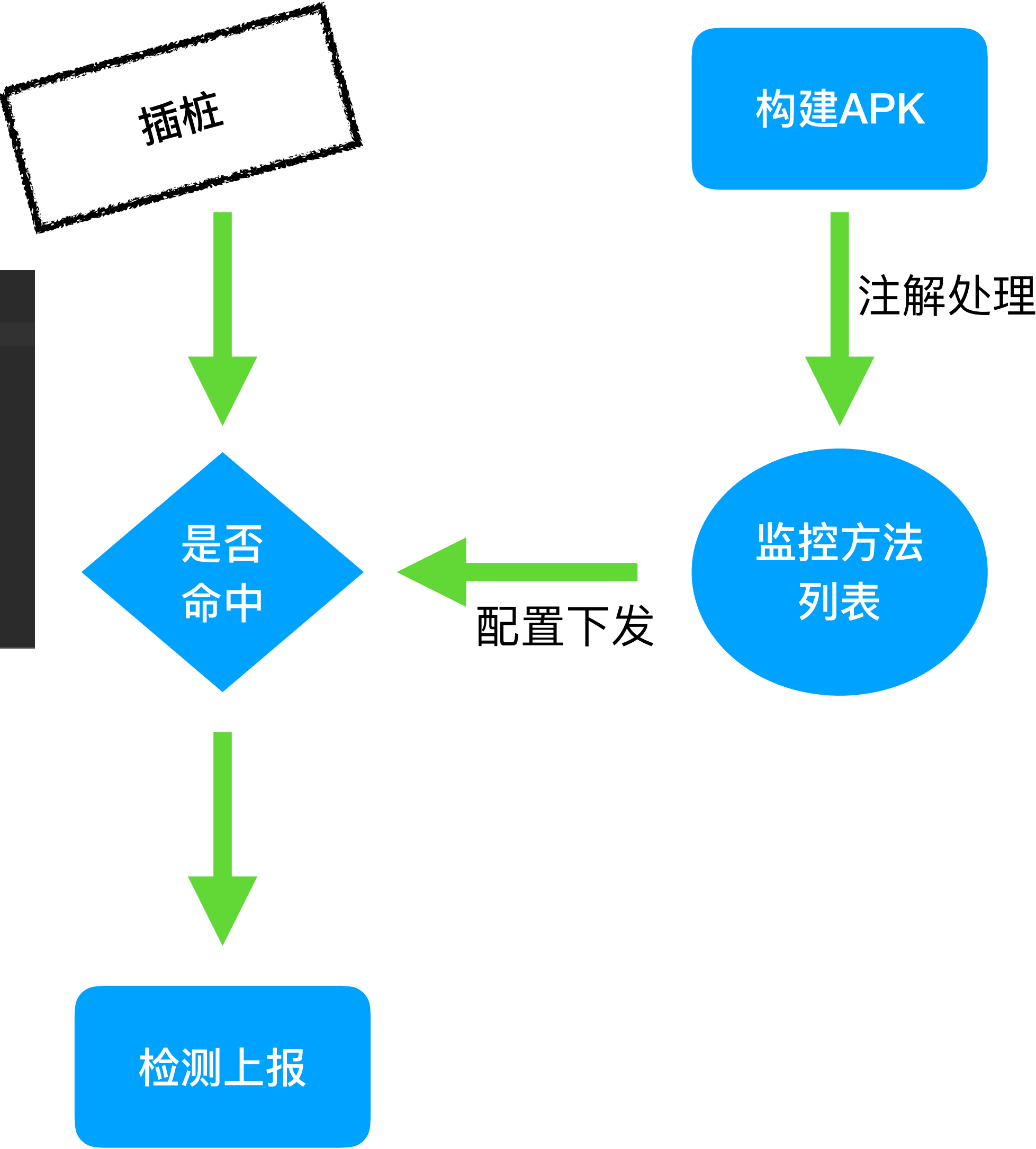
主动  
半自动埋点

```
@Override
@SnifferView(module = "subtle_2", enabled = FALSE, id = R.id.button5, scope = PAGE, interval = 3000)
@SnifferHttp(module = "subtle_2", host = "mapping.sankuai.com", path = {"/index/class", "/index/method"})
protected void onStart() {
    super.onStart();
    fakeFetch(1, (short) 1, new boolean[3]);
}

@SnifferStart(key = "fake network", module = "subtle_3", scope = PAGE, timeout = 510)
private void fakeFetch(int a, short b, boolean[] bs) {
    new Handler().postDelayed(() -> { fakeResponse(); }, 500);
}

@SnifferEnd(key = "fake network")
private void fakeResponse() { Log.i("Subtle", "in fakeResponse"); }
```

```
public String concat(String one) {
    if (Sniffer.isMonitor(.....)) {
        // do nothing
    }
    return "hello" + one;
}
```



解决方案  
场景复现





解决方案

# 线上行为分析

TOP1

## 路径信息

需要知道用户在出现问题时的操作路径。我们将其分解为两类

- 1.用户操作路径
- 2.方法执行路径

TOP2

## 网络信息

希望能记录所有的网络请求的请求体，响应体和头部信息。

TOP3

## 其他信息

程序运行时代码执行逻辑有时候跟运行当时的状态相关

- 1.用户权限列表
- 2.方法快照
- 3.持久层数据
- 4.其他信息

解决方案

# 线上行为分析



路径信息

# 用户操作路径

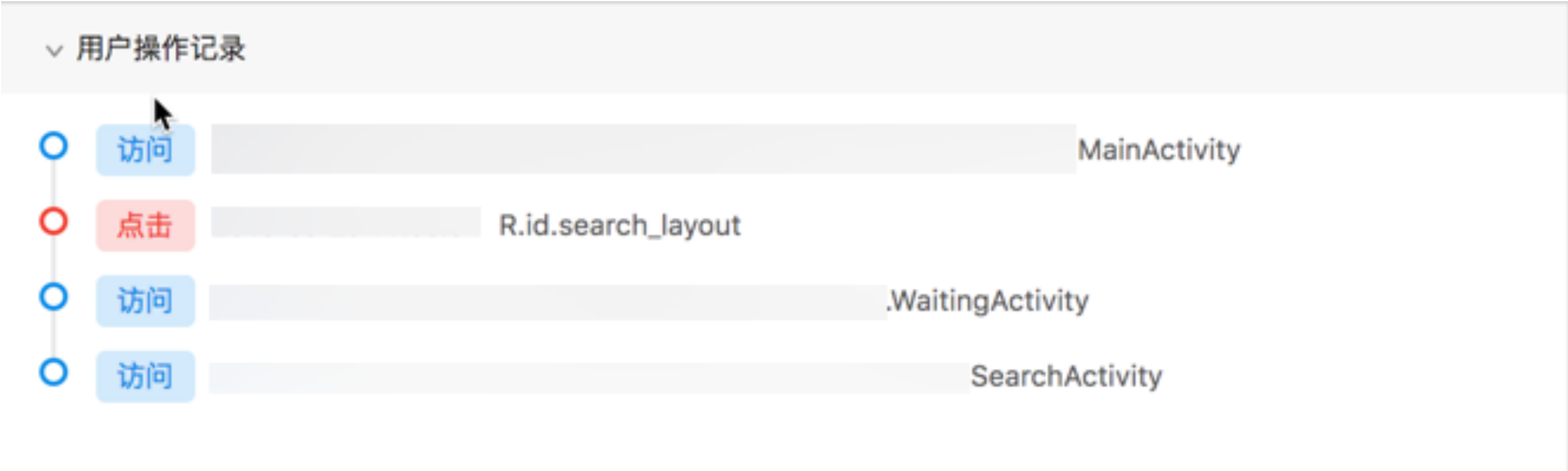


路径信息

# 用户操作路径

## 原理

- 生命周期回调
- Hook onClick事件



技术挑战

# 用户操作路径

用户操作记录		
访问		WelcomeActivity
访问		.MainActivity
点击	2131692283	
点击	2131692357	
访问		LocateManuallyActivity
访问		.MainActivity

## 问题1 如何翻译资源Id?

- 资源Id被内联
- 资源名称被混淆
- 系统资源名称

技术挑战

# 用户操作路径



技术挑战

# 用户操作路径

√用户操作记录			
○	访问		.WelcomeActivity
○	访问		MainActivity
○	点击	2131692283	
○	点击	2131692357	
○	访问		LocateManuallyActivity
○	访问		MainActivity

修改之前

√用户操作记录			
○	访问	2018-06-07 10:50:56	WelcomeActivity
○	访问	2018-06-07 10:50:57	MainActivity
○	点击	2018-06-07 10:51:02	R.id.close
○	点击	2018-06-07 10:51:05	R.id.layout_location_box
○	访问	2018-06-07 10:51:05	LocateManuallyActivity
○	访问	2018-06-07 10:51:06	.MainActivity

修改之后



技术挑战

## 用户操作路径

```
<Button
    android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="84dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:onClick="goThird"
    android:text="goThird"
    app:layout_constraintBottom_toTopOf="@+id/mytext"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />
```

```
public void goThird(View view) {
    Log.e("Tag", "view id : "+view.getId());
}
```

### 问题2 如何记录XML中点击事件？

- xml布局里面的onClick标签
- 点击事件不一定是onclick方法



技术挑战

## 用户操作路径

v7包

### AppCompatActivity#createView

```
if (view != null) {  
    // If we have created a view, check it's android:onClick  
    checkOnClickListener(view, attrs);  
}
```

```
/**  
 * An implementation of OnClickListener that attempts to lazily load a  
 * named click handling method from a parent or ancestor context.  
 */  
private static class DeclaredOnClickListener implements View.OnClickListener {  
    private final View mHostView;  
    private final String mMethodName;  
  
    private Method mResolvedMethod;  
    private Context mResolvedContext;  
  
    public DeclaredOnClickListener(@NonNull View hostView, @NonNull String methodName) {  
        mHostView = hostView;  
        mMethodName = methodName;  
    }  
  
    @Override  
    public void onClick(@NonNull View v) {  
        if (mResolvedMethod == null) {  
            resolveMethod(mHostView.getContext(), mMethodName);  
        }  
    }  
}
```

## Hook

DeclaredOnClickListener  
onClick方法

路径信息

## 方法执行路径

系统提供的调用堆栈

java.lang.RuntimeException:

An error occurred while executing doInBackground()

at android.support.v4.content.ModernAsyncTask\$3.done()

at java.lang.Thread.run(Thread.java:764)

... 3 more

- 性能损耗严重
- 无关键信息

路径信息

# 方法执行路径

自己记录调用堆栈

## 原理

- 编译期
  - 字节码插桩
- 运行期
  - 异步线程记录

方法调用史

历史结果集: 1

方法签名: 多个关键字请用空格隔开

线程: ☒ 全选

- ☒ AsyncTask #1 ☒ pool-24-thread-2
- ☒ fakeMainThread ☒ main ☒ pool-2-thread-1

搜索 清空

共 151 条数据, 当前是第1-151条

RootDetectionProcessor.getIsRoot

RootDetectionProcessor.getHasMalWare

EmulatorDetectionProcessor.getIsEmulator

EmulatorDetectionProcessor.getEmulatorInfo

DataProcessor.getPlatformInfo

DataProcessor.getLocationInfo

AsyncTask #1 ApkSigningBlock.addPayload

pool-24-thread-2 LocationInfoReporter.checkAndUpdateConfig

pool-24-thread-2 FakeMainThread.getInstance

pool-24-thread-2 FakeMainThread.post

pool-24-thread-2 LocateThreadPool\$ErrorHandleThreadPool.afterExecute

fakeMainThread LocationInfoReporter\$2\$1.run

- 时间序列
- 线程信息
- 关键字搜索
- 安全加密

技术挑战

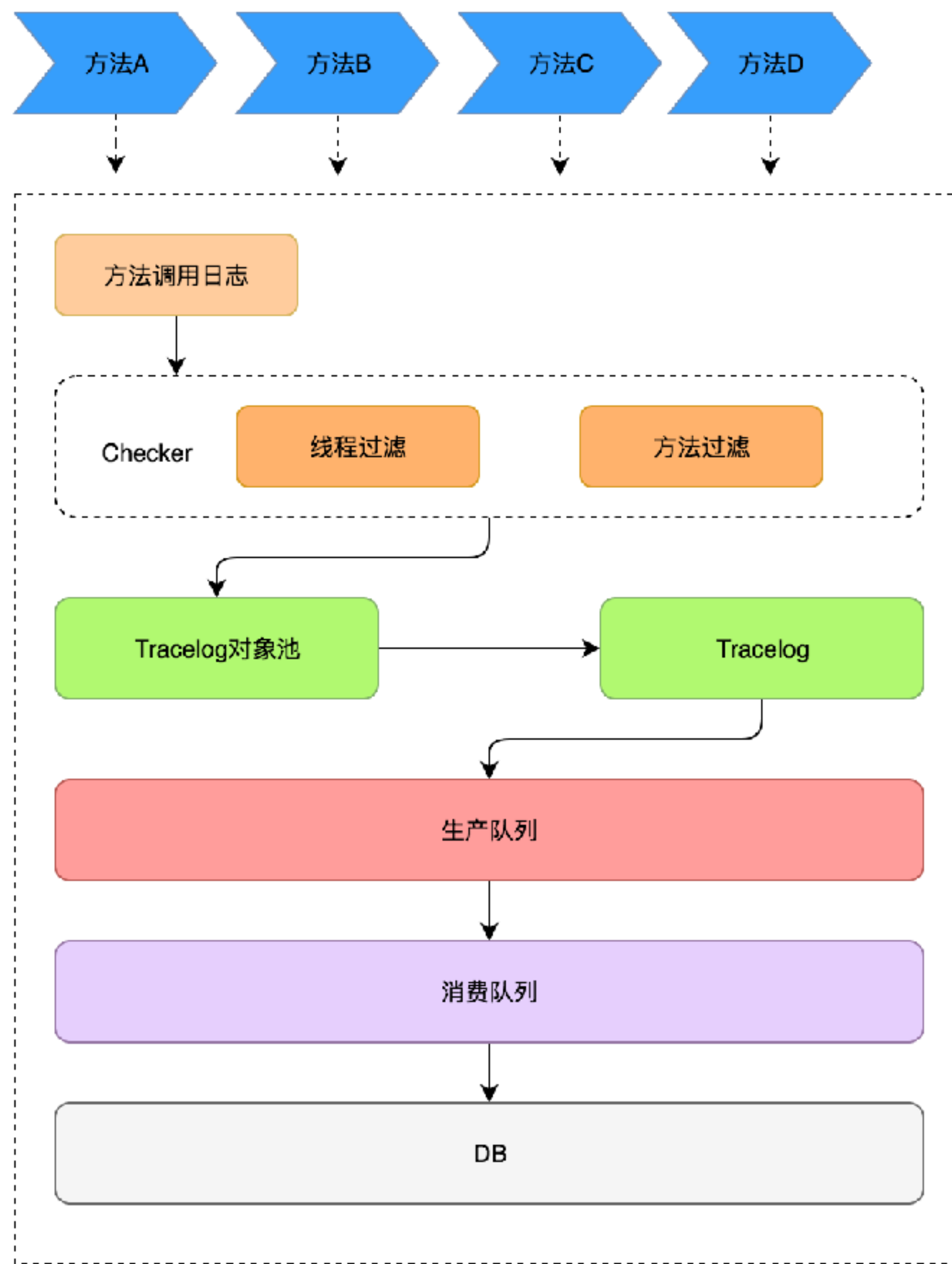
## 方法调用链路

### 问题

- 进入美团首页11万方法
- 平均1ms调用300方法

### 完全修复

- 插桩过滤
- 动态识别机制
- 对象池方案



网络信息

## 网络请求

### Retrofit

针对Retrofit\$Builder类的build方法进行插桩处理，通过添加一个拦截器进行相关信息的获取。

### Volley

针对Request类的parseNetworkResponse方法进行插桩处理



其他信息

# 用户权限列表

## 原理

- PackageManager.GET\_PERMISSIONS

▼ 用户权限信息

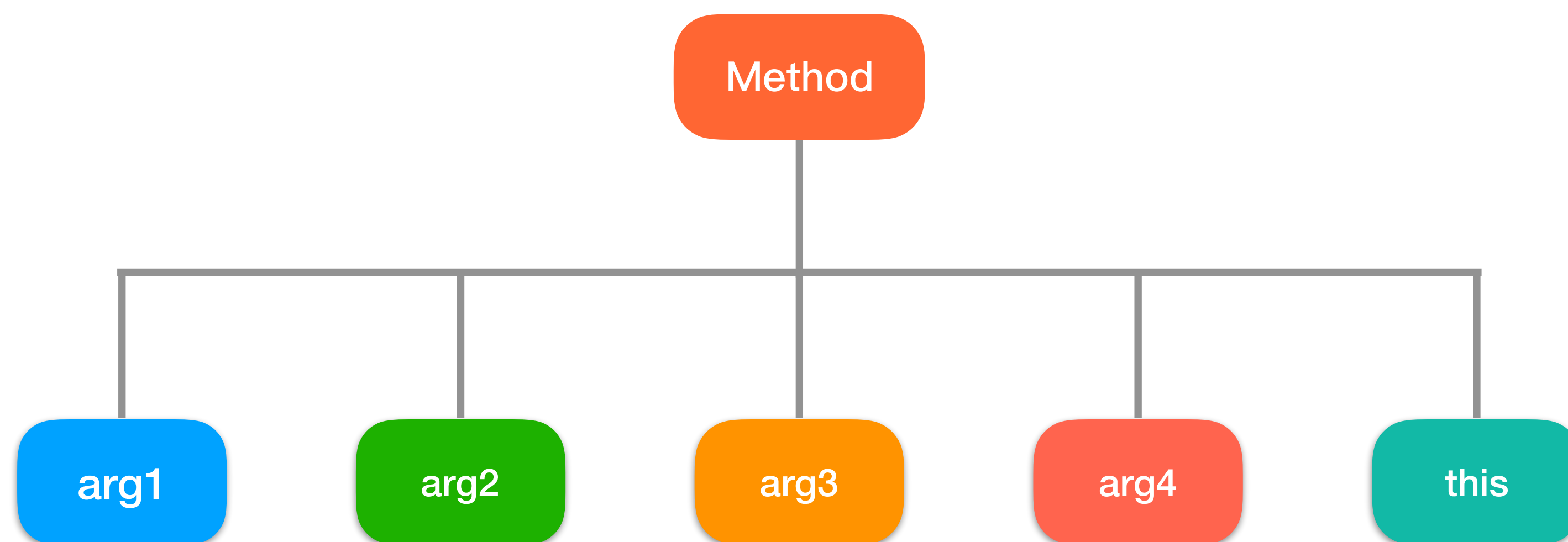
权限名	状态
	✔ 开启
android.permission.ACCESS_NETWORK_STATE	✔ 开启
android.permission.ACCESS_FINE_LOCATION	✔ 开启
android.permission.ACCESS_COARSE_LOCATION	✔ 开启
android.permission.CHANGE_WIFI_STATE	✔ 开启
android.permission.INTERNET	✔ 开启
android.permission.WRITE_EXTERNAL_STORAGE	✔ 开启
android.permission.ACCESS_WIFI_STATE	✔ 开启
android.permission.READ_PHONE_STATE	✔ 开启
android.permission.GET_TASKS	✔ 开启
android.permission.WAKE_LOCK	✔ 开启
android.permission.RECORD_AUDIO	✔ 开启
android.permission.DOWNLOAD_WITHOUT_NOTIFICATION	✔ 开启
android.permission.CAMERA	✔ 开启
android.permission.READ_CONTACTS	✔ 开启
android.permission.FLASHLIGHT	✔ 开启
android.permission.CALL_PHONE	✔ 开启
android.permission.READ_SMS	✔ 开启
	✔ 开启
android.permission.REQUEST_INSTALL_PACKAGES	✔ 开启
android.permission.VIBRATE	✔ 开启
android.permission.NFC	✔ 开启

其他信息

## 方法快照

### 原理

- 插桩
- 异步记录当前对象、方法参数
- Gson序列化
- 快照



其他信息

# 方法快照

## 支持类型

- 基本类型
- 对象类型
- 容器类型

方法对象快照

执行线程:

```
▼ ● 实参[0]: com.meituan.metrics.traffic.TrafficRecord
  S url: java.lang.String = "http://p0.meituan.net/0.0.100/wmbanner/0c79bef43e59a4d30770345d57a4da7a372533.jpg"
  S date: java.lang.String = "2018-06-07"
  N startTime: long = 1528359670476
  N duration: long = 339
  ▼ requestHeaders: java.util.Map = {java.util.Map(2)}
    ▼ ● 键值对[0]
      S 键: java.lang.String = "Connection"
      > A 值: java.util.Collections$SingletonList
        1)}
    ▼ ● 键值对[1]
      S 键: java.lang.String = "User-Agent"
      > A 值: java.util.Collections$SingletonList
        1)}
  > responseHeaders: java.util.Map = {java.util.Map(2)}
    N requestBodySize: long = 0
    N responseBodySize: long = 364435
    N requestHeaderSize: long = 104
    N responseHeaderSize: long = 617
    N type: int = 2
    N responseCode: int = 200
    N txBytes: long = 185
    N rxBytes: long = 365669
```

执行线程: main-1

```
▼ this: com.sankuai.meituan.setting>AboutMeituanActivity
  clickCount: int = 0
  MetaChannel: java.lang.String = 'undefined'
  updateClickable: boolean = true
  isActive: boolean = false
  mStateSaved: boolean = false
  mThemeld: int = 2131493062
  mCreated: boolean = false
  mNextCandidateRequestIndex: int = 0
  mReallyStopped: boolean = true
  mRequestedPermissionsFromFragment: boolean = false
  mResumed: boolean = false
```



解决方案

增强灵活性



信令下发

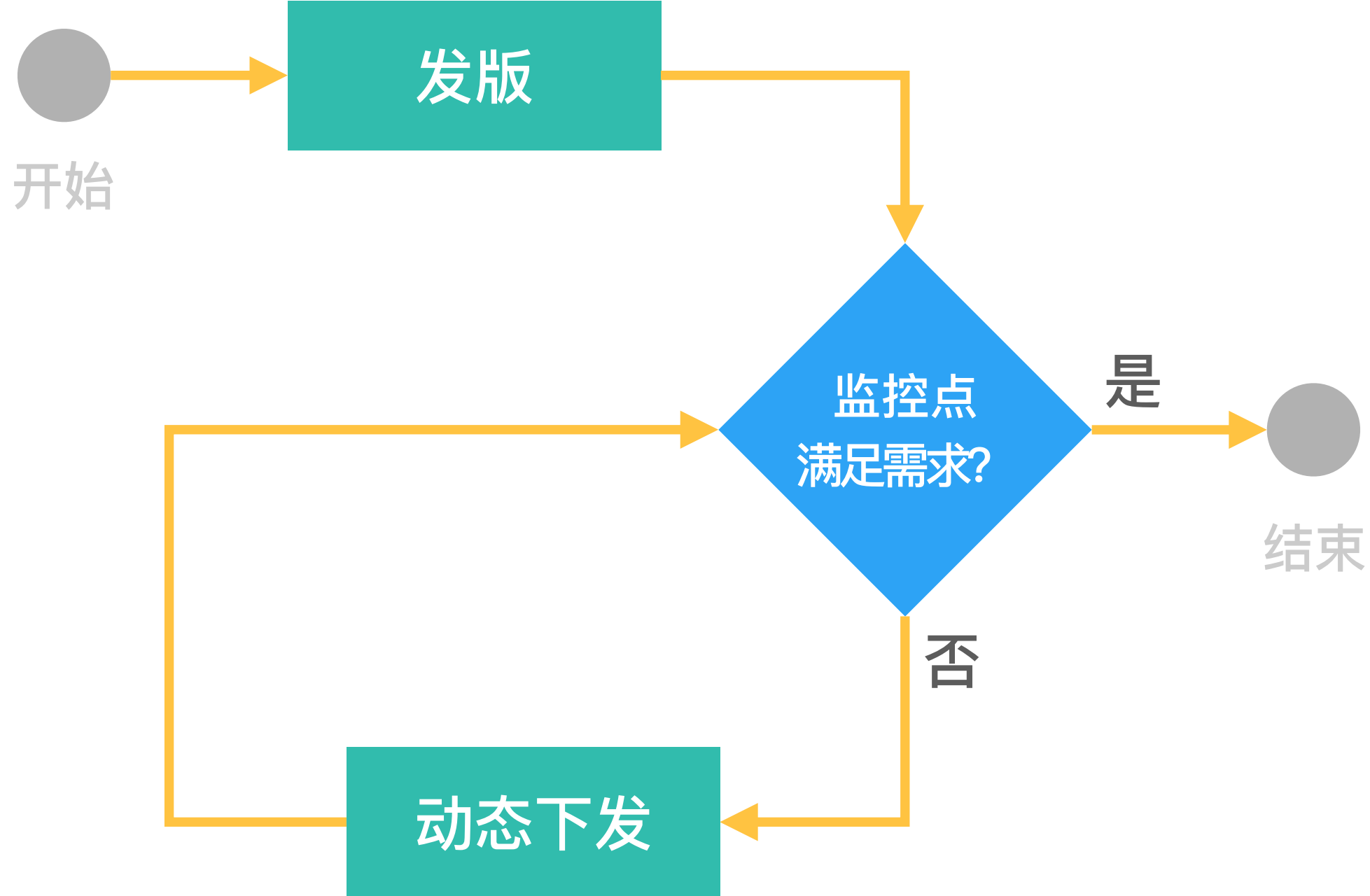
不依赖版本，根据监控配置执行相应监控逻辑。



脚本引擎

动态执行脚本逻辑

增强灵活性  
信令下发



实现监控点增删改

增强灵活性

指令举例



指令类型	指令作用
Throw	异常回调函数监控 一旦标记方法产生回调，立即上报
View	视图监控 主要检查点 1.启用 enable 2.可点击 clickable 3.可见 visibility 4.宽高是否正常 width/height
Start	路径监控 记录事件发生起点
End	路径监控 判断一个动作是否从一个开始点走到一个结束点
Forward	跳转监控 判断一个跳转动作是否完成
Http	网络信息监控 host+path
Customer	自定义信息监控 会调用业务方传进来的handler进行处理

解决方案

## 脚本引擎

期望客户端动态修改运行逻辑的能力。

- 业务逻辑微调
- 业务降级
- 疑难杂症调试

解决方案

# 脚本引擎

## 原理

键	函数指针
bindClass	javaBindClass
newInstance	javaNewInstance
newArray	javaNewArray



- 初始化时在jni层建立方法映射表【JNI】
- lua向共享栈中放入方法名,方法参数【LUA】
- jni层通过方法名称和方法映射表查找到对应方法并将入参检验后调用,函数指针实际是调用java层代码,实质是反射【JNI】
- java层代码执行函数后将方法结果入栈【JAVA】
- lua调用处可通过出栈得到结果【LUA】

解决方案

# 脚本引擎

描述： 在外卖运营日常监控中发现某个页面跳转错误异常上升，经查该页面跳转时没有带目标页面地址，所以无法定位问题。

方案： 下发一段脚本，将目标页面地址**动态**上报。

结果： 根据上报的地址定位到对应的页面以及页面负责人进而修改。

收益： **实时性**->分钟级别，**灵活性**->获取上下文信息

🐾 TRACE

TRACE

Luajava在线编辑助手

系统关键字

luajava: 实例化对象  
\_this: 当前java对象  
\_args[]: java方法参数  
\_return: java方法返回值

系统方法

getContext()  
logLocal(text)  
instanceof(object, className)  
uploadFileAsync(path)  
report(text)  
luajava.newInstance()  
luajava.bindClass()  
  
reportSimpleObject(name, object)  
reportSp(name)  
reportSp(name, key)  
reportPermission()  
reportPermission(name)  
reportInternalDb(name, sql)  
reportExternalDb(name, sql)

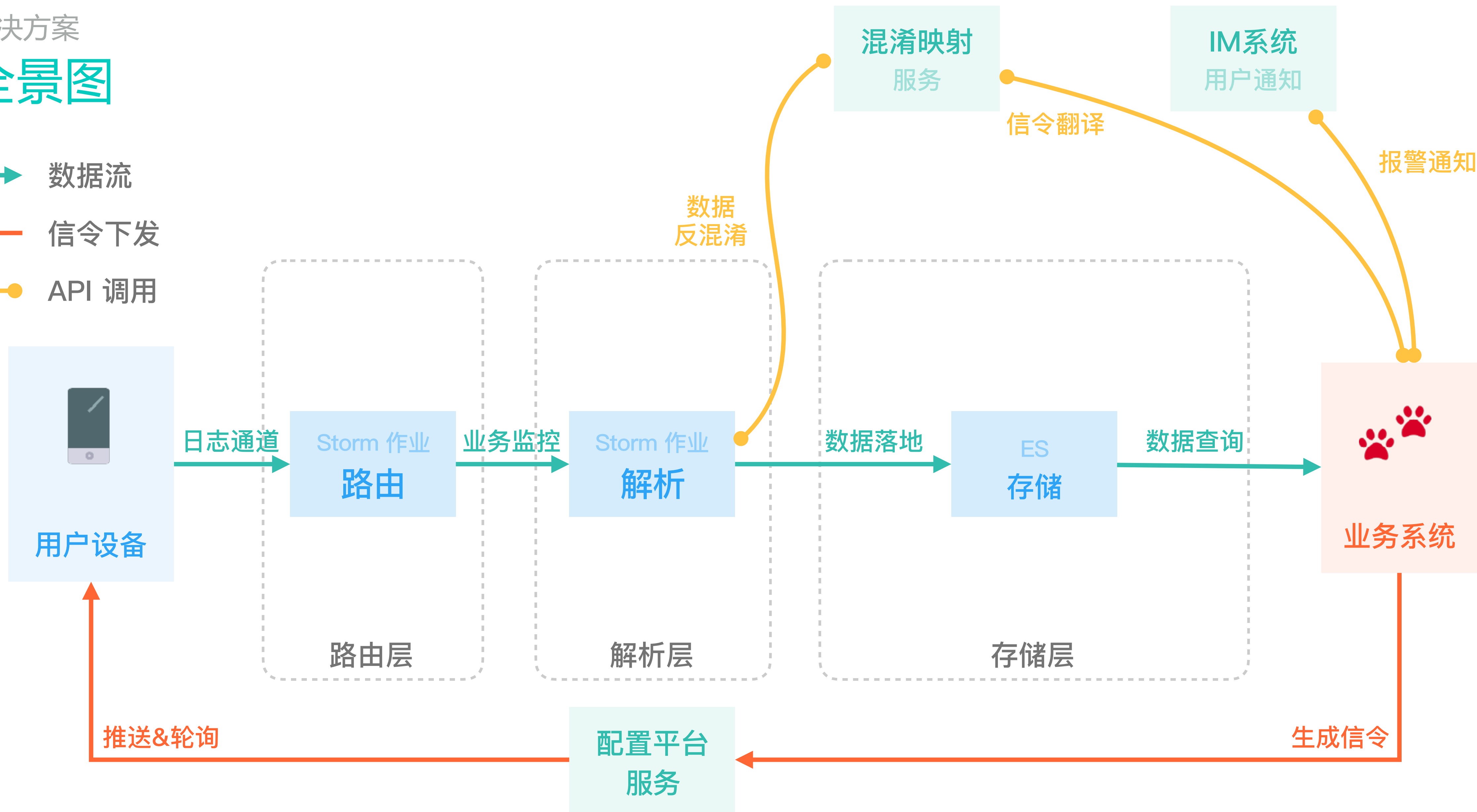
一键复制

Luajava-Editor

1 -- 新建哈希表  
2 local map = luajava.newInstance("java.util.HashMap")  
3 -- 设置对应参数  
4 map.put("url", \_args[0].c)  
5 -- 调用内置函数上报  
6 sniffer\_extra(map)

解决方案  
全景图

- 数据流
- 信令下发
- API 调用



# 目录

01 背景

02 解决方案

03 总结



总结

What ?

监控 [jiān kòng]

1. 监督控制。  
实行物价监控。
2. 监测控制机器、仪表等的工作状态或某些事物的变化。  
监控水位。

监测线上问题的状态和变化

综述  
Why ?



综述

How ?

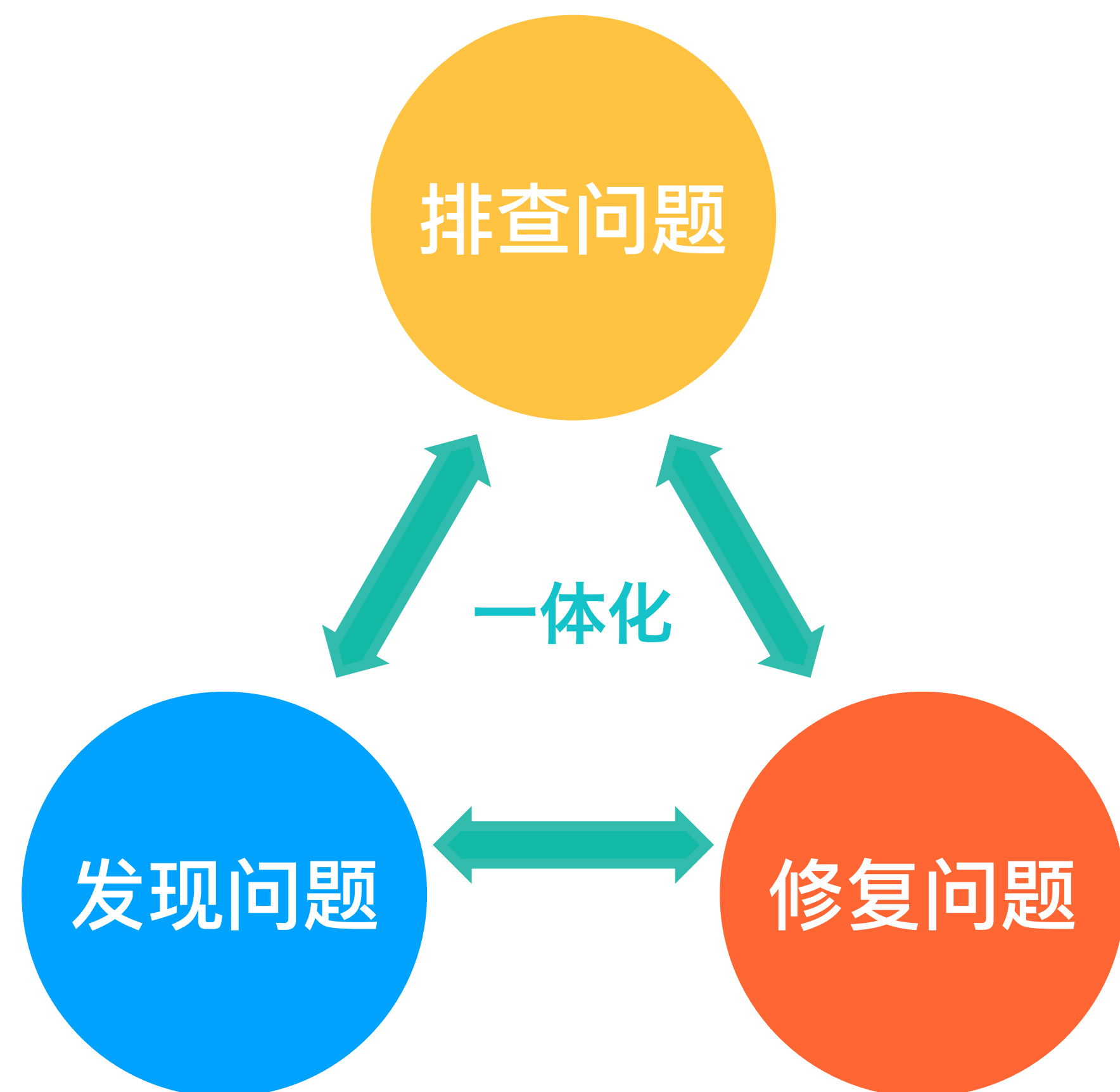
变被动为主动

提供充足的辅助信息

摆脱版本周期，灵活多变

总结

## 未来愿景



### 自动止损修复

- Native to H5
- So to java

### 自动完全修复

- 问题自动归类
- 问题自动分析

Q&A

综述  
方法论

What



Why

How

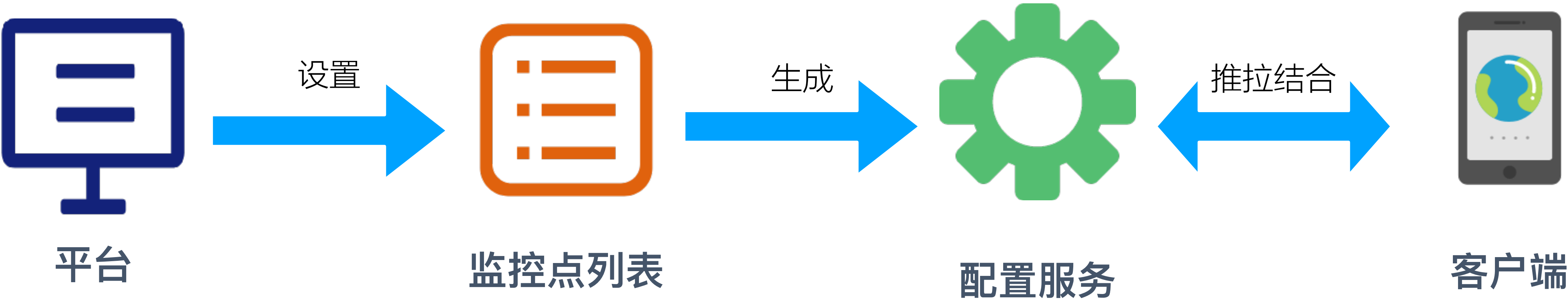
综述

# How ?

- 多种监控方式
- 实时数据指标
- 分钟级监控报警
- 可视化监控视图



技术挑战  
动态埋点





线上监控介绍

## 混淆翻译服务

- Android 应用经过了混淆，下发的信令和上报的数据均需经过翻译

方案：调用 **Mapping API**，实现对信令和  
数据翻译

- Android 应用经过了混淆，下发的信令和上报的数据均需经过翻译

方案：methodNumber与方法名的映射

线上监控介绍

## 主动采集原理介绍

```
@Override
@SnifferView(module = "subtle_2", enabled = FALSE, id = R.id.button5, scope = PAGE, interval = 3000)
@SnifferHttp(module = "subtle_2", host = "mapping.sankuai.com", path = {"/index/class", "/index/method"})
protected void onStart() {
    super.onStart();
    fakeFetch(1, (short) 1, new boolean[3]);
}

@SnifferStart(key = "fake network", module = "subtle_3", scope = PAGE, timeout = 510)
private void fakeFetch(int a, short b, boolean[] bs) {
    new Handler().postDelayed(() -> { fakeResponse(); }, 500);
}

@SnifferEnd(key = "fake network")
private void fakeResponse() { Log.i("Subtle", "in fakeResponse"); }
```

AOP  
Annotation Processor

匿名内部类 无效

transform  
读注解 命中-》生成

# CODE A BETTER LIFE

一行代码 亿万生活



更多技术干货  
欢迎关注“美团技术团队”

## Q&A