HUAWEI

< HDC.Together >

HUAWEI DEVELOPER CONFERENCE 2021

HarmonyOS DFX框架
--卓越产品的基石

1 DFX介绍

2 HarmonyOS DFX框架与能力全景

3 记录:日志、事件、跟踪

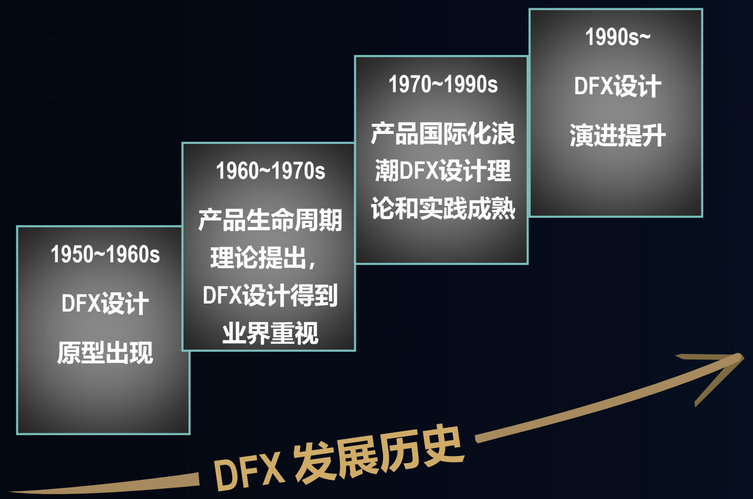
4 故障检测

5 观测剖析：信息导出、分布式调试、分布式调优

6 展望与演进

什么是DFX？ -Design For X

产品的非功能性设计的总称，X指产品的某个特性或者产品生命周期的某个阶段



DFX- Design For eXcellence，即面向卓越的设计

DFR：可靠性设计，Design For Reliability

DFT：可测试性设计，Design For Testability

DFM：可制造性性设计，Design For Manufacturability

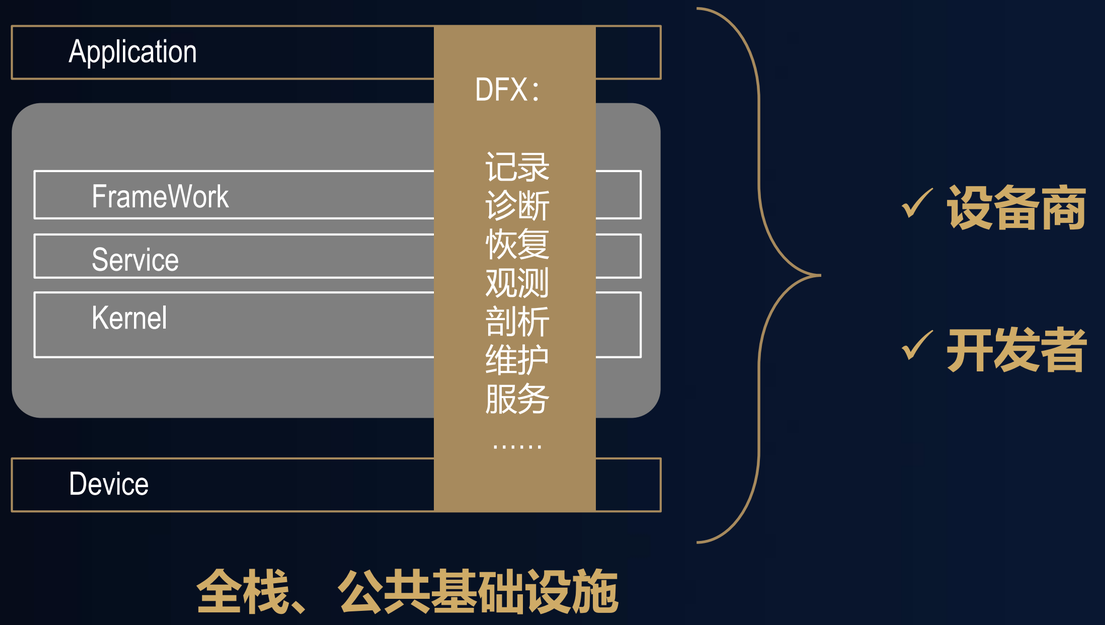
DFS：可服务性设计，Design For Serviceability

DFLC：生命周期设计，Design For LifeCycle

.....

什么是操作系统DFX?

操作系统提供的DFX是公共基础设施，用来使能开发者和设备商设计出卓越的产品

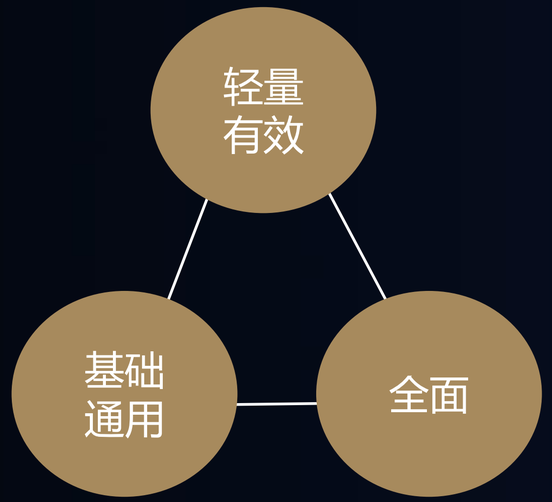


全栈、公共基础设施

HarmonyOS对DFX能力的要求

轻量有效：

- 系统资源开销少（RAM/ROM/CPU...）
- 易用
- 精准有效（检测，定位，分析，度量）

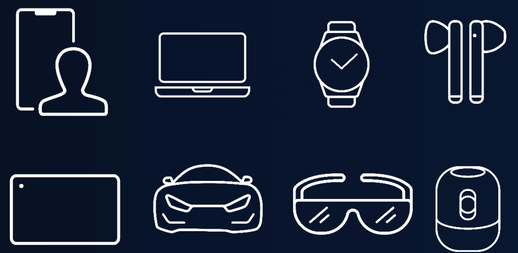


基础通用：

- 关键、基础
- 通用、易扩展

全面：

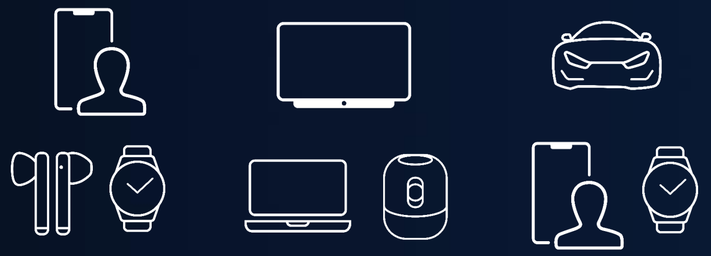
- 全面服务应用和设备品类
- 全面服务开发者、设备商
- 全面覆盖产品全生命周期



资源差异：
RAM：128K-
ROM：2M-

多语言全栈支持

可大可小、灵活部署



分布式日志
分布式跟踪
分布式调试调优...

HarmonyOS DFX框架与能力

记录：日志、事件、跟踪

凡走过 必留下痕迹

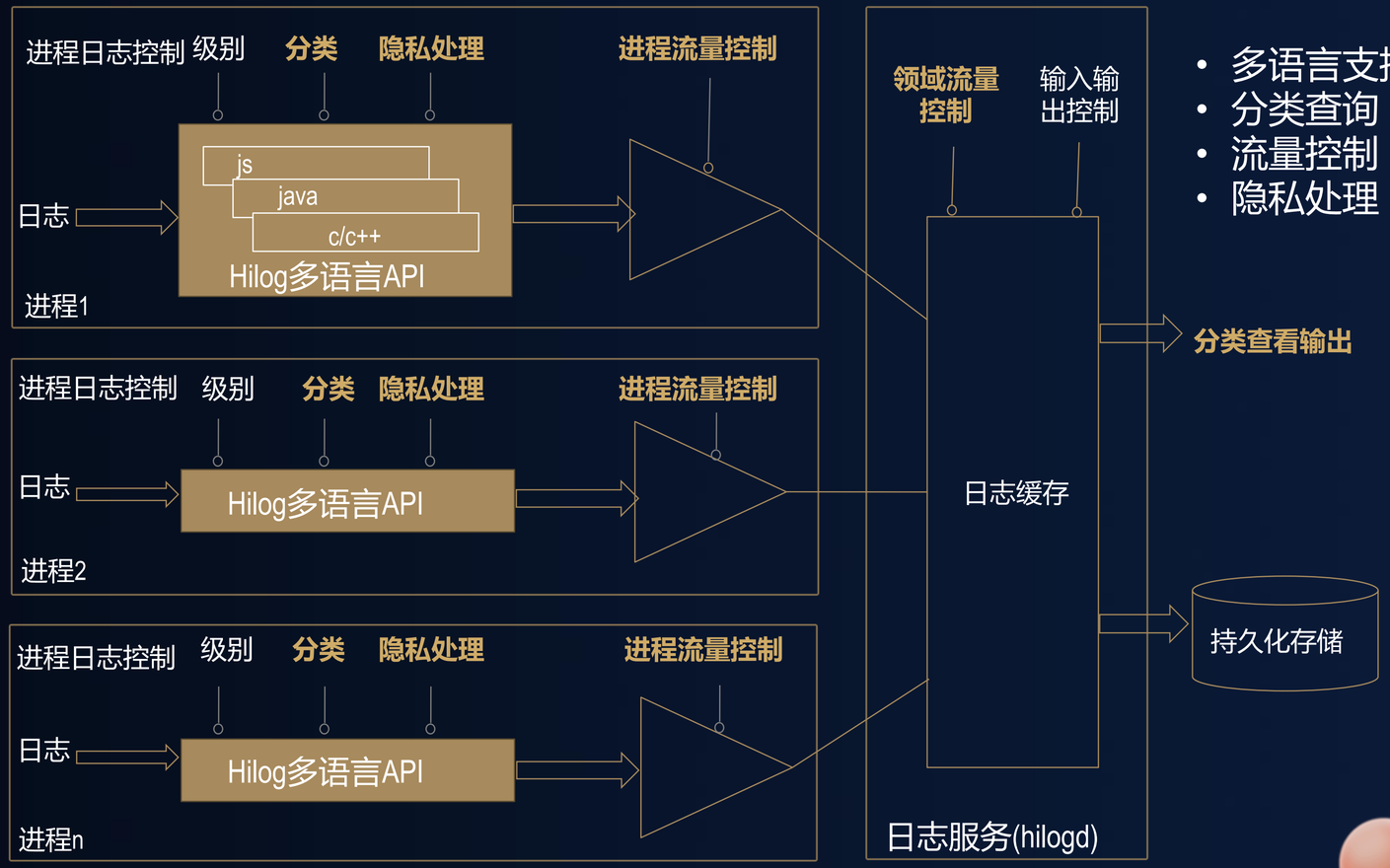
日志

- 日志就像车辆保险，平时不愿意为保险付钱，出问题的时候又都想有保险可用 *-- 来自网络*
- 记录日志的黄金法则是不要让你的日志无必要地冲掉别人的日志，正如你希望别人也这样 *-- 谷歌的日志打印要求*

日志查询不便

日志流量太大

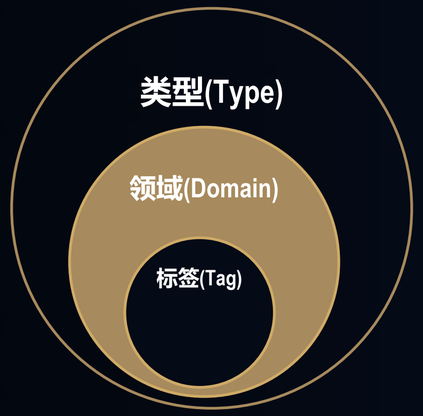
隐私安全问题



- 多语言支持
- 分类查询
- 流量控制
- 隐私处理

日志：分类查看

- 日志按类型、**领域**、标签分类：



- 类型(Type)：core, app, init, kmsg
- 领域(Domain): camera, bluetooth, …
- 标签(Tag): xxx, yyy, zzz, …

日志领域(Domain)：跨软件栈层次的业务垂域

- 查看某个领域日志：

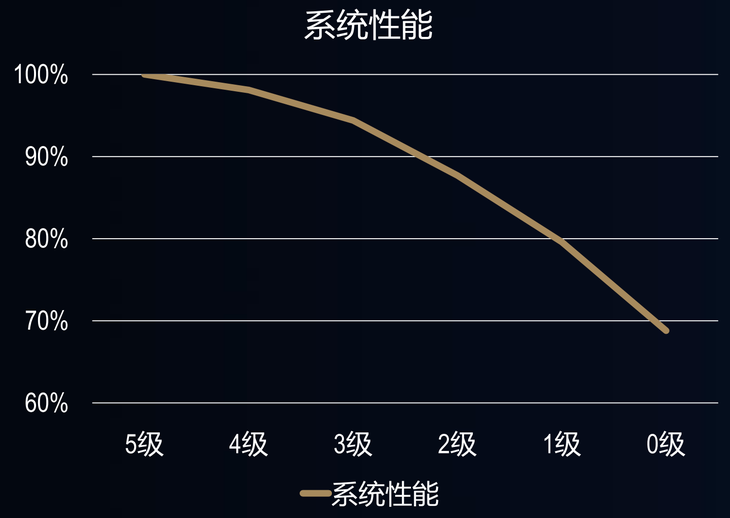
```
Phoenix:/ # hilog -D 0xD001800
09-27 20:02:58.422  8683  8683 E 01800/SAM: Service SamgrService didn't start. Returning nullptr
09-27 20:02:58.422  8683  8683 I 01800/SA: Waiting for samgr...
09-27 20:02:58.422  8683  8683 E 01800/SA: CheckSystemAbilityManagerReady:Wait for samgr time out (10s)
09-27 20:02:58.422  8683  8683 W 01800/SA: CheckSystemAbilityManagerReady failed!
09-27 20:02:58.422  8684  8684 E 01800/SAM: Service SamgrService didn't start. Returning nullptr
09-27 20:02:58.422  8684  8684 I 01800/SA: Waiting for samgr...
09-27 20:02:58.422  8684  8684 E 01800/SA: CheckSystemAbilityManagerReady:Wait for samgr time out (10s)
09-27 20:02:58.422  8683  8683 I 01800/SA: SystemAbility:9526 destroy!
09-27 20:02:58.422  8684  8684 W 01800/SA: CheckSystemAbilityManagerReady failed!
09-27 20:02:58.422  9028  9028 I 01800/SAM: Waiting for service SamgrService...
09-27 20:02:58.471  9552  9552 I 01800/SA: safwk---->main entry
```

- 丰富的日志分级分类查询命令：

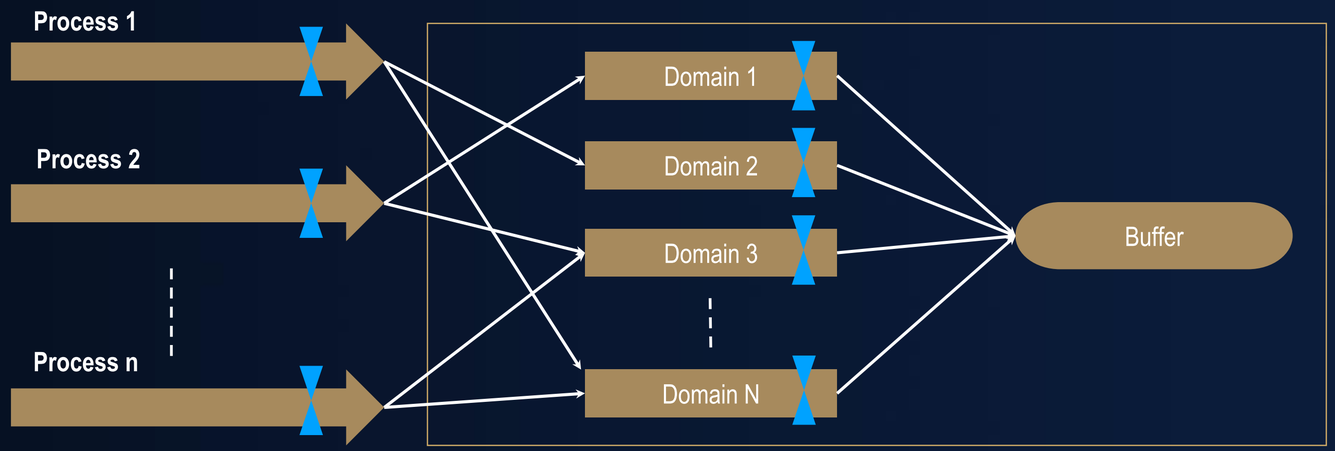
功能	hilog命令
查看日志: HOS系统	hilog -t core
查看日志: HOS应用	hilog -t app
清空HOS日志缓存	hilog -r
查看某个tag的日志	hilog -T tag
查看多个tag的日志	hilog -T tag1,tag2
不查看某个tag的日志	hilog -T ^tag
不查看多个tag的日志	hilog -T ^tag1,tag2
查看某个domain的日志	hilog -D 0xd0xxxxx
查看多个domain的日志	hilog -D 0xd0xxxxx,0xd0xxxxx
不查看某个domain的日志	hilog -D ^0xd0xxxxx
不查看多个domain的日志	hilog -D ^0xd0xxxxx,0xd0xxxxx
查看某个级别的日志	hilog -L D
查看多个级别的日志	hilog -L D,I
不查看某个级别的日志	hilog -L ^D
不查看多个级别的日志	hilog -L ^D,I

日志：流量管控

- 日志量对系统性能造成显著影响：



- 流控机制可以有效地识别出滥打日志的领域



- Debug模式 vs Release模式

Debug模式输出：

05-26 11:01:06.870 1051 1051 W 02d18/test: Test burst logs, index: 1000.
05-26 11:01:06.870 1051 1051 W 02d18/test: 100 line(s) **maybe** dropped **in release**!

Release模式输出：

05-26 11:01:06.870 1051 1051 W 02d18/test: 100 line(s) **were** dropped!

日志：隐私管控

- 采集日志最小化，仅为提供必须服务:

“为保障你正常使用我们的服务，我们会收集你的设备型号、操作系统、唯一设备标识符、登录IP地址、软件版本号、接入网络的方式和类型、设备极速器、**操作日志**等信息，这类信息为提供服务必须的基础信息。”

- HiLog变量打印控制:

打印代码：

```
HILOG_WARN(LOG_CORE, "%s failed to visit %{private}s, reason: %{public}d.", username, url, errno);
```

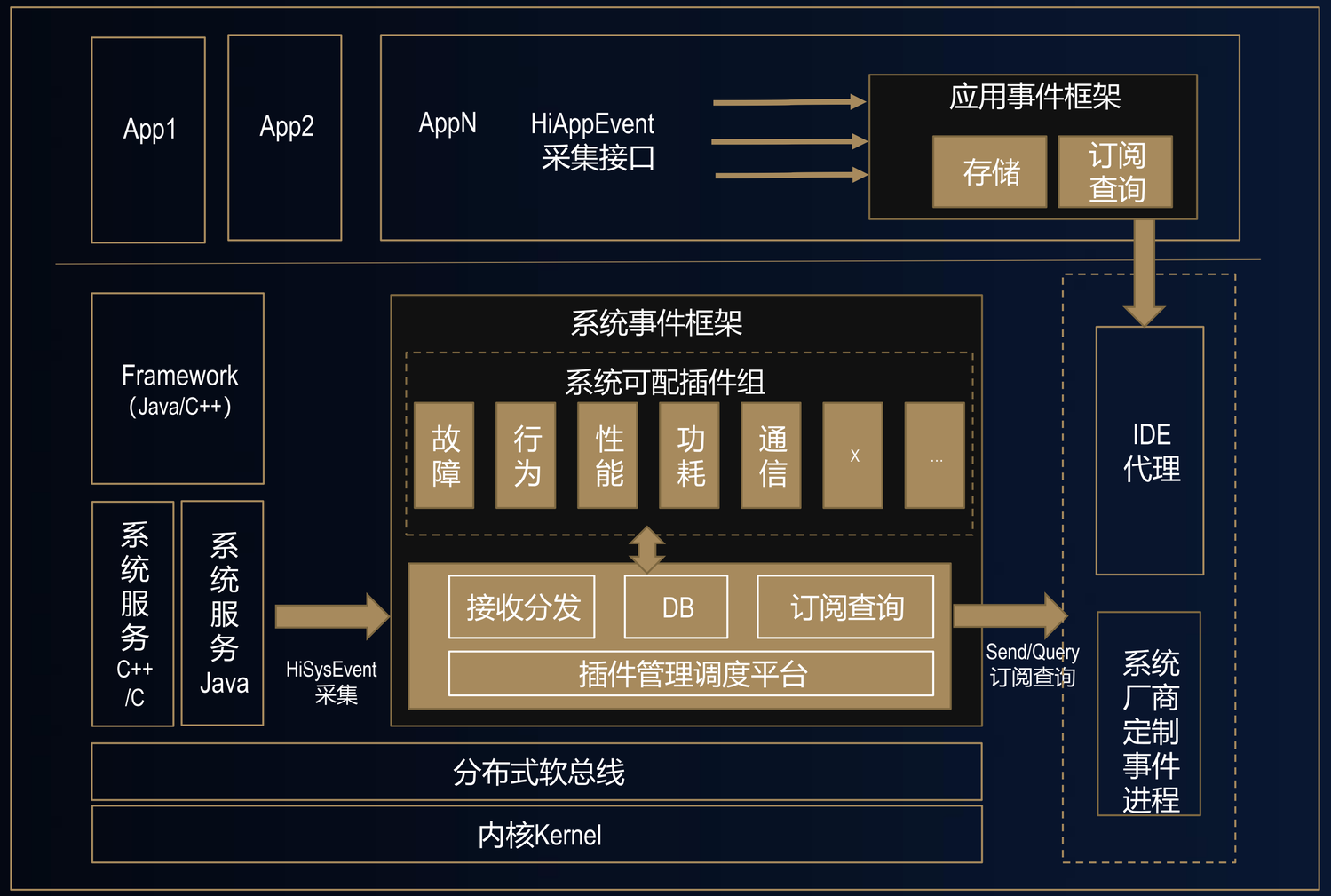
Debug模式输出：

```
05-26 11:01:06.870 1051 1051 W 02d18/test: XiaoMing failed to visit https://x.y.z, reason: 402.
```

Release模式输出：

```
05-26 11:01:06.870 1051 1051 W 02d18/test: <private> failed to visit <private>, reason: 402.
```

事件框架



- 完备的事件打点接口
- 方便的事件查看工具
- 轻量灵活的平台部署

实例：事件定义&打点

- 事件定义包含领域、事件名、基础信息、事件参数:

```
domain: RELIABILITY

APP_CRASH:
  __BASE: {type: FAULT, level: CRITICAL, tag: reliability,
desc: application crash}
  SUB_EVENT_TYPE: {type: STRING, desc: sub event type}
  EVENT_TAG: {type: STRING, desc: event tag}
  EVENT_TIME: {type: STRING, desc: event time}
  PACKAGE: {type: STRING, desc: application package name}
  APPVERSION: {type: STRING, desc: application version}
  PNAME: {type: STRING, desc: application process name}
  FWK_TYPE: {type: STRING, desc: framework type}
  APP_TYPE: {type: STRING, desc: application type}
  REASON: {type: STRING, desc: fault reason}
  LIFETIME: {type: UINT32, desc: time from startup to crash}
  DIAG_INFO: {type: STRING, desc: diagnose info}
  DETAILED_LOG: {type: STRING, desc: detailed log}
  FG: {type: INT8, desc: foreground}
  FINGERPRINT: {type: STRING, desc: fingerprint}
```

- 按照事件定义记录事件:

```
#include <string>
#include "appcrashinfo.h" // 开发者自定义的struct AppCrashInfo结构
#include "hisysevent.h"   // HarmonyOS提供的系统事件打点API头文件

using std::string;

void AppCrashDeterctor(const struct AppCrashInfo& appCrashInfo)
{
    string domain = "RELIABILITY";     // 对应yaml中系统事件所属领域的定义
    string eventName = "APP_CRASH";    // 对应yaml中系统事件名的定义
    HiSysEvent::Write(domain, eventName, HiSysEvent::EventType::FAULT, // 对应yaml中系统事件类型的定义
        // 以下对应yaml中APP_CRASH事件的参数定义
        "SUB_EVENT_TYPE", appCrashInfo.subEventType,
        "EVENT_TAG", appCrashInfo.tag,
        "PACKAGE", appCrashInfo.package,
        "APPVERSION", appCrashInfo.version,
        "PNAME", appCrashInfo.pname,
        "FWK_TYPE", appCrashInfo.fwkType,
        "APP_TYPE", appCrashInfo.appType,
        "REASON", appCrashInfo.reason,
        "LIFETIME", appCrashInfo.lifeTime,
        "DIAG_INFO", appCrashInfo.info,
        "DETAILED_LOG", appCrashInfo.log,
        "FG", appCrashInfo.fg,
        "FINGERPRINT", appCrashInfo.fingerPrint
    );
}
```

实例：事件订阅&读取

- 可以方便地通过订阅接口实时订阅系统中发生的事件:

```
// 侦听回调接口：侦听的事件响应
void HiSysEventToolListener::OnHandle(const std::string& domain, const std::string&
eventName,
    const int eventType, const std::string& eventDetail)
{
    std::cout << eventDetail << std::endl;
}

// 设置侦听回调
auto listener = std::make_shared<HiSysEventToolListener>();

// 设置事件的查询条件
std::vector<struct ListenerRule> rules;

// 侦听所属的域为RELIABILITY，事件名为APP_CRASH的系统事件
struct ListenerRule rule1 = { 1, "RELIABILITY", "APP_CRASH" };
rules.emplace_back(rule1);

// 侦听所属的域为HIVIEWDFX的系统事件
struct ListenerRule rule2 = { 1, "HIVIEWDFX", "" };
rules.emplace_back(rule2);

if (HiSysEventManager::AddEventListener(listener, sysRules)) {
    return true;
}
```

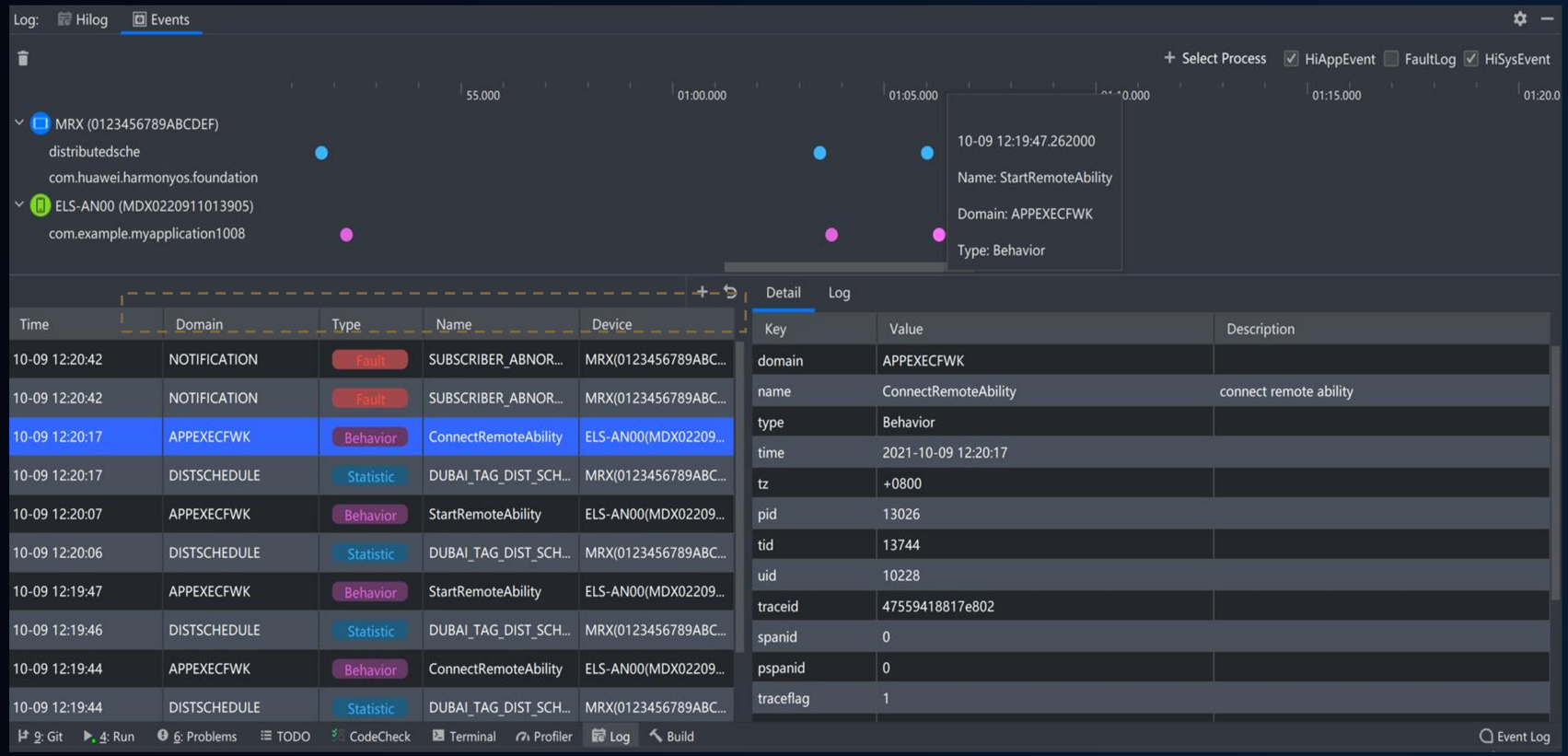
- 也可以方便地通过查询接口查询系统中发生的历史事件:

```
// 回调接口：处理返回的事件结果集
void HiSysEventToolQuery::OnQuery(const ::std::vector<std::string>& sysEvent,
    const ::std::vector<int64_t>& seq)
{
    for_each(sysEvent.cbegin(), sysEvent.cend(), [](const std::string &tmp) {
        std::cout << tmp << std::endl;
    });
}

// 回调处理
auto queryCallBack = std::make_shared<HiSysEventToolQuery>();
// 设置查询事件的起始时间、结束时间以及返回最大的条数
struct QueryArg args(clientCmdArg.beginTime, clientCmdArg.endTime,
clientCmdArg.maxEvents);
// 设置事件的查询条件
std::vector<struct QueryRule> rules;

// 查询事件所属的域为RELIABILITY，事件名为APP_CRASH
struct QueryRule rule1 = { 0, "RELIABILITY", {"APP_CRASH"} };
rules.emplace_back(rule1);

// 查询事件所属的域为HIVIEWDFX
struct QueryRule rule2 = { 0, "HIVIEWDFX", {} };
rules.emplace_back(rule2);

if (HiSysEventManager::QueryHiSysEvent(args, rules, queryCallBack)) {
    return true;
}
```

- 接口可以做二次开发

实例：事件查看工具

- 通过IDE查看系统事件以及应用事件

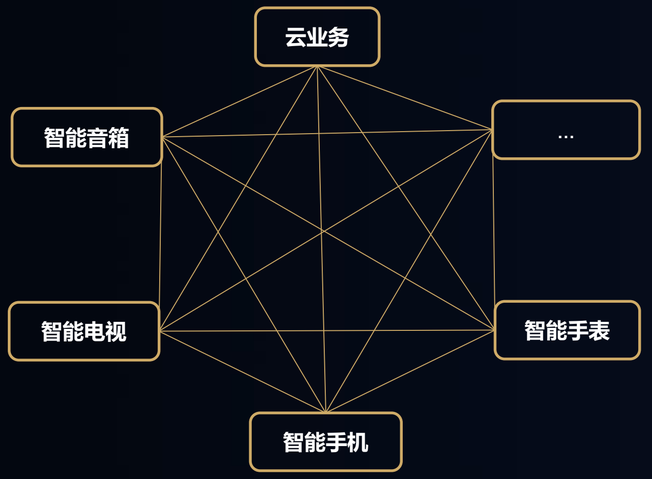


- 通过命令行查看系统事件

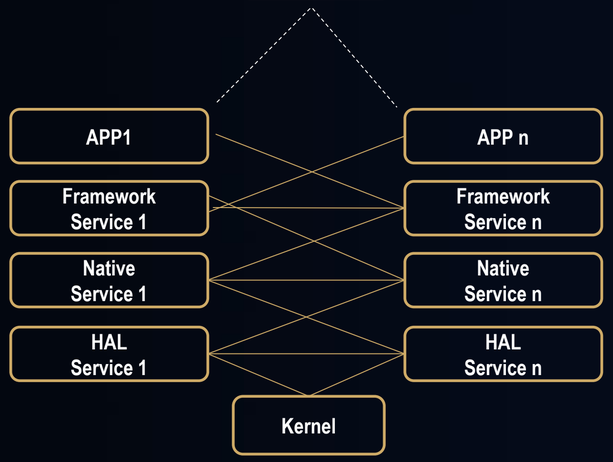
```
HWNOH:/ # hisysevent
hisysevent [-r [-d] | -l [-s <time> -e <time> -m <count>]]
-r    get real hisysevent log.
-r -d set debug mode, both options must appear at the same time.
-l -s <begin time> -e <end time> -m <max hisysevent count>
      get history hisysevent log, begin time should not be earlier than end time.
```

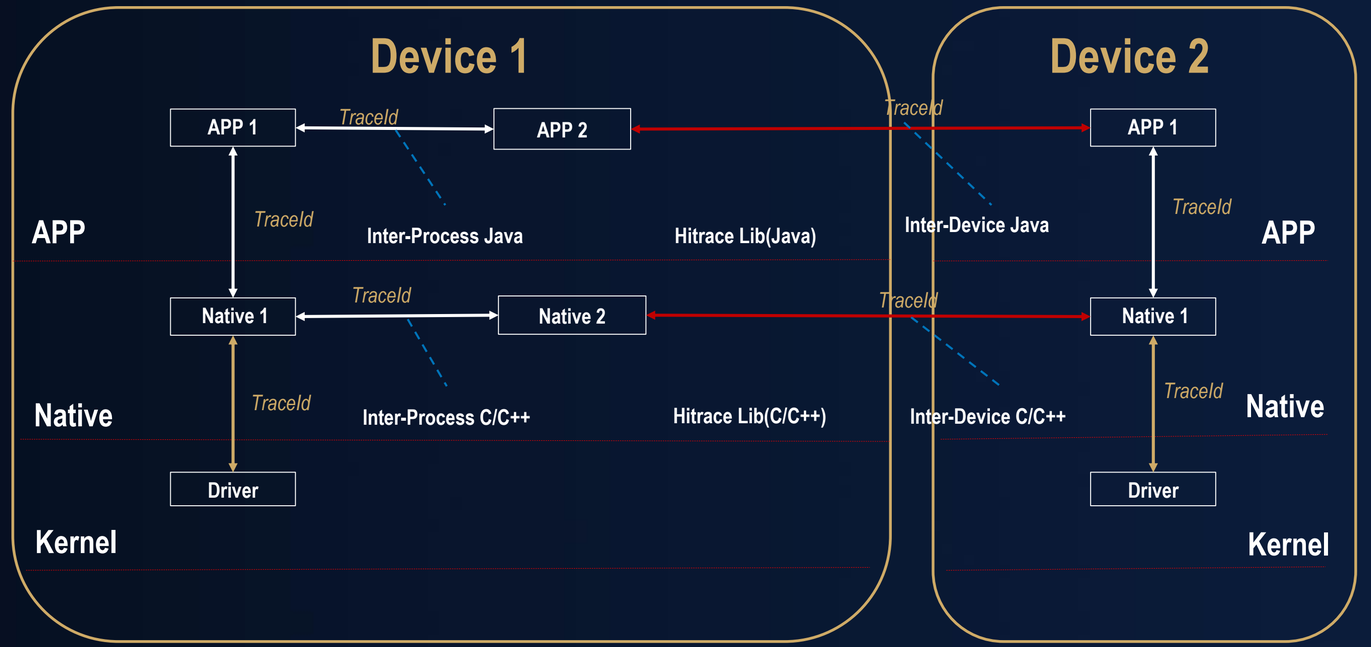
跟踪

设备间业务流程



设备内业务流程





轻量跟踪机制，微秒级的开销

应用实例：业务链跟踪

- 在业务开始和结束的地方分别调用begin和end接口:

HAP1中点击button，触发启动HAP2：

```
Button idButton = (Button) findComponentById(ResourceTable.Id_button);
idButton.setClickedListener(new Component.ClickedListener() {
    @Override
    public void onClick(Component component) {
        HiTraceId traceId = HiTrace.begin("sync_id",
HiTrace.HITRACE_FLAG_DEFAULT);

        String id = "id_6BC58E27AC02FAF70E35EE4";
        HiLog.info(label, "in hap1, id=%{public}s", id);
        HiAppEvent.write("sync_id", HiAppEvent.EventType.BEHAVIOR, "name", "hap1",
"id", id);

        Operation operation = new
Intent.OperationBuilder().withBundleName("com.example.hap2")
            .withAbilityName("com.example.hap2.MainAbility").build();
        Intent intent = new Intent().setParam("sync_id", id);
        intent.setOperation(operation);
        startAbilityForResult(intent, 1);

        HiTrace.end(traceId);
    }
});
```

HAP2被启动，获取同步数据sync_id：

```
public void onStart(Intent intent) {
    super.onStart(intent);
    super.setUIContent(ResourceTable.Layout_ability_peer_main);

    String id = intent.getStringParam("sync_id");
    if (id != null) {
        HiLog.info(label, "in hap2, id=%{public}s", id);
        HiAppEvent.write("sync_id", HiAppEvent.EventType.BEHAVIOR,
            "name", "hap2", "id", id);
    }
}
```

HAP1的日志打印和事件打点:

```
09-24 15:50:25.735 29266 29266 I 02D03/HiTraceC: [1bf7eb8341b37e5, 0, 0] HiTraceBegin name:sync_id
flags:0.
09-24 15:50:25.736 29266 29266 I 00000/MAIN_TAG: [1bf7eb8341b37e5, 0, 0] in hap1,
id=id_6BC58E27AC02FAF70E35EE4
09-24 15:50:25.737 29266 29628 D 02D07/HiAppEvent: [1bf7eb8341b37e5, 1ad3eeb, 0] write eventName is
sync_id, eventType is 4.
09-24 15:50:25.737 29266 29628 D 02D07/HiAppEvent_write: [1bf7eb8341b37e5, 1ad3eeb, 0] WriteEvent
eventInfo={"name_":"sync_id","type_":4,"time_":1632469825736,"tz_":"+0800","pid_":29266,"tid_":29628,
"traceid_":bf7eb8341b37e5,"spanid_":0,"pspanid_":0,"trace_flag_":1,"name":"hap1","id":"id_6BC58E27AC0
2FAF70E35EE4"}
09-24 15:50:25.737 29266 29266 I 01320/AafwkKeyBound: [1bf7eb8341b37e5, 0, 0] [Ability][start ability
for result][START]: element: null/com.example.hap2/com.example.hap2.MainAbility
09-24 15:50:25.737 29266 29266 I 01100/AppKit: [1bf7eb8341b37e5, 0, 0] ContextDeal::startAbility
called
09-24 15:50:25.738 29266 29266 D 01100/AbilityShell: [1bf7eb8341b37e5, 0, 0]
AbilityShellConverterUtils::convertToShellInfo Shell package: com.example.hap2, class:
com.example.hap2.MainAbilityShellActivity
09-24 15:50:25.750 29266 29266 I 01320/AafwkKeyBound: [1bf7eb8341b37e5, 0, 0] [Ability][start ability
for result][END]:
09-24 15:50:25.750 29266 29266 I 01320/AafwkKeyBound: [1bf7eb8341b37e5, 0, 0]
[AbilitySliceManager][start ability for result][END]:
09-24 15:50:25.750 29266 29266 E 02D03/HiTraceC: [1bf7eb8341b37e5, 0, 0] HiTraceEnd.
```

HAP2的日志打印和事件打点:

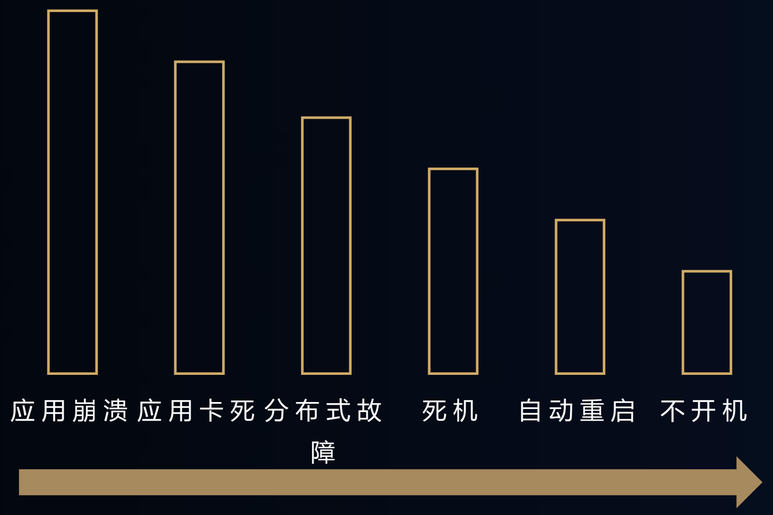
```
09-24 15:50:25.797 29374 29374 D 02D08/HISYSEVENT: [1bf7eb8341b37e5, 2533e53, 24cd7b2]
size=355,
sysevent={"domain_":"APPEXECFWK","name_":"ABILITY_START_RESULT","type_":1,"time_":1632469825
797,"tz_":"+0800","pid_":29374,"tid_":29374,"uid_":10221,"traceid_":"1bf7eb8341b37e5","spani
d_":"2533e53","pspanid_":"24cd7b2","trace_flag_":1,"ERROR_TYPE":0,"ABILITY_NAME":"com.exampl
e.hap2.MainAbility","ABILITY_TYPE":1,"PACKAGE_NAME":"com.example.hap2","START_TYPE":0}
09-24 15:50:25.799 29374 29374 D 01100/AbilityShell: [1bf7eb8341b37e5, 2533e53, 24cd7b2]
AbilityShellActivity::onStart called
09-24 15:50:25.807 29374 29374 I 00001/PEER_TAG: [1bf7eb8341b37e5, 2533e53, 24cd7b2] in
hap2, id=id_6BC58E27AC02FAF70E35EE4
09-24 15:50:25.808 29374 29632 D 02D07/HiAppEvent: [1bf7eb8341b37e5, f3c0b0, 2533e53] write
eventName is sync_id, eventType is 4.
09-24 15:50:25.808 29374 29632 D 02D07/HiAppEvent_write: [1bf7eb8341b37e5, f3c0b0, 2533e53]
WriteEvent
eventInfo={"name_":"sync_id","type_":4,"time_":1632469825807,"tz_":"+0800","pid_":29374,"tid
_":29632,"traceid_":1bf7eb8341b37e5,"spanid_":39009875,"pspanid_":38590386,"trace_flag_":1,"
name":"hap2","id":"id_6BC58E27AC02FAF70E35EE4"}
```

故障检测

提出正确的问题，往往等于解决了问题的大半

HarmonyOS 故障检测器分类

故障分布

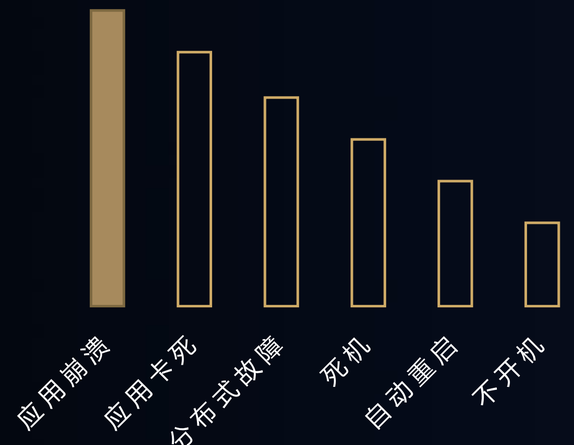


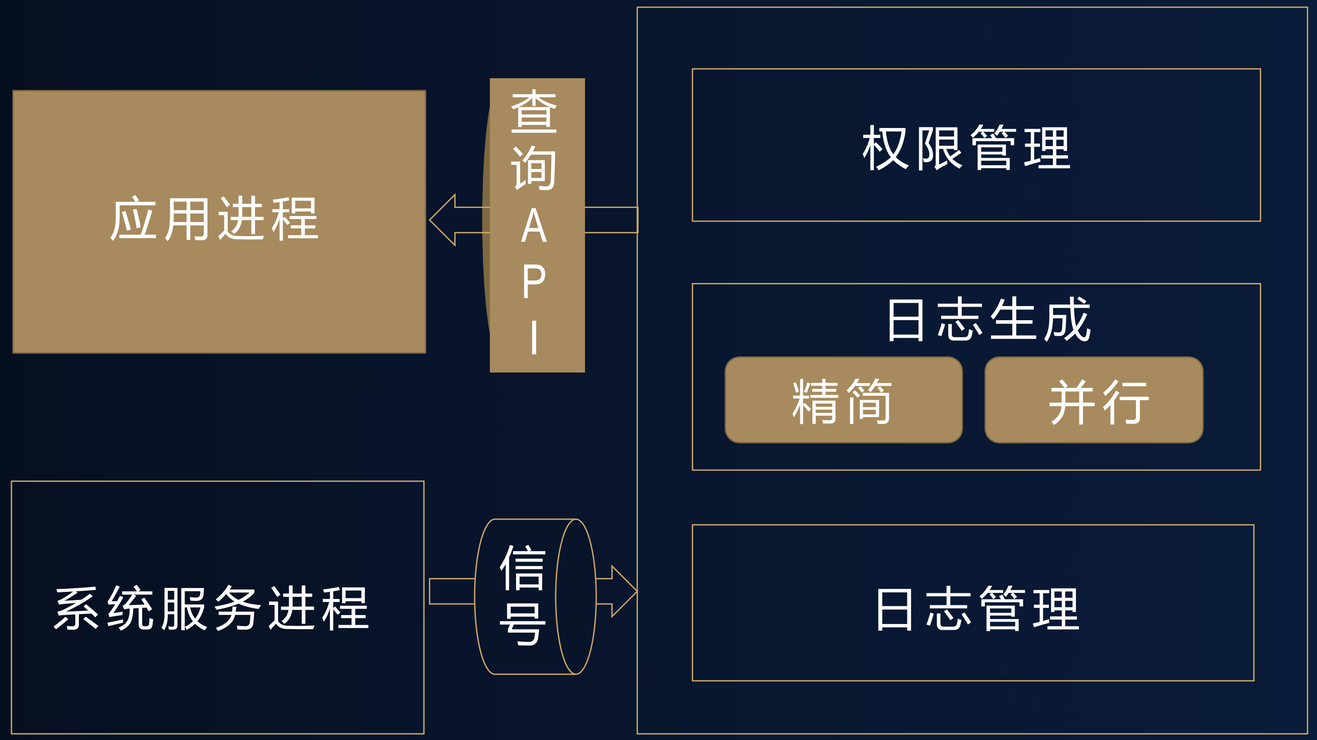
开发定位难度逐渐增加

7类单系统故障检测器 + 分布式故障检测器

进程崩溃检测

故障分布





关键技术：全栈检测、日志精简、精准定位

进程崩溃日志

CPP CRASH

```
Device info:TAS-AL00
Build info:TAS-AL00 11.0.0.555
Module name:com.XXX.ohos.XXX.XXX
Version:1.0.0.80  --- 应用版本号
Pid:28137         --- 进程ID
Uid:10056         --- 应用UID
Reason:Signal:SIGSEGV(SEGV_ACCERR)@0x79f6d1e110   --- 故障原因
Process name:com.XXX.ohos.XXX.XXX    --- 进程名
Fault thread Info:
Tid:28137, Name:os.XXX.XXX
#00 pc 000000000011e110 [anon:libc_malloc]
#01 pc 00000000001d6b50 /system/lib64/libagpcoreui.z.so (OHOS::AGP::UILayerGroup::RemoveFromSceneGraph()+44)
#02 pc 00000000001d6b50 /system/lib64/libagpcoreui.z.so ()
Register Info:
x0 fffffffffffffffc x1 0000007ff1892540 x2 0000000000000010 x3 00000000000493e0  ----- 寄存器信息
x4 0000000000000000 x5 0000000000000008 x6 0000007d540e3000 x7 00000000014d3168
x8 0000000000000016 x9 96b7332bbfde6c12 x10 0000007ccea0b0f8 x11 0000007cc0000000
x28 00000000701d72c0 x29 0000007ff18926a0
sp 0000007ff1892500 lr 0000007d51ca2aa8 pc 0000007d509e6a68
Memory Stack:    ---- 故障发生附近内存信息
0000007ff1892520 0000000000000000 0000000000000058 ........X.......
Other thread stacktrace:
Maps:
43d000-43f000 r--p 00000000 /system/bin/appspawn
43f000-444000 r-xp 00001000 /system/bin/appspawn ........X.......
-----------   辅助信息  -----------
1633678770 APP_START ---- 故障发生附近SysEvent
1633678771 APP_RESUMED
1633678772 APP_EVENT InitXXsdk ok ---- 故障发生附近AppEvent
1633678800 APP_SWITCH_TO_BACKGROUND
1633678921 APP_RESUMED
1633678950 APP_EVENT Failed to read from db
1633678951 CPP_CRASH SIGSEGV
```

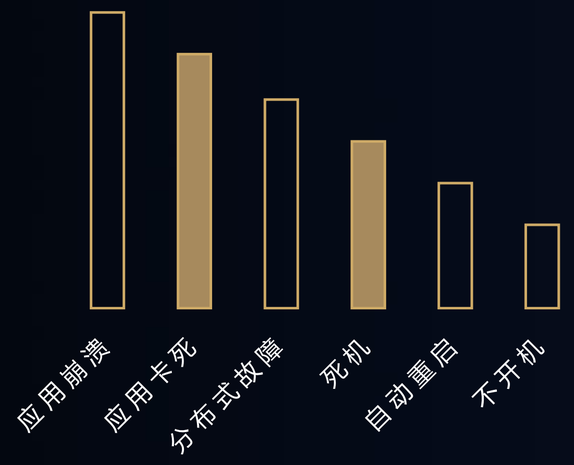
- 运行设备信息
- 运行模块进程信息
- 崩溃线程调用栈

根据需求灵活配置

- 崩溃线程寄存器
- 崩溃线程栈内存
- 其他线程栈
- 相关系统/应用事件

应用卡死&系统死机检测

故障分布



32个检测点

APP	UI线程卡死	渲染线程卡死	
Framework	ACE框架卡死	界面切换超时	...
Native	核心服务卡死	IPC超时	...
Kernel	Hungtask	亮灭屏超时	...

关联判决 ＋ **灵活组合**

快速点击屏幕	快速点键BACK键	快速点击HOM键	快速点击PWR键

关键技术：多维检测、按需部署、灵活组合

应用卡死日志

1. 公共信息

Device info:TAS-AL00
Build info:TAS-AL00 2.0.0.908_Z(DEVC00E1R5P3dexlog)
Module name:com.huawei.helloworld
Version:10
Pid:12848
Uid:10213

2. 故障关键信息

Reason:UI_BLOCK_6S
Summary: at java.lang.Thread.sleep(Native method)
- sleeping on <0x044a5ea3> (a java.lang.Object)
at java.lang.Thread.sleep(Thread.java:443)
- locked <0x044a5ea3> (a java.lang.Object)
****

DOMAIN:APPEXECFWK
STRINGID:UI_BLOCK_6S
TIMESTAMP:1628225686006
PID:12848
UID:10213
PACKAGE_NAME:com.huawei.helloworld
PROCESS_NAME:com.huawei.helloworld
MSG:APP_FREEZE, has taken 6s
****

UI_BLOCK_3S
PID = 12848
UID = 10213
MSG = APP_FREEZE_WARNING\nactivityName = com.huawei.helloworld.MainAbilityShellActivity\nversionName = 10\n, has taken 3s
PACKAGE_NAME = com.huawei.helloworld
PLATFORM = Z
PROCESS_NAME = com.huawei.helloworld
eventLog_action = s,b,pb:1eventLog_interval = 0

3. 故障辅助信息

----- **OpenStacktraceCatcher** –
pid==12848 packageName is com.huawei.helloworld----- pid 12848 at 2021-08-06 12:54:43 -----Cmd line: com.huawei.helloworldBuild ----- end 12848
-----**BinderCatcher** --:
1603:2114 to 1105:1130 code 4 wait:0.287267187 s
1603:2173 to 772:2367 code 5 wait:28.848666154 s
----- **PeerBinderCatcher** -- :
pid==12848 layer_ == 1

1 硬件信息、软件信息、软件模块名、软件模块版本、Pid、Uid

2 故障根因

故障线程调用栈

卡死检测点事件

3 故障进程调用栈、IPC耗时信息、关联进程调用栈

应用卡死事件序列

问题描述:

某视频APP,观看中，弹出应用无响应的框，见截图


```
time [20210528221715-00010], ID [01][APP_WARNING], pid 8706, tgid 8380
time [20210528221717-00011], ID [02][APP_FREEZE], pid 8706, tgid 8380
time [20210528221727-00012], ID [23][MULTI_TOUCH], pid 1307, tgid 1072
time [20210528221730-00013], ID [04][APP_RECOVER], pid 8706, tgid 8380
```

事件回放:

10秒内发现有多次触屏事件，卡死15秒后弹框提示

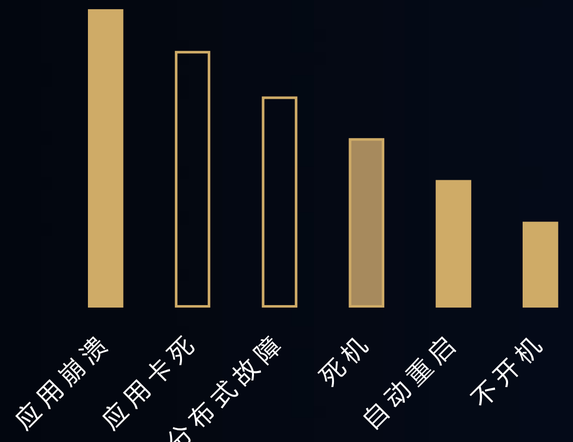
定位:

{ what=1 target=com.xxxx.xxxx.LocationClient$a }, has taken 5104ms
单事件在主线程处理超过5秒

越界代码检测

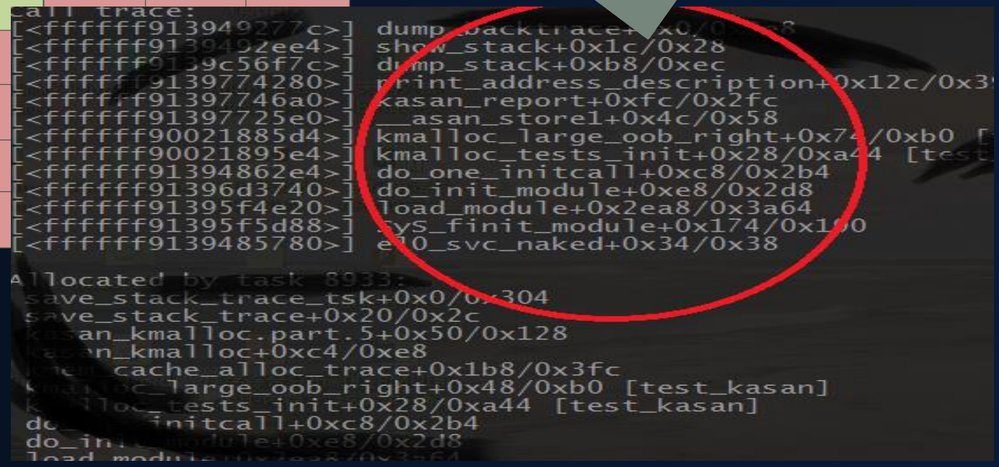
可检测类型：堆越界、栈越界、全局缓冲区溢出、野指针、内存重复释放、整型溢出

故障分布



内存标记

调用追踪



关键技术：精准检测、定位越界代码行、低开销运行时部署

观测剖析

工欲善其事，必先利其器

信息导出

使用场景：为开发、调试、测试人员提供统一的系统信息获取工具

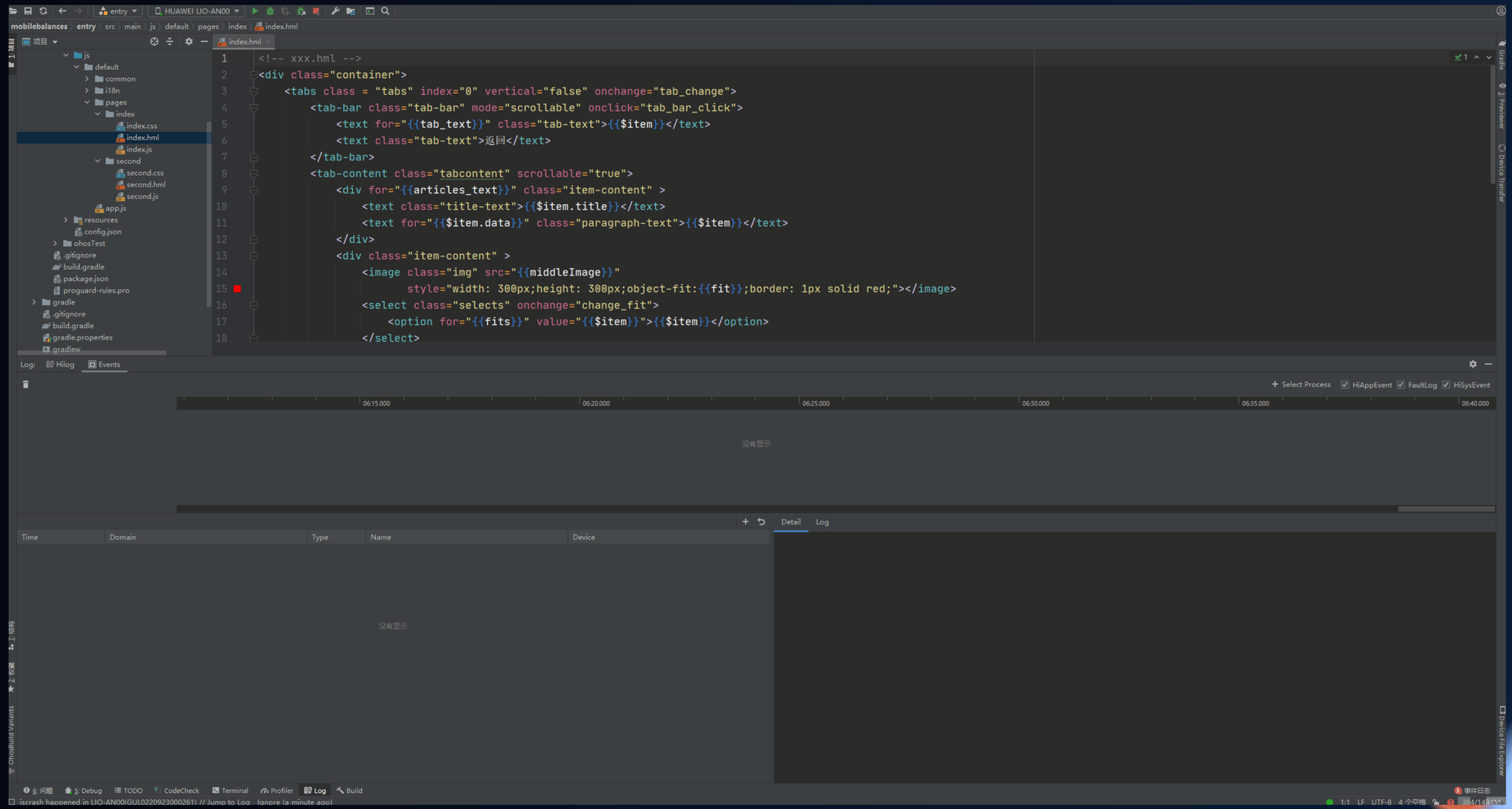
信息导出命令

命令接口	命令含义
hidumper	dump cpu usage, memory usage and all tasks
hidumper -dc	dump the cpu usage
hidumper -df	dump the latest fault logs
hidumper -dm	dump the memory usage
hidumper -dt	dump all the tasks
hidumper -h	help text for the tool
hidumper -ikc	inject kernel crash
hidumper -iuc	inject user crash
hidumper -m	dump memory to stdout in hex format
hidumper -m filepath	dump memory to filepath in the device in hex format
hidumper -m memstart memsize	dump memory with starting address memstart(hex) and size memsize(hex) to stdout in hex format
hidumper -m memstart memsize filepath	dump memory with starting address memstart(hex) and size memsize(hex) to filepath in the device in hex format
hidumper -t [timeout]	timeout (Second)
hidumper -L [size]	maximum limit size （BYTE）
hidumper -z [path]	compress in the specified path
hidumper -l [-s/-c]	list all information about clusters or services
hidumper -c	dump information about all clusters
hidumper -s	dump information about all services
hidumper -s sa0 sa1	dump information about sa0 and sa1 services
hidumper -a “-x -y”	specify parameters "-x" and "-y"
hidumper -p pid1 pid2	dump process stack about pid1 and pid2

分布式联动调试

关键技术：自动捕获异常，关联日志、事件、跟踪、故障日志，异常堆栈定位代码行

自动捕获异常案例

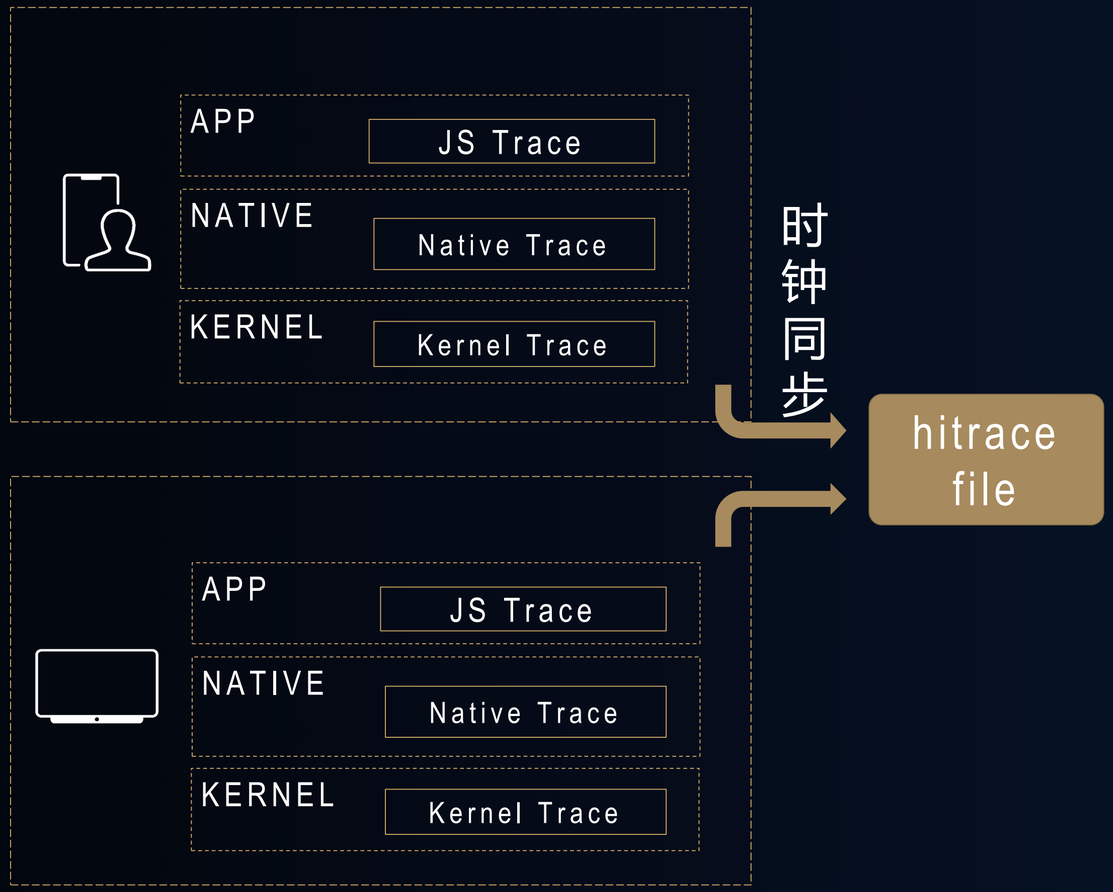

mobilebalances entry src main js default pages index index.hml
index.hml
<!-- xxx.hml -->

<tabs class = "tabs" index="0" vertical="false" onchange="tab_change">
<tab-bar class="tab-bar" mode="scrollable" onclick="tab_bar_click">
<text for="{{tab_text}}" class="tab-text">{{$item}}</text>
<text class="tab-text">返回</text>
</tab-bar>
<tab-content class="tabcontent" scrollable="true">

<text class="title-text">{{$item.title}}</text>
<text for="{{$item.data}}" class="paragraph-text">{{$item}}</text>

<image class="img" src="{{middleImage}}"
style="width: 300px;height: 300px;object-fit:{{fit}};border: 1px solid red;"></image>
<select class="selects" onchange="change_fit">
<option for="{{fits}}" value="{{$item}}">{{$item}}</option>
</select>
Log: Hilog Events
Select Process HiAppEvent FaultLog HiSysEvent
06:15.000 06:20.000 06:25.000 06:30.000 06:35.000 06:40.000
没有显示
Detail Log
Time Domain Type Name Device
没有显示
Debug TODO CodeCheck Terminal Profiler Log Build

分布式调优



时钟同步

hitrace file

关键技术：分布式跟踪采集，自动分析跨设备调用分段时延，高耗时函数预警

演进与展望

- 缺陷检测
- 故障恢复
- 大数据分析
- 更多调试调优工具

<HDC.Together>
华为开发者大会2021

扫码参加1024程序员节

<解锁HarmonyOS核心技能，赢取限量好礼>

开发者训练营

Codelabs 挑战赛

HarmonyOS技术征文

HarmonyOS开发者创新大赛

扫码了解1024更多信息

报名参加HarmonyOS开发者创新大赛

谢谢

欢迎访问HarmonyOS开发者官网

欢迎关注HarmonyOS开发者微信公众号