

基于 DEX 插桩的 自动应用质量监控

夏鸣远

AppetizerIO

theappetizerio@gmail.com



以数据说质量

- 业务埋点 → 用户路径，留存活跃
- 开发 log → logcat 日志
- 正确性 → 捕获异常
- 性能数据 → 帧率，卡顿，耗电
- HTTP 流量 → 接口稳定性、性能、流量等
- 覆盖率，性能图表 → 测试有效性

DEX 插桩

- 手工埋点：难维护，单应用
- SDK：接入、配置成本，涵盖有限
- 系统层工具：各种局限性，采集数据有限
- 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
- 数据透明统一

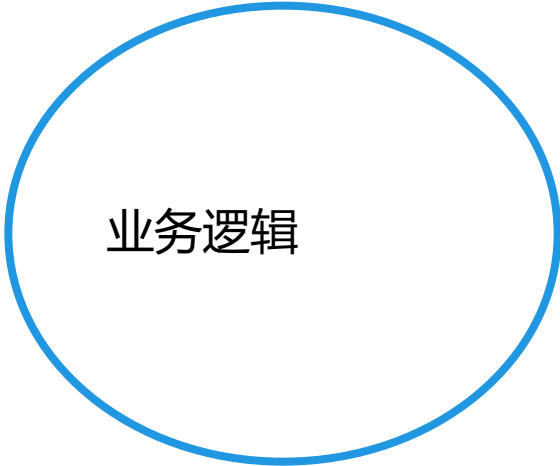
传统采集方式



DEX插桩

一键插桩

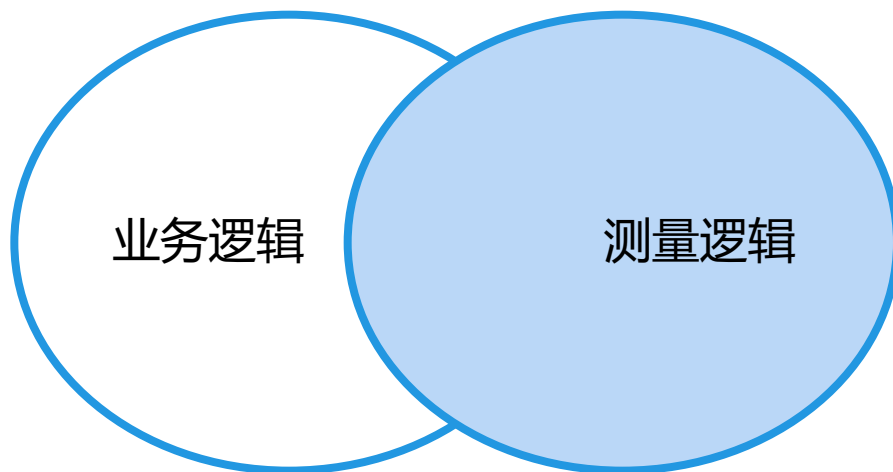
in.apk



业务逻辑

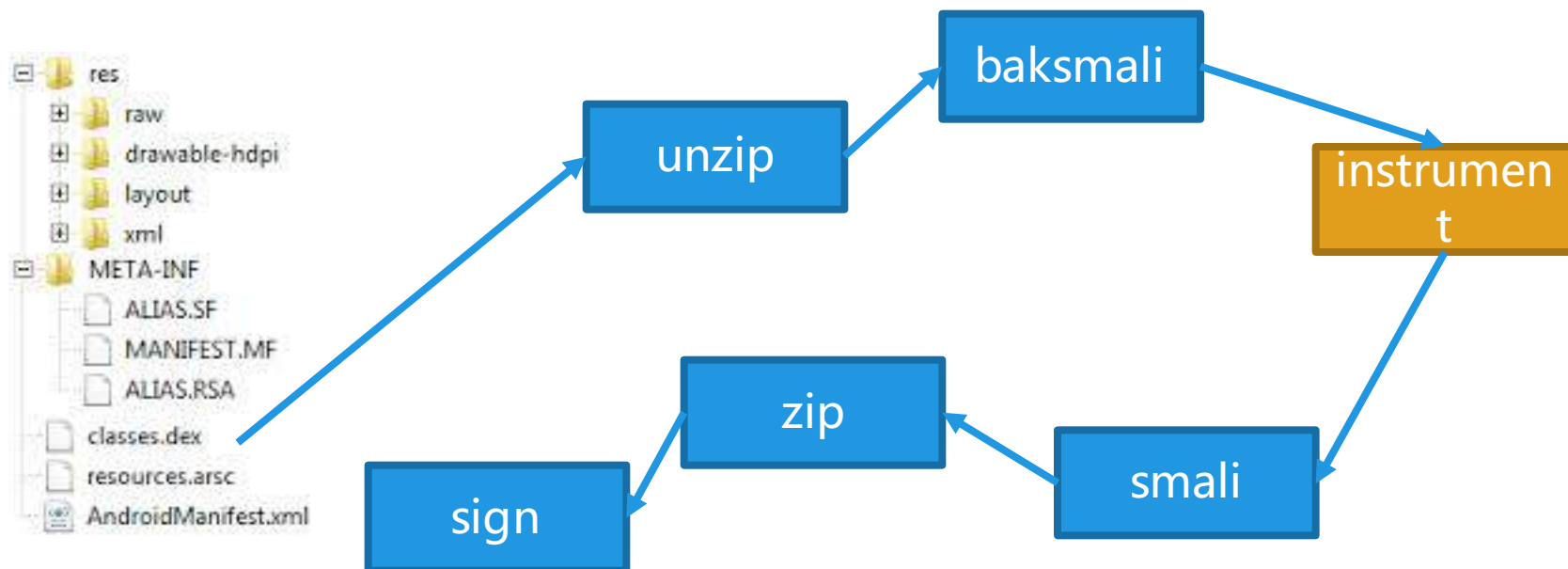
一键插桩

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



DEX插桩技术内幕

基于smali/baksmali



插桩举例

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

} Class information

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

} Static fields

```
# direct methods  
.method public constructor <init>()V  
  .registers 1  
  
  .prologue  
  .line 5  
  invoke-direct {p0}, Ljava/lang/Object;-><init>()V  
  
  return-void  
.end method
```

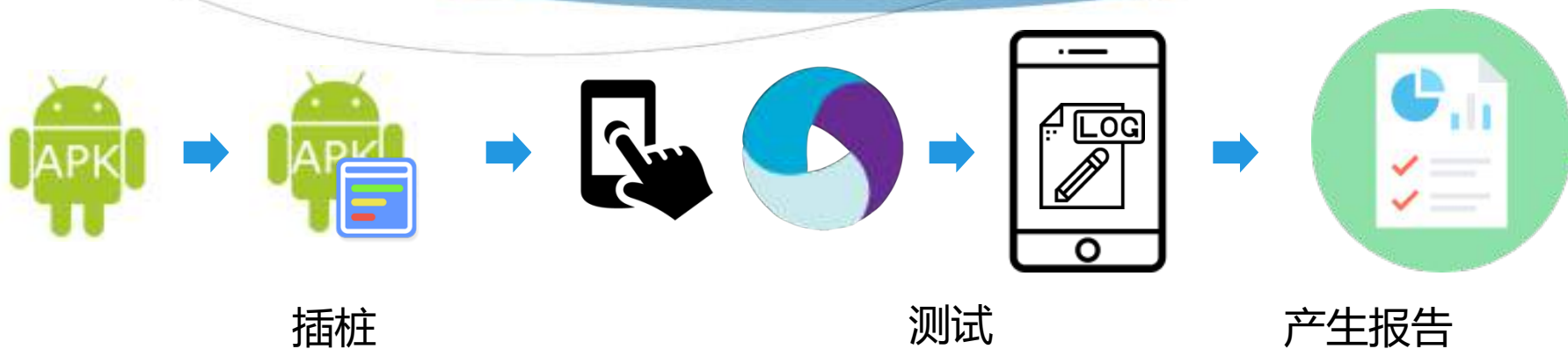
} Methods
Direct
Virtual

Appetizer有**近百条**插桩规则，插桩内容从简单的计时，到复杂的配对追踪等

插桩举例

```
@Override
public void onCreate(Bundle savedInstanceState) {
+   Appetizer.onStart(this, savedInstanceState);
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Toast.makeText(getApplicationContext(),"2. onCreate()", Toast.LENGTH_SHORT).show();
+   Appetizer.onEnd(this);
}
```


质量监控流程



支持的APP开发技术



混淆
加固



插桩

测试

产生报告



React Native

WEEX



Kotlin



热补丁：Tinker, AndFix



RxJava

插桩包测量内容



- Java层全线程异常
- 主线程卡顿 (事件、回调)
- 界面切换 (Activity/Fragment)
- 完整HTTP请求响应
(HttpURLConnection, okhttp 2/3, Apache HTTP Client)

- CPU/内存使用率
- 测试的界面覆盖率
- 网络流量
- 前后台时间

插桩方式



Jenkins



Python 命令行

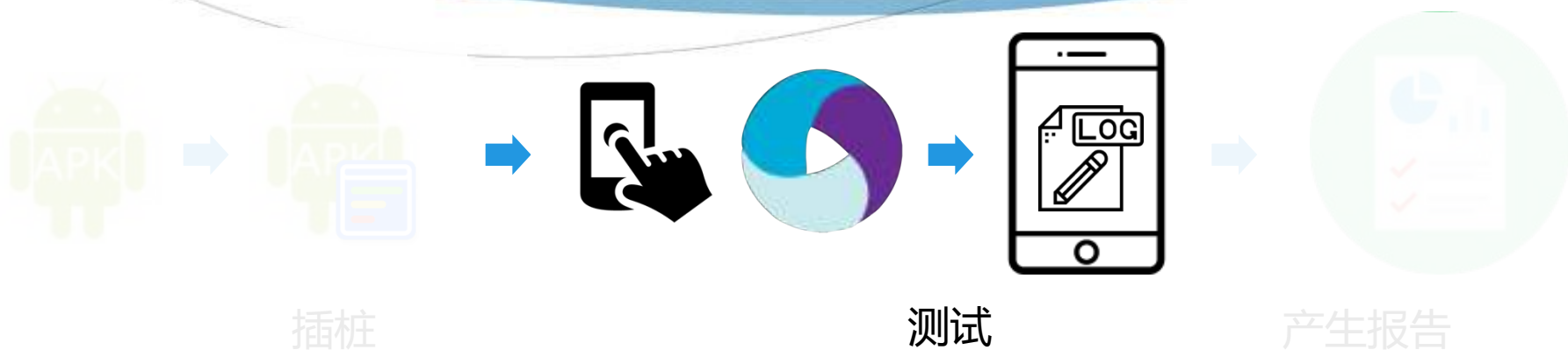


客户端图形化
界面



Gradle

测试阶段



- 原生系统，不需要Root，不需要特殊设备配置
- 手工测试
- UI自动化：Appium, Calabash, Robotium
- 自动探索：Monkey, AppCrawler
- Log数据存在设备本地，约为1k~20k/分钟

报告与分析



免费版本

- 客户端查看最近报告
- 导出问题 (csv, html, pdf, json)
 - Exceptions
 - HTTP错误
 - HTTP性能问题
 - 主线程卡顿

收费版本

- 查看完整历史报告
- 下载完整数据导入MongoDB/ELK做深度分析
- 按照每个插装包收取费用

报告 - 测试统计信息

质量监控 **beta**

API



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

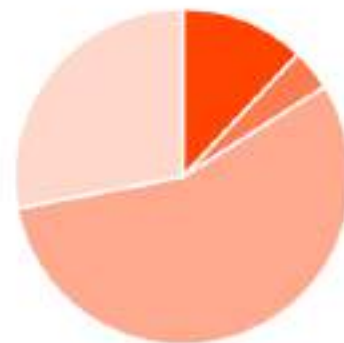
平均资源下载速度

593 B/s



问题分布

25 个问题



报告 – 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)

报告 – CPU/内存/堆栈



报告 - 启动切换缓慢



耗时的生命周期函数

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

耗时: 505 ms

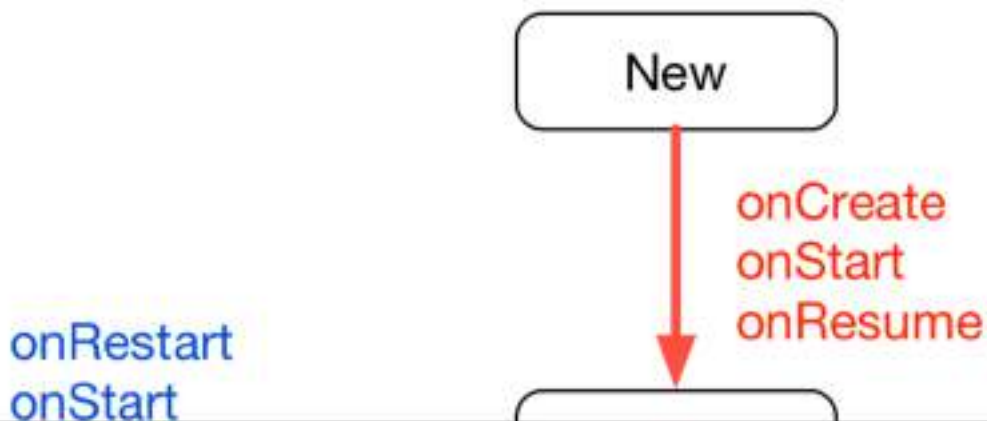
建议值: 200 ms

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

启动时间: APP启动时间包含onCreate, onStart和onResume三个步骤。如果耗时较长,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)。建议将这个过程的响应时间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), onStart(100ms), onResume(80ms)。



报告 – HTTP请求错误

Error Status Code

URL: <http://bilibili-service.daoapp.io/video/15102341?quality=4&type=mp4>

请求时间: 1318ms (total) = 1312ms (latency) + 6ms (transmission)

HTTP方法: GET

API类型: okhttp

返回码: 404

响应内容: text/html

响应内容长度: 空

缓存控制域: {"max-age": "360", "public": true}

原始的HTTP Request Header

```
{
  "User-Agent": "OhMyBiliBili Android Client/2.1 (100332338@qq.com)"
}
```

原始的HTTP Response Header:

```
{
  "Accept-Ranges": "bytes",
  "Access-Control-Allow-Origin": "*",
  "Access-Control-Expose-Headers": "X-Log, X-Reqid",
  "Access-Control-Max-Age": "2592000",
  "Age": "33603",
  "Cache-Control": "public, max-age=3600",
  "Content-Disposition": "inline; filename=\"503_appError.html\"",
  "Content-Transfer-Encoding": "binary",
  "Date": "2017-08-14T08:00:00.000Z",
  "Expires": "2017-08-14T08:00:00.000Z",
  "Server": "nginx/1.10.3",
  "Vary": "Accept-Encoding",
  "X-Log": "100332338",
  "X-Reqid": "100332338"
}
```

报告 – HTTP首包太慢

High latency 一键复现...

URL: http://i0.hdslb.com/bfs/bangumi/17a1ee348544dbd677e05cbeaa85c52c11e1a06.jpg

请求时间: 1319ms (total) - **846ms (latency)** ⚠️ 473ms (transmission)

HTTP方法: GET

API类型: **uriconnection** ⚠️

返回码: **308**

响应内容: image/jpeg

响应内容长度: 308 kB

缓存控制域: {"max-age":"31536000"}

原始的HTTP Request Header

```
{
  "Accept-Encoding": "identity",
  "User-Agent": "Dalvik/2.1.0 (Linux; U; Android 6.0; Nexus 5 Build/MRA58K)"
}
```

原始的HTTP Response Header:

```
{
  "Age": "3099168",
  "Cache-Control": "max-age=31536000",
  "Connection": "keep-alive",
  "Content-Length": "307950",
  "Content-Type": "image/jpeg",
  "Date": "Sun, 12 Feb 2017 11:50:18 GMT",
  "EagleId": "dec0ba0414899993864041790e",
  "Etag": "17a1ee348544dbd677e05cbeaa85c52c11e1a06"
```

报告 – HTTP下载过慢

Slow transmission 一键复现...

URL: http://0.hdsib.com/bfs/archive/070ccb6de80911e13fdd54b9b589791db6f74dd.jpg

请求时间: 258ms (total) = 47ms (latency) + **211ms (transmission)** ⚠

HTTP方法: **GET**

API类型: **urlconnection** ⚠

返回码: **200**

响应内容: image/jpeg

响应内容长度: 42.5 kB

缓存控制域: ("max-age":"31536000")

原始的HTTP Request Header

```
{
  "Accept-Encoding": "identity",
  "User-Agent": "Dalvik/2.1.0 (Linux; U; Android 6.0; Nexus 5 Build/MRA58K)"
}
```

原始的HTTP Response Header:

```
{
  "Age": "333813",
  "Cache-Control": "max-age=31536000",
  "Connection": "keep-alive",
  "Content-Length": "42476",
  "Content-Type": "image/jpeg",
  "Date": "Thu, 16 Mar 2017 11:58:31 GMT",
  "EagleId": "dec0ba0514899993247622376e",
}
```

报告 - 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.AppCompatActivity.setContentView(AppCompatActivity.java:288)
- 1: android.support.v7.app.AppCompatActivity.onCreate(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)

插桩的性能影响

- ◆ 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
- ◆ 可定制化减少 log，开关测量项目（未来高级功能）

我们的用户



有效测试时间

23.9 小时

HTTP请求	2438
OK	913
HTTP错误	259
已缓存	
高延时	1266
可缓存	36

HomeBangumiBobySection.java line 75 crash by Invalid index 0, size is 0 #37

Closed

Azard opened this issue on Feb 27 · 1 comment

Appetizer

+

AppCrawler

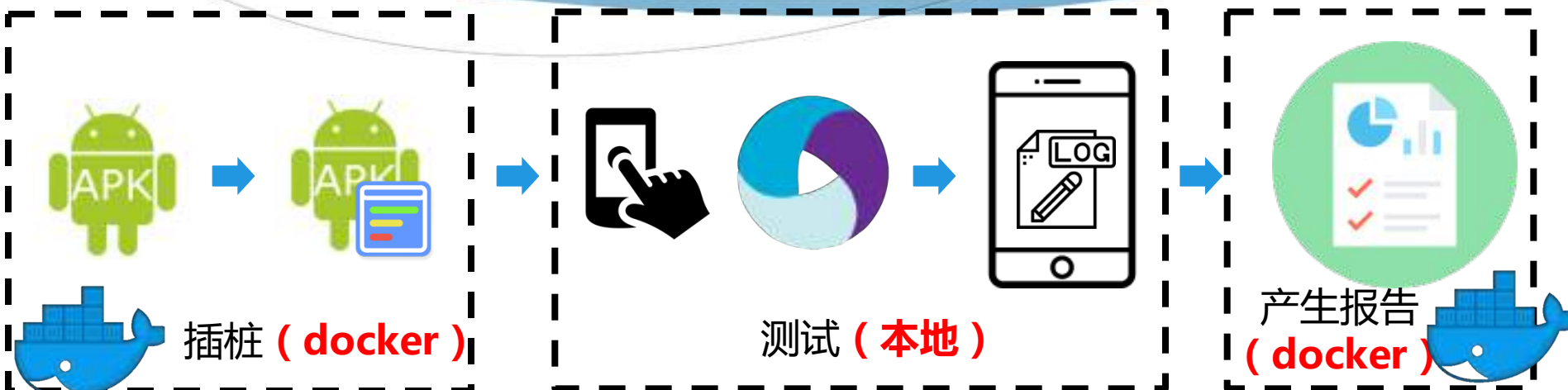
Azard commented on Feb 27 • edited



我们的用户

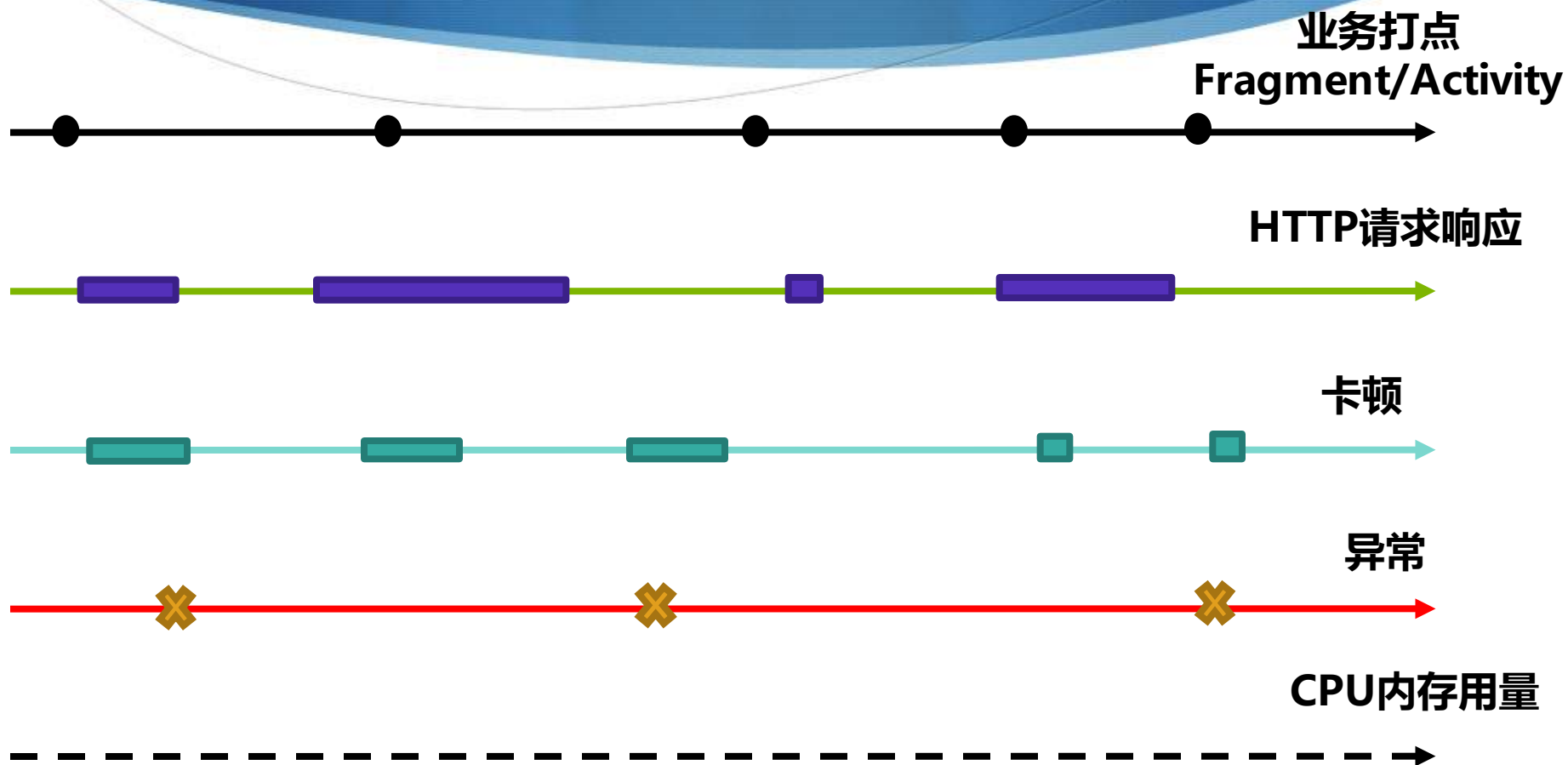


私有化部署

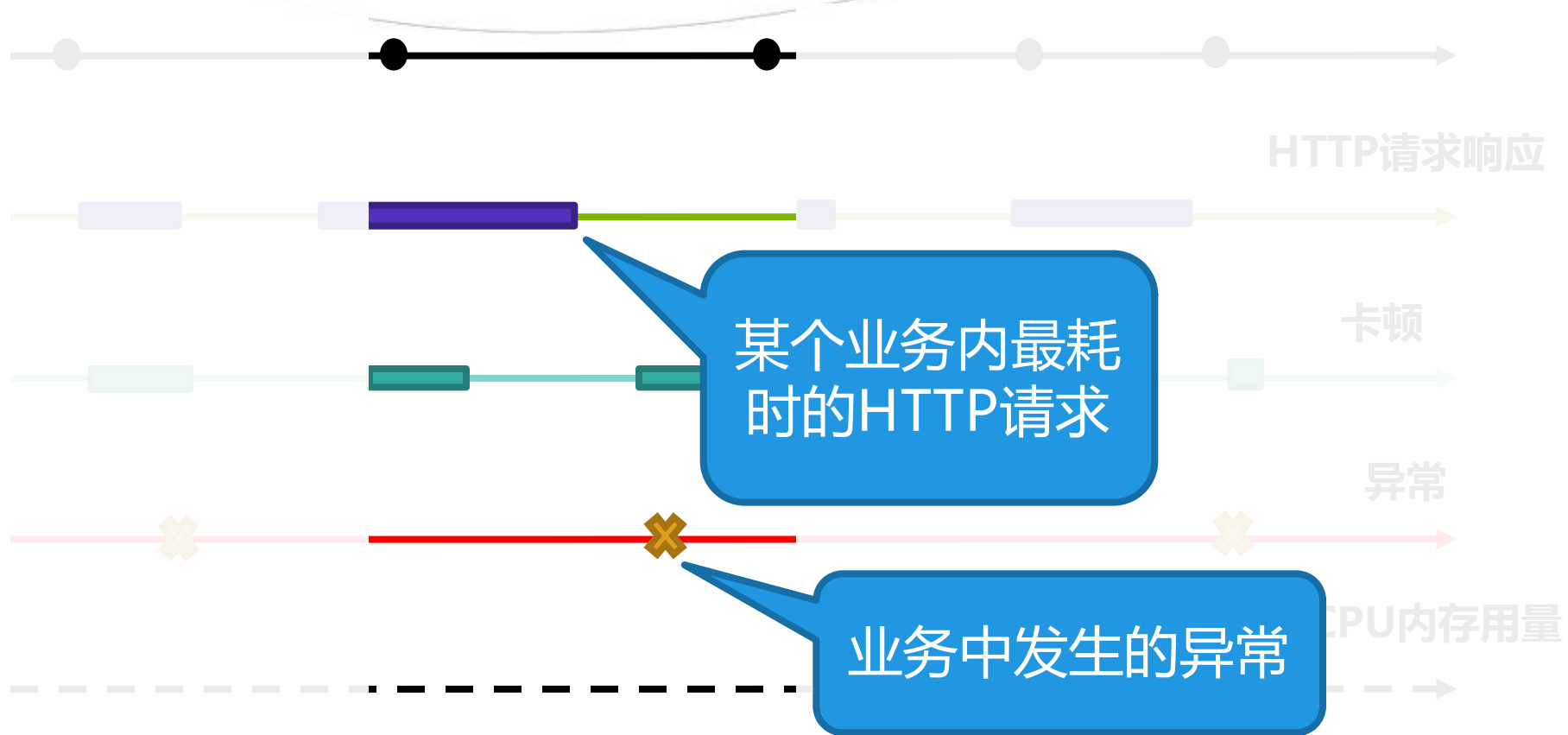


输出APP执行数据 (json/mongodb/ELK)
→ 深度分析

APP执行数据

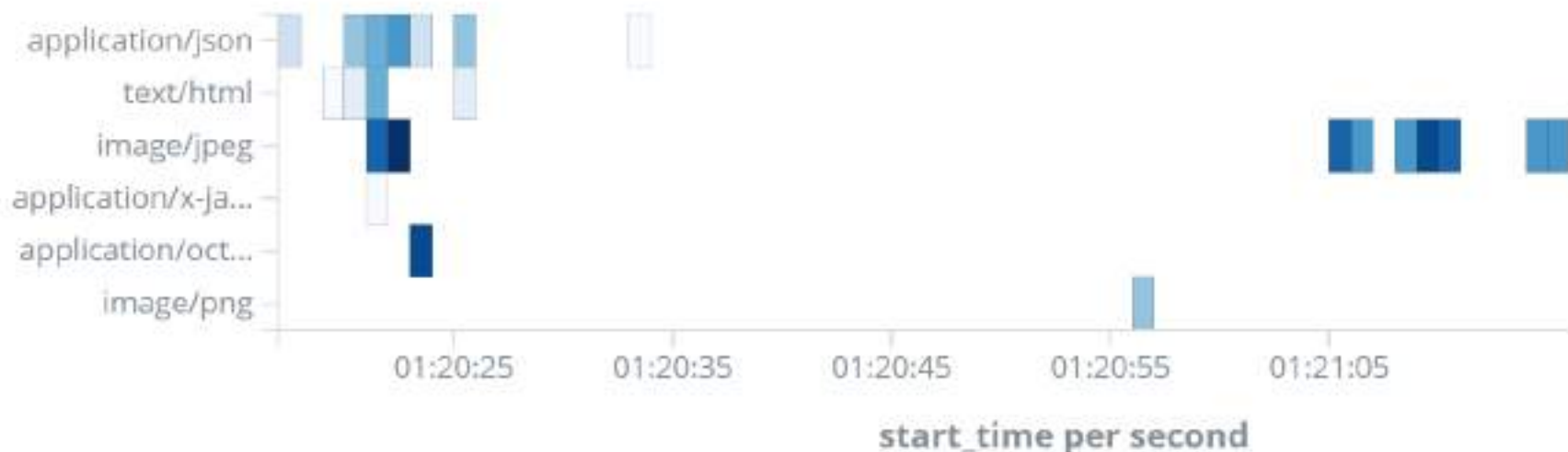


深度分析：搜索，排序

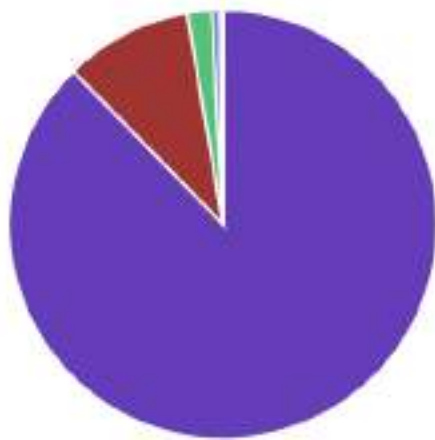


深度分析：流量分析

http_traffic_heatmap_MIME_TYPE



http_traffic_pie_MIME_TYPE



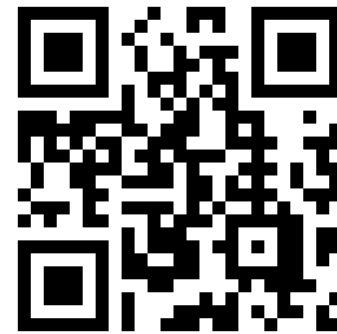
- image/jpeg
- application/octet-stre...
- application/json
- text/html
- image/png

资源

- ◆ Appetizer 官网及客户端下载：<https://appetizer.io>
- ◆ 插桩工具Python命令行版：
<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、商务合作：theappetizerio@gmail.com

未来计划

- ◆ Appetizer BLSDK (aar格式) 更精确的业务埋点
 - ◆ 目前到Fragment层面的业务精细度
- ◆ HTTP 更精细化 : DNS, TCP连接时间
- ◆ WebView
- ◆ 导入其他 SDK 数据 , 例如 Leak Canary , JaCoCo



基于 DEX 插桩的 自动应用质量监控

夏鸣远

AppetizerIO

theappetizerio@gmail.com

