APP质量监控与性能优化

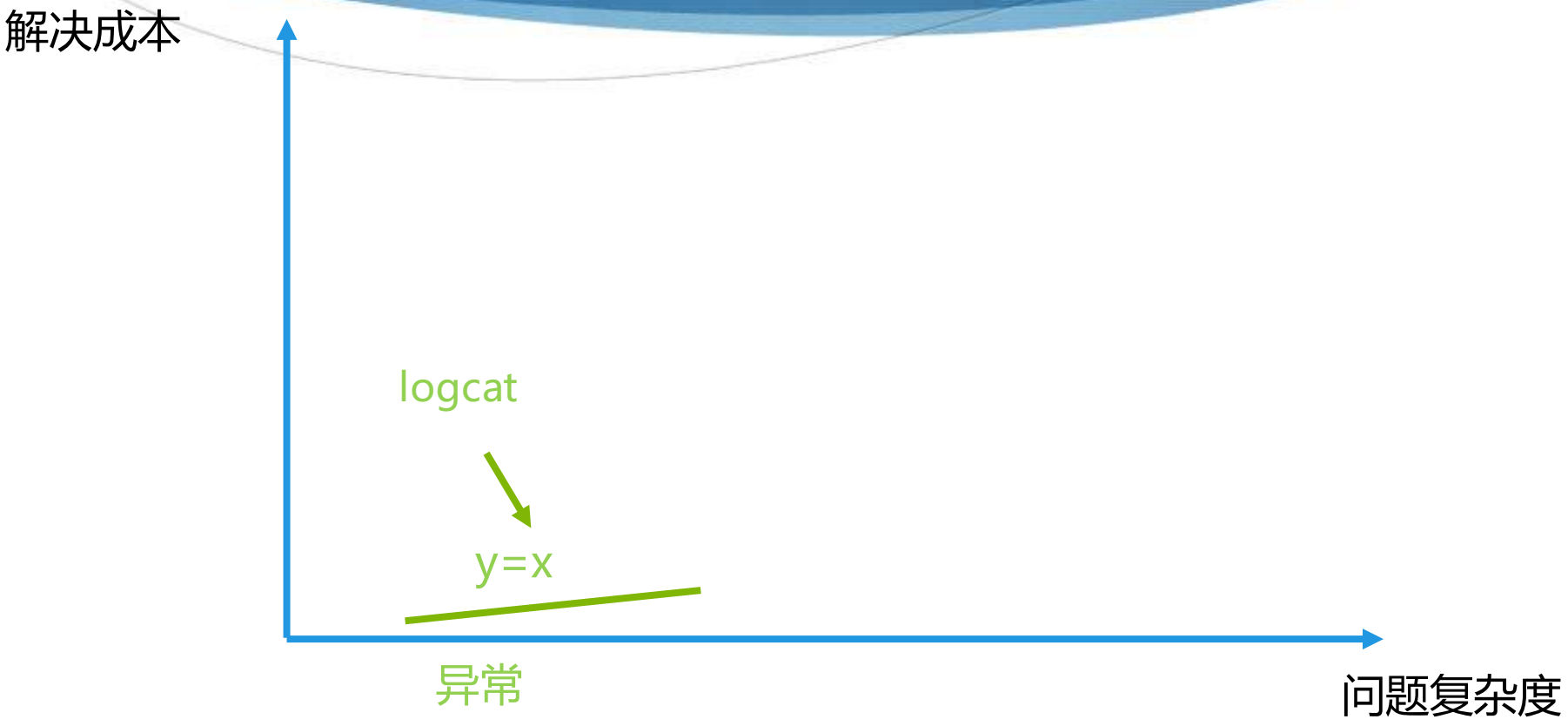
夏鸣远
AppetizerIO
theappetizerio@gmail.com

没有数据的调试是痛苦的！

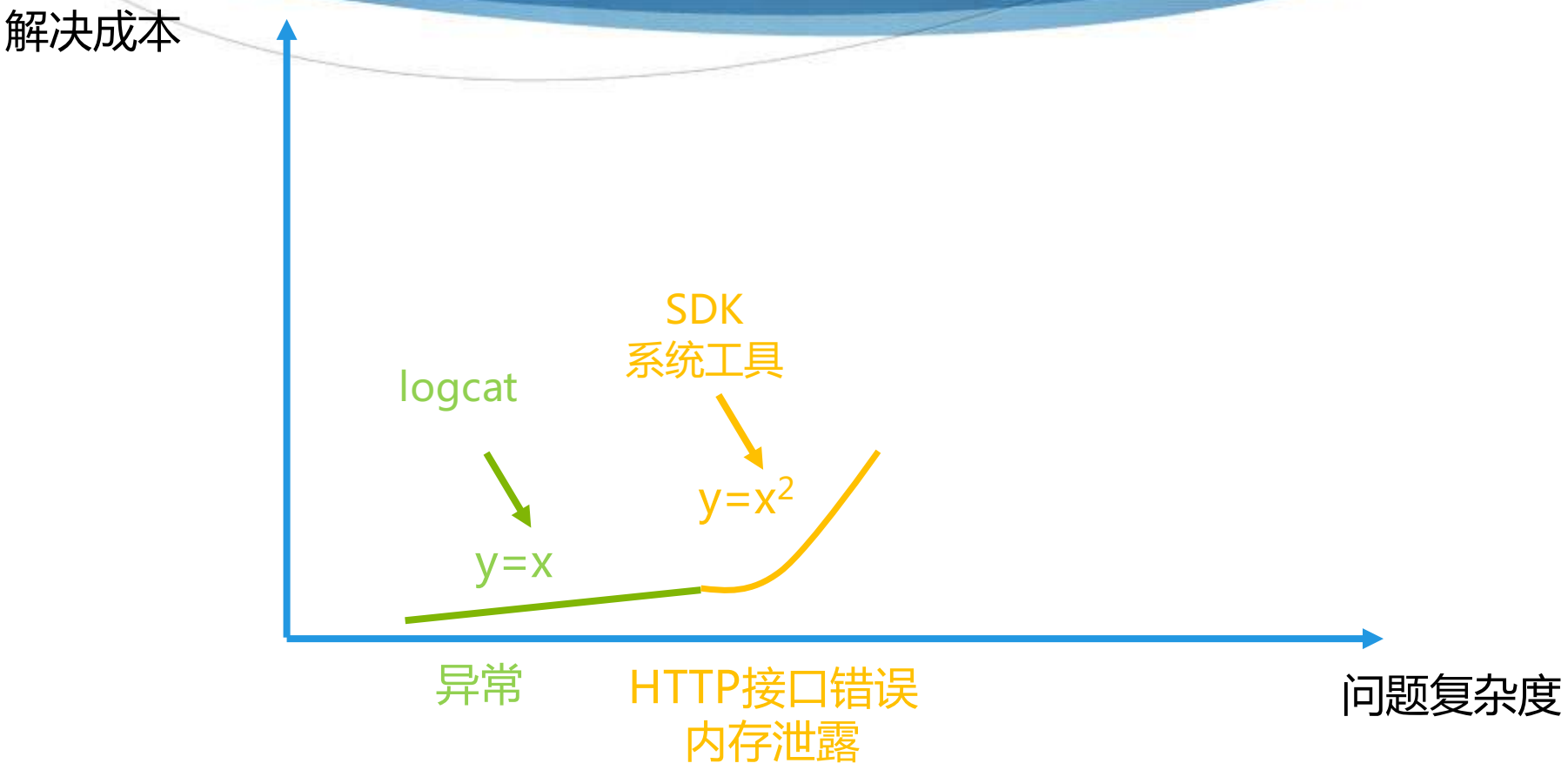
没有数据的调试是痛苦的！

- 没有堆栈的闪退
- 没有抓包的HTTP接口错误
- 没有Tracing的卡顿
- 没有埋点的业务错误

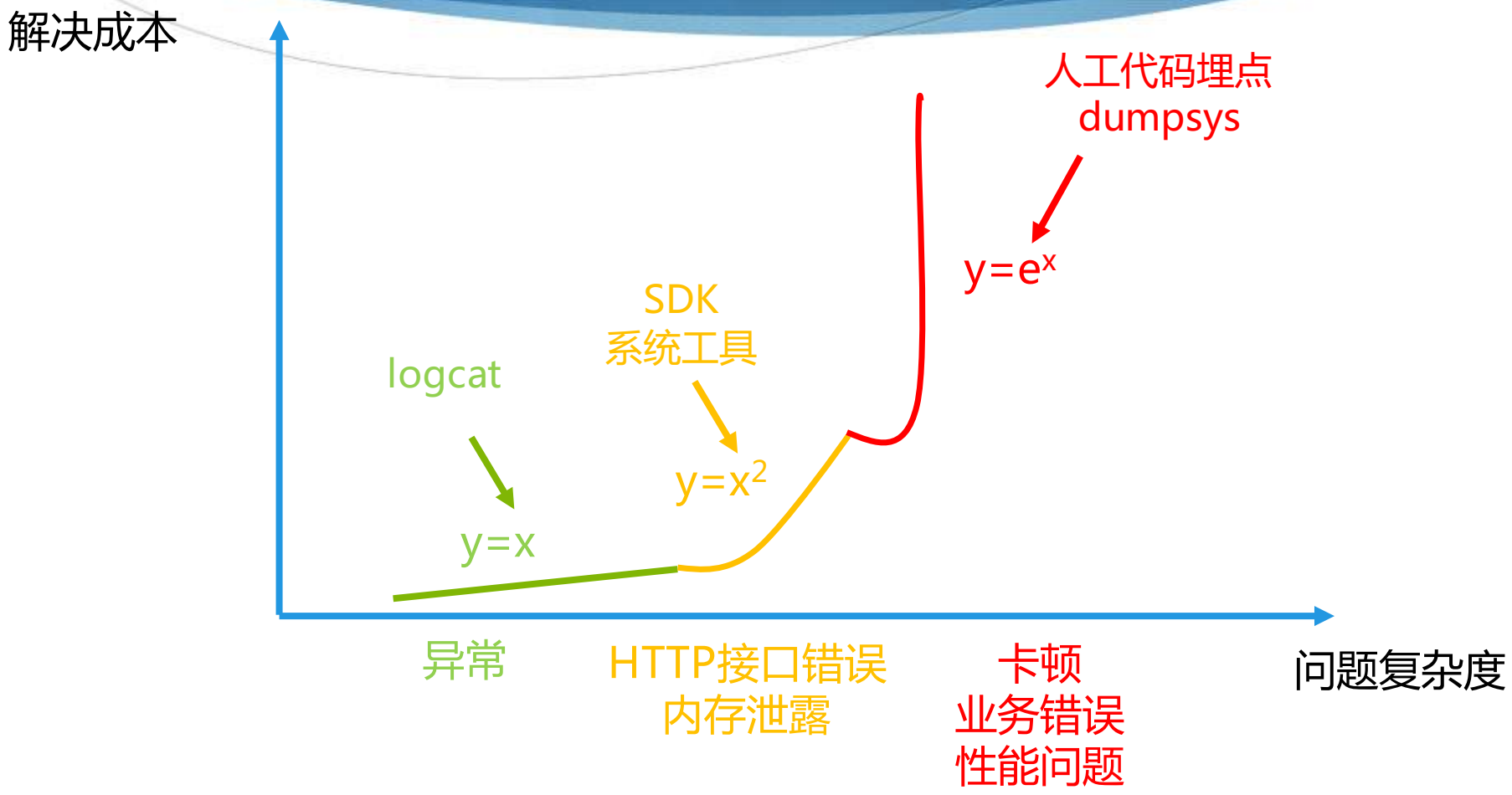
问题 与 解决问题的成本



问题 与 解决问题的成本

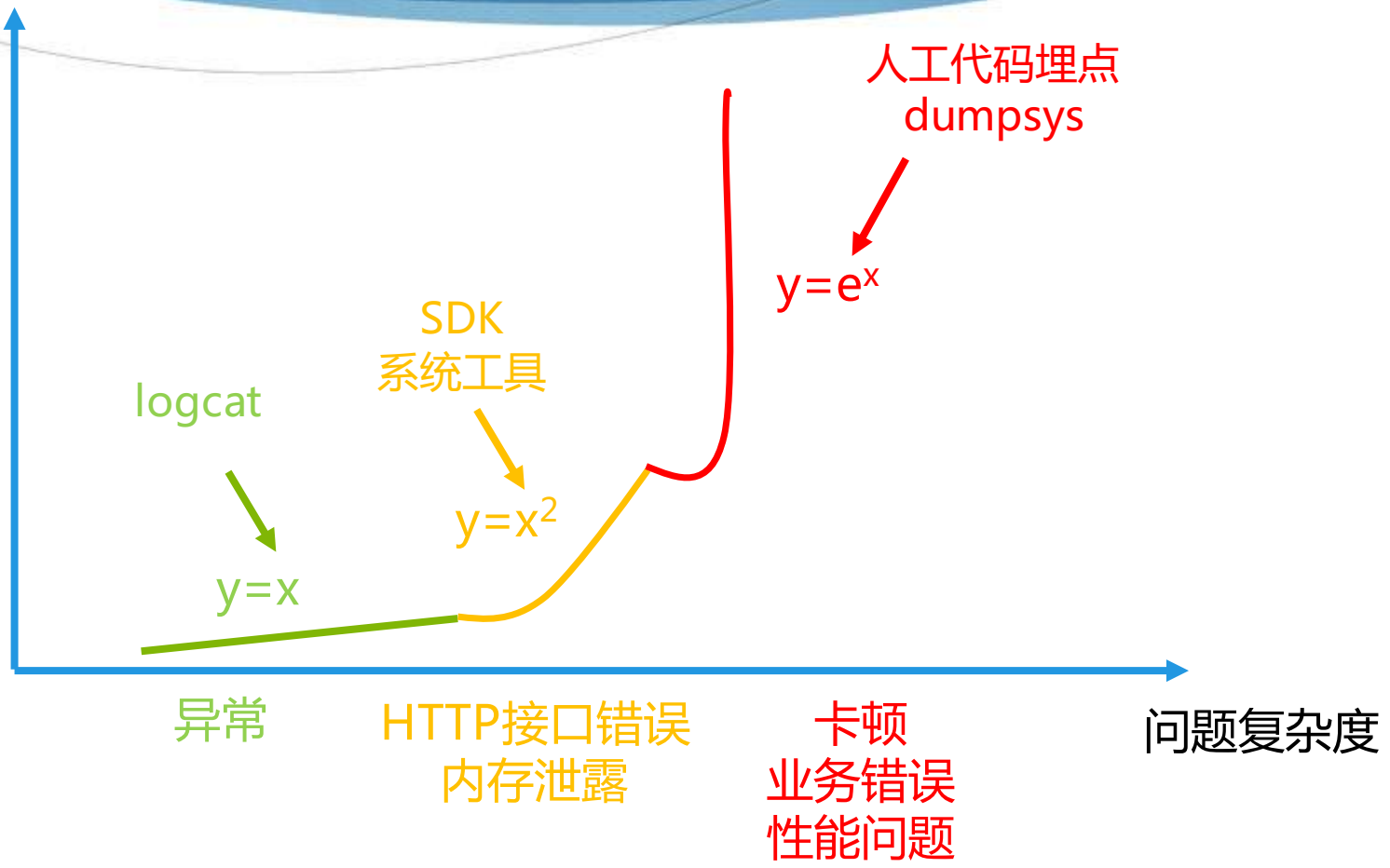


问题 与 解决问题的成本


解决成本
人工代码埋点
dumpsys
y=e^x
SDK
系统工具
logcat
y=x^2
y=x
异常
HTTP接口错误
内存泄露
卡顿
业务错误
性能问题
问题复杂度

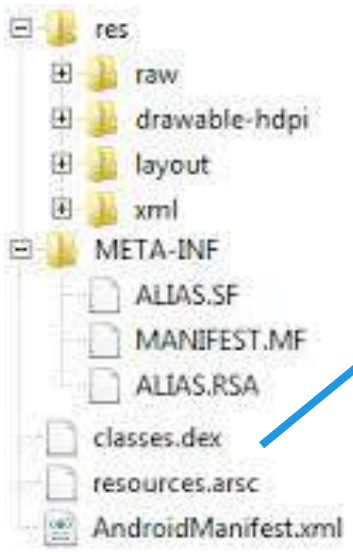
问题 与 解决问题的成本

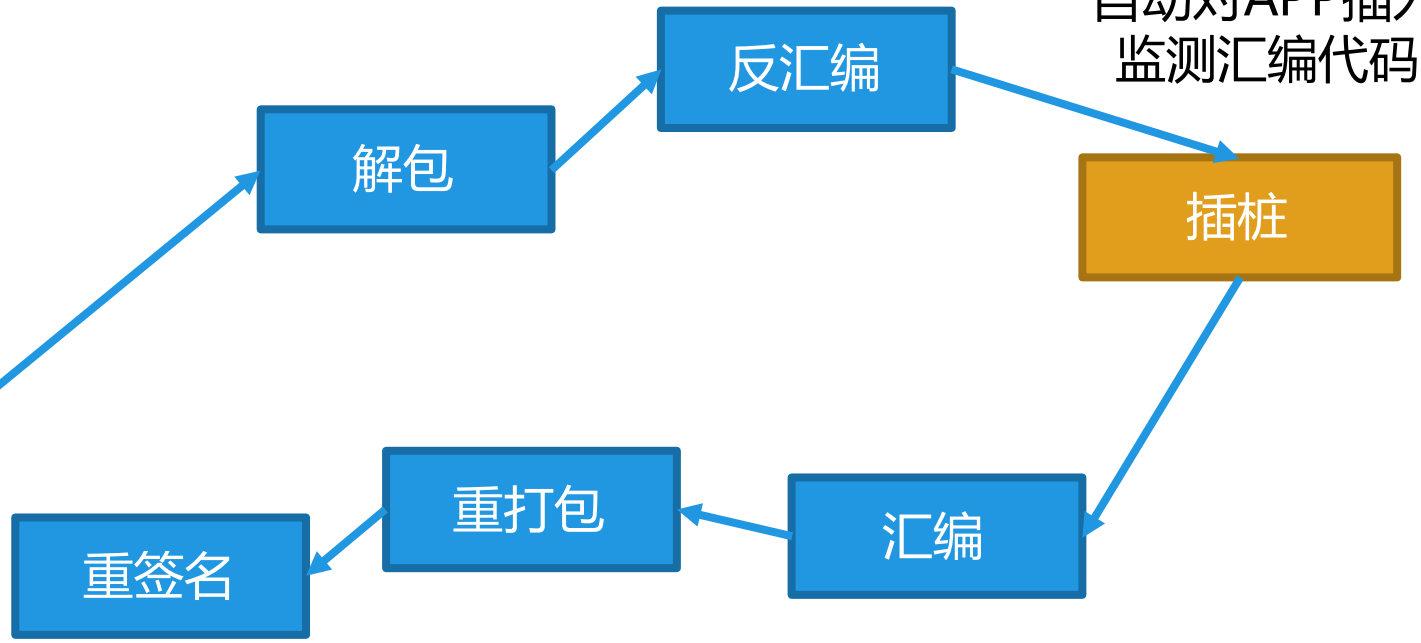
解决成本
代码维护成本
业务升级成本
更新适配成本
副作用
发包流程复杂度
团队理解复杂度



DEX插桩技术

- 基于smali（dex汇编器）





按照设计的规则
自动对APP插入
监测汇编代码

Smali 代码（DEX汇编）

```
.class public Lcom/apkudo/util/Serializer;
.super Ljava/lang/Object;
.source "Serializer.java"

# static fields
.field public static final TAG:Ljava/lang/String; = "ApkudoUtils"

# direct methods
.method public constructor <init>()V
    .registers 1

    .prologue
    .line 5
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    return-void
.end method
```

Class information

Static fields

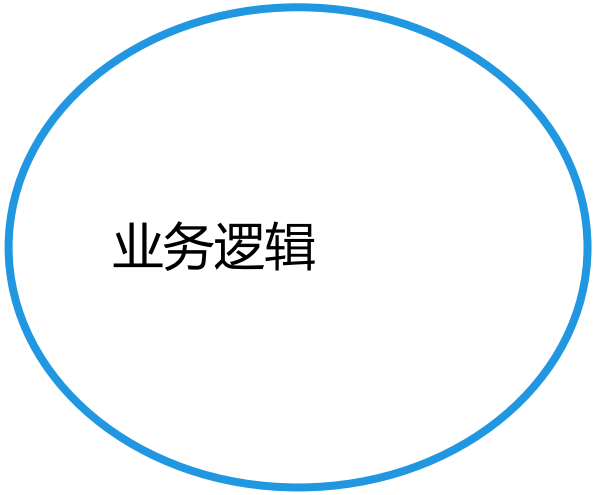
Methods
Direct
Virtual

插桩举例

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
+       Appetizer.logTimestamp().logActivityStart(MyAct.class, savedInstanceState)
+               .logUIThreadEvent(this);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast.makeText(getApplicationContext(),"2. onCreate()", Toast.LENGTH_SHORT).show();
+       Appetizer.logTimestamp().logActivityEnd(MyAct.class, savedInstanceState)
+               .logUIThreadEvent(this);
    }
```

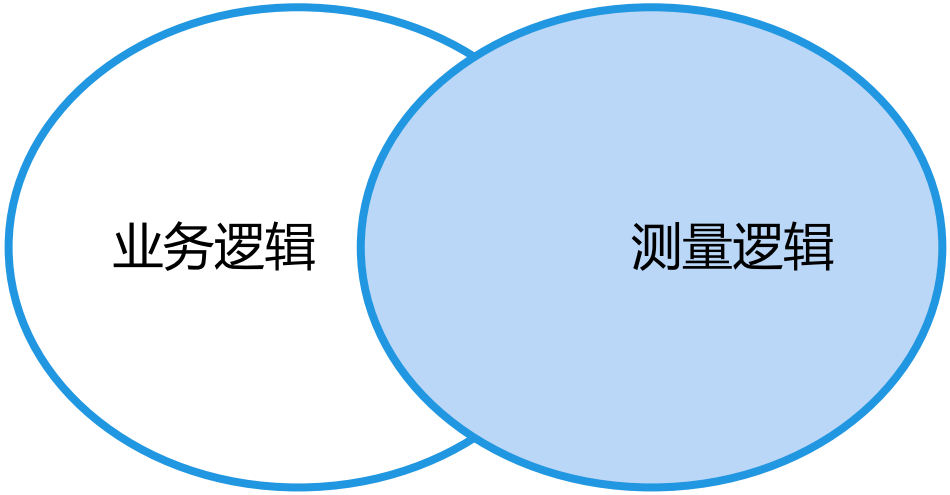
一键插桩

in.apk

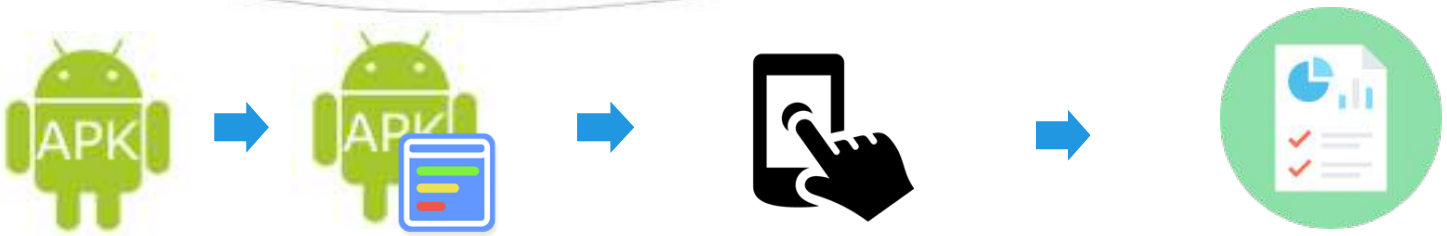


一键插桩

```
$> python insights.py process in.apk out.apk
```

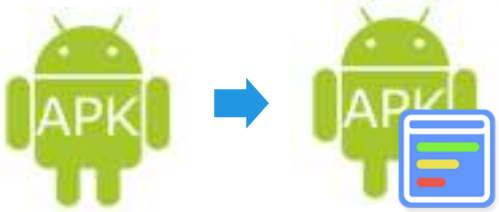


质量监控流程



插桩　　运行　　分析

插桩支持的APP开发技术



混淆 ✔
加固 ×

插桩

运行

分析

React Native

WEEX

Kotlin

Java 8

热补丁：Tinker, AndFix

RxJava

插桩测量内容

	SDK	工具	Appetizer
Java层全线程异常	✔	✔	✔
ANR	✔	✔	✔
主线程卡顿（事件、回调）	✔		✔
功能切换（Activity/Fragment）	**埋点**		✔
HTTP抓包（完整请求响应参数）	**埋点**	**配置**	✔
CPU/内存使用率		✔	✔
功能覆盖率	**埋点**		✔
网络流量		**配置**	✔
时序关系		**超慢**	✔

插桩接入方式

Jenkins

Python 命令行

图形化客户端

Gradle

公有云插桩服务

Docker

私有云部署

运行环境

插桩　　运行　　分析

	Appetizer
Root	×
配置特殊网络	×
安装特殊守护APP	×
连接USB线 ADB	×*

分析

错误 + 问题 + 深度分析 + 导出

Java异常
HTTP 400, 500

卡顿
HTTP响应慢

时间轴
流量分析
自动建模

JSON
HTML

报告 - 测试统计信息

界面覆盖率

36.67%

详细情况

有效测试时间

3.3 分钟

前台时间占比

56.3%

最长界面停留时间

26.3 秒

总计流量消耗

9.85 MB

平均网路延迟

42.45 ms

平均资源下载速度

593 B/s

问题分布

25 个问题

JVM层异常

java.lang.NullPointerException

异常全名： java.lang.NullPointerException

Stack trace:

加载Proguard规则反混淆堆栈

0: com.hotbitmapgg.bilibili.module.entry.HistoryFragment.finishCreateView(HistoryFragment.java:61)

1: com.hotbitmapgg.bilibili.base.RxLazyFragment.onViewCreated(RxLazyFragment.java:55)

2: android.support.v4.app.FragmentManagerImpl.moveToState(FragmentManager.java:1315)

3: android.support.v4.app.FragmentManagerImpl.moveFragmentsToInvisible(FragmentManager.java:2323)

4: android.support.v4.app.FragmentManagerImpl.executeOpsTogether(FragmentManager.java:2136)

5: android.support.v4.app.FragmentManagerImpl.optimizeAndExecuteOps(FragmentManager.java:2092)

6: android.support.v4.app.FragmentManagerImpl.execPendingActions(FragmentManager.java:1998)

功能切换缓慢

耗时的生命周期函数

问题： Life cycle method VipActivity.onCreate slows down UI switch

耗时： 505 ms

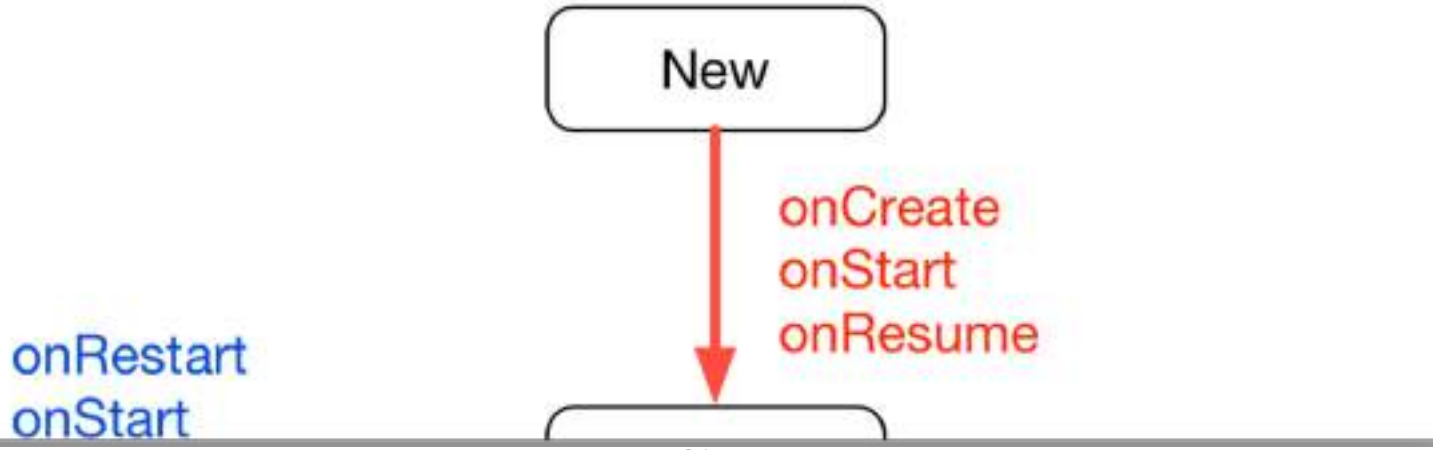
建议值： 200 ms

Activity： com.hotbitmapgg.bilibili.module.entry.VipActivity

启动时间： APP启动时间包含onCreate, onStart和onResume三个步骤，如果耗时较久，用户会在启动APP后看到一个空白的窗口，影响体验。建议控制三个生命周期函数的执行时间：onCreate(200ms), onStart(100ms) and onResume(80ms)，从而保证启动在半秒内完成。

暂停响应时间： 暂停响应时间发生在APP被部分遮盖的时候（例如被一个弹出框部分遮盖），建议将这个过程缩短以防用户感觉到卡顿。onResume和onPause的建议执行时间均为80ms。

切换时间： 切换时间指代APP从一个Activity切换到另一个Activity，或者切换到另一个APP的时间。为确保UI的流畅度，建议切换要在切换动画完成前完成。与UI切换有关的生命周期函数建议执行时间为：onStop(100ms), onRestart(100ms), onStart(100ms), onResume(80ms)。



自动HTTP抓包

URL: http://static.hdslb.com/live-static/images/mobile/android/small/xxhdpi/9.png?201703141338

请求时间： 211ms (总计时间) = 18ms (首包时间) + 193ms (传输时间)

HTTP方法： GET

API类型： urlconnection

返回码： 200

响应内容： image/png

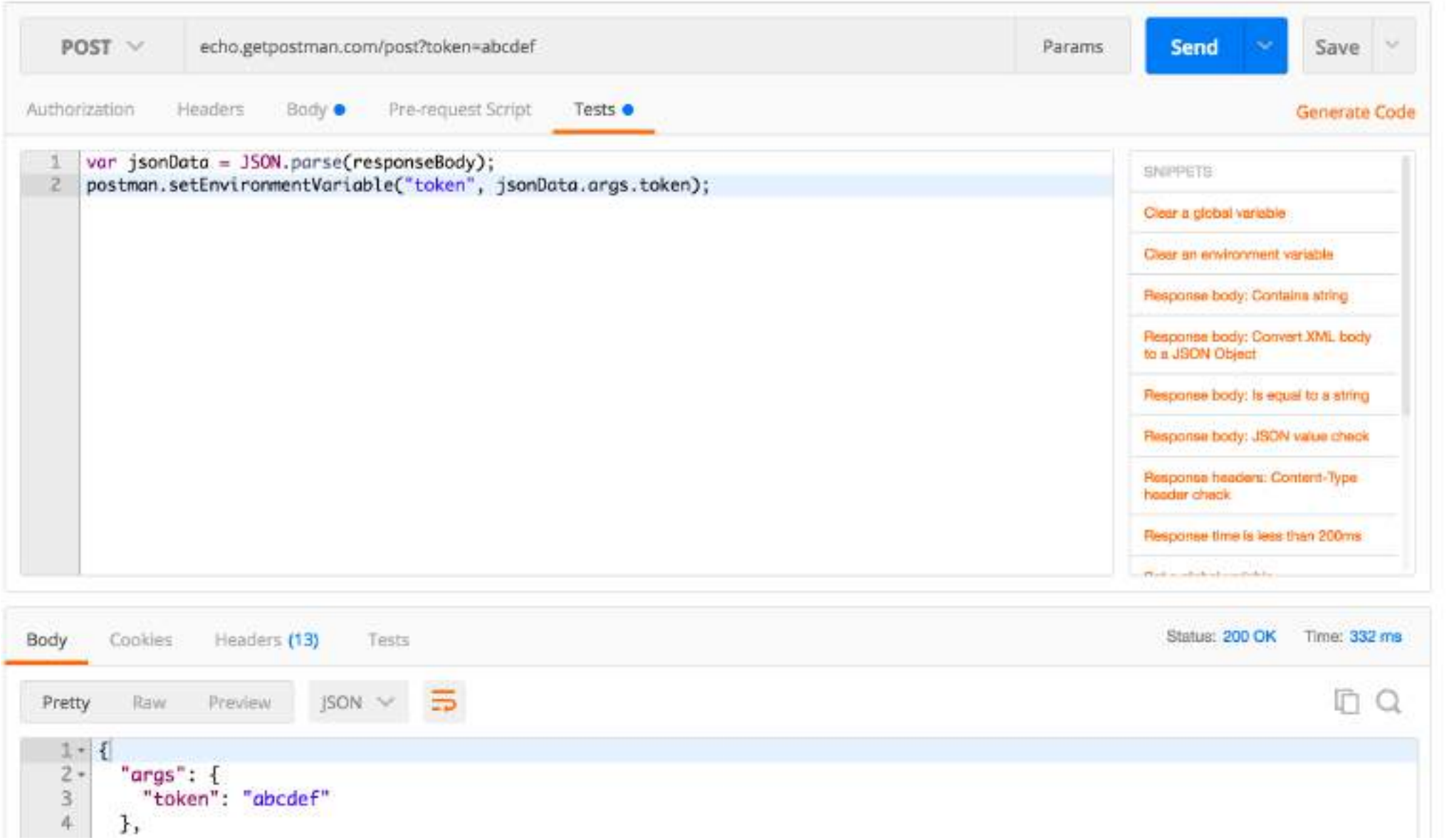
响应内容长度： 2.2 kB

缓存控制域： {"max-age":"2880"}

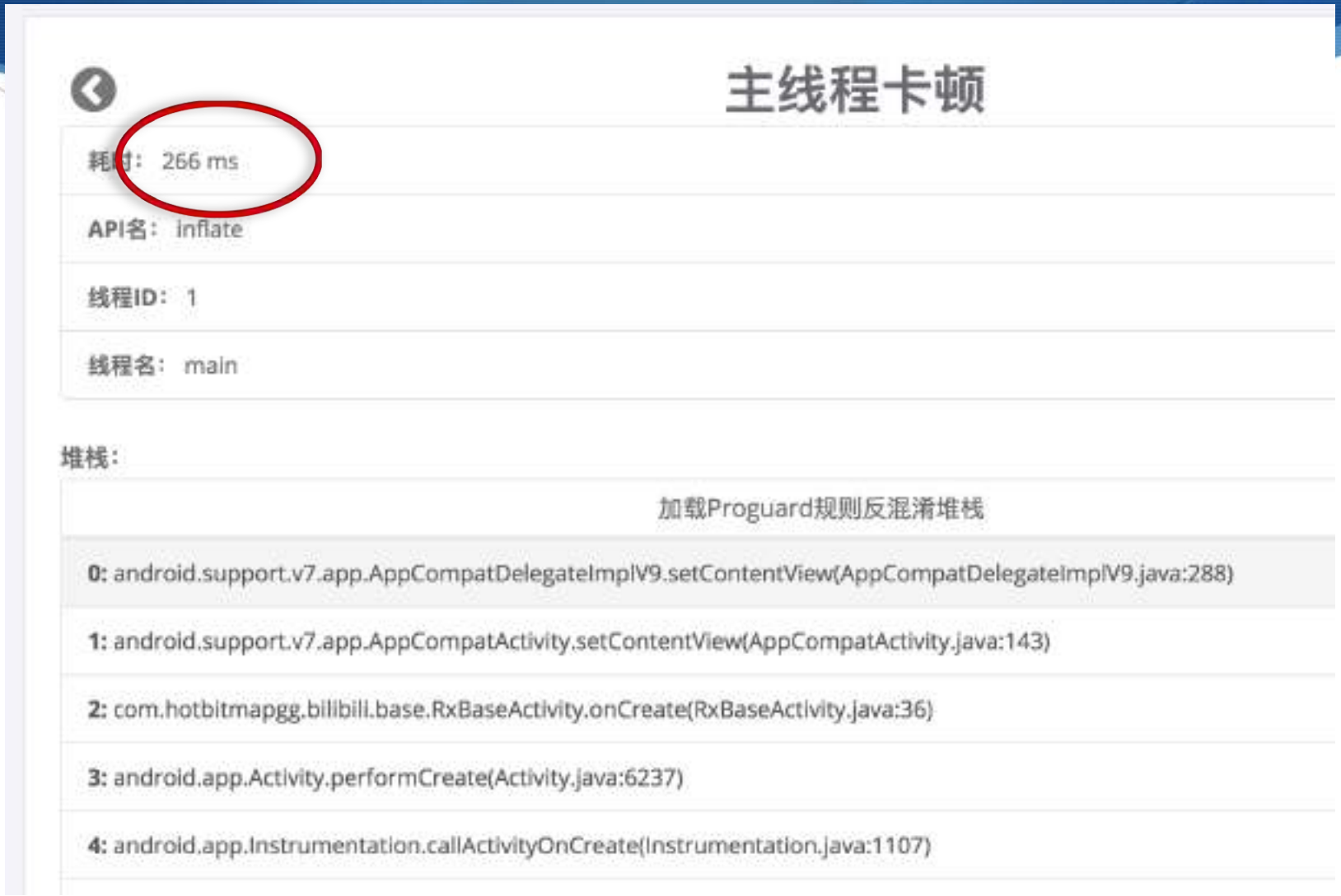
原始的HTTP Request Header

```
{
  "Accept-Encoding": "identity",
  "User-Agent": "Dalvik/2.1.0 (Linux; U; Android 6.0;
}
```

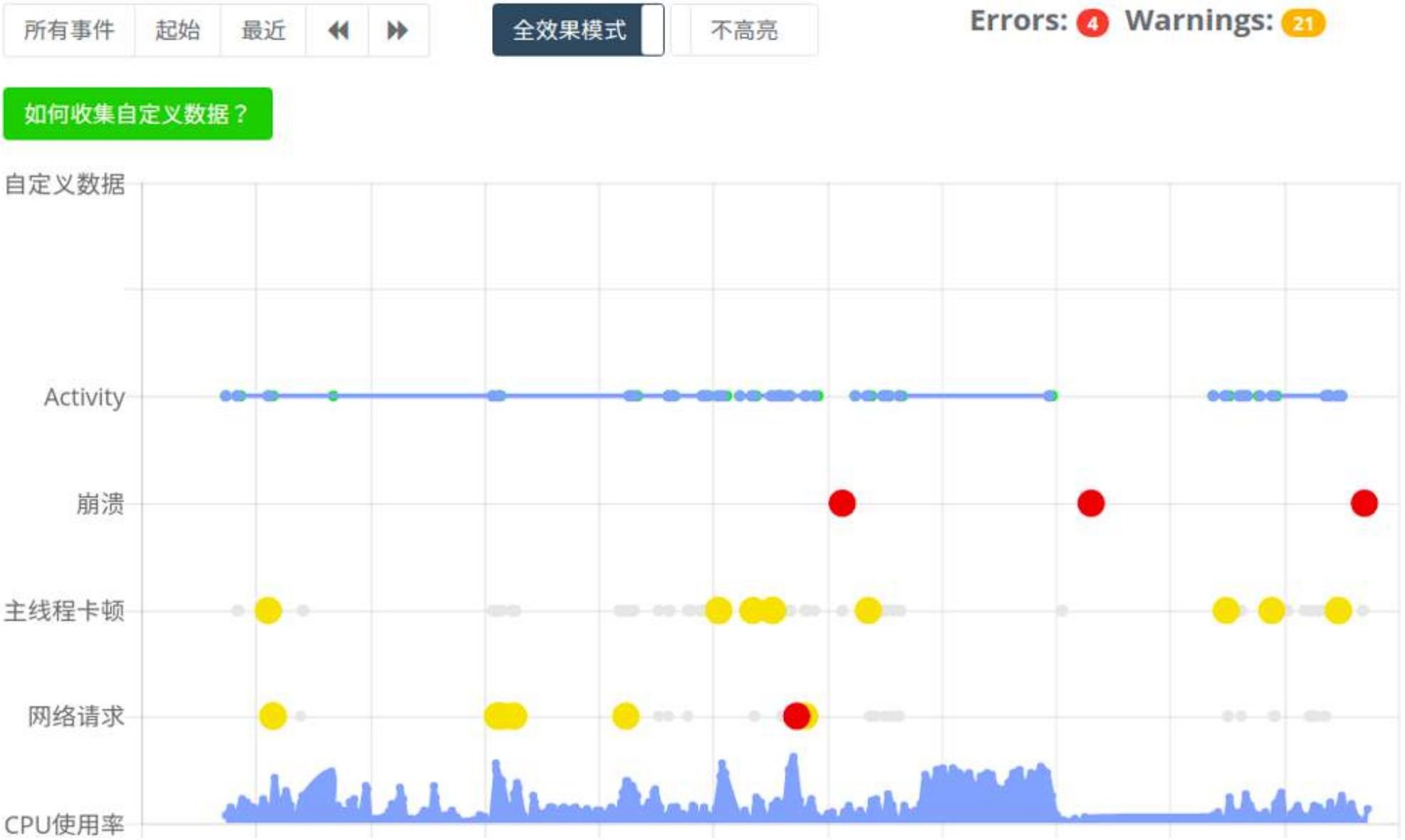
一键Postman重现



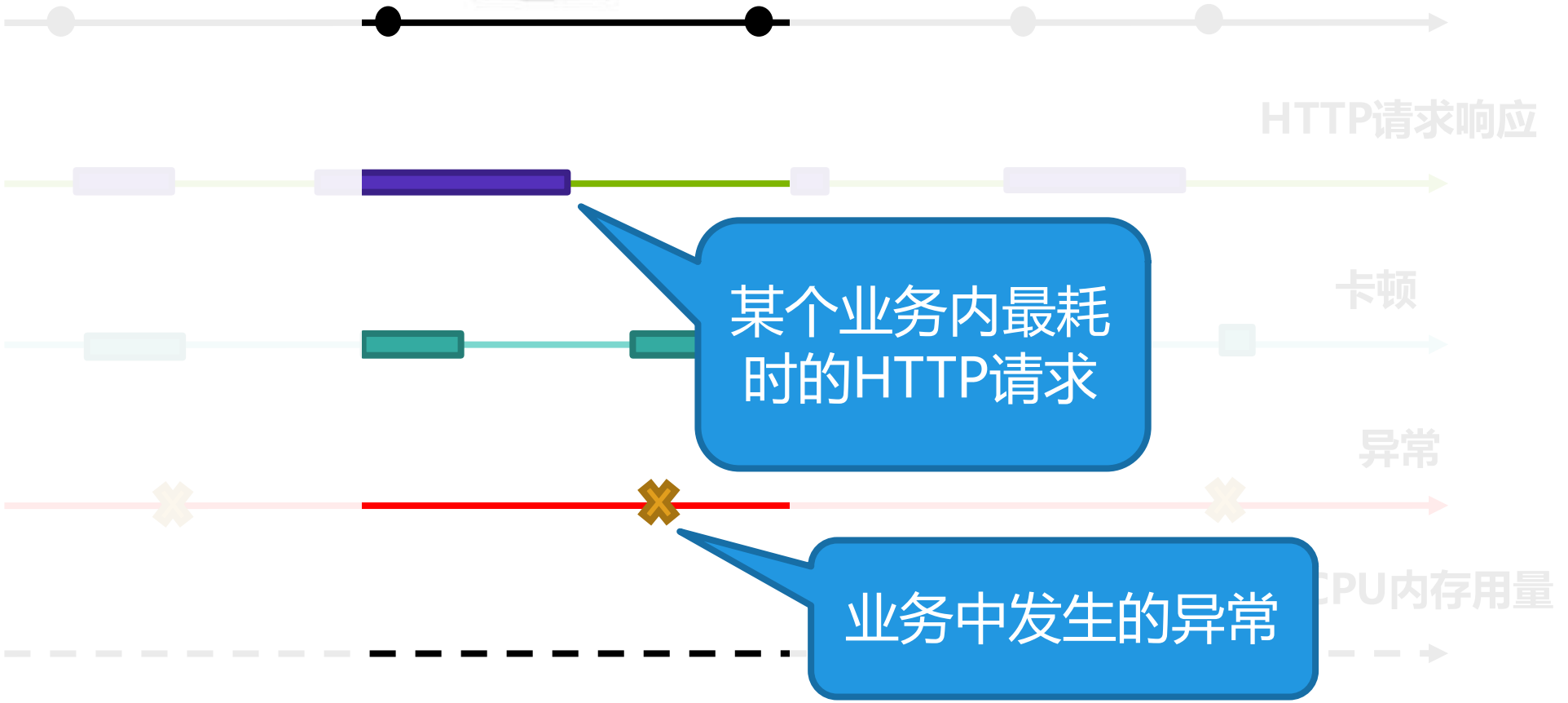
主线程卡顿


主线程卡顿
耗时：266 ms
API名：inflate
线程ID：1
线程名：main
堆栈：
加载Proguard规则反混淆堆栈
0: android.support.v7.app.AppCompatDelegateImplV9.setContentView(AppCompatDelegateImplV9.java:288)
1: android.support.v7.app.AppCompatActivity.setContentView(AppCompatActivity.java:143)
2: com.hotbitmapgg.bilibili.base.RxBaseActivity.onCreate(RxBaseActivity.java:36)
3: android.app.Activity.performCreate(Activity.java:6237)
4: android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1107)

时间轴图

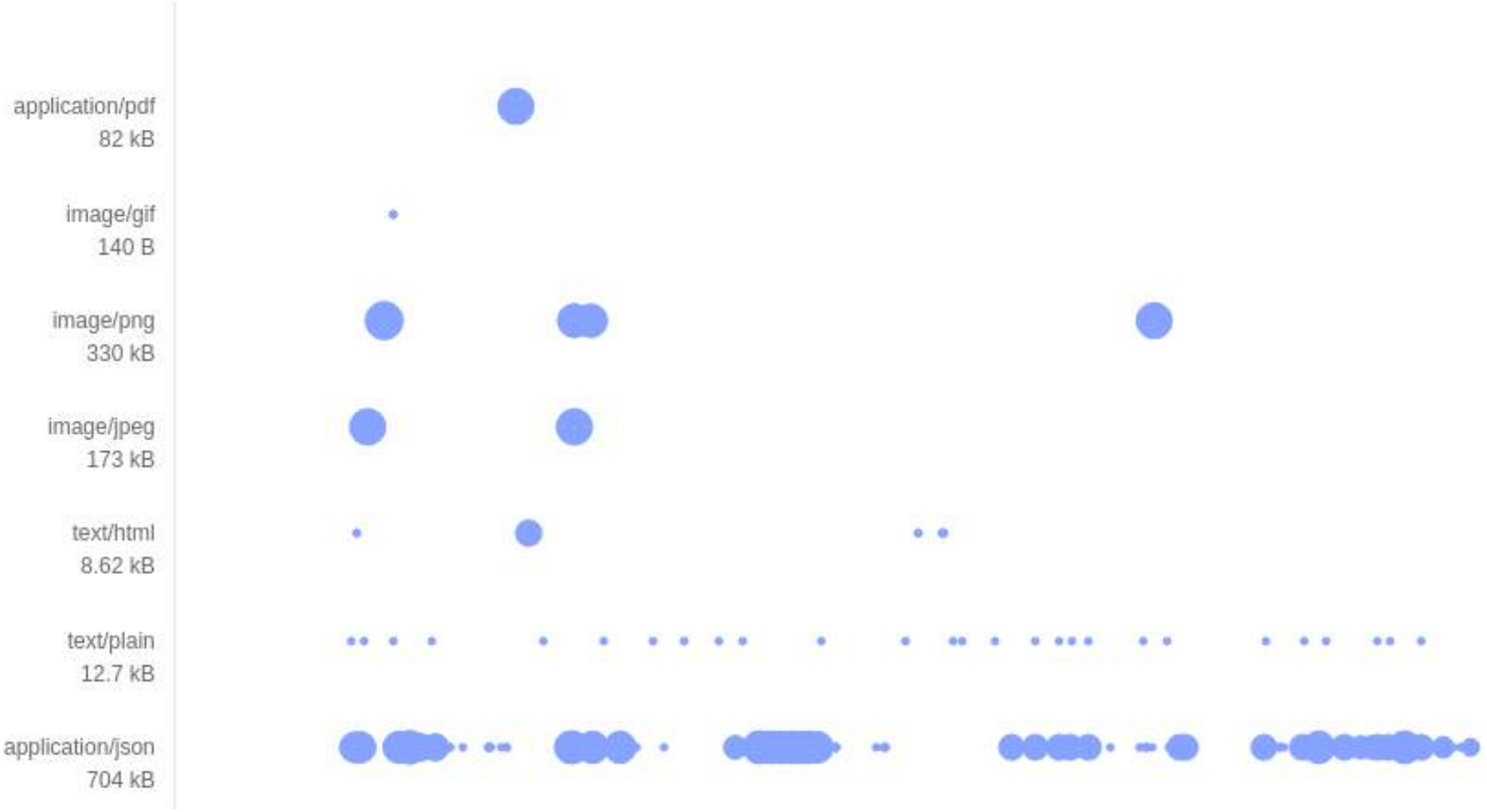

所有事件
起始
最近
全效果模式
不高亮
Errors: 4
Warnings: 21
如何收集自定义数据？
自定义数据
Activity
崩溃
主线程卡顿
网络请求
CPU使用率

深度分析：导出，搜索，排序

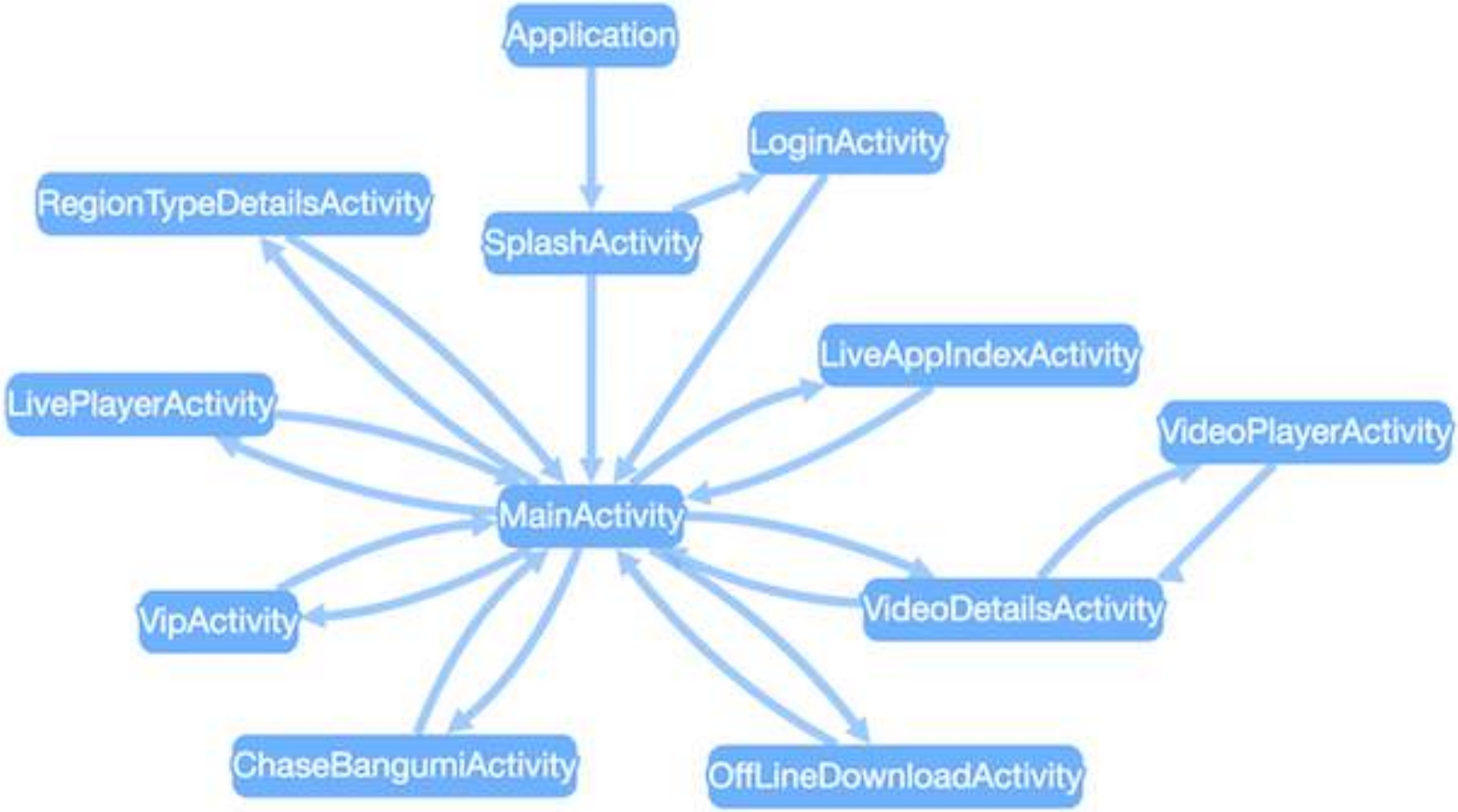


流量分析

按类型分类 | 按MIME类型分类 | 按根域名分类 | 按子域名分类 | 按请求方法 | 按网络库



自动功能建模


Application
LoginActivity
RegionTypeDetailsActivity
SplashActivity
LiveAppIndexActivity
LivePlayerActivity
VideoPlayerActivity
MainActivity
VipActivity
VideoDetailsActivity
ChaseBangumiActivity
OffLineDownloadActivity

插桩的性能、安全影响

- 测量开销
  - HTTP – 非主线程 测量开销<1ms
  - CPU/内存数据采集 约 2% CPU占用
  - 页面切换 测量开销<1ms
- 安全
  - 不搜集请求body，私有化部署，数据透明，插桩代码透明

DEX 插桩

- SDK：接入、配置成本，涵盖有限，副作用
- 手工埋点：难维护，混乱
- 系统层工具：各种局限性，采集数据有限

传统采集方式

- 全自动一键插桩，零维护
- 持续化集成生态
- 测量广泛：闪退、性能、网络
- 数据开放、整齐、透明

DEX插桩

Appetizer 资源

- 官网及图形化客户端下载： https://appetizer.io/cn/
- Python客户端：https://github.com/appetizerio/insights.py
- 社区，文档：https://testerhome.com/topics/node127
- 知乎专栏： https://zhuanlan.zhihu.com/smartmobdev
- QQ群：467889502
- 私有化部署：theappetizerio@gmail.com

APP质量监控与性能优化

夏鸣远
AppetizerIO
theappetizerio@gmail.com