

应用探索测试服务使用指南

2024-07-31





华为终端有限公司

版权所有 © 华为终端有限公司 2024。 保留一切权利。

本材料所载内容受著作权法的保护，著作权由华为公司或其许可人拥有，但注明引用其他方的内容除外。未经华为公司或其许可人事先书面许可，任何人不得将本材料中的任何内容以任何方式进行复制、经销、翻印、播放、以超级链路连接或传送、存储于信息检索系统或者其他任何商业目的的使用。

商标声明

、 **HUAWEI**、华为，以上为华为公司的商标（非详尽清单），未经华为公司书面事先明示许可，任何第三方不得以任何形式使用。

注意

华为会不定期对本文档的内容进行更新。
本文档仅作为使用指导，文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为终端有限公司

地址：广东省东莞市松山湖园区新城路 2 号
网址：<https://consumer.huawei.com>

目 录

1 概述.....	3
2 版本记录.....	4
3 应用探索测试	5
3.1 操作指导	5
3.1.1 创建任务.....	5
3.1.2 执行测试.....	7
3.1.3 查看报告	8
4 测试报告解读	12
4.1 故障类型	12
4.2 故障定位	12
4.2.1 JS_ERROR.....	12
4.2.2 APP_FREEZE.....	13
4.2.2.1 日志解读.....	13
4.2.2.2 日志示例.....	14
4.2.3 CPP_CRASH.....	14
4.2.3.3 日志解读.....	14
4.2.3.4 日志示例.....	15
5 FAQ.....	16

1 概述

DevEco Testing 应用探索测试为 HarmonyOS Next 应用开发者提供面向应用的智能遍历测试手段，帮助开发者识别影响应用、系统稳定性的多类常见异常问题，助力开发者打造高稳定性、可靠性的产品。

应用探索测试：针对应用稳定性测试，DevEco Testing 提供基于专家经验的智能遍历手段，借助智能 AI 实现场景智能感知及控件语义分析，驱动测试高效执行，并通过对测试数据的持续学习，推动遍历执行持续优化，帮助用户识别应用故障问题及定位问题。

2 版本记录

日期	修订版本	修改描述
2023-11-18	01	发布应用探索测试服务
2024-05-31	02	测试报告优化
2024-07-31	03	故障上报优化；支持生成应用遍历模型

3 应用探索测试

3.1 操作指导

3.1.1 创建任务

步骤 1：进入 DevEco Testing 客户端，在左侧菜单栏选择“稳定性测试”，点击“应用探索测试”服务卡片，即进入任务创建界面。



图 2.1 应用探索测试服务入口

步骤 2：配置服务参数：

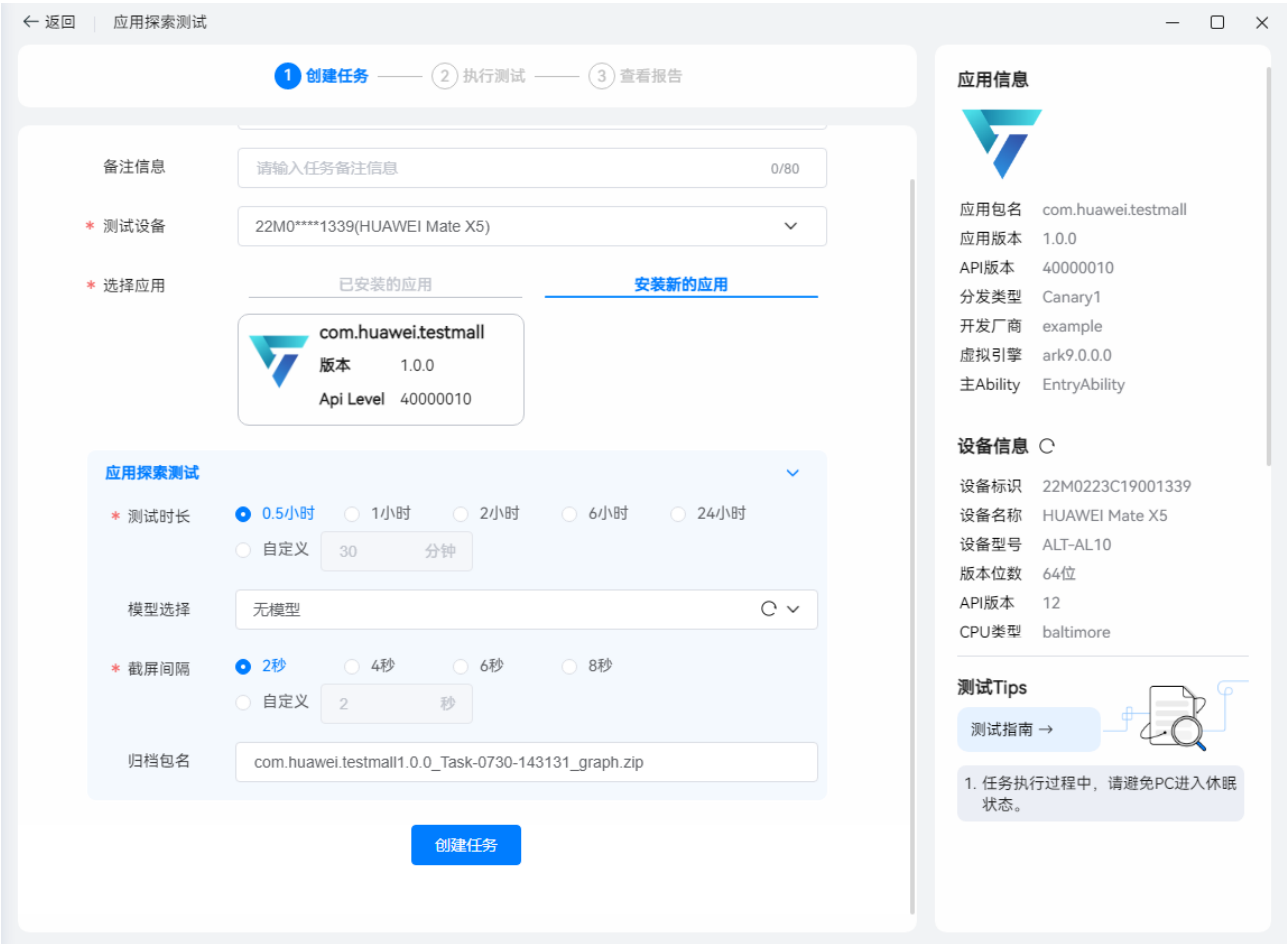


图 2.2 任务配置页

- 任务名称：用于标识任务，系统会根据时间生成默认任务名，支持用户自定义修改。
- 备注信息：按需填写任务备注信息，便于快速筛选报告。
- 测试设备：待测设备。系统版本支持 HarmonyOS Next 版本。
- 选择应用：可选择测试设备上已安装的应用；或安装新的应用，即在测试设备上安装新的应用包。

应用探索测试

- 测试时长：任务总时长，建议时长不低于 1 小时，时长过低的测试结果不具代表性。
- 模型选择：初次创建任务时，默认为无模型；后续创建任务，将自动生成遍历模型，可选择已有模型，辅助提升遍历效率。
- 截屏时间：截屏时间按需选择，截屏间隔越短，路径地图中的截图数量相对越多。
- 归档包名：每次任务结束后，将基于本次测试过程生成遍历测试模型，将自动存储至 D:\ProgramData\DevEco Testing\testGraph\exploreTest 路径*下，可供后续应用探索测试及 UX 基础质量测试高级配置场景下使用。

*模型存储路径为基于自定义数据路径下的存储路径，可前往【设置-基本设置】中修改数据路径。

步骤 3：创建任务。参数配置完成后，点击“创建任务”即开始测试。

3.1.2 执行测试

任务创建后即进入测试执行页面（图 2.3），测试过程中，在测试页面可以看到测试进度、遍历路径地图、设备镜像、及语义分析过程。

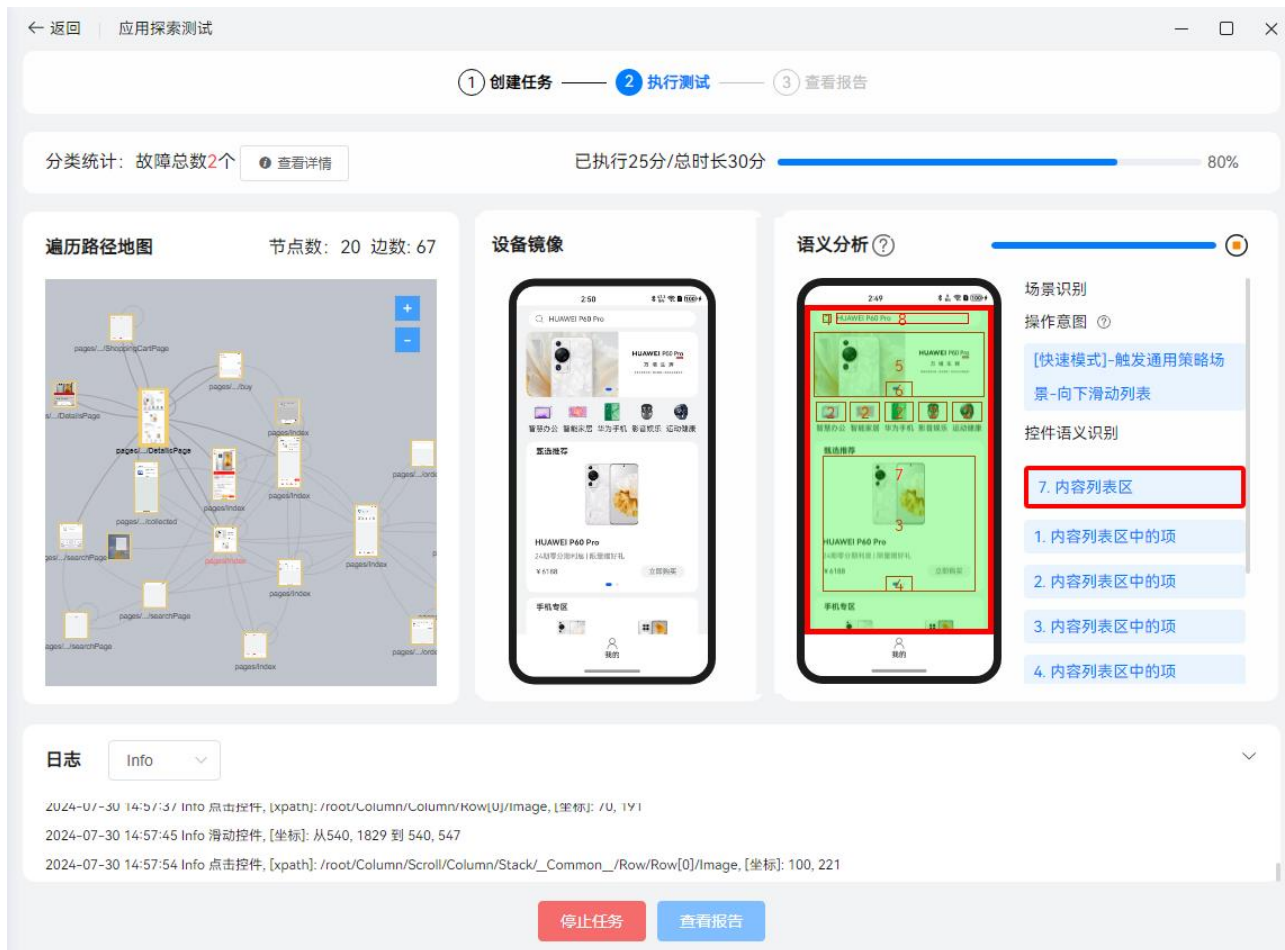


图 2.3 测试执行页面

以图 2.3 购物类应用为例，测试过程中，会对应用界面上的控件进行实时的语义分析，推断出每个控件代表的具体含义（分类、订单、添加、支付），结合对场景的智能感知，识别出当前正在对购物场景进行测试，据此生成对应的测试意图，如浏览商品、购物车结算等，驱动测试高效执行。

语义分析：智能 AI 将自动为界面控件排序，测试过程将参照顺序执行。测试执行中支持用户暂停或启动语义分析，语义分析暂停时，任务会继续执行，直至任务时间结束。

测试执行时，可查看测试进度及实时故障识别数据，点击“查看详情”，可以查看问题列表，包含问题类型、模块、发现时间、发现次数、概要信息及定位日志。

详情

问题详情

NO.	问题类型	细分类型	模块	发现时间	发生次数	概要信息	定位日志
1	应用崩溃故障	JS_ERROR	com.huawei.testmall	2024-07-31 08:29:25	1	查看	查看
2	应用冻屏故障	APP_FREEZE	com.huawei.testmall	2024-07-31 08:33:42	2	查看	查看

100条/页

共 2 条

<

1

>

1

Go

图 2.4 问题详情

执行过程中如果发生设备断连、重启等情况，遍历暂停但任务会继续计时，当设备重新连接（或重启）完毕，遍历任务继续执行，断连（或重启）前的测试信息依然存在，若设备断连并且在生成报告前都没有重新连接会导致生成的报告数据不完整。

3.1.3 查看报告

任务结束后，测试报告如下图所示：

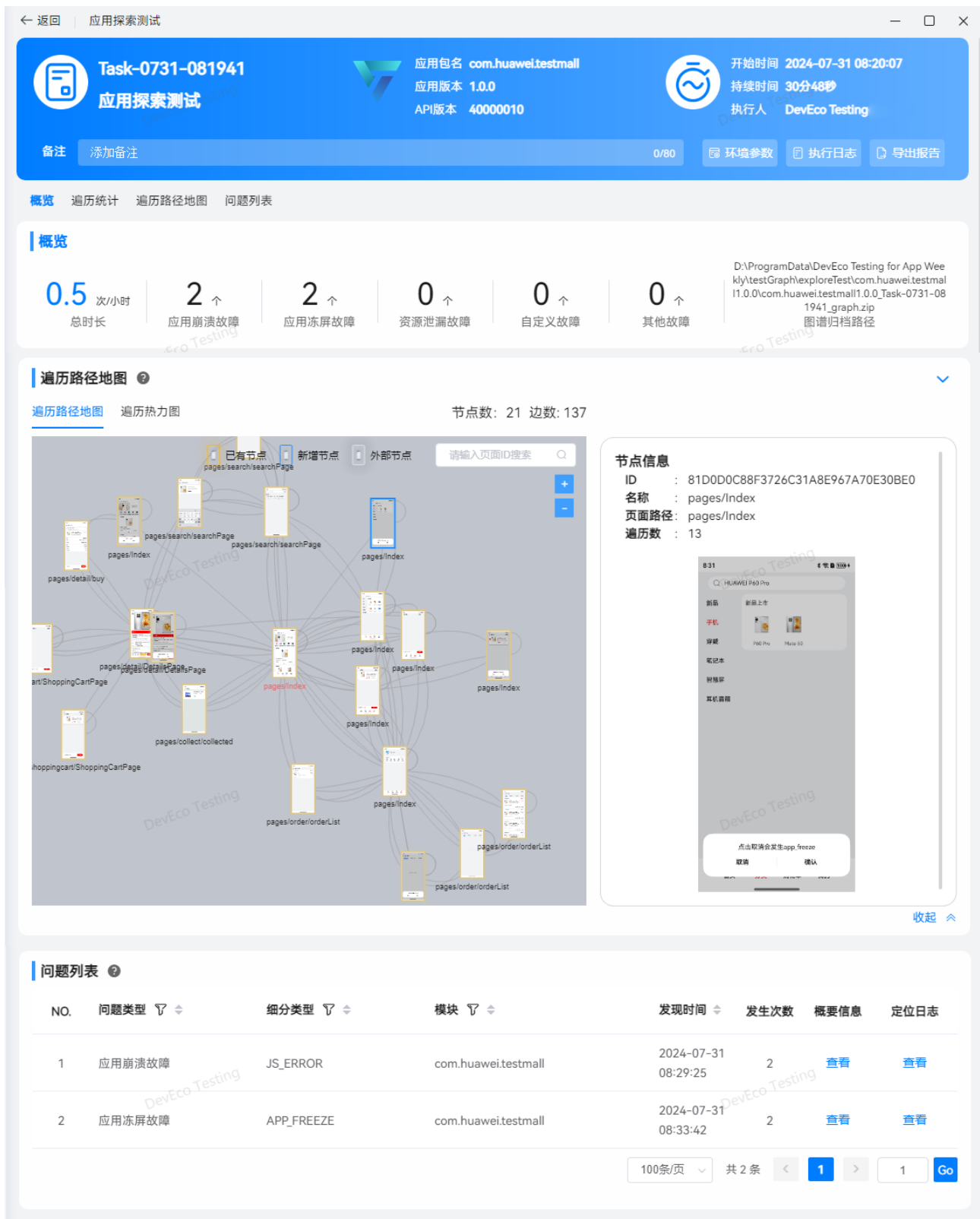


图 2.5 测试报告

任务信息：在报告的最上方可查看本次任务的设备信息、运行时间、参数配置、执行日志、应用信息，点击“导出报告”按钮可以将报告导出 html 格式的报告。

概览：本次任务的主要数据概览及本次任务模型包存放路径。

遍历路径地图：

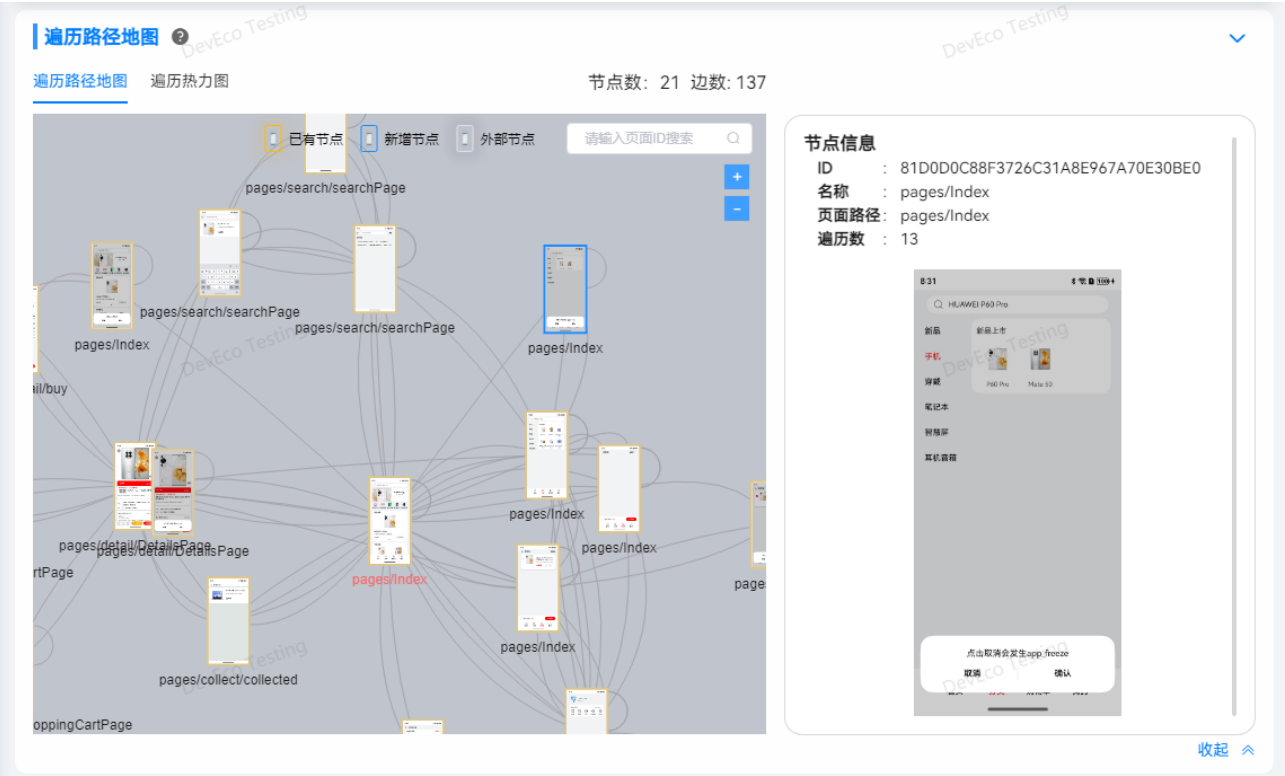


图 2.6 遍历路径地图

问题列表：对测试过程中产生的问题信息的分类统计。点击列表中的 筛选符号 能够对指定列的数据进行筛选，点击列表中的 三角符号 可以对指定列进行正序或倒序的排序，默认按照发生时间的正序进行排序。点击“概要信息”列的“查看”能够查看对应故障的概要信息（图 2.7）。点击“定位日志”列的“查看”能够跳转到存放了 faultlog 日志及故障发生时段 hilog 日志的文件夹（图 2.8）。

详情

name	JS_ERROR
domain	AAFWK
tag	STABILITY
level	CRITICAL
time	2024-07-31 08:29:25
timeZone	+0800
id	06889762447881776078
type	1
pid	43199
tid	43199
uid	20020149
seq	69805
PRE_INSTALL	No
HAPPEN_TIME	1722385765428
LOG_PATH	/data/log/faultlog/faultlogger/jscrash-com.huawei.testmall-20020149-20240731082925
PNAME	com.huawei.testmall

共 2 条12

图 2.7 故障概要信息

problem_1

文件主页共享查看

data > logs > problem_1

搜索"problem_1"

名称	修改日期	类型	大小
hilog.txt	2024/7/31 8:29	文本文档	878 KB
hisysevent.json	2024/7/31 8:29	JSON 文件	2 KB
jscrash-com.huawei.testmall-20020149-20240731082925.l...	2024/7/31 8:29	文本文档	1 KB

3 个项目

问题列表

NO.	问题类型	细分类型	模块	发现时间	发生次数	概要信息	定位日志
1	应用崩溃故障	JS_ERROR	com.huawei.testmall	2024-07-31 08:29:25	2	查看	查看
2	应用冻屏故障	APP_FREEZE	com.huawei.testmall	2024-07-31 08:33:42	2	查看	查看

100条/页 共 2 条12Go

图 2.8 定位日志

*更多应用稳定性体验优化建议及问题定位，请查阅：[应用稳定性体验建议](#) 及 [CppCrash 故障定位指导](#)

4 测试报告解读

4.1 故障类型

DevEco Testing 探索测试服务会发现并收集测试过程中发生的故障，故障类型包含以下几种：

1. JS_ERROR：应用异常，应用程序在 JS 层发生了崩溃；
2. APP_FREEZE：应用无响应，前台应用无法及时响应用户操作；
3. CPP_CRASH：进程崩溃，C++编写的 Native 进程（包含 c++应用进程和统服务进程）发生崩溃；
4. FD_LEAK：句柄泄漏故障，是由于进程句柄数过高且持续增长，以此判定该进程可能存在句柄泄漏。
5. THREAD_LEAK：线程泄漏故障，是由于进程的线程数过高且持续增长，以此判定该进程可能存在线程泄漏。
6. MEMORY_LEAK：内存泄漏故障，是由于进程的 PSS 内存大小过高且持续增长，以此判定该进程可能存在内存泄漏。
7. HiAppEvent 故障：HiAppEvent 故障是来自应用开发者在应用内预埋的 HiAppEvent 故障类打点事件，每一个 HiAppEvent 故障类事件会生成一个对应的故障记录

4.2 故障定位

4.2.1 JS_ERROR

对于 JS_ERROR 类型的故障，可以查看定位日志文件夹中的 faultlogger 日志（文件名：jscrash-[bundle name]-[uid]-[time]），根据日志中的 Error message 和 stacktrace 来定位错误（图 6.1），对于托管代码的导致的 JS_ERROR，日志还会定位到产生错误的具体代码（图 6.2）。

```

jsrsh-com.example.crashapplication:20010030-20221206160829[3]
1 Device info:OpenHarmony 2.0 Canary
2 Build info:OpenHarmony 3.2.7.5
3 Module name:com.example.crashapplication
4 Version:1.0.0
5 Pid:1882
6 Uid:20010030
7 Lifetime: 0.000000s
8 Js-Engine: ark
9 page: <JS> index.js
10 Error message: OutOfMemory when trying to allocate 262144 bytes function name: OldSpace::Merge
11 Stacktrace:
12   at onAppClick (/pages/index/index.js:33:13)
13

```

图 6.1 JS_ERROR 日志 1

```

jsrsh-com.example.crashapplication:20010030-20221207104714[3]
1 Device info:OpenHarmony 2.0 Canary
2 Build info:OpenHarmony 3.2.7.5
3 Module name:com.example.crashapplication
4 Version:1.0.0
5 Pid:12461
6 Uid:20010030
7 Lifetime: 0.000000s
8 Js-Engine: ark
9 page: <JS> index.js
10 Error message: Cannot read property volume of null
11 SourceCode:
12   var volume = test.value();
13
14 Stacktrace:
15   at onAppClick (/pages/index/index.js:20:26)

```

图 6.2 JS_ERROR 日志 2

4.2.2 APP_FREEZE

4.2.2.1 日志解读

APP_FREEZE 类型的故障可在定位日志文件夹中查看对应的 faultlogger 日志（文件名：appfreeze-[bundle name]-[uid]-[time]），日志中提供了触发事件及检测点信息，进程调用栈，关联调用栈，Binder 调用状态以及内存和 CPU 状态，再结合流水日志可以对故障进行分析定位。定位 APP_FREEZE 故障时可以按照以下思路进行故障分析：

（1）首先查看 APP_FREEZE 日志中的 Reason，了解导致 APP_FREEZE 的事件类型，表 6.1 为不同 Reason 对应的事件含义。

表 6.1 appFreeze 故障类型

Reason	事件含义
UI_BLOCK_6S	UI 线程响应超时
THREAD_BLOCK_6S	应用主线程响应超时
APPLICATION_BLOCK_INPUT	用户输入响应超时
NO_DRAW	应用可见窗口绘制超时
LIFECYCLE_TIMEOUT	ability 生命周期切换超时
APP_LIFECYCLE_TIMEOUT	app 生命周期切换超时
SCREEN_ON_TIMEOUT	按下 power 键 10s 屏幕未亮

（2）关注日志的 MSG，根据 MSG 确定大致方向。

（3）分析 OpenStackTraceCatcher 里面的应用栈信息，并且结合流水日志一起确定当前正在做的操作。

（4）查看 PeerBinderCatcher，确认当前进程是否有对端的 binder 卡住，如果有跟当前进程相关的同步 wait，则会有相应的 PeerBinder Stacktrace 信息，即卡住当前进程的对端进程的栈信息。

（5）查看整机的 CPU 状态和当前进程的内存信息辅助定位问题。

4.2.2.2 日志示例

本小节以一个 UI_BLOCK_6S 的故障为例来查看 APP_FREEZE 日志。

(1) 查看 Reason 为 UI_BLOCK_6S，确定是 UI 线程响应超时导致的 APP_FREEZE。

```
1 appfreeze: com.example.myapplication:20221206180623
2 Device info: OpenHarmony 2.0 Canary
3 Build info: OpenHarmony 3.2.7.5
4 Module name: com.example.myapplication
5 Version: 1.0.0
6 Pid: 8178
7 Uid: 20010030
8 Reason: UI_BLOCK_6S
9 appfreeze: com.example.myapplication UI_BLOCK_6S at 20221206180623
```

(2) 查看 MSG，UI_BLOCK 提供两个栈信息，一个 3s 栈和一个 6s 栈，两个栈可以对比看，就这份日志而言，两个栈都是先 id = 8189 的线程等待，这份日志还提供了 stacktrace，知道最近做了两次按钮点击的操作。

```
575 start time: 1670349983448
576 DOMAIN = ACE
577 EVENTNAME = UI_BLOCK_6S
578 TIMESTAMP = 1670349983441
579 PID = 8178
580 UID = 20010030
581 TID = 8182
582 PACKAGE_NAME = com.example.myapplication
583 PROCESS_NAME =
584 eventLog_action = s,cmd:c,cmd:m
585 eventLog_interval = 0
586 MSG = Blocked thread id = 8189
587 JSVM instance id = 0
588 Page: pages/index.js
589 Stacktrace:
590   at onAppButtonClick (\$page:\$index.js:43:15)
591   at onAppButtonClick (\$page:\$index.js:18:18)
592

26 start time: 1670349981423
27 DOMAIN = ACE
28 EVENTNAME = UI_BLOCK_3S
29 TIMESTAMP = 1670349981411
30 PID = 8178
31 UID = 20010030
32 TID = 8182
33 PACKAGE_NAME = com.example.myapplication
34 PROCESS_NAME =
35 eventLog_action = s,pb:0
36 eventLog_interval = 0
37 MSG = Blocked thread id = 8189
38 JSVM instance id = 0
39
40
```

(3) 通过查看 MSG 了解到卡住的线程为 8189，结合 OpenStacktraceCatcher 对应线程的信息，日志以及上述的 stacktrace 看一下是卡在什么操作上了。

```
200 Tid:8189, Name:jsThread-1
201 #00 pc 0011b36a /system/lib/libart_ksruntime.so
202 #01 pc 00169755 /system/lib/libart_ksruntime.so
203 #02 pc 001ec3d5 /system/lib/libart_ksruntime.so
204 #03 pc 001e9e35 /system/lib/libart_ksruntime.so
205 #04 pc 00140141 /system/lib/libart_ksruntime.so
206 #05 pc 001ec3d5 /system/lib/libart_ksruntime.so
207 #06 pc 001e9e35 /system/lib/libart_ksruntime.so
```

(4) 查看 PeerBinderCatcher（目前 PeerBinderCatcher 只在 3s 的栈中提供），看当前进程是否有在和其他进程做 binder 通信，这份日志里是没有的，如果有的话，日志也会提供关联进程的栈信息，可以结合日志进行分析，对问题进行定位。

```
487 PeerBinderCatcher -- pid=8178 layer_== 0
488 :
489
490 BinderCatcher --
491
492
493 pid context request started max ready free_async_space
494 8178 binder 0 3 16 4 520192
495 7873 binder 0 2 16 3 520192
496 1747 binder 0 4 16 5 520192
```

(5) 日志中还提供了 CPU 和内存信息，可以辅助定位，看是不是因为 CPU 负载过高或者内存泄漏导致线程卡住的。

4.2.3 CPP_CRASH

4.2.3.3 日志解读

CPP_CRASH 类型的故障可在定位日志文件夹中查看对应的 faultlogger 日志（文件名：cppcrash-[bundle name]-[uid]-[time]）和 temp 日志（文件名：cppcrash-[pid]-[time]），faultlogger 日志基于 temp 日志生成，省去了内存信息。在定位 cppcrash 故障时可以按照以下步骤结合 temp 日志进行分析：

(1) 查看日志中 Reason，了解错误信号，确定错误类型，表 6.2 为 CPP_CRASH 不同错误信号对应的故障原因。

表 6.2 CPP_CRASH 故障类型

信号值	信号	解释	触发原因
4	SIGILL	非法指令	执行了非法指令，通常是因为可执行文件本身出现错误，或者试图执行数据段，堆栈溢出时也有可能产生此信号
5	SIGTRAP	断点或陷阱异常	由断点指令或其它 trap 指令产生
6	SIGABRT	abort 发出的信号	调用 abort 函数生成的信号
7	SIGBUS	非法内存访问	非法地址，包括内存地址对齐（alignment）出错。比如访问一个四个字长的整数，但其地址不是 4 的倍数。它与 SIGSEGV 的区别在于后者是由于对合法存储地址的非法访问触发的（如访问不属于自己存储空间或只读存储空间）
8	SIGFPE	浮点异常	在发生致命的算术运算错误时发出，不仅包括浮点运算错误，还包括溢出及除数为 0 等其它所有的算术的错误
11	SIGSEGV	无效内存访问	试图访问未分配给自己的内存，或试图往没有写权限的内存地址写数据
16	SIGSTKFLT	栈溢出	堆栈溢出
31	SIGSYS	系统调用异常	非法的系统调用

(2) 通过 addr2line 解析调用栈，确定错误行号，如果不能解析到行号，需要借助反汇编进行下一步分析。

(3) 反汇编

(a) 先通过 llvm-objdump 命令导出对应版本的带符号库的 so 文件所对应的汇编文件；

(b) 在导出的汇编文件中搜索日志中调用栈的第 0 帧偏移，找到发生问题的位置，确定发生问题的指令；

(c) 建立汇编和源码的对应关系，在 cpp 代码中。当进入一个函数时，x0 寄存器存的是 this，r1、r2、r3...分别存的是第 1、2、3...个入参，如果发生问题的变量不是入参，需要结合周围的汇编代码进行逆推，此时的突破口是函数跳转时的符号信息，可以使用 c++filt 命令得到易读的符号信息，辅助锁定代码，再结合具体的代码进行分析确定故障具体原因。

4.2.3.4 日志示例

接下来看一份具体的 cppcrash 的日志。

(1) 首先观察错误类型为 SIGSEGV，非法地址访问，访问地址为 nil，很明显的空指针问题。

```

cppcrash:com.example.crashapplication:20010030-20170810143141[2]  cppcrash-3721-1502375501700[2]
1  Timestamp:2017-08-10 14:31:41.000
2  Pid:3721
3  Uid:20010030
4  Process name:com.example.crashapplication
5  Reason:Signal:SIGSEGV(SEGV_MAPERR)@ (nil)
6  Fault thread Info:
7  Tid:3733, Name:jsThread-1
8  #00 pc d3bda268 /data/storage/c11/bundle/libs/arm/libc.so
9  #01 pc d3bda218 /data/storage/c11/bundle/libs/arm/libc.so
10 Registers:
11 r0:00000000 r1:00000000 r2:02436a50 r3:02436a50

```

(2) 通过 addr2line 对调用栈进行解析，确定发生问题的文件，函数和行号。

```

PS C:\Users\...> addr2line -e D:\DevEcoStudio\Projects\... build\default\intermed... ps\defa
ult\armeabi-v7a\libentry.so 000000000003268 -f -a -p -C
0x000003268: Box::volume() at D:/EcoStudioProjects/crashapplication/entry/src/main.cpp:19
PS C:\Users\...>

```


(3) 找到对应的行，这里 crash 的原因比较简单，看到函数就基本能够确认是调用这个函数的对象为空导致的。

```
18 double::double() {
19     r=length * breadth * height;
20 }
21};
```

(4) 如果发生了编译优化，addr2line 是不能确定行号的，需要通过汇编辅助分析，找出具体错误

在 Linux 环境中执行 `llvm-objdump -d [so 文件路径] > libace.asm`，在导出的 `libace.asm` 中搜索 3268（`cppcrash` 中第零帧的偏移）

```
234 _ZN3Box6volumeEv:
235 3254: 00 48 2d e9    push    {r11, lr}
236 3258: 0d b0 a0 e1    mov     r11, sp
237 325c: 08 d0 4d e2    sub     sp, sp, #8
238 3260: 04 00 8d e5    str     r0, [sp, #4]
239 3264: 04 00 9d e5    ldr     r0, [sp, #4]
240 3268: 00 10 90 e5    ldr     r1, [r0]
241 326c: 04 20 90 e5    ldr     r2, [r0, #4]
242 3270: 08 30 90 e5    ldr     r3, [r0, #8]
```

这一行的指令是 `ldr r1, [r0]` 把 `r0` 寄存器中内存地址的内容读到 `r1` 中，查看日志中 `r0` 寄存器的值是 0，对 0 地址的非法访问，验证了 `so` 和日志是能够对应上的，同时也确认了是调用函数的对象为空指针导致的。

```
Registers:
r0:00000000 r1:00000000 r2:01647bd0 r3:0000000c
```

5 FAQ

1、为什么测试页面截图会出现为页面切换状态？

由于应用界面切换存在较大的不确定性，测试截图时，存在一定的可能性截到页面切换的中间过程，该场景为正常情况，不影响应用探索测试整体功能。



2、测试过程中，有些场景为什么存在语义分析不准确？

控件语义分析依赖 AI 模型智能分析，存在一定的误识别概率，该模型会持续更新，力求识别更准确。

3、查看测试报告时，报告加载卡顿的原因是什么？

由于探索测试执行时间比较长，会产生大量的报告数据，当某些报告节点比较多时，报告加载会消耗较多资源。可考虑适当缩短测试时长，或选择更高内存的主机进行测试执行。

4、应用中嵌入了 Web 窗口，能进行应用探索测试吗？

应用中若嵌入 Web 窗口，不影响应用内原生界面的测试，Web 窗口内的界面暂不支持遍历，相关能力将在后续版本中提供。