



ArkUI Analyzer 设计开发文档

基于方舟字节码文件的鸿蒙应用功能地图静态分析工具

队伍名称:	三个臭皮匠顶个诸葛亮
所属赛道:	OS 原理开发 / OS 系统工具开发
项目成员:	明孟立、蒋贤潇、李恩兆
校内导师:	苏亭、孙海英
所属高校:	华东师范大学
完成日期:	2024 年 12 月 25 日

目录

第 1 章 需求分析	1
1.1 研究背景	1
1.2 相关技术	2
1.2.1 静态分析在 UI 建模中的经典应用	2
1.2.2 目标驱动的 UI 自动化探索	3
1.3 功能需求说明	4
第 2 章 概要设计	5
2.1 架构设计	5
2.2 集成设计	6
2.3 关键数据结构	7
第 3 章 静态分析能力的设计与实现	11
3.1 功能说明	11
3.2 Panda IR 指令到 UI 信息逆向映射的设计与实现	11
3.3 ArkTS 模块之间 import 关系分析的设计与实现	12
3.4 跨 ArkTS 模块数据流分析的设计与实现	13
3.5 函数调用关系分析的设计与实现	15
第 4 章 鸿蒙应用功能地图的设计和实现	17
4.1 功能说明	17
4.2 UI-Import Graph 分析的设计与实现	17
4.3 组件树分析的设计与实现	18
4.4 UI 迁移关系分析的设计与实现	21
第 5 章 项目测试	27
5.1 单元测试	27
5.2 核心功能测试	27
5.2.1 代码功能分析完整性测试	27
5.2.2 代码功能关联映射准确率测试	31
5.3 回归测试	32
参考文献	33
附录 1: 编译与使用教程	36
环境准备	36

快速部署	36
附录 2: 术语表	38
附录 3: OpenHarmony 应用开发框架概述	40

第 1 章 需求分析

1.1 研究背景

鸿蒙操作系统（以下简称鸿蒙系统）是华为主导开发的智能终端操作系统的面向全场景、全连接、全智能时代实现万物互联（包括手机、汽车座舱、IoT 设备等）。鸿蒙系统从 23 年年底开始才陆续引入原生应用，截至 2024 年 9 月鸿蒙生态迎来历史性突破，已经有 15000 个左右的原生应用加入了鸿蒙生态；24 年第四季度华为将推出 HarmonyOS Next 商业版，该版本将只支持鸿蒙原生应用^[1]。因此，有效保障鸿蒙原生应用的质量和用户体验对鸿蒙生态的健康发展尤为重要。

自动化界面遍历测试技术（Automated GUI Testing Techniques）^{[2][3]}是一种保障移动应用质量的有效手段，被广泛用于应用兼容性测试、功能测试和回归测试。比如，在应用市场的场景下，此类技术可对海量应用实现云测试，快速识别出存在质量缺陷和安全风险的应用，提高鸿蒙生态中应用质量。在开发者测试场景下，此类技术也可以帮助开发者快速实现功能测试，或者对新版本的应用实现快速的回归测试，尽早拦截潜在的功能稳定性和不一致性错误^[4]。在实践中，此类自动化界面遍历测试技术往往与静态时序分析技术^[5]相结合，以提升应用自动化测试的效率和深度。

然而，由于鸿蒙原生应用的开发采用了自研的 ArkTS 作为主要编程语言、ArkUI 作为界面编程框架，并支持混合语言实现（如 C++ 等 Native 代码），使得以往针对其他移动操作系统平台（如谷歌安卓系统、苹果 iOS 系统等）的自动化界面遍历测试技术和静态程序分析技术无法适用。比如，一些针对安卓应用的知名自动化测试工具，如 FastBot^[4]和 DroidBot^[8]等，无法应用到鸿蒙原生应用中。同时，一些被广泛用于分析安卓应用的知名静态分析工具 Soot^[9]和 FlowDroid^[10]（其主要分析对象为 Java 语言），也无法应用于分析 ArkTS 语言，很难为针对鸿蒙原生应用的自动测试技术提供支撑。

虽然鸿蒙系统目前提供了一些自动化测试能力，比如 OpenHarmony 自动化测试框架代码部件仓 arkXtest^[11]和 OpenHarmony 稳定性测试自动化工具

Wukong^[12]，但是 arkXtest 依赖于人工编写测试用例，而 Wukong 主要的测试策略只是简单的随机测试，类似于安卓测试工具 Monkey。因此，针对鸿蒙原生应用，如何构建有效的自动化界面遍历测试技术和静态程序分析技术以支撑高质量的鸿蒙原生应用开发是一个值得探索的重要课题。

1.2 相关技术

本节主要介绍项目相关的现有技术研究进展，并基于调研结果探索这些技术如何在鸿蒙原生应用的 UI 自动化探索中更好地应用。具体涉及静态分析在 UI 建模中的经典应用、目标驱动的 UI 自动化探索两个方面。

1.2.1 静态分析在 UI 建模中的经典应用

静态分析在 UI 自动化探索中的应用尤为重要，特别是在 UI 建模的场景中。传统的 UI 自动化探索通常依赖于动态执行，然而，这种方式需要在应用运行时生成和执行大量测试用例，耗时且效率低下。为了提高效率，研究者开始探索基于静态分析的 UI 建模方法。Shengqian Yang 等（2015）提出的静态窗口迁移图（WTG）便是一个成功的尝试，该方法通过静态分析应用的 UI 组件（主要是 Activity、Menu、Dialog 等高频组件），构建了一个描述 UI 状态及其迁移关系的图模型（图 1-1）。在此基础上，自动化工具可以通过遍历迁移图快速地识别 UI 界面间的迁移顺序，从而提高探索效率。

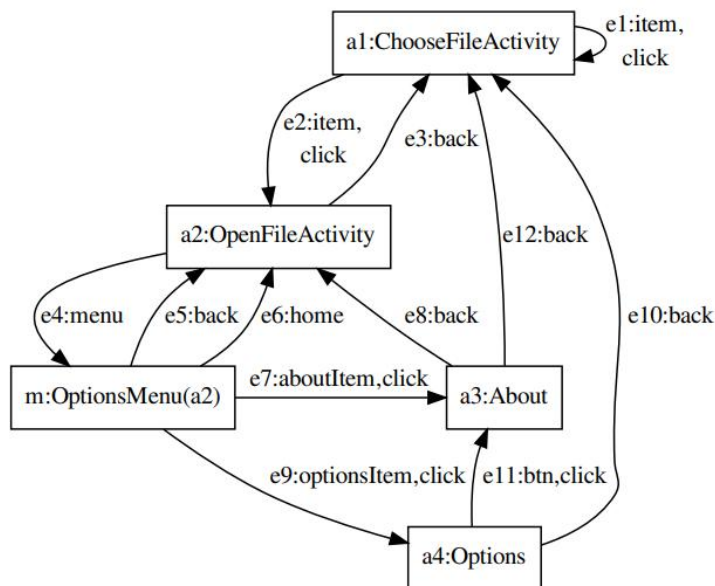


图 1-1 WTG 示例图

尽管静态分析指导自动化工具进行 UI 探索，在安卓平台取得了成功应用。然而，由于鸿蒙原生应用的开发采用了自研的 ArkTS 作为主要编程语言、ArkUI 作为界面编程框架，并支持混合语言实现（如 C++ 等 Native 代码），使得以往针对安卓平台的静态分析无法直接适用。

具体而言，例如现有的通过静态分析指导 UI 自动化探索的工作，例如 Gator，其通过分析、构建出安卓应用的 Component Constraint Graph^[15]、Window Transition Graph^[16]、Callback Control Flow Graph^[17]，以指导自动化工具进行 UI 探索。但是，Gator 工具依赖于 Soot^[9]静态分析框架，Soot 为 Gator 提供了分析 Component Constraint Graph、Window Transition Graph、Callback Flow Graph 需要的基础分析能力。此外，Soot 目前仅能处理 Java 字节码、Android 字节码、Jimple 和 Jasmin 格式的程序，而无法处理 ArkTS、ArkUI 和方舟字节码格式的程序。

1.2.2 目标驱动的 UI 自动化探索

目标驱动的 UI 自动化探索方法是一种相对较新的自动化探索策略，其核心思想是针对特定的功能目标进行探索，而非随机地遍历所有可能的应用路径。Duling Lai (2019) 在其研究中提出了 GalexPorer 这一工具，通过构建 Screen Transition Graph (STG) 来引导自动化探索的执行^[14]。STG 是基于 UI 屏幕状态及其之间迁移关系的图模型，GalexPorer 根据该图快速地识别如何从初始界面跳

转到目标功能所在的界面，以及所需的功能行为。目标驱动的 UI 自动化探索通过精确导航至目标功能区域，能够减少无效路径的探索，提高测试执行的效率。同样，该工具无法直接用于鸿蒙应用的 UI 自动化探索，因为其底层依赖的静态分析基础能力，并不支持对鸿蒙应用的分析。

1.3 功能需求说明

基于上述讨论，本项目旨在构建鸿蒙生态中首款基于方舟字节码文件的应用功能地图静态分析工具。该工具的设计与开发基于方舟字节码文件，为了更好地支持没有应用源码的场景，从而具有更广泛的适用性。在功能地图构建之前，我们将基于方舟编译器的中间表示（Panda IR）构建一套基础静态分析能力。

鸿蒙应用功能地图具体包括：应用中 UIAbility 之间的 UI 迁移关系、Page 之间的 UI 迁移关系、Page 内部的组件之间的 UI 迁移关系，以及 Page 内部的组件之间的结构关系。功能地图主要用于引导 UI 自动化探索的过程。通过给动态测试工具提供路径信息，使其能够有针对性地覆盖关键路径，以及探索此前动态测试中尚未遍历到的路径。

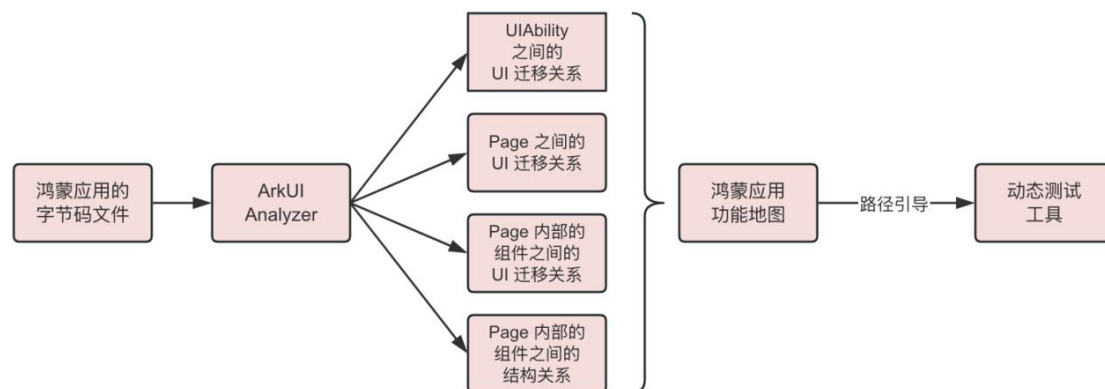


图 1-2 ArkUI Analyzer 应用场景示例图

第 2 章 概要设计

2.1 架构设计

ArkUI Analyzer 的总体设计分为两大模块：一个模块是基于 Panda IR 的静态分析基础能力的构建, 另一个模块是在该基础能力之上的鸿蒙应用功能地图的分析、构建。

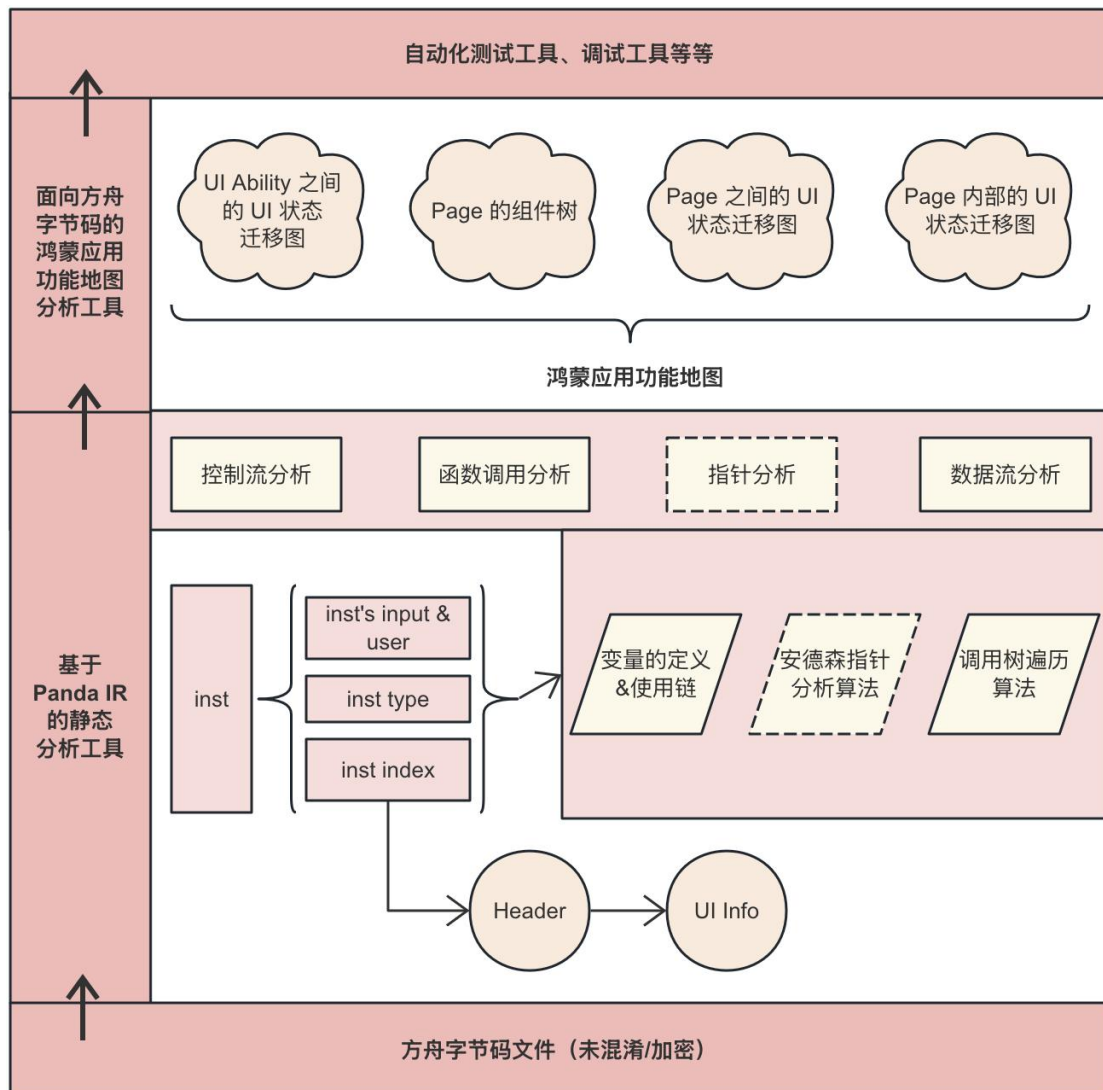


图 2-1 ArkUI Analyzer 架构图

首先, 方舟字节码文件会通过方舟编译器的运行时被解析为 Panda IR 这一中间表示。基于 Panda IR, 我们将设计并实现控制流分析、函数调用分析、指针分析和数据流分析等基础分析功能。然后在此基础上, 继续分析出鸿蒙应用的 UI

Ability 之间的 UI 状态迁移图、Page 的组件树、Page 之间的 UI 状态迁移图，以及 Page 内的 UI 状态迁移图，这些图模型的组合即为最终的鸿蒙应用功能地图。ArkUI Analyzer 源码的大致目录结构如图 2-2 所示。



图 2-2 ArkUI Analyzer 源码目录结构图

2.2 集成设计

ArkUI Analyzer 作为一个独立模块置于方舟编译器 (Ark Compiler) 中，其编译过程仅依赖于方舟编译器本身，无需任何其他第三方依赖。具体关系如下所示：方舟编译器主要分成两个部分：编译工具链与运行时。

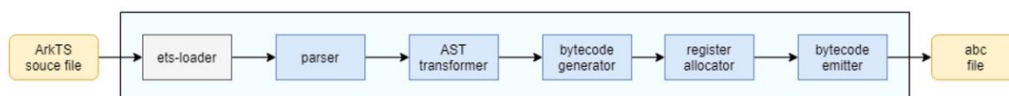


图 2-3 Ark Compiler 编译工具链流程图

编译工具链负责将 ArkTS 源码通过多层转换最终生成 Panda 字节码文件 (.abc 文件)，运行时 (Ark Runtime) 负责加载和执行生成的 Panda 字节码。如图 2-4 所示为 Ark Compiler 与 ArkUI Analyzer 的关系。

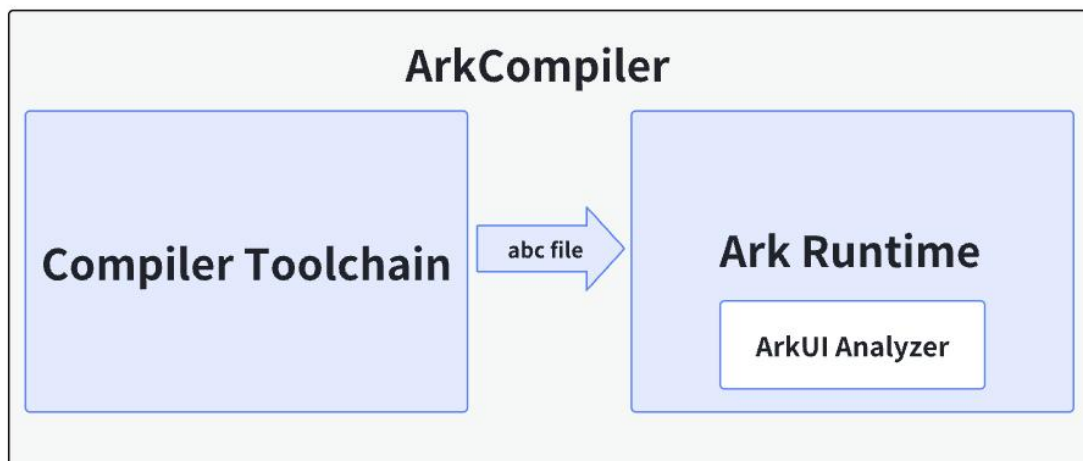


图 2-4 ArkCompiler 与 ArkUI Analyzer 关系示例图

2.3 关键数据结构

```

01. class Rel {
02. public:
03.     /// 创建 UIAbility 和 Page 之间的包含关系
04.     static Rel CreateCntmt(std::weak_ptr<UIAbility> from_ability,
05.                           std::weak_ptr<Page> to_page, bool is_init) noexcept;
06.
07.     /// 创建 UIAbility 之间的迁移关系
08.     static Rel CreateTransUIAbility(std::weak_ptr<UIAbility> from_ability,
09.                                     std::weak_ptr<UIAbility> to_ability,
10.                                     TransType ttype,
11.                                     TriggerReason reason) noexcept;
12.
13.     /// 创建 Page 之间的迁移关系
14.     static Rel CreateTransPage(std::weak_ptr<Page> fromPage,
15.                                std::weak_ptr<Page> to_page, TransType ttype,
16.                                TriggerReason reason) noexcept;
17.
18.     /// 获取关系 (迁移/包含) 类型
19.     RelType GetRelType() const noexcept;
20.     /// 获取触发描述
21.     std::string GetTriggerDesc() const;
22.     /// 获取迁移类型
23.     TransType GetTransType() const;
24.     /// 在包含关系中, 检查是否是初始化页面
25.     bool IsInitPage() const;
26.     /// 获取关系 (迁移/包含) 的起点
27.     RelEndpoint GetFrom() const noexcept;
28.     /// 获取关系 (迁移/包含) 的终点
29.     RelEndpoint GetTo() const noexcept;
30.
31. private:
32.     Rel() = default;
33.
34.     RelType rel_type_;
35.     RelEndpoint from_;
36.     RelEndpoint to_;
37.
38.     TransType trans_type_;
39.     TriggerReason trigger_reason_;
40.
41.     bool is_init_ = false;
42. };

```

图 2-5 Rel 数据结构

Rel 是一个描述 UIAbility 和 Page 之间关系的类 (如图 2-5 所示), 支持

创建包含关系和迁移关系（UIAbility 迁移或 Page 迁移），并提供方法获取关系类型、触发原因、起点终点等信息。内部通过私有成员变量存储关系类型、迁移类型、触发原因和是否为初始化页面等属性，用于管理 UI 组件和页面间的动态交互。

```

01. class UIAbility : public std::enable_shared_from_this<UIAbility> {
02. public:
03.     explicit UIAbility(std::string name);
04.     const std::string &GetName() const noexcept;
05.
06.     /// 创建 UIAbility 和 Page 之间的包含关系
07.     void AddCntmt(const std::shared_ptr<Page> &page, bool is_init = false);
08.
09.     /// 添加 UIAbility 之间的迁移关系
10.     void AddUIAbilityTrans(const std::shared_ptr<UIAbility> &to_ability,
11.                           TransType type, TriggerReason reason);
12.
13.     /// 获取当前对象所拥有的全部关系
14.     const std::vector<Rel> &GetRels() const noexcept;
15.
16. private:
17.     std::string name_;
18.     std::vector<Rel> rels_;
19. };

```

图 2-6 UIAbility 数据结构

UIAbility 表示一个 UI 功能单元（如图 2-6 所示），管理其与 Page 之间的包含关系和与其他 UIAbility 之间的迁移关系。它支持添加包含关系和迁移关系，提供方法获取名称及所有相关关系，内部通过 `std::vector<Rel>` 存储所有关系信息。

```

01. class Page : public std::enable_shared_from_this<Page> {
02. public:
03.     explicit Page(std::string name);
04.     const std::string &GetName() const noexcept;
05.
06.     /// 添加 Page 之间的迁移关系
07.     void AddPageTrans(const std::shared_ptr<Page> &to_page, TransType type,
08.                      TriggerReason reason);
09.
10.     /// 获取当前对象所拥有的全部关系
11.     const std::vector<Rel> &GetRels() const noexcept;
12.
13. private:
14.     std::string name_;
15.     std::vector<Rel> rels_;
16. };

```

图 2-7 Page 数据结构

Page 表示一个页面单元（如图 2-7 所示），支持管理与其他页面之间的迁移关系。它提供方法添加页面迁移关系、获取页面名称及其全部相关关系，内部通

过 `std::vector<Rel>` 存储所有关系信息。

```

01. struct ComponentInfoNode {
02.     // 组件类型的枚举
03.     enum class Type { SYSTEM, CUSTOM, UNKNOWN };
04.     // 系统组件子类型的枚举
05.     enum class SysSubType {
06.         BASIC,    /// 例如, Button, Text 等
07.         DUMMY,    /// 例如, for(n), if(state), if(action) 等
08.         MENU,     /// 菜单组件
09.         DIALOG,   /// 对话框组件
10.         UNKNOWN,
11.     };
12.     // 自定义组件子类型的枚举
13.     enum class CusSubType {
14.         PAGEROOT,    /// 由 @Component 装饰的根节点, 位于 pages/*.ets
15.         VIEWPUROOT,  /// 由 @Component 装饰的根节点, 位于 view 或 feature/component/*.ets
16.         BUILDER,     /// 由 @Builder 装饰的虚拟节点
17.         /// 理论上, BUILDER 包括 MENU 和 DIALOG
18.         MENU,        /// 菜单组件
19.         DIALOG,      /// 对话框组件
20.         UNKNOWN,
21.     };
22.     // 来自 ComponentTreeNode 的成员变量
23.     std::string component;    /// 组件名称
24.     std::string line_number;  /// 行号
25.     Type type;               /// 组件类型
26.     union SubType {
27.         SysSubType sys_sub_type;
28.         CusSubType cus_sub_type;
29.         SubType() : sys_sub_type(SysSubType::UNKNOWN) {}
30.         ~SubType() {}
31.     } sub_type;             /// 组件子类型
32.     std::string import_path;    /// 自定义组件的导入路径
33.     std::string text_content;   /// 组件的文本内容
34.     std::vector<std::string> attribute; /// 组件的属性
35.     json attributes;           /// 用于动态树结构的属性
36.     // 来自 ComponentIDHandlerNode 的成员变量
37.     std::string id_value;       /// ID 值
38.     std::string handler_type;   /// 处理器类型
39.     std::string handler_name;   /// 处理器名称
40.     const panda::defect_scan_aux::AbcFile *abc_file; /// ABC 文件对象
41.     const panda::defect_scan_aux::Function *handler_func; /// 事件处理器对象
42.     bool is_page_root;         /// 是否是页面根节点
43.     std::vector<ComponentInfoNode> children; /// 子组件列表
44.     std::string xpath;         /// 用于定位节点的 xpath
45.     std::string unique_xpath;  /// 唯一标识路径
46. };

```

图 2-8 ComponentInfoNode 数据结构

ComponentInfoNode 结构体表示静态组件树中的每个节点（如图 2-8 所示），包含了组件的基本信息，如组件类型、名称、行号及属性等。

```

01. struct ComponentInfoTree {
02.     ComponentInfoNode root; // 树的根节点
03.     std::vector<ComponentInfoNode> node_without_parent; // Menu/Dialog
04.     std::string name; // 树的名称 由 import_path + ';' + class_name 构成
05.     std::string file_name;
06.     std::string class_name;
07.     std::unordered_map<std::string,
08.     std::unordered_set<ComponentInfoNode, ComponentInfoNodeHash, ComponentInfoNodeEqual>> xpath2node; // xpath 到节点的映射
09.     std::vector<std::string> xpath_list; // xpath 集合
10.     std::string page_path; // 用于确定动态树所在页面
11.     std::unordered_map<ComponentInfoNode, ComponentInfoNode,
12.     ComponentInfoNodeHash, ComponentInfoNodeEqual> child2parent; // 子节点到父节点的映射
13.
14.     std::unordered_map<ComponentInfoNode,
15.     std::unordered_set<ComponentInfoNode, ComponentInfoNodeHash, ComponentInfoNodeEqual>,
16.     ComponentInfoNodeHash, ComponentInfoNodeEqual> node2possibleNodes; // 节点到可能节点的映射
17.
18.     ComponentInfoTree() = default;
19.
20.     // 将 ComponentTreeNode 转换为 ComponentInfoNode。
21.     ComponentInfoNode ConvertToComponentInfoNode(const ComponentTreeNode &tree_node);
22.
23.     // 将 parent2child 映射关系转换为组件树。
24.     void Map2Tree(const std::unordered_map<ComponentTreeNode, std::vector<ComponentTreeNode>,
25.     ComponentTreeNodeHash, ComponentTreeNodeEqual> &parent2child,
26.     std::string file_name, const std::string &record_name);
27.
28.     // 通过递归处理父子关系构建组件树。
29.     ComponentInfoNode BuildTree(const std::unordered_map<ComponentTreeNode, std::vector<ComponentTreeNode>,
30.     ComponentTreeNodeHash, ComponentTreeNodeEqual> &parent2child,
31.     ComponentTreeNode root, const std::string &record_name);
32.
33.     // 将子树合并到当前树中，扩展层级结构。
34.     void MergeTree(ComponentInfoNode& current, ComponentInfoNode& root, ComponentInfoNode& true_parent, size_t cnt);
35.
36.     // 为组件树生成 DOT 文件，从给定节点开始。
37.     void GenComponetTreeDot(ComponentInfoNode &current, std::ofstream &outFile, size_t &node_id,
38.     const std::string &parent_node_id = "");
39.
40.     // 构建 XPath 路径。
41.     void BuildXPath(ComponentInfoNode &current, ComponentInfoNode &parent);
42.
43.     // 打印树的结构。
44.     void PrintTree();
45. };

```

图 2-9 ComponentInfoTree 数据结构

ComponentInfoTree 结构体表示静态组件树（如图 2-9 所示），主要用于存储组件之间的父子关系和兄弟关系等。它提供的方法包括：将父子节点关系转换为组件树、递归构建树形结构、合并子树、生成可视化树结构的 DOT 文件、构建组件的 XPath 路径以及打印树结构等。其核心功能是通过管理组件树的节点关系、动态合并子节点和路径映射，帮助构建静态组件树。

第 3 章 静态分析能力的设计与实现

3.1 功能说明

基于 Panda IR 的静态分析模块旨在从方舟编译器（Ark Compiler）的编译工具链入手，该模块主要包括数据流分析、控制流分析、函数调用链分析、模块之间的导入、导出关系分析等核心功能。通过该模块，ArkUI Analyzer 实现了 Panda IR 指令到 UI 信息的逆向映射、ArkTS 模块之间 import 关系的分析、跨 ArkTS 模块级的数据流分析、函数之间调用关系的分析等基础静态分析能力。

3.2 Panda IR 指令到 UI 信息逆向映射的设计与实现

该部分逻辑对应 abc_file.cpp 文件中的 GetStringByInst() 函数。

在 ArkUI 中，Button 和 Text 等 UI 组件的代码在进入方舟编译器后，会经历逐步的编译过程。其间，ArkTS 代码先是会被转为 JS/TS 代码，然后进一步转为 Panda IR 指令（如图 3-1 所示）。

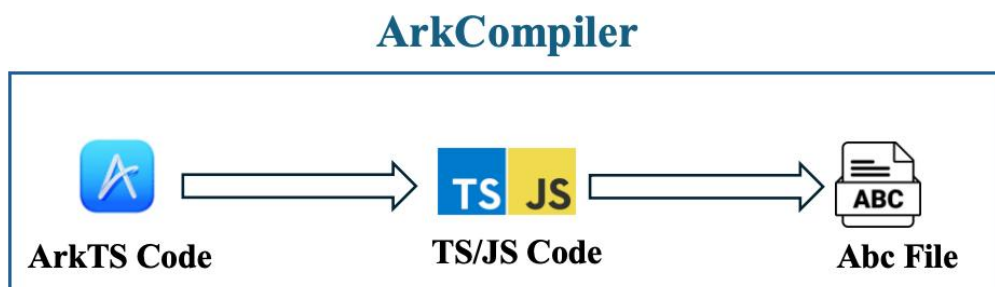


图 3-1 ArkTS 到 Abc 转换流程图

对于 Button() 这一 ArkTS 代码，其转为的 JS/TS 代码为 Button.create()，对应的 Panda IR 指令为 TRYLDGLOBALBYNAME_IMM8_ID16 和 LDOBJBYNAME_IMM8_ID16，具体含义为：首先，加载一个全局对象（即 Button）放入累加器中，然后，从累加器中提取 Button 对象的 create 方法放入累加器中，便于后续完成对 Button.create() 的调用。根据 Panda IR 指令中的 ID（索引），可以从方舟字节码文件的 Header[1] 结构中，确定 TRYLDGLOBALBYNAME_IMM8_ID16 指令加载的对象名称，也就是 UI 组件的名称。以此确定了该指令在 Ar

kTS 代码中的确切含义，从而实现了 Panda IR 指令到 UI 信息的逆向映射（如图 3-3 所示）。GetStringByInst 函数中的部分相关代码：

```

01. case InstType::TRYLDGLOBALBYNAME_IMM8_ID16:
02.   uint32_t string_id = inst.GetImms()[1];
03.   // 检查返回值是否为UI组件名称
04.   return GetStringByStringId(EntityId(string_id));
05.

```

图 3-2

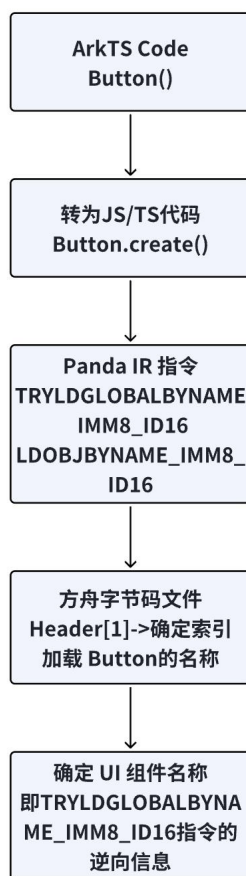


图 3-3 UI 信息逆向映射流程图

3.3 ArkTS 模块之间 import 关系分析的设计与实现

图 3-4 中展示了两个 ArkTS 模块之间的 import 关系。UserPage.ets 模块通过 import 引用了 UserInfoCard.ets 模块，并在其顶层组件 Row 中使用了来自 UserInfoCard.ets 的 UserInfoCard 组件。ArkUIAnalyzer 能够捕获这种模块间的依

赖关系，同时解析各模块的组件树结构，如 `UserInfoCard.ets` 中 `Row` 组件包含 `Text` 和 `Image`，以及 `UserPage.ets` 中 `Row` 组件包含 `UserInfoCard`、`Column` 和 `Button`，从而清晰地反映模块间的依赖和层次结构关系。

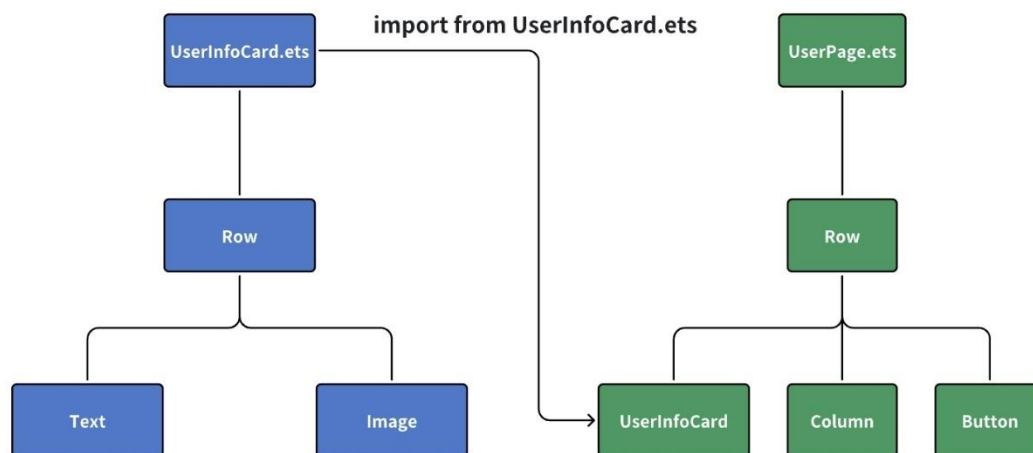


图 3-4 模块之间的 import 关系示例图

该部分逻辑对应 `abc_file.cpp` 中的 `ExtractModuleRecord` 函数。

该函数是方舟编译器的内置模块，主要实现思路同“从 Panda IR 指令逆向到 UI 信息”类似，也是从方舟字节码文件中解析目标信息，这里的目标信息即模块之间的 import 关系。

3.4 跨 ArkTS 模块数据流分析的设计与实现

图 3-5 展示了跨 ArkTS 模块的数据流分析过程。`Constant.ets` 模块中定义了一个静态变量 `modvar`，其值为字符串 `"pages/target_page"`，通过 `export` 将其导出。`Home.ets` 模块通过 `import` 语句引入了 `modvar`，并在组件 `Index` 的 `onClick` 事件中使用了 `router.pushUrl` 调用，将 `modvar` 作为参数传递。

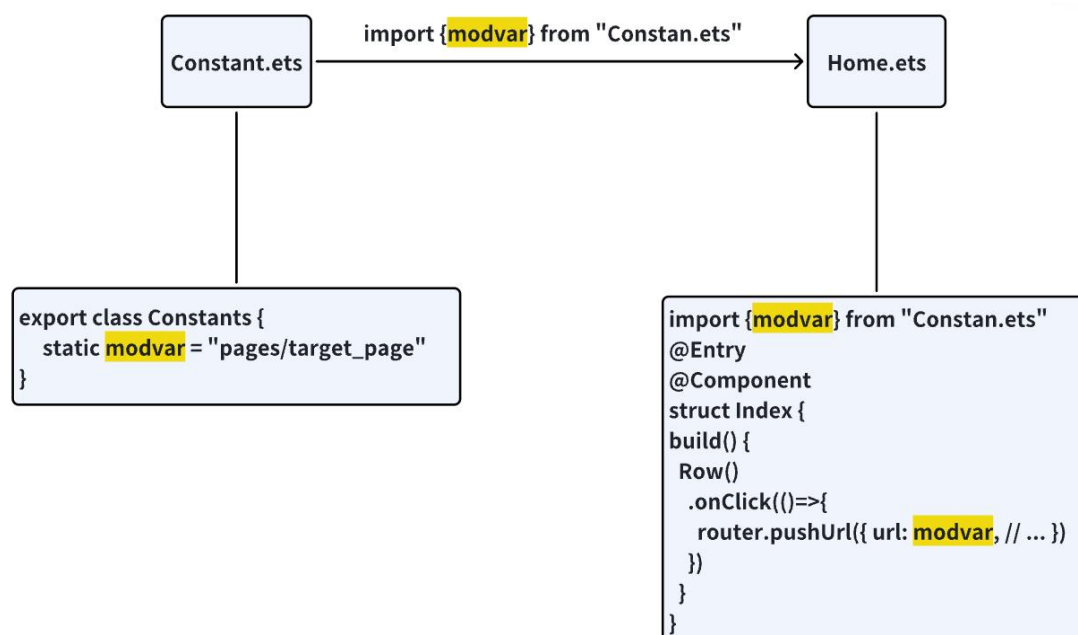


图 3-5 跨模块数据流分析示例图

该部分逻辑对应 `analyze_modvar_definition.cpp` 文件。在鸿蒙应用开发规范中，常量通常集中存储在一个独立的 ArkTS 文件中，以用于定义路由的目标位置等内容。因此，对于 UI 迁移行为的分析，跨模块的数据流分析能力尤为重要。通过解析 Panda IR 中指令级别的 `input-user` 关系，并结合 Panda IR 的语义信息，构建模块变量的基础的定义 - 使用关系。具体实现可以参考代码文件 `analyze_modvar_definition.cpp`。如图 3-6 为例，具体说明。

```

01. // file: constant.ets
02. export class Constants { static modvar = "pages/target_page" }
03.
04. // file: custom_component.ets
05. import { modvar } from "constant.ets"
06. router.pushUrl({ url: modvar, // ... })
  
```

图 3-6 跨模块引入变量并使用的伪代码

首先，在遍历函数调用链时，分析到 `pushUrl` 这一 API 后（相关逻辑参见 `analyze_ui_state_transition_graph.cpp` 中的 `AnalyzeTargetGraph` 函数），根据 `pushUrl` 的使用规则，可以确定其输入之一为 `CREATEOBJECTWITHBUFFER` 指令，这对应源码中的 `{ url: modvar, // ... }` 部分。此外，结合 `CREATEOBJECTWITHBUFFER` 指令的语义，可以确定 `CREATEOBJECTWITHBUFFER` 指令的使用者之一必然是与 `url` 对应的指令。

当定位到 url 对应的指令后，如果 mod_var 是模块变量的话，便进入该模块变量的定义点所在的函数，解析相关指令的输入关系。相关逻辑依次对应：analyze_ui_state_transition_graph.cpp 中的 AnalyzeUrldefinition 函数、analyze_mod_var_definition.cpp 中 AnalyzeModVarDef 函数。此时，可能存在以下三种情况。

1. 输入指令为 LOADSTRING 指令

在这种情况下，可以直接分析被 load 的字符串内容，以确定 url 的定义值。相关逻辑对应到 GetClosestLoadString 函数中的 if (input_inst_type == panda:defect_scan_aux:InstType:OPCODE_LOADSTRING) 分支。

2. 输入指令为 PHI 指令

当输入是 PHI 指令时，需要解析 PHI 指令的输入关系。随后，沿着 PHI 指令的每一个输入分支，分别追踪距离最近的 LOADSTRING 指令。相关逻辑对应到 GetClosestLoadString 函数中 if (input_inst_type == panda:defect_scan_aux:InstType:OPCODE_PHI) 分支。

3. 输入指令为外部参数指令

如果输入是外部参数指令，则需要构建一个局部的跨过程间控制流图，并追踪到其对应的实参指令所在的函数。在该函数中，沿着实参指令的输入路径，寻找距离实参指令最近的 LOADSTRING 指令。相关逻辑对应到 GetClosestLoadString 函数中的 if (input_inst_type == panda:defect_scan_aux:InstType:OPCODE_PARAMETER) 分支。

3.5 函数调用关系分析的设计与实现

该部分逻辑对应 analyze_call_chain.cpp 文件。

在 callee_info.cpp 文件中，提供了基本的函数调用链分析能力。这是方舟编译器内置的模块，然而，对于涉及对象方法调用的分析，该模块存在一定的不足。以下是针对这一不足的改进和完善。

首先是优化局部变量的 load 和 store 依赖关系的分析，通过基于工作列表 (worklist) 的分析算法，结合一定的启发式策略，尽可能准确地识别并关联最近的 store 指令，构建与之相关的 load 指令之间的数据依赖关系。相关逻辑对应

到 `abc_file.cpp` 中的 `GetStLexInstByLdLexInst` 函数。

在此基础上，然后分析 JavaScript/TypeScript 中 `this` 指针的类指向。如果发现最近的 `store` 指令的输入是与 `new class` 相关的指令，则进一步判断这些指令的输入是否加载了当前模块的类，或加载了外部模块的类，从而识别出调用的类名和方法。最终，这种分析能够更加完整和准确地识别涉及对象方法的调用关系，提升静态分析的精确性。

`newclass` 相关的指令类型如下：`CREATEEMPTYOBJECT`、`CREATEOBJECTWITHBUFFER`、`CREATEOBJECTWITHEXCLUDEDKEYS`、`NEWOBJAPPLY`、`NEWOBJRANGE`、`WIDE_NEWOBJRANGE_PREF`。

例如，在静态分析过程中，当遇到 `CALL` 指令时，会对该指令的指令级别的输入流进行分析。如果定位到 `LDOBJBYNAME` 指令，这意味着这是一次对象方法的调用。通常情况下，`LDOBJBYNAME` 指令的 `input` 之一为 `LDLEXVAR` 指令（即 `this`）。进一步地，可以通过 `GetStLexInstByLdLexInst` 函数找到与之对应的 `store` 指令。具体到实现中，接下来的步骤是调用 `AnalyzeClassMethod` 函数，继续从 `store` 指令的 `input` 中追踪与 `newclass` 相关的指令。通过这样的分析流程，最终能够确定该方法是哪个类的方法。

第 4 章 鸿蒙应用功能地图的设计和实现

4.1 功能说明

该模块旨在从基于 Panda IR 的静态分析模块入手，分析应用中 UIAbility 之间的 UI 迁移关系、各 Page 之间的 UI 迁移关系，以及 Page 内部组件之间的 UI 迁移关系和结构关系，构建出鸿蒙应用的功能地图。该功能地图主要用以引导动态测试工具的 UI 自动化探索过程。通过提供路径信息，使动态测试工具能够有针对性地覆盖关键路径，以及探索此前动态测试中尚未遍历到的路径。

4.2 UI-Import Graph 分析的设计与实现

该部分逻辑对应 `analyze_ui_dependency.cpp` 文件。

基于之前对模块之间 `import` 关系的分析，进一步分析 Page、自定义组件、Model 层之间的 `import` 关系。这两者的关系在于：Page、自定义组件、Model 层之间的 `import` 关系是模块之间 `import` 关系的一个子集。在此基础上，我们参考鸿蒙应用的基本开发规范^{[20][21]}：页面文件统一存放在 `pages` 目录，自定义组件存放在 `view` 或 `feature/components` 目录，Model 层存放在 `viewmodel` 或 `model` 目录。通过这种目录结构，我们进一步实现了对页面与自定义组件之间 `import` 关系的分析。

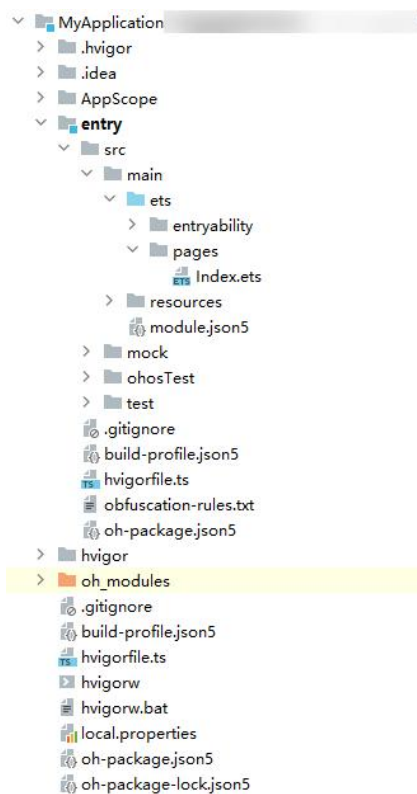


图 4-1 ArkTS 工程目录结构 (Stage 模型)

如图 4-2 所示是页面与自定义组件之间导入、导出关系的示例图:



图 4-2 页面与自定义组件 import 关系示例图

4.3 组件树分析的设计与实现

该部分逻辑对应 analyze_component_tree.cpp 文件。

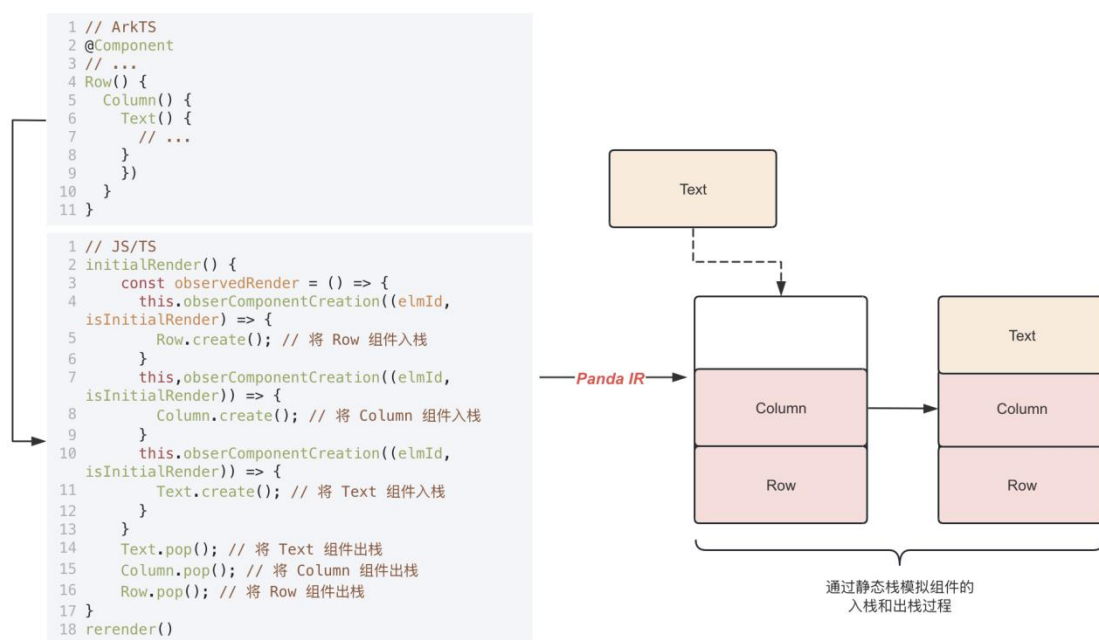


图 4-3

如图 4-3 所示，首先，ArkTS 代码被编译器翻译为 JS/TS 代码，再转为 Panda IR。在此过程中，JS 中的 `initialRender` 函数负责构建 UI 组件树，每个组件都对应一个 `observeComponentCreation` 函数，该函数内包含与组件属性定义相关的代码，例如 `Text.id()` 和 `Text.onClick()`。

在 ArkUI 引擎中，组件的层次结构关系通过一个栈来构建。具体而言，`create()` 函数用于将 UI 节点推入栈，当节点构建完成后，再通过 `pop()` 函数将其移出栈。这种入栈和出栈的顺序体现了组件的层次结构关系（父子关系和兄弟关系）。具体分析逻辑对应到 `AnalyzeBuildFunc` 函数。

因此，在基于 Panda IR 进行分析时，我们使用一个静态栈来模拟组件的入栈和出栈过程，从而得出组件之间的层次关系。具体可参见 `ComponentTreeContext` 结构体。在栈中，每个元素被视为其上方直接相邻元素的“父节点”，从而形成一种层级关系。具体而言，栈中的每个元素都与其上方的元素构成父子层级。例如，当栈中初始包含元素 A，将元素 B 压入栈后，B 被视为 A 的子节点。当 B 从栈中弹出，再将元素 C 压入栈时，C 也被视为 A 的子节点。此外，B 和 C 之间则具有兄弟节点的关系。



图 4-4 动态渲染下的组件关系示例图



图 4-5 静态组件树示例图

这里的组件树描述了 UI 组件之间的层次关系，具体指的是 UI 页面中组件按照父子或兄弟关系组织的方式。例如，如图 4-4 所示，在一个备忘录应用的页面中，根节点作为整个页面的容器，包含多个关键组件。顶部的标题栏包括菜单按钮和添加按钮，分别用于打开侧边栏和跳转到添加笔记页面。标题栏下方是搜索框，帮助用户快速查找笔记内容。接着是笔记列表区域，展示多个笔记项，每个笔记项包含标题、时间等信息，点击后可跳转到相应的笔记详情页面。此外，页面还包含未渲染的子树结构，如笔记分类侧边栏和笔记内容，这些内容在当前页面不可见，但可能通过用户操作动态加载或显示。

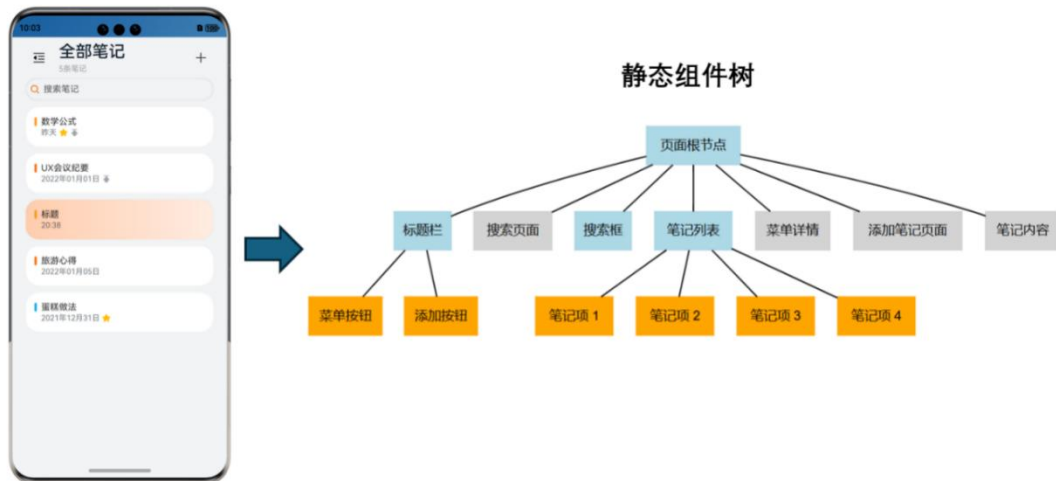


图 4-6 备忘录应用首页组件树示例图

针对如图 4-6 所示的 UI 界面状态，我们希望构建相应的静态组件树（右侧）。为了简化表达与分析，构建过程对该组件树图进行了适当的调整，因此与实际工具分析出的结果存在一定差异。在图中，蓝色节点和橙色节点表示当前页面已渲染的组件，橙色节点特别指代具有 UI 迁移行为的组件；而灰色节点则表示在源码中存在但未在当前页面渲染的组件。

静态组件树关注的是页面所有组件的完整结构，包括已渲染和未渲染的部分。而动态组件树则关注运行时的实际状态，仅包含当前页面中已渲染并可见的组件，并会随着用户操作实时更新。例如，当用户点击菜单按钮或笔记项时，动态组件树会动态添加或移除相应节点，以反映界面的变化。

4.4 UI 迁移关系分析的设计与实现

该部分逻辑对应 `analyze_ui_state_transition_graph.cpp` 文件。具体 UI 迁移关系可分为 —— UIAbility 之间的 UI 迁移关系、Page 之间的 UI 迁移关系、Page 内部的组件之间的 UI 迁移关系（例如弹窗、菜单等）。

首先，收集应用中所有的生命周期函数和事件处理函数。接着，以这些生命周期函数和事件处理函数为起点，沿着函数调用链递归遍历每个函数的控制流图，借助 `callee name` 等信息以定位与 UI 迁移相关的 API（如 `startAbility`、`terminateSelf` 等）。一旦定位到目标 API，再结合 API 的使用规则，分析 UI 迁移的起始位置和目标位置。

```

01.   if (callee_name.find("startAbility") != std::string::npos ||
02.       callee_name.find("terminateSelf") != std::string::npos) {
03.       // 继续沿着相关指令的输入流定位到 getContext 相关指令,
04.       // 从而确定这是涉及UIAbility启动、关闭相关的API
05.   }

```

图 4-7 UIAbility 代码示例图



图 4-8 UIAbility 状态迁移图

以 router.pushUrl (Page 之间的 UI 迁移) 为例具体说明分析思路:

```

01.   // constant.ets
02.   export class Constants { static modvar = "pages/target_page" }
03.
04.   // custom_component.ets
05.   import { modvar } from "constant.ets"
06.   router.pushUrl({ url: modvar, // ... })
07.
08.   // source_page.ets
09.   import { custom_component } from custom_component

```

图 4-9

步骤 1: 确定 UI 迁移的目标位置。在 custom_component.ets 文件中, 分析 pushUrl 中的 url 值。url 的赋值可能是字面量或变量。如果是字面量, 那么该值即为 UI 迁移的目标位置, 相关逻辑对应 ExtractTargetPage 函数; 如果是变量, 则需要沿着变量的定义 - 使用链, 逐步查找其定义值, 相关逻辑对应 AnalyzeUrldefinition 函数。例如, 如果该变量是模块变量, 则需要在 constant.ets 文件中进一步分析其定义, 找到最终的赋值以确定目标位置。

步骤 2: 确定 UI 迁移的起始位置。因为 router.pushUrl 出现在自定义组件文件中, 所以通过分析页面与自定义组件的导入/导出关系, 可以找出哪些页面导入了该自定义组件。所有导入该组件的页面即为 UI 迁移的起始位置, 例如 source_page 页面。

步骤 3: 在确定 Page 之间的 UI 迁移起始位置和目标位置后, 需要在迁移路径上标注以下信息: 是哪个 UIAbility、Page 或 Component 的生命周期函数或是对哪个组件的操作 (如单击、双击等) 触发了此次 UI 迁移? 迁移过程中, 是否会关闭当前页面, 再跳转到目标页面? 为实现步骤 3 中描述的这一需求, 整个分析过程中将维护两类信息:

1. 以生命周期函数或事件处理函数为起点的调用链。若调用链上的某个函数 F 调用了 `pushUrl`, 可以直接追溯到 F 的调用链起点。
2. 针对事件处理函数, 维护其上下文信息, 包括: 触发该事件处理函数的组件类型、该函数在 JS/TS 源代码中的行号以及组件的 id 等。如果组件的 id 为空, 则根据之前对 Page 的组件树的分析结果, 使用 XPath 来唯一标识该组件。

图 4-10 是 Page 之间的 UI 迁移示例图, 图 4-11 是 UIAbility 之间的 UI 迁移示例图, 这里, `action` 字段表示对组件的操作类型, 而 `opt` 字段则用于描述 UI 迁移的效果, 即是否在跳转到目标页面前关闭当前页面。

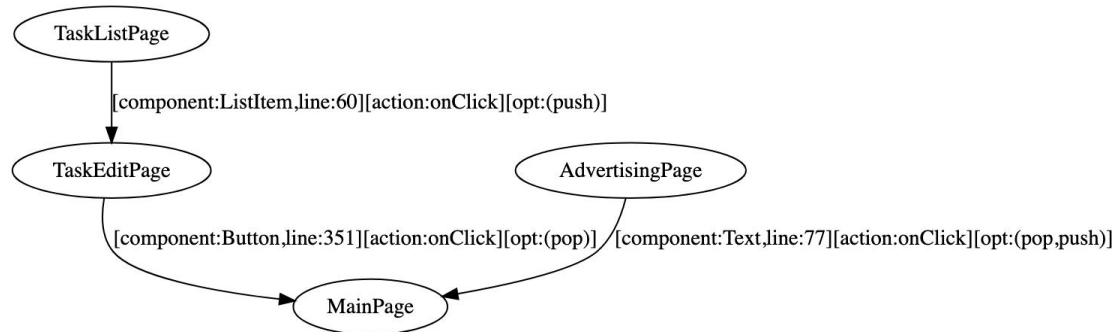


图 4-10 Page 之间的 UI 迁移示例图

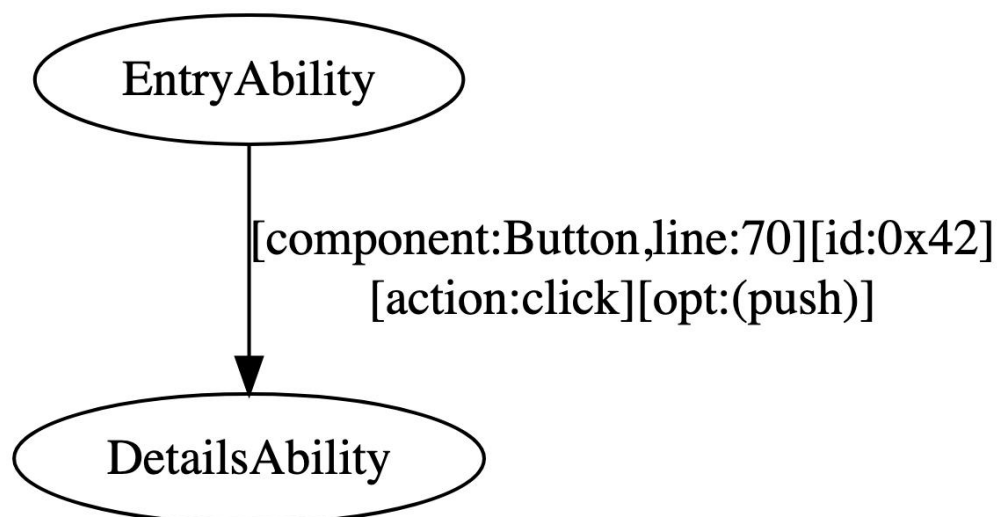
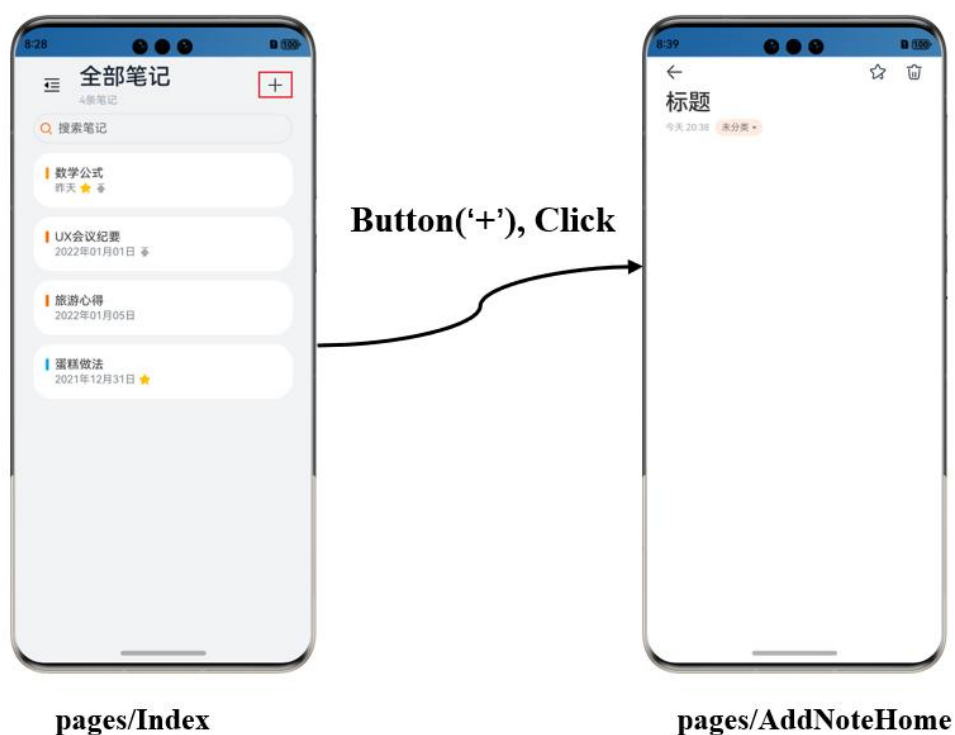


图 2-11 UIAbility 之间的 UI 迁移示例图

Page Transition Graph (PTG) 描述了鸿蒙原生应用中各个 UI 页面之间的跳转关系。例如，如图 4-12 所示，当用户在 `pages/Index` 页面上点击右上角的“+”按钮时，应用通过事件处理函数 `onClick` 实现页面跳转，转到 `pages/AddNoteHome` 页面。

图 4-12 页面跳转示例：从 `pages/Index` 到 `pages/AddNoteHome`

为了分析 Page Transition Graph (PTG)，我们将首先分析页面中 UI 组件和

事件处理函数之间的映射关系。接着，对于事件处理函数（如 `onclick` 等）中涉及页面跳转的函数（如 `router.pushUrl`、`router.replaceUrl`、`router.back` 等），我们会从 Panda IR 中解析目标页面的路由，建立起 `button - onclick - source page - target page`（即 `source page` 上的 `button` 被点击之后，将跳转到哪个 `target page`）之间的关系。

由于静态分析的误报可能会导致 PTG 中出现动态运行时实际不可达的边，我们将在 Page Transition Graphh 中引入 Path Validity 处理机制，该机制将尽可能修正那些动态运行时不可达的边。为了实现这一目标，我们将在页面跳转过程中模拟一个静态页面栈。例如，对于 `router.pushUrl` 路由，我们将在相应的页面跳转边上标注 `[push target page]` 信息。对于 `router.replaceUrl` 路由，我们将在相应的页面跳转边上标注 `[pop source page, push target page]` 信息。最后，我们会逐一遍历 Page Transition Graphh 中的路径，并根据路径上边的栈操作信息（即 `pop` 操作和 `push` 操作）进行模拟，以尽可能修正动态运行时不可达的边。

例如，对于从 `page1` 到 `page6` 的路径（`page1` 经过一系列跳转后到达 `page6`），此时，相应的栈操作标注是：`[push page2, push page3, pop page3, push page6]`，然后对于 `page6` 页面的 `router.back` 路由，静态分析的结果指向目标 `page3` 页面（如图 4-13 所示）。经过模拟从 `page1` 到 `page6` 的这些栈操作后，发现此时的栈中的元素自低向上依次是 `page2` 页面、`page6` 页面，因此，对于 `page6` 页面的 `router.back` 路由之后目标应该是 `page2` 页面而不是 `page3` 页面。由此，说明该静态分析所得到的该条路径是存在问题的，我们将按照栈操作的模拟结果进行相应修正（如图 4-14 所示）

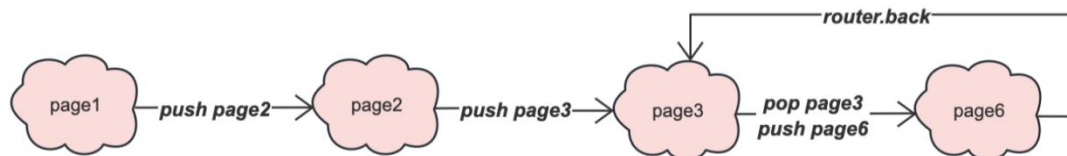


图 4-13 PTG Before Validating Path

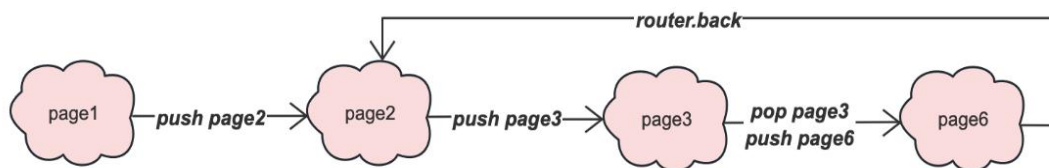


图 4-14 PTG After Validating Path

此外，对于 Page 内的 UI 迁移行为，也是采取了这样的分析思路，这里给出分析之后的效果示例图。

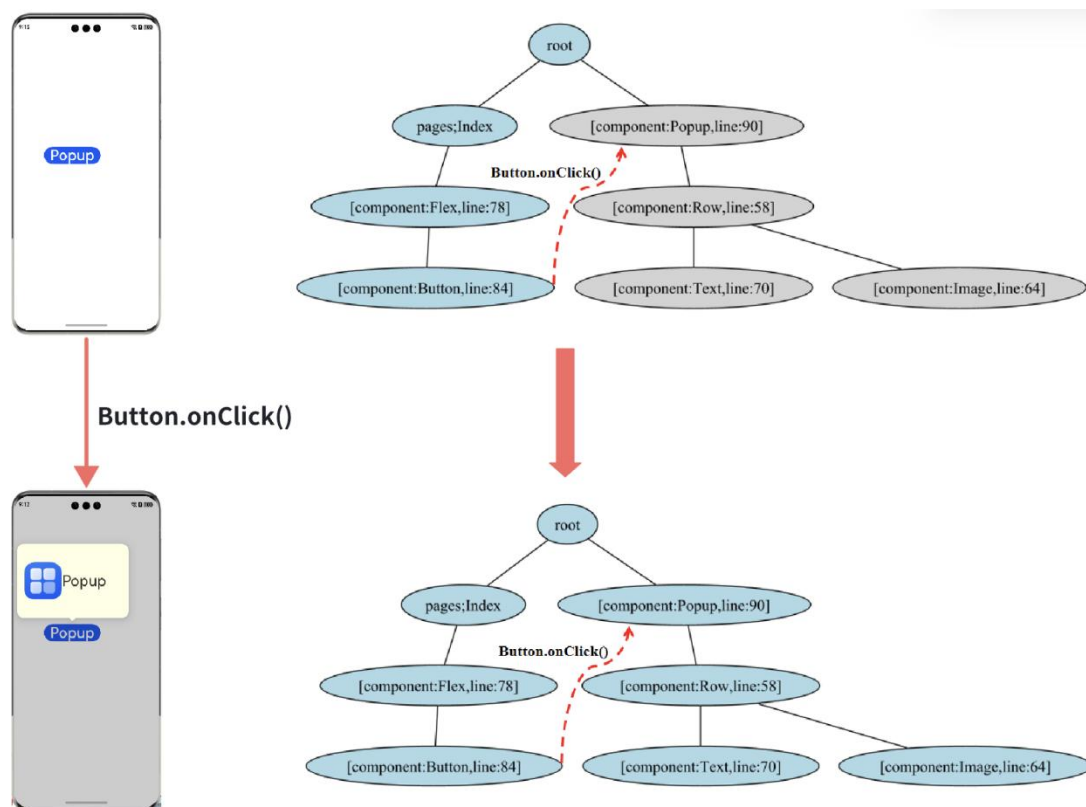


图 4-15 Page 内的 UI 迁移行为示例图

第 5 章 项目测试

本小节将介绍 ArkUI Analyzer 项目的项目测试结果。

测试平台为单台物理机，该机的硬件配置及操作系统如下：

- 内存：16G
- 核数：8 核
- SSD：250G
- 操作系统：Ubuntu 22.04.4 LTS
- 内核版本：GNU/Linux 5.15.0-113-generic x86_64

测试集来源于华为鸿蒙官方开发文档中提供的最佳开源应用实践“一次开发，多端部署”，涵盖了丰富的 ArkUI 特性。具体内容可参考：<https://developer.huawei.com/consumer/cn/doc/best-practices-V5/bpta-develop-once-deploy-everywhere-V5>。

5.1 单元测试

在 /workspace/Ark Compiler/runtime_core/libark_defect_scan_aux/tests/unittest/ 目录下存放着 js 单元测试脚本：async_function_test.js、callee_info_test.js、debug_info_test.js、define_info_test.js、graph_test.js、module_info_test.js，主要用于测试 ArkUI Analyzer 静态分析工具的基础分析能力，包括模块之间 import 关系分析、函数调用链分析等。

5.2 核心功能测试

5.2.1 代码功能分析完整性测试

代码功能分析完整性评测标准定义：静态分析找到的 UI 迁移行为事件数 / 人工识别的 App 全量 UI 迁移行为事件数。人工识别方法：通过人工遍历 App 界面，逐一识别和统计所有的 UI 迁移行为数。关于“静态分析找到的 UI 迁移行为数”的说明：对 UI 迁移行为数的统计，本质上是通过通过对 abc 字节码进行静态分

析，统计出 UI 迁移行为相关 API 的数量。

结合鸿蒙应用开发文档，梳理的几类 UI 迁移行为及其相关 API 如下：

UI 迁移行为的类型	对应的 API
页面路由、导航（点击行为）	<p>(1) router: pushUrl、pushNamedRoute、replaceNamedRoute、replaceUrl、back、showAlertBeforeBackPage</p> <p>(2) navagation: pushPath、pushPathByName、pushDestination、pushDestinationByName、replacePath、replacePathByName、removeByIndexes、removeByName、removeByNavDestinationId、pop、popToName、popToIndex、moveToTop、moveIndexToTop、clear</p> <p>(3) 通过 `visibility` 属性来实现的 UI 界面跳转</p>
关于 UIAbility 的跳转（点击行为）	startAbility、startAbilityForResult、terminateSelf、terminateSelfWithResult
菜单的打开或者关闭（点击行为）	bindMen、bindContextMenu、ContextMenu.close
弹窗的打开或者关闭（点击行为）	<p>AlertDialog.show、ActionSheet.show、CalendarPickerDialog.show、DatePickerDialog.show、TimePickerDialog.show、TextPickerDialog.show、this.getUIContext().showAlertDialog、this.getUIContext().showActionSheet、this.getUIContext().showCalendarPickerDialog、this.getUIContext().showDatePickerDialog、this.getUIContext().showTimePickerDialog、this.getUIContext().showTextPickerDialog、以及</p>

	通过 CustomDialogController 类的 open、close 方法所控制的弹窗的打开或者关闭
切换行为	通过 Tab、Stepper、ListItem 组件触发的切换行为
轮播行为	通过 Swiper 组件触发的轮播行为
滚动、滑动行为	通过 Scroll 组件、List 组件、Grid 组件触发的滚动行为
拖拽行为	onDragStart 、 onDragEnter 、 onDragMove 、 onDragLeave、onDrop、onDragEnd、onPreDrag
组件的区域、尺寸变化	onAreaChange、onSizeChange、onVisibleAreaChange

表 5-1 UI 迁移行为类型与对应 API 表

如下实验流程图，体现了在实验过程中，我们是如何建立一个映射，将动态遍历期间观察到的 UI 界面与静态分析得到的可迁移行为关联起来的。

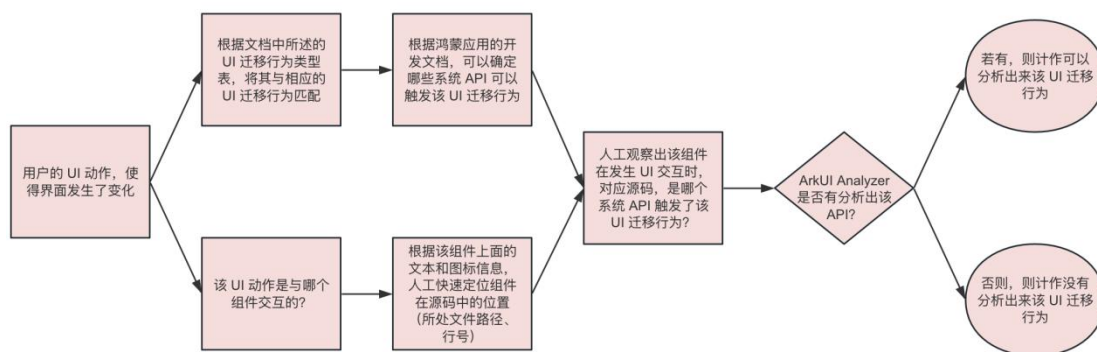


图 5-2 实验流程图

如图 5-2 所示，以此为例，具体说明。

当用户点击热搜、剧场、电影等标签时，会跳转到该标签对应的内容界面（建筑人文事迹等）。这明显属于切换行为，根据表 5-1 UI 迁移行为类型与对应 API 表 UI 迁移行为类型与对应 API 表所述，我们可获知 Tab、Stepper 等组件可触发切换行为。然后我们根据“热搜”这样一个文字标识，加以人工阅读、调试被测应用的相关代码，以定位“热搜”标签所在的代码文件路径（P）和行号（L）。如此，人工便可观察出“热搜”标签被点击时，对应源码，具体是哪一个组件 C

被点击，以及该组件的文件路径 P、行号 L 的具体值。然后，以此作为判定标准，去静态分析得到的结果中检查，在文件路径 P、行号 L 这样一个约束下，是否有分析出一个组件 C，其能够触发切换行为。



图 5-3

开源应用	静态分析找到的 跳转事件数	人工识别的界面 跳转事件数	代码功能分析 完整性
一多开发-长视频	32	34	93%
一多开发-即时 通讯	38	40	95%
一多开发-社区 评论	25	26	96%
一多开发-银行 理财	67	71	94%
一多开发-购物 比价	66	71	92%
一多开发-股票类	23	23	100%

一多开发-移动支付	5	5	100%
一多开发-音乐	40	42	95%
一多开发-旅行订票	9	9	100%
一多开发-商务办公	44	44	100%

表 5-4 代码功能分析完整性评估对照表

5.2.2 代码功能关联映射准确率测试

代码功能关联映射准确率的定义：静态分析找到的有效控件与跳转关系 pair 对/人工提取的 App 全量控件与界面跳转 pair 对。人工提取方法：通过人工遍历源码，逐一识别和统计控件与界面跳转 pair 对。

开源应用	静态分析找到的有效控件与跳转关系 pair 对	人工提取的 App 全量控件与界面跳转 pair 对	代码功能关联映射准确率
一多开发-长视频	32	34	93%
一多开发-即时通讯	38	39	97%
一多开发-社区评论	25	25	100%
一多开发-银行理财	67	69	97%
一多开发-购物比价	66	71	92%
一多开发-股票类	23	23	100%
一多开发-移动	5	5	100%

支付			
一多开发-音乐	40	42	95%
一多开发-旅行 订票	9	9	100%
一多开发-商务 办公	44	44	100%

表 5-5 代码功能关联映射准确率评估对照表

5.3 回归测试

为了保障开发规范与开发质量，我们编写了回归测试脚本，每增加一个功能点，会进行一次回归测试，以保证添加的新功能点没有引入新的 bug。在/Ark Compiler/runtime_core/libark_defect_scan_aux/tests/apptest/apps/目录下存放待分析的 abc 文件（以项目名称命名各个 abc 文件），执行位于/workspace/Ark Compiler/runtime_core/libark_defect_scan_aux/tests/apptest/script/目录下的 apptest.sh 脚本，将会执行一次 ArkUI Analyzer 工具的编译，编译成功后会执行 apptest.js 脚本，分析 apps 目录下的 abc 文件，并与前一次回归测试的分析结果进行差分测试，输出每个 abc 文件对应的静态分析结果和 diff 差异结果。

参考文献

- [1] 华为鸿蒙系统. <https://baike.baidu.com/item/华为鸿蒙系统/23500650>
- [2] Ting Su, Jue Wang, Zhendong Su: Benchmarking automated GUI testing for Android against real-world bugs. ESEC/SIGSOFT FSE 2021: 119-130
- [3] Wenyu Wang, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng, Tao Xie: An empirical study of Android test generation tools in industrial cases. ASE 2018: 738-748
- [4] Zhengwei Lv, Chao Peng, Zhao Zhang, Ting Su, Kai Liu, Ping Yang: Fastbot2: Reusable Automated Model-based GUI Testing for Android Enhanced by Reinforcement Learning. ASE 2022: 135:1-135:5
- [5] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, Zhendong Su: Guided, stochastic model-based GUI testing of Android apps. ESEC/SIGSOFT FSE 2017: 245-256
- [6] Zhe Liu, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, Qing Wang: Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing. ICSE 2023: 1355-1367
- [7] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, Qing Wang: Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. ICSE 2024: 100:1-100:13
- [8] Yuanchun Li, Ziyue Yang, Yao Guo, Xiangqun Chen: DroidBot: a lightweight UI-guided test input generator for Android. ICSE (Companion Volume) 2017: 23-26
- [9] Soot: <https://github.com/soot-oss/soot>
- [10] FlowDroid: <https://github.com/secure-software-engineering/FlowDroid>
- [11] arkXtest: https://gitee.com/openharmony/testfwk_arkxtest
- [12] Wukong: https://gitee.com/openharmony/ostest_wukong
- [13] Shengqian Yang, Haowei Wu, Hailong Zhang, Yan Wang, Chandrasekar

- Swaminathan, Dacong Yan, and Atanas Rountev. 2018. Static window transition graphs for Android. *Automated Software Engg.* 25, 4 (December 2018), 833–873.
- [14] Duling Lai and Julia Rubin. 2020. Goal-driven exploration for Android applications. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*. IEEE Press, 115–127. <https://doi.org/10.1109/ASE.2019.00021>
- [15] Atanas Rountev, Dacong Yan, "Static Reference Analysis for GUI Objects in Android Software"CGO '14: Proceedings of Annual IEEE/ACM International Symposium on Code Generation and OptimizationFebruary 2014 Page 143-153 doi: <https://doi.org/10.1145/2581122.2544159>.
- [16] S. Yang, H. Zhang, H. Wu, Y. Wang, D. Yan and A. Rountev, "Static Window Transition Graphs for Android (T)," 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 2015, pp. 658-668, doi: 10.1109/ASE.2015.76.
- [17] S. Yang, D. Yan, H. Wu, Y. Wang and A. Rountev, "Static Control-Flow Analysis of User-Driven Callbacks in Android Applications,"2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 2015, pp. 89-99, doi: 10.1109/ICSE.2015.31.
- [18] 方舟字节码文件格式: <https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-bytecode-file-format-V5>
- [19] 方舟字节码的基本原理: <https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-bytecode-file-format-V5>
- [20] Stage 模型的应用程序包结构: <https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/application-package-structure-stage-V5>
- [21] ArkTS 工程的目录结构: <https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/ide-project-structure-V5>
- [22] 页面路由、导航相关:<https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/js-apis-router-V5>

- [23] UIAbility 跳转相关:<https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/js-apis-inner-application-uiabilitycontext-V5>
- [24] 菜单相关:<https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/ts-universal-attributes-menu-V5>
- [25] 弹窗相关 <https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/js-apis-promptaction-V5>、https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/4_17_u5f39_u7a97-V5、<https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/ts-methods-custom-dialog-box-V5>
- [26] 切换行为相关 <https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/ts-basic-components-stepper-V5>、<https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/ts-container-tabs-V5>
- [27] 轮播行为相关 <https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/ts-container-swiper-V5>
- [28] 滚动、滑动行为相关 https://developer.huawei.com/consumer/cn/doc/harmonyos-references-V5/4_4_u6eda_u52a8_u4e0e_u6ed1_u52a8-V5

附录 1：编译与使用教程

本节将重点阐述在 Ubuntu 20.04 环境中编译与使用 ArkUIAnalyzer 工具的教程。需要注意的是，本节仅提供大致流程说明，实际操作可直接运行 `/script/build_arkui_analyzer.sh` 脚本。

环境准备

- 操作系统： Ubuntu 20.04
- 硬盘空间： 至少 200 GB 空间
- OpenHarmony V5.0.0 Release

编译 OpenHarmony 操作系统及相关工具需要大量的硬盘空间，特别是在首次拉取代码和编译过程中。因此，建议确保至少有 200 GB 空间，以避免由于硬盘空间不足导致编译失败。

快速部署

1. 建议使用 Docker 搭建环境，拉取并启动 Ubuntu 20.04 容器。
2. 在容器中安装以下必要依赖：

```
01. sudo apt update -y && \  
02.     DEBIAN_FRONTEND=noninteractive sudo apt install -y \  
03.     bc bison build-essential ccache cpio curl default-jdk default-jre flex g++-multilib gcc-arm-none-eabi \  
04.     gcc-multilib gdb genext2fs git git-lfs gnupg lib32ncurses-dev lib32z1-dev libc++1 \  
05.     libc6-dev-i386 libelf-dev libgl1-mesa-dev liblz4-tool libssl-dev libtinfo5 libx11-dev libxcursor-dev \  
06.     libxinerama-dev libxml2-utils libxrandr-dev m4 mtools mtd-utils npm openjdk-8-jre-headless python3-pip ruby \  
07.     scons u-boot-tools unzip wget x11proto-core-dev xsltproc zip zlib1g-dev
```

图 0-1

3. 配置 Git 用户信息：

```
01. git config --global user.name "ubuntu"  
02. git config --global user.email "ubuntu@localhost"
```

图 0-2

4. 初始化 OpenHarmony 源代码仓库：

```

01. if [ ! -d ".repo" ]; then
02.     repo init -u https://gitee.com/openharmony/manifest -b refs/tags/OpenHarmony-v5.0.0-Release --no-repo-verify
03. fi
04. repo sync -c
05. repo forall -c 'git lfs pull'

```

图 0-3

5. 拉取并同步 ArkUI Analyzer 仓库:

```

01. if [ ! -d "arkui_analyzer" ]; then
02.     git clone https://gitee.com/ml_m/arkui_analyzer
03. else
04.     cd arkui_analyzer/
05.     git pull
06.     cd ..
07. fi

```

图 0-4

6. 调整文件与目录结构: 删除原有的 libark_defect_scan_aux 目录, 并将 arkui_analyzer 重命名为 libark_defect_scan_aux 以适配编译需求然后替换编译文件 BUILD.。

```

01. rm -rf libark_defect_scan_aux
02. mv arkui_analyzer libark_defect_scan_aux
03. rm -rf BUILD.gn
04. mv libark_defect_scan_aux/script/BUILD.gn libark_defect_scan_aux/BUILD.gn

```

图 0-5

7. 下载预构建工具:

```

01. cd /workspace/build/
02. export PATH=~/.local/bin:$PATH
03. sudo chmod +X prebuilts_download.sh
04. sudo ./prebuilts_download.sh

```

图 0-6

8. 执行构建命令:

```

01. cd /workspace/
02. sudo chmod +X build.sh
03. sudo ./build.sh --product-name rk3568 --build-target ark_js_host_linux_tools_packages
04. sudo ./build.sh --product-name rk3568 --build-target arkcompiler/runtime_core:ark_host_linux_tools_packages
05. sudo ./build.sh --product-name rk3568 --build-target ark_host_linux_defectscanaux_unittest

```

图 0-7

输出构建完成提示: 编译完成后, 显示构建成功信息。

```
01. echo "*****"
02. echo "build process completed for rk3568 with specified targets"
03. echo "enjoy it :-D"
04. echo "cd /workspace/out/rk3568/clang_x64/arkcompiler/runtime_core/"
05. echo "./arkui_analyzer -h"
06. echo "*****"
```

图 0-8

附录 2：术语表

术语	定义
ArkUI Analyzer	用于分析鸿蒙应用 UI 行为迁移模型的静态分析工具，主要目标是指导动态测试工具更有效地覆盖关键路径。
方舟编译器	鸿蒙系统中的高级语言编译器，支持多语言的高效编译和运行，包括编译工具链和运行时两个部分。
Panda IR	方舟编译器中的中间表示语言，介于高级语言和机器码之间，用于优化和静态分析。
UIAbility	鸿蒙系统中的一种能力模型，主要用于实现用户界面逻辑。
Page	鸿蒙应用中的页面单元，负责呈现和管理用户界面内容。
组件	鸿蒙应用 UI 的基本构成单元，例如按钮、文本框等。
ArkTS	方舟编译器支持的一种类型安全的高级编程语言，用于开发鸿蒙应用。
生命周期函数	应用或组件在特定生命周期阶段被调用的函数，如初始化、销毁等。
事件处理函数	用于响应用户操作或系统事件的函数，例如点击按钮或页面加载事件的回调函数。
模块变量	在 ArkTS 模块中定义的变量，通常用于共享信息或配置，如路由目标位置等。
数据流分析	分析变量在程序中的定义和使用关系，以跟踪变量的变化和数 据流动过程。
调用链	从一个函数调用另一个函数形成的链式关系，通常用于跟踪函数之间的依赖和调用路径。
静态栈	用于模拟程序执行时的栈操作，通常用于分析程序中的层次结构关系，如 UI 组件的层次关系。

组件树	表示 UI 组件之间层次结构的树形结构，其中每个节点代表一个 UI 组件。
Foreach 节点	在组件树中表示子节点数量可动态变化的特殊节点，其数量通常由用户交互行为决定。
If 节点	在组件树中表示条件渲染的节点，其子节点是否渲染取决于指定条件的真假值。
路由	管理页面导航和切换的机制，如通过 <code>'router.pushUrl'</code> 方法实现的页面跳转。
PHI 指令	中间表示语言中的一种指令，用于表示多个分支变量的合并点。
XPath	用于唯一标识 UI 组件的路径表达式。

表 0-9 术语表

附录 3：OpenHarmony 应用开发框架概述

OpenHarmony 系统相较于 Android 系统，采用了一些独特的设计理念，特别是在 UI 架构和应用开发模式上。鸿蒙系统的 UI 框架采用了声明式 UI 设计，强调灵活的界面布局和元素渲染，同时支持跨设备的应用部署和执行。这样一来，开发者能够在多个设备和场景之间保持一致的用户体验，然而这也使得 UI 交互、状态管理等方面变得更加复杂。因此，现有的基于 Android 的自动化测试工具可能不适用于 OpenHarmony，尤其是在 UI 建模、界面状态迁移和事件响应机制等方面。

Stage 模型是 OpenHarmony 平台的核心应用架构，用于管理跨设备、跨场景的应用生命周期和资源调度。它通过多个关键组件（如 AbilityStage、UIAbility、ExtensionAbility、WindowStage 等）协同工作，实现了应用的灵活生命周期管理和统一的资源调度。

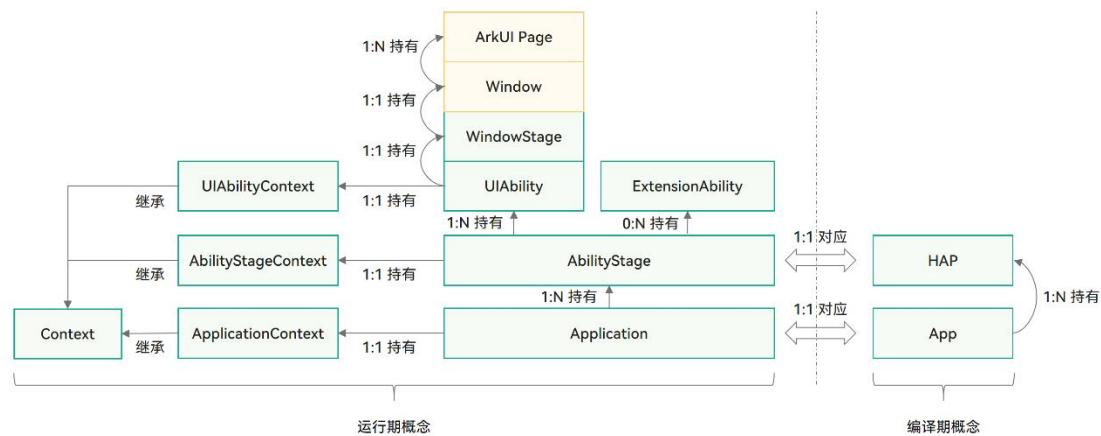


图 0-10 Stage 模型概念图

Stage 模型的关键组件如下：

1. AbilityStage 是 OpenHarmony 中应用的核心生命周期管理类。每个应用运行时都会创建一个 AbilityStage 实例。它负责管理应用的生命周期，并协调应用内部的各个模块和组件。AbilityStage 主要负责应用启动时的初始化、资源调度、能力分配等工作。它为 UI 组件（如 UIAbility）提供运行环境，并管理这些 UI 组件的生命周期。
2. UIAbility 是 OpenHarmony 中的用户界面（UI）组件，类似于 Android

中的 Activity。每个 UIAbility 代表一个界面或页面，并处理与用户的交互。UIAbility 主要用于显示界面，并提供与用户的交互逻辑。其生命周期包括从创建、显示到销毁的全过程。开发者可以通过生命周期回调如 onCreate、onForeground、onBackground、onDestroy) 等来控制 UI 的显示和资源管理。

3. ExtensionAbility 面向后台任务或特殊场景需求，不直接处理 UI 展示。它处理诸如输入法、任务调度等功能，生命周期由系统管理，开发者根据场景需求进行扩展。
4. WindowStage 是与 UIAbility 绑定的窗口管理类，负责 UI 渲染和窗口状态管理。它确保 UI 的显示、事件处理以及窗口状态的切换。
5. Context 类为 UIAbility 和 ExtensionAbility 提供资源访问接口，允许它们调用系统资源、服务和网络能力。

Stage 模型通过精细化的生命周期管理，使得应用在不同设备和场景中的表现保持一致。关键生命周期如下：

1. AbilityStage 生命周期：应用加载时创建 AbilityStage 实例，负责应用的资源和状态管理。
2. UIAbility 生命周期：UIAbility 生命周期包括创建（onCreate）、前台（onForeground）、后台（onBackground）和销毁（onDestroy）四个阶段，开发者可在回调中管理 UI 资源。
3. ExtensionAbility 生命周期：ExtensionAbility 处理后台任务，由系统管理，开发者根据需求选择扩展类（如输入法、任务调度等）。
4. WindowStage 生命周期：每个 UIAbility 与一个 WindowStage 绑定，负责 UI 渲染和状态切换。WindowStage 在 UIAbility 创建时加载 UI 内容，销毁时释放资源。

Stage 模型为 OpenHarmony 提供了一种跨设备、跨场景的应用生命周期管理方式，通过对 AbilityStage、UIAbility、ExtensionAbility 和 WindowStage 等组件的精细管理，确保了 UI 展示和后台服务的一致性。尽管其增加了生命周期管理

的复杂性，但也带来了灵活性和跨平台的优势，为开发者提供了更高效的资源调度和 UI 展示能力。

在 OpenHarmony 中，UIAbility 是应用界面的核心组件，它的生命周期涉及多个状态，包括 Create（创建）、Foreground（前台）、Background（后台）和 Destroy（销毁）四个主要状态。每个状态变化都对应着不同的回调方法，这些回调使开发者能够在不同的生命周期阶段执行必要的操作，如资源加载、UI 更新、后台任务处理等。

1. **Create 状态**: 当应用启动并加载 UIAbility 时，系统会触发 `onCreate()` 回调。在此阶段，开发者可以进行页面初始化操作，如资源加载、UI 设置等，为后续的渲染做好准备。
2. **Foreground 状态**: 当 UIAbility 从后台切换到前台时，系统会调用 `onForeground()` 回调。在此阶段，开发者可以重新申请前台资源，恢复 UI 状态或重新加载资源。
3. **Background 状态**: 当 UIAbility 页面切换到后台时，`onBackground()` 回调被触发。在此阶段，开发者应释放不再需要的资源，或者执行一些背景任务，如保存状态、暂停动画等。
4. **Destroy 状态**: 当 UIAbility 实例销毁时，`onDestroy()` 回调被触发。此时，开发者应释放所有相关资源并保存必要的数​​据，确保应用退出时不会有资源泄漏。

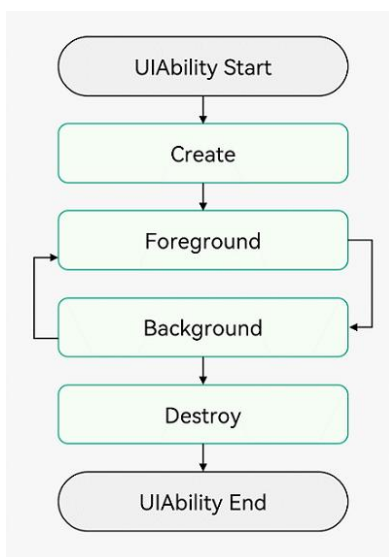


图 0-11 UIAbility 生命周期状态图

每个 UIAbility 实例都与一个 WindowStage 关联，WindowStage 负责管理该 UIAbility 的主窗口。其生命周期包括两个阶段：当 UIAbility 进入前台时，WindowStageCreate 回调会被触发，开发者可以在此阶段设置 UI 页面、订阅事件并加载页面内容；当 UIAbility 销毁时，WindowStageDestroy 回调被调用，此时开发者应释放与窗口相关的所有资源，确保没有内存泄漏。

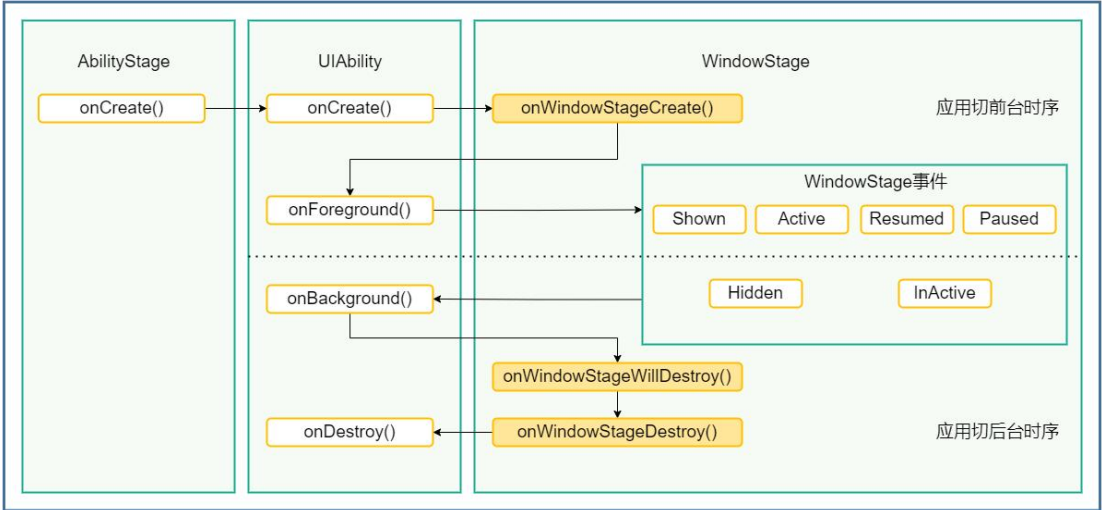


图 0-12 WindowStageCreate 和 WindowStageDestroy 生命周期状态图