HapTest 自动化测试工具

一、工具概述

1.1 定位

HapTest 是基于 HapRay 性能测试框架开发的**自动化 UI 探索与性能测试工具**，专门针对 HarmonyOS 应用进行智能化测试。

1.2 核心价值

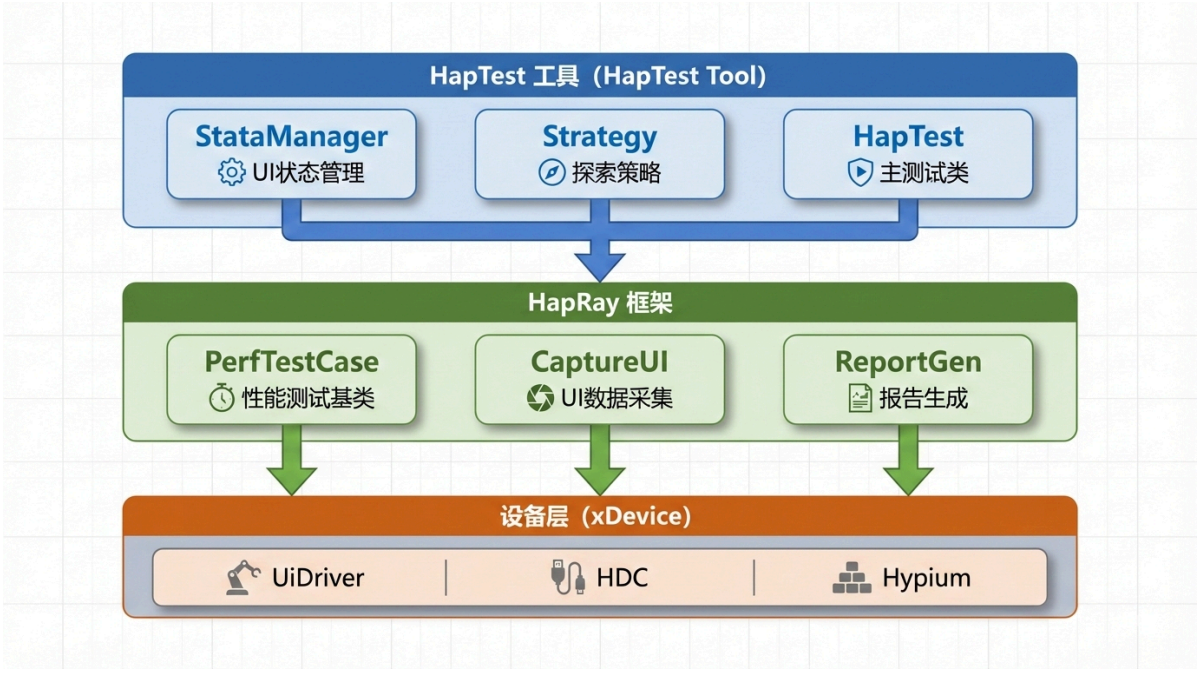
- ✅ **零代码测试**：无需编写测试脚本，通过命令行即可启动自动化测试
- ✅ **智能探索**：基于策略自动探索应用 UI，发现潜在问题
- ✅ **性能采集**：同步采集 CPU、内存、trace 等性能数据
- ✅ **可视化报告**：生成 HTML 性能分析报告，直观展示测试结果

1.3 应用场景

- 应用稳定性测试：长时间运行，发现 crash、ANR 等问题
- 性能回归测试：对比不同版本的性能表现
- UI 兼容性测试：验证不同设备上的 UI 表现
- 探索式测试：快速了解应用功能和交互流程

二、技术架构

2.1 整体架构图



2.2 模块职责

模块	文件	职责
HapTest	`haptest_case.py`	主测试类，继承 PerfTestCase，实现探索循环
StateManager	`state_manager.py`	UI 状态管理，元素提取，状态去重
Strategy	`strategy.py`	探索策略，决策下一步操作
HapTestAction	`haptest_action.py`	命令行入口，动态生成测试用例

三、核心模块详解

3.1 StateManager - UI 状态管理

核心职责：

1. **UI 元素提取**：从 Inspector JSON 中递归提取可点击元素
2. **bundleName 过滤**：只保留目标应用的元素，过滤系统组件
3. **状态去重**：基于元素树哈希判断是否访问过相同页面
4. **历史记录**：记录所有操作历史，标记已点击元素

类结构：

```
class UIState:
    """单个UI状态的封装"""
    - step_id: 步骤编号
    - screenshot_path: 截图路径
    - element_tree_path: 元素树路径
    - inspector_path: Inspector JSON 路径
    - app_package: 目标应用包名
    - clickable_elements: 可点击元素列表（延迟加载）
    - state_hash: 状态哈希值（延迟计算）

class StateManager:
    """状态管理器"""
    - visited_states: set() - 已访问状态的哈希集合
    - state_history: list - 所有状态历史
    - action_history: list - 所有操作历史
```

关键实现：

```
class UIState:
    def __init__(self, step_id, screenshot_path, inspector_path, app_package):
        self.app_package = app_package
        self._clickable_elements = None
        self._state_hash = None

    def _extract_clickable_recursive(self, node, clickable, path='',
parent_bundle=''):
        """递归提取可点击元素"""
```

```
        attrs = node.get('attributes', {})
        node_type = attrs.get('type', '')
        clickable_attr = attrs.get('clickable', '') == 'true'
        bundle_name = attrs.get('bundleName', parent_bundle)

        # 只提取目标应用的元素
        if clickable_attr:
            if self.app_package and bundle_name != self.app_package:
                Log.debug(f'跳过非目标应用元素: {node_type}')
            else:
                # 解析 bounds 并添加到列表
                bounds = self._parse_bounds(attrs.get('bounds', ''))
                if bounds:
                    clickable.append({...})

        # 递归处理子节点（传递bundleName）
        for child in node.get('children', []):
            self._extract_clickable_recursive(child, clickable, current_path,
bundle_name)
```

状态去重机制:

```
def _compute_hash(self) -> str:
    """基于element tree计算状态哈希"""
    with open(self.element_tree_path, 'r', encoding='utf-8') as f:
        tree_content = f.read()

    # 提取树结构签名（提取所有元素ID）
    signature = self._extract_tree_signature(tree_content)
    return hashlib.md5(signature.encode('utf-8')).hexdigest()

def _extract_tree_signature(self, tree_content: str) -> str:
    """提取树结构签名(忽略动态内容)"""
    lines = tree_content.split('\n')
    signature_lines = []

    for line in lines:
        if '| ID:' in line:
            parts = line.split('| ID:')
            if len(parts) > 1:
                id_part = parts[1].strip()
                signature_lines.append(id_part)

    return '|'.join(signature_lines)
```

- 基于 element tree 的元素 ID 序列生成状态指纹
- 使用 MD5 哈希，相同 UI 结构产生相同哈希值
- 通过 `StateManager.visited_states` 集合记录已访问状态
- 相同状态只访问一次，避免无限循环

元素点击追踪:

```
def get_unvisited_elements(self, ui_state: UIState) -> list:
```

```
    """获取当前状态中未访问过的可点击元素"""
    all_clickable = ui_state.clickable_elements
    clicked_elements = set()

    # 从历史记录中提取已点击元素的path
    for action in self.action_history:
        if action['type'] == 'click' and action['target']:
            clicked_elements.add(action['target'].get('path', ''))

    # 过滤出未点击的元素
    unvisited = [elem for elem in all_clickable
                 if elem['path'] not in clicked_elements]
    return unvisited
```

- 每个元素有唯一的 `path`（如 `/root[]/Column[]/Button[login]`）
- 通过比较 path 判断元素是否被点击过
- 支持跨页面的元素追踪（同一元素在不同页面不会重复点击）

3.2 Strategy - 探索策略

支持三种策略：

策略	特点	适用场景
depth_first	深度优先，优先点击未访问元素	快速探索主流程
breadth_first	广度优先，平衡各页面访问	全面覆盖测试
random	随机选择，引入不确定性	压力测试、发现边界问题

决策逻辑：

```
def decide_next_action(self, ui_state, state_mgr):
    """决策下一步操作"""
    unvisited = state_mgr.get_unvisited_elements(ui_state)

    if unvisited:
        # 有未访问元素，选择一个点击
        target = self._select_target(unvisited)
        return 'click', target
    else:
        # 无未访问元素，尝试滑动或返回
        if self.consecutive_backs < 3:
            self.consecutive_backs += 1
            return 'back', None
        else:
            return 'stop', None  # 探索完成
```

3.3 HapTest - 主测试类

核心流程:

```
def process(self):
    """主测试流程"""
    self.start_app()

    while self.current_step < self.max_steps:
        # 1. 采集当前 UI 状态
        ui_state = self._execute_ui_capture()

        # 2. 状态管理（去重、记录）
        is_new_state = self.state_mgr.add_state(ui_state)

        # 3. 策略决策
        action_type, target = self.strategy.decide_next_action(
            ui_state, self.state_mgr
        )

        if action_type == 'stop':
            break

        # 4. 执行操作 + 性能采集
        self._execute_action_with_perf(action_type, target)
```

关键特性:

- **禁用重复 UI 采集**：使用 `NoOpCaptureUI` 避免 `execute_performance_step` 中的重复采集
- **双日志系统**：控制台显示 INFO 级别，文件记录 DEBUG 级别
- **错误处理**：坐标转换、元素点击、bundle 解析均有容错机制

3.4 性能数据采集

采集内容:

- **hiperf 数据**：CPU 占用、函数调用栈（用于火焰图）
- **trace 数据**：系统事件、线程调度、帧率
- **UI 数据**：截图、元素树、Inspector JSON

四、使用方式

4.1 命令行运行

```
# 基础用法
python -m scripts.main haptest \
  --app-package com.example.app \
  --app-name "示例应用" \
  --max-steps 50

# 指定策略
python -m scripts.main haptest \
```

```
  --app-package com.example.app \
  --app-name "示例应用" \
  --strategy breadth_first \
  --max-steps 100
```

4.2 参数说明

参数	说明	默认值
`--app-package`	应用包名（必填）	-
`--app-name`	应用名称（必填）	-
`--strategy`	探索策略	depth_first
`--max-steps`	最大步数	50

4.3 输出结果

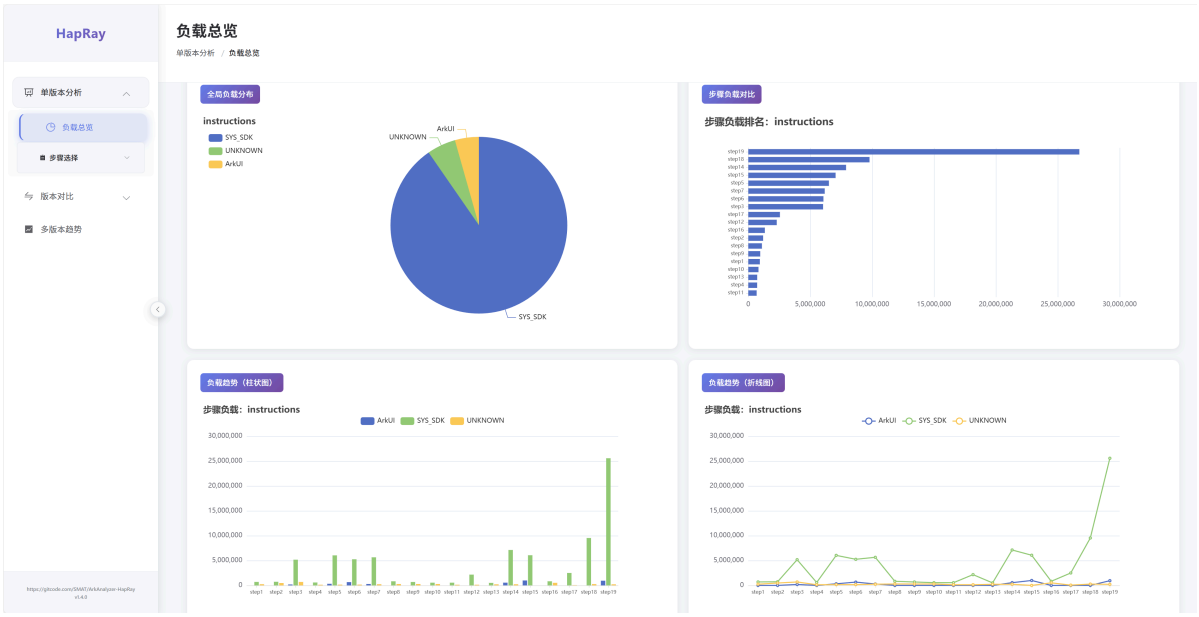
```
reports/haptest_<package>_<timestamp>/
├── HapTest_<package>/
|   ├── logs/
|   |   └── haptest.log          # 详细日志（DEBUG级别）
|   ├── ui/
|   |   ├── step1/
|   |   |   ├── screenshot_current_1.png
|   |   |   ├── element_tree_current_1.txt
|   |   |   └── inspector_current.json
|   |   └── step2/...
|   ├── hiperf/                  # 性能数据
|   |   ├── step1/
|   |   |   ├── perf.db
|   |   |   └── pids.json
|   |   |   └── ...
|   |   └── step2/...
|   ├── htrace/                  # Trace 数据
|   |   ├── step1/
|   |   |   ├── trace.htrace
|   |   |   └── trace.db
|   |   |   └── ...
|   |   └── step2/...
|   └── report/
|       └── hapray_report.html   # 可视化报告
```

4.3.1 详细日志示例

```
============================================================
15:38:54 - HapTest - INFO - Step 3/20
15:38:54 - HapTest - INFO -
============================================================
15:38:57 - HapTest - INFO - UI状态: 已访问
15:38:57 - HapTest.State - DEBUG - 解析到 17 个可点击元素（bundleName匹配:
com.example.wsywechat）
```

```
15:38:57 - HapTest.State - DEBUG - 解析到 17 个可点击元素（bundleName匹配:
com.example.wsywechat）
15:38:57 - HapTest - INFO - 可点击元素数: 17
15:38:57 - HapTest.State - DEBUG - 总可点击: 17, 已点击: 1, 未访问: 13
15:38:57 - HapTest.State - DEBUG - 总可点击: 17, 已点击: 1, 未访问: 13
15:38:57 - HapTest - INFO - 未访问元素数: 13
15:38:57 - HapTest - DEBUG - 未访问元素示例: ['Search', 'SearchField',
'SymbolGlyph']
15:38:57 - HapTest - DEBUG - 完整target信息:
{
  "type": "ListItem",
  "id": "",
  "text": "",
  "bounds": {
    "left": 0,
    "top": 1247,
    "right": 1216,
    "bottom": 1442
  },
  "path": "[]/root[]/Tabs[]/Swiper[]/TabContent[]/Column[]/List[]/ListItem[]",
  "bundleName": "com.example.wsywechat"
}
15:38:57 - HapTest - INFO - 决策: 点击 ListItem
15:39:12 - HapTest - INFO -
============================================================
15:39:12 - HapTest - INFO - Step 4/20
15:39:12 - HapTest - INFO -
============================================================
15:39:15 - HapTest - INFO - UI状态: 已访问
15:39:15 - HapTest.State - DEBUG - 解析到 4 个可点击元素（bundleName匹配:
com.example.wsywechat）
15:39:15 - HapTest.State - DEBUG - 解析到 4 个可点击元素（bundleName匹配:
com.example.wsywechat）
15:39:15 - HapTest - INFO - 可点击元素数: 4
15:39:15 - HapTest.State - DEBUG - 总可点击: 4, 已点击: 2, 未访问: 4
15:39:15 - HapTest.State - DEBUG - 总可点击: 4, 已点击: 2, 未访问: 4
15:39:15 - HapTest - INFO - 未访问元素数: 4
15:39:15 - HapTest - DEBUG - 未访问元素示例: ['Image', 'Image', 'TextInput']
15:39:15 - HapTest - DEBUG - 完整target信息:
{
  "type": "TextInput",
  "id": "input",
  "text": "",
  "bounds": {
    "left": 146,
    "top": 1513,
    "right": 1071,
    "bottom": 1627
  },
  "path": "[]/root[]/Column[]/Column[]/Row[]/TextInput[input]",
  "bundleName": "com.example.wsywechat"
}
15:39:15 - HapTest - INFO - 决策: 点击 TextInput
15:39:30 - HapTest - INFO -
============================================================
.......
```

4.3.2 可视化报告示例



五、技术亮点

5.1 智能状态管理

- **去重算法**：基于 MD5 哈希的状态指纹
- **增量探索**：只点击未访问过的元素
- **深度控制**：避免陷入无限循环

5.2 性能无损采集

- **并行采集**：UI 操作与性能数据采集同步进行
- **完整数据**：CPU、内存、trace、screenshot 全覆盖
- **低开销**：不影响应用正常运行

5.3 高可扩展性

- **策略可插拔**：新增策略只需实现 `decide_next_action`
- **分析器可扩展**：基于 HapRay 的分析器框架
- **配置驱动**：通过配置文件调整采集参数