移动App专项测试

Mobile APP Subject Tests

集团研发中心 / QA 何小伟

WHY 为什么做专项测试？

WHAT 什么是专项测试？

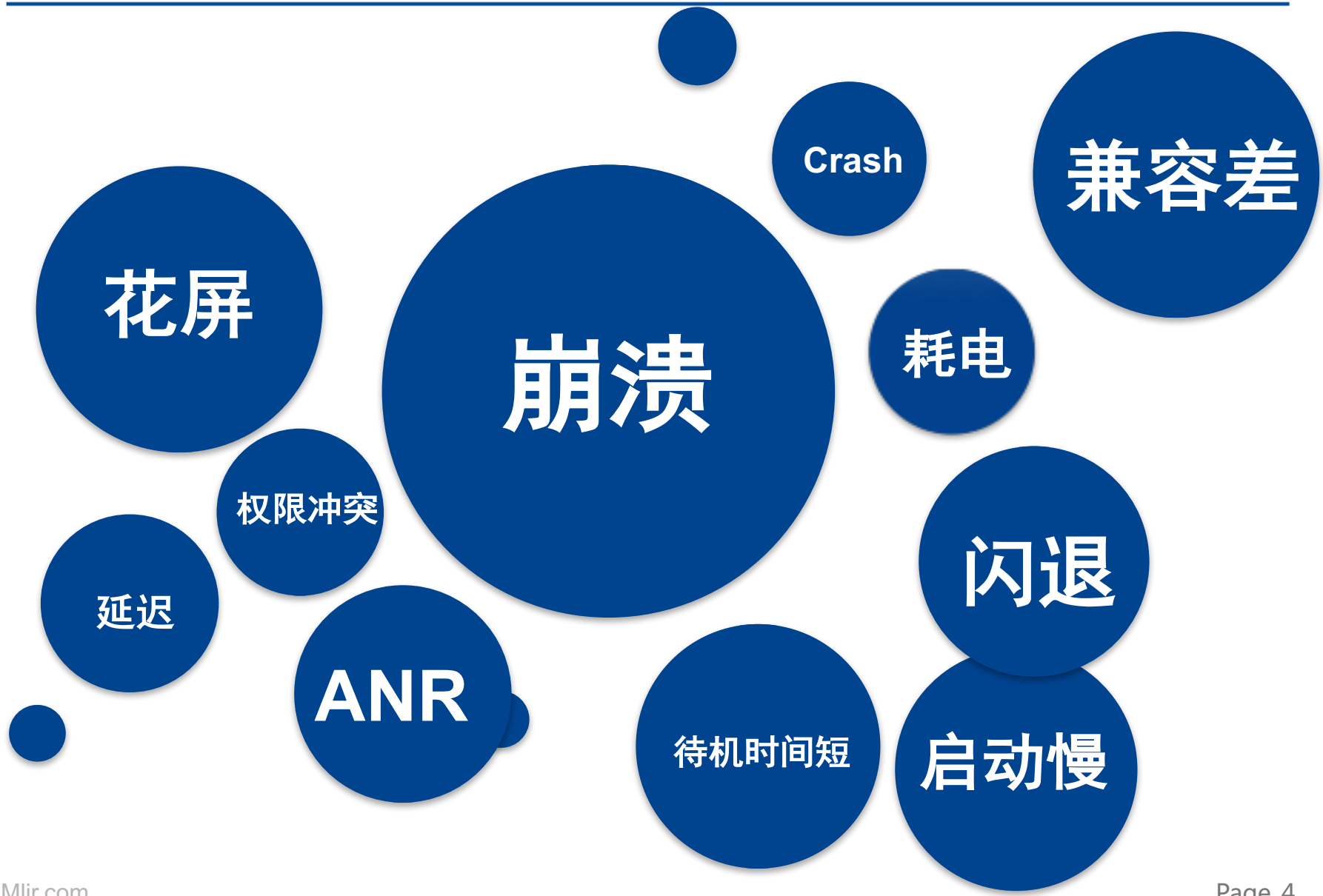
HOW 怎么做专项测试？

美利

WHY

为什么做专项测试

Why 为什么做专项测试？



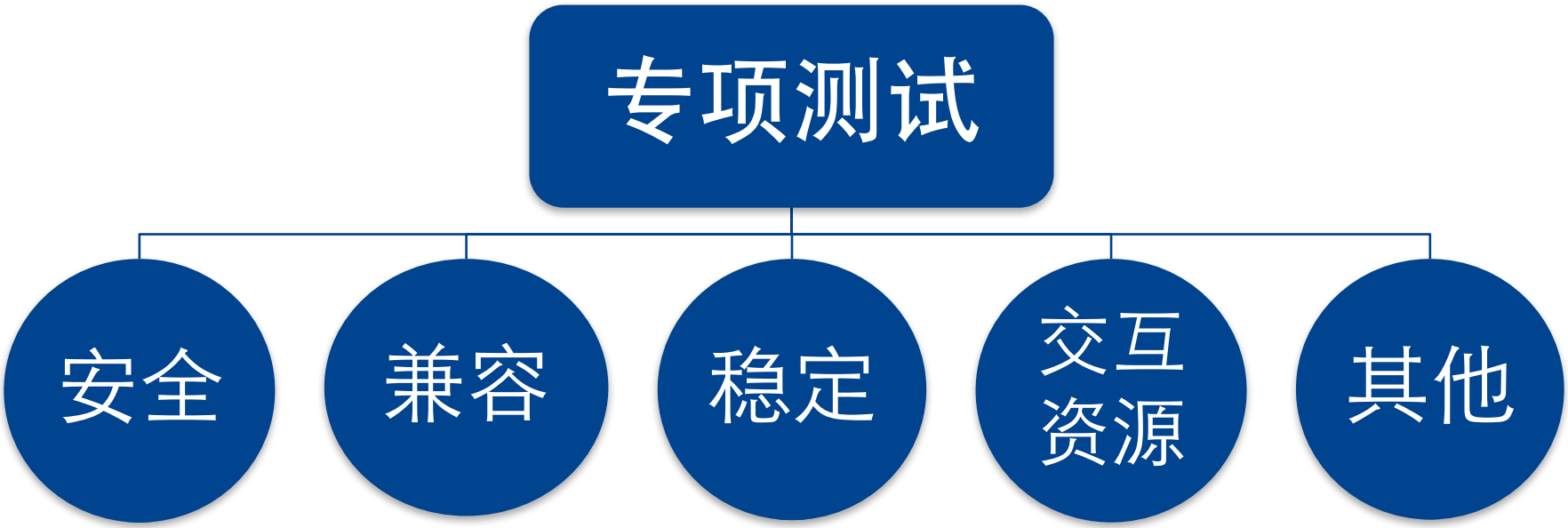
从客户端、服务端共「五大专项」维度对应用质量进行全面评估，决定产品上线时间及是否能上线发布。

高品质App定义？

WHAT

什么是专项测试？

针对某个特殊方面或者问题而展开的测试我们统称为**「专项测试」**


专项测试
安全
兼容
稳定
交互
资源
其他

安全测试-专项测试

App评估常见问题

客户端评估

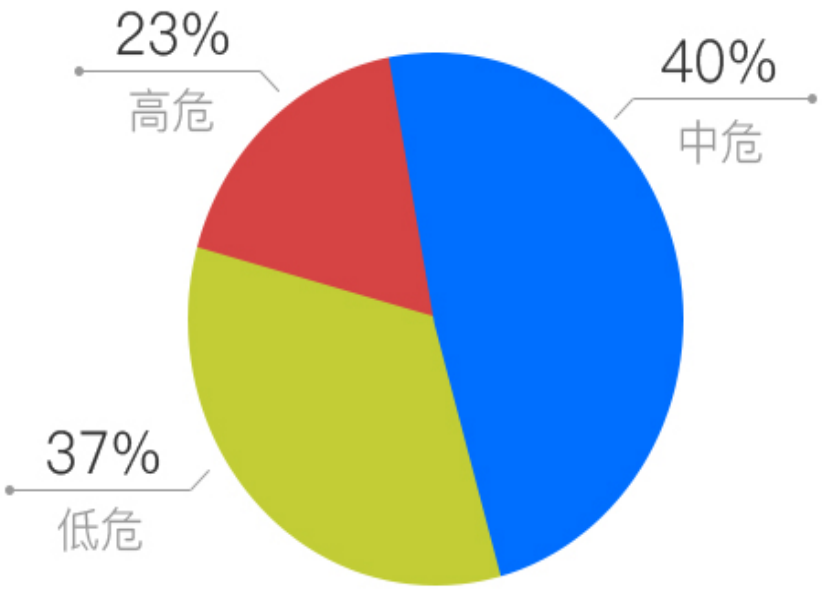
业务评估

- 证书问题
- 可调试
- 可反编译
- 敏感信息泄漏
- Log日志
- 内存管理
- 越权漏洞
- 短信炸弹
- 其他业务逻辑漏洞


APP
安全
服务器安全
「数据存储」
客户端到服务
器「数据传输」
客户端安全
「数据录入」

金融行业移动 APP 漏洞评测

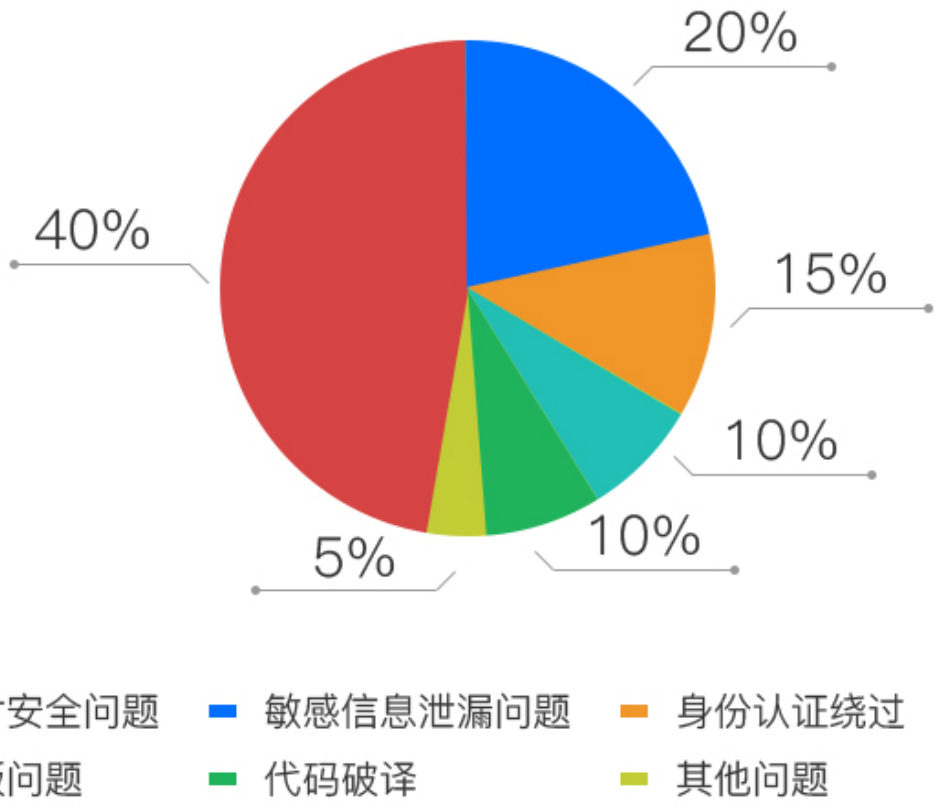
统计来源：腾讯云乐固团队



- **高危** 数据传输不安全导致盗取用户钱财损害平台利益
- **中危** 用户敏感信息泄露，应用被重打包后加入恶意代码和广告
- **低危** 应用崩溃，APP主要逻辑被逆向

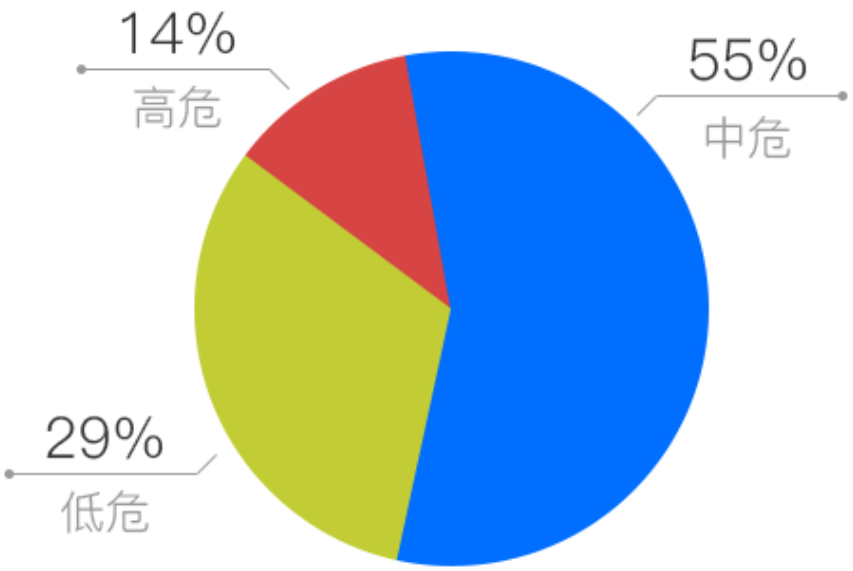
金融行业移动 APP 安全问题类别

统计来源：腾讯云乐固团队



电商行业移动 APP 漏洞评测

统计来源：腾讯云乐固团队



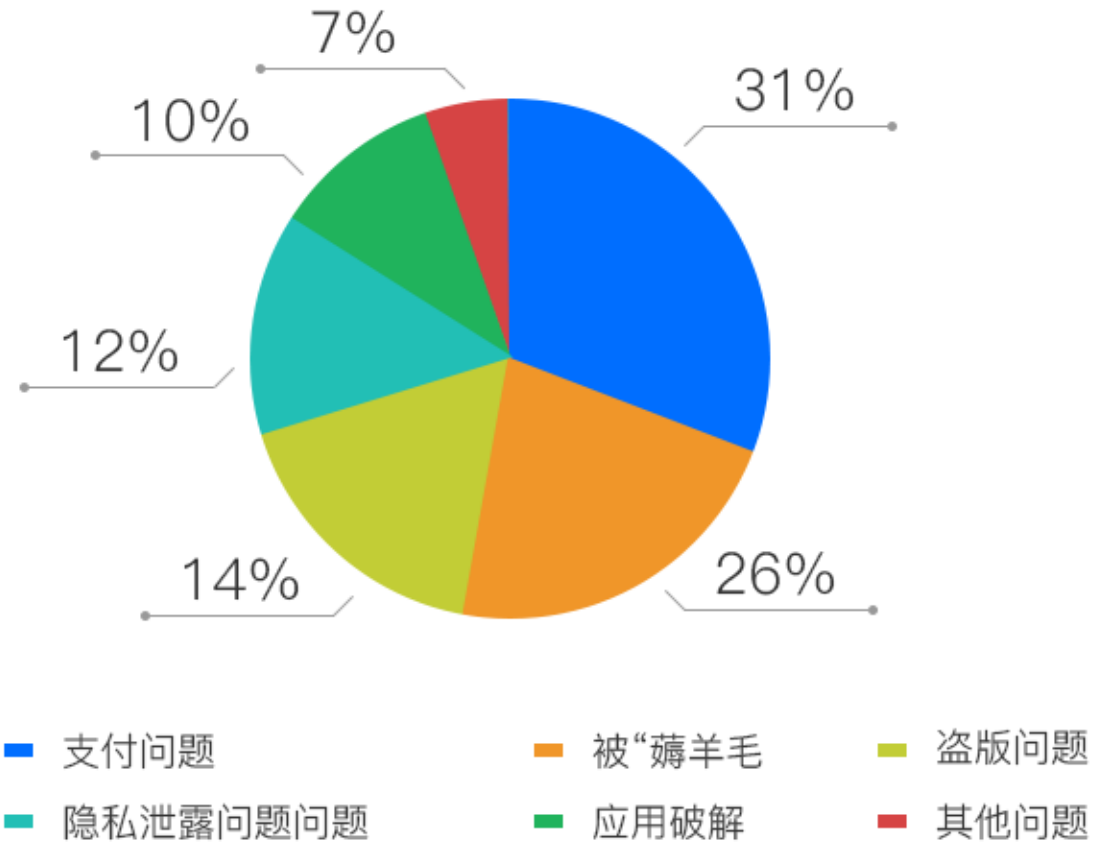
高危 数据传输不安全导致用户订单泄露、篡改

中危 本地数据存储不安全、用户隐私泄露

低危 APP 业务逻辑被破解、算法剽窃

电商 APP 安全问题类别

统计来源：腾讯云乐固团队

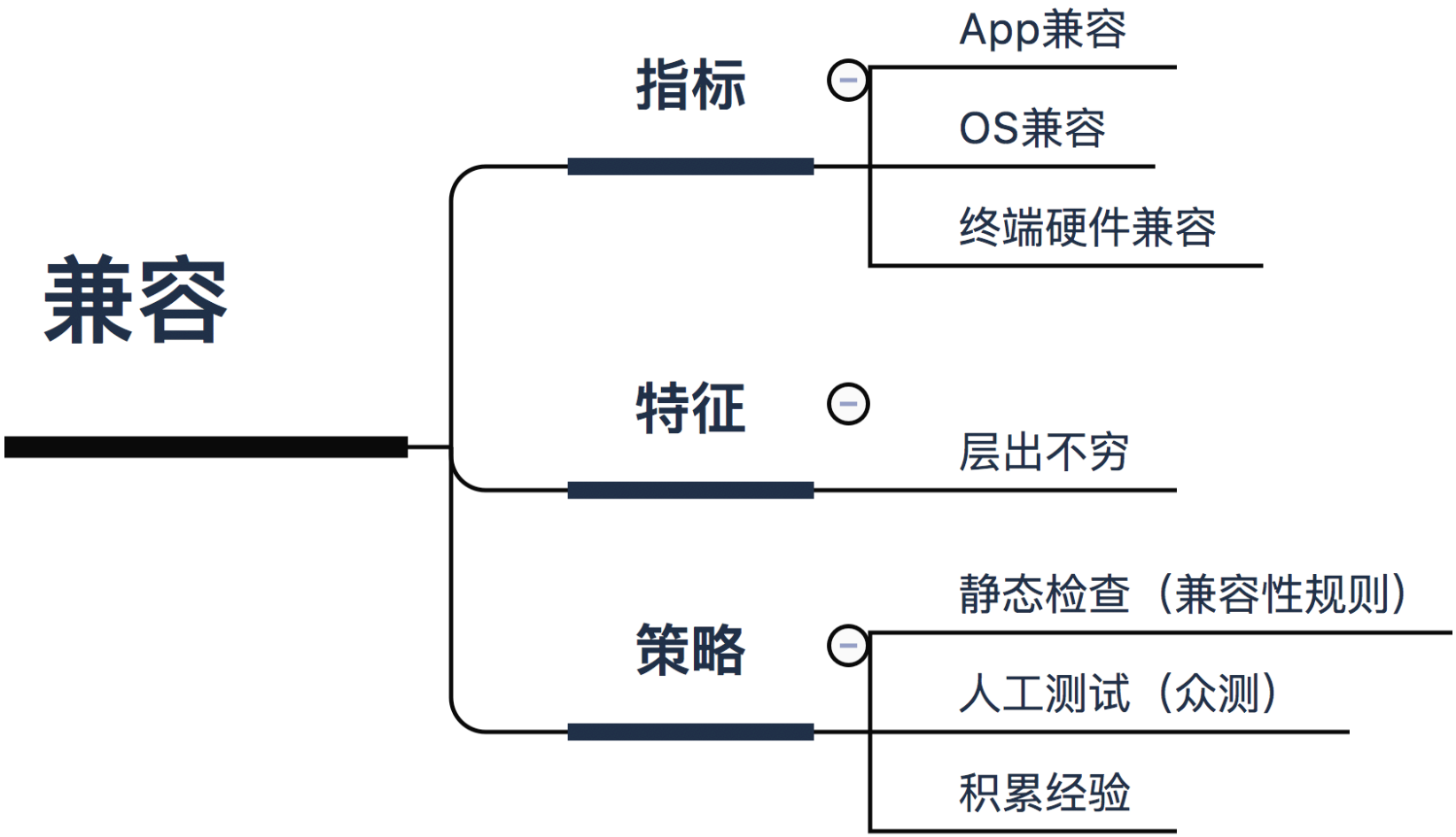


专项测试-安全性测试

检查项目

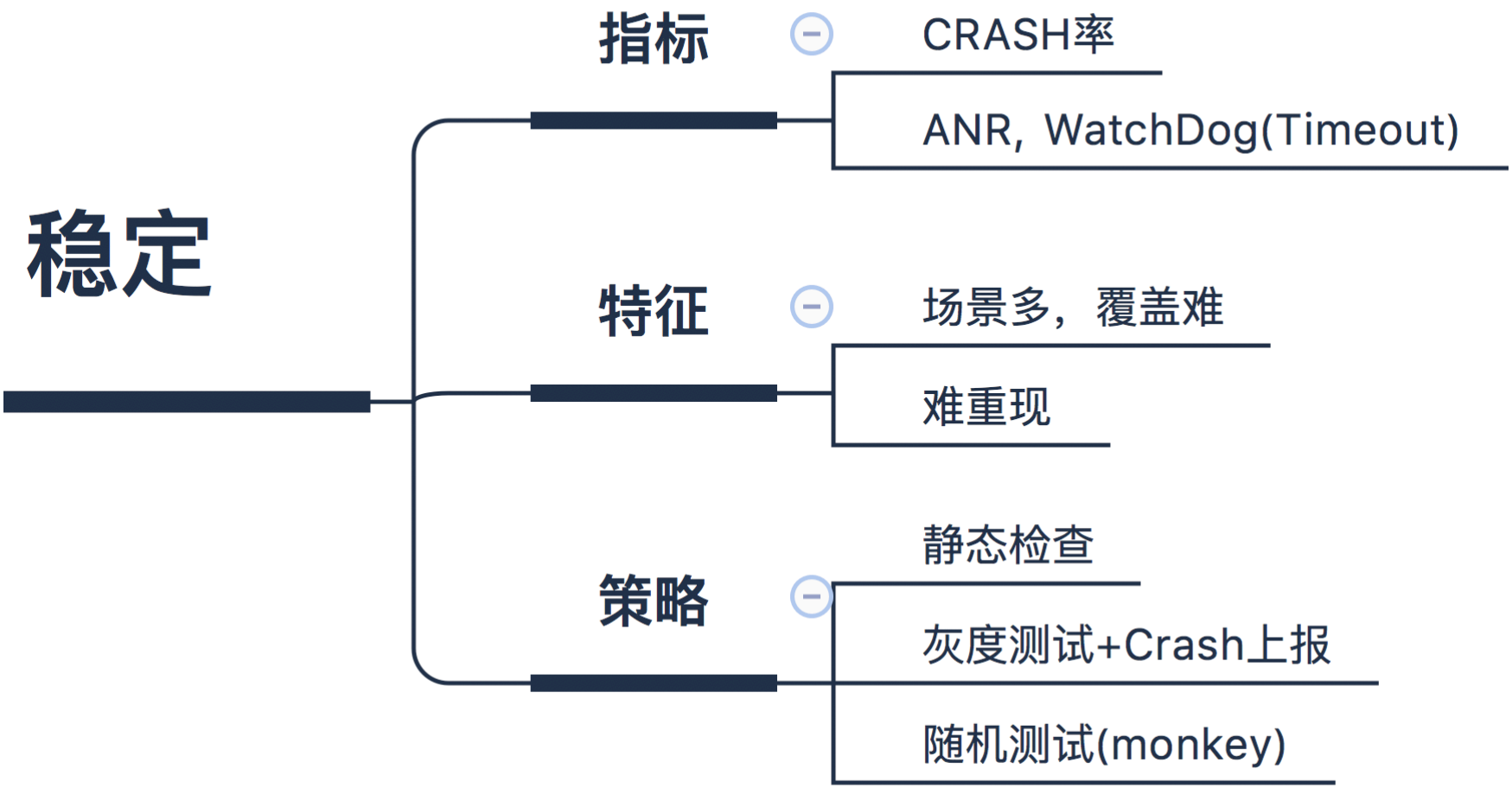
1.明文传输用户名、密码和验证码等敏感信息。
2.不安全的本地存储。
3.泄漏后台服务器地址，导致服务器可控
4.边信道信息泄漏
5.未使用有效的token机制，导致可以绕过鉴权
6.传输数据可修改，造成越权访问
7.登录设计缺陷，存在被暴力破解风险
8.利用业务逻辑缺陷制作短信炸弹
9.关键页面存在钓鱼劫持风险，导致用户信息泄露
10.可以重新编译打包
11.WebView漏洞
12.Web表单设计缺陷，存在SQL注入漏洞，XSS跨站
13.组件Content Provider配置错误，导致数据泄漏
14.组件Activity配置错误，导致登录页面被绕过
15.组件Service配置错误，导致非法权限提升「越权」
16.组件Broadcast Receiver配置错误，导致拒绝服务、非法越权
17.开启allowbackup备份权限，存在备份数据泄露风险
18.开启Debuggable属性，存在应用信息篡改泄露风险
19.下载非官方开发工具，导致IOS版本APP被植入恶意代码
20.开发者证书不规范，导致开发者身份信息不明

兼容性测试-专项测试

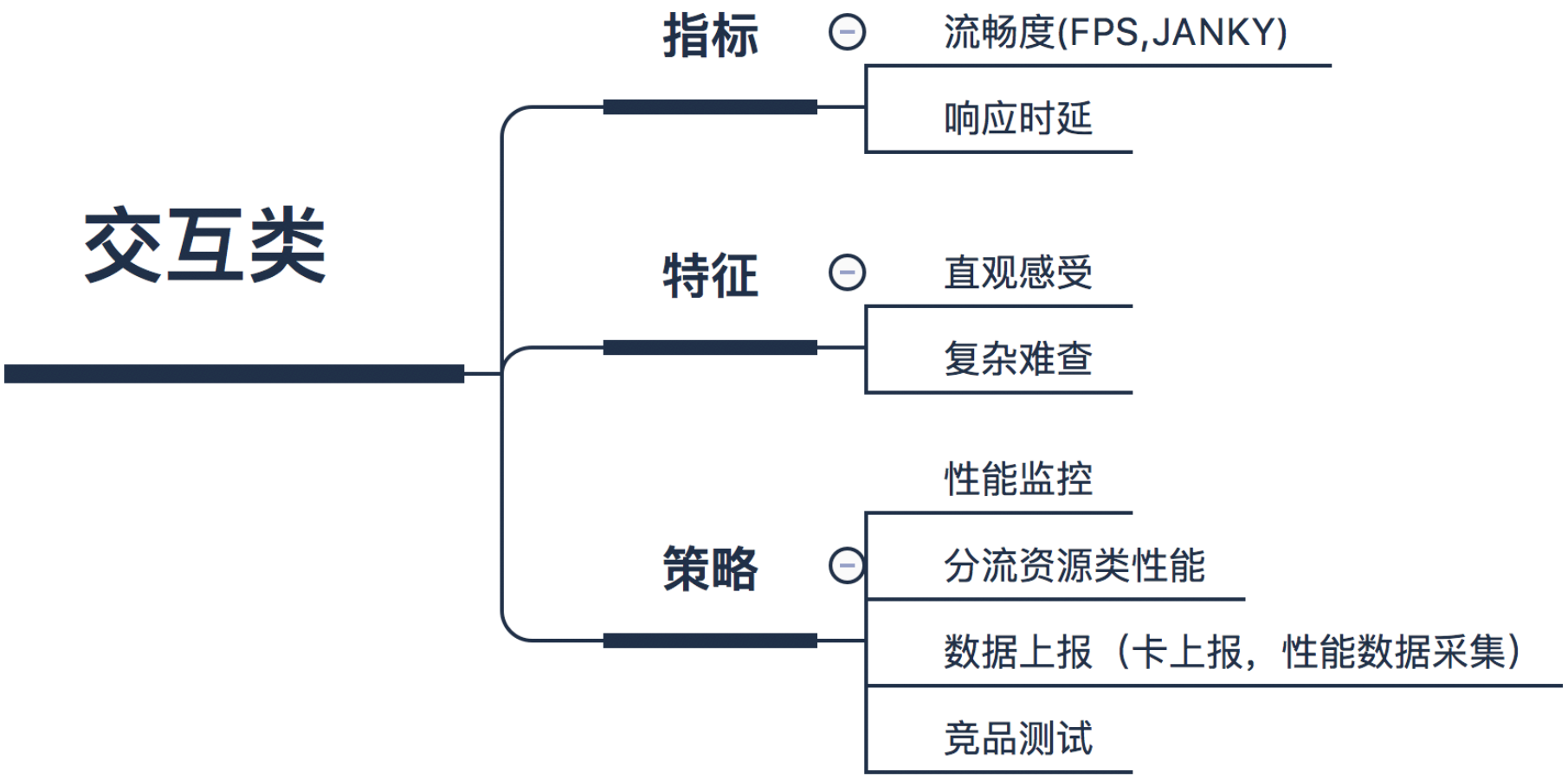


类型	释 意
安装失败	在某款手机安装没有成功
闪退	在app运行过程中导致崩溃意外退出
启动失败	安装成功无法启动，或者立马退出
卡顿	App运行不流畅，画面切换卡屏
卡死	系统无法正常运行，不接受输入，触发系统崩溃
黑屏	无法运行，没有显示任何ui画面
数据异常	客户端和服务器的数据不一致
Ui异常	Ui界面显示混乱
程序功能异常	程序运行时出现程序停止，提示error，功能无响应现象

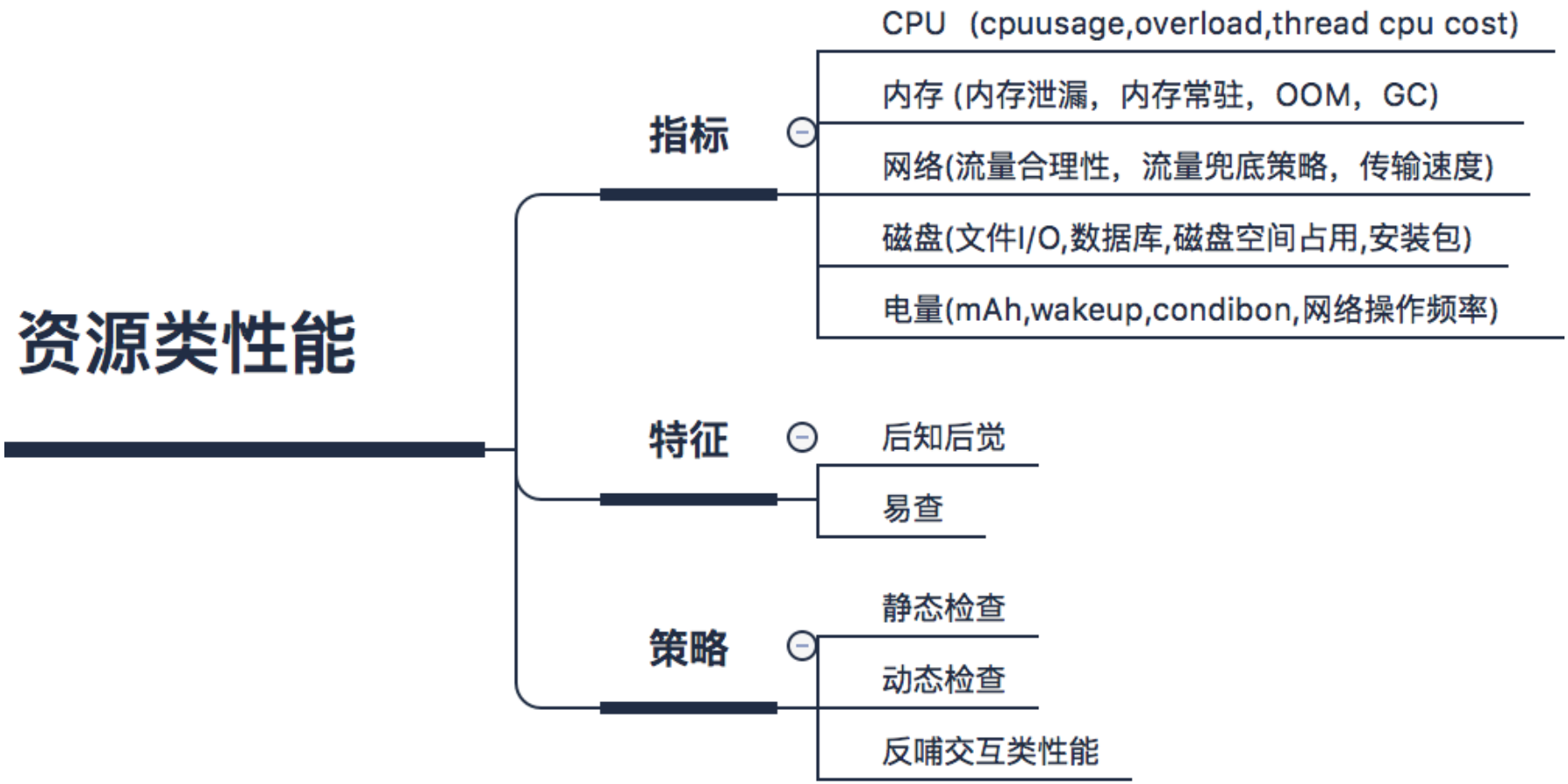
稳定性测试-专项测试


稳定
指标
CRASH率
ANR，WatchDog(Timeout)
特征
场景多，覆盖难
难重现
策略
静态检查
灰度测试+Crash上报
随机测试(monkey)

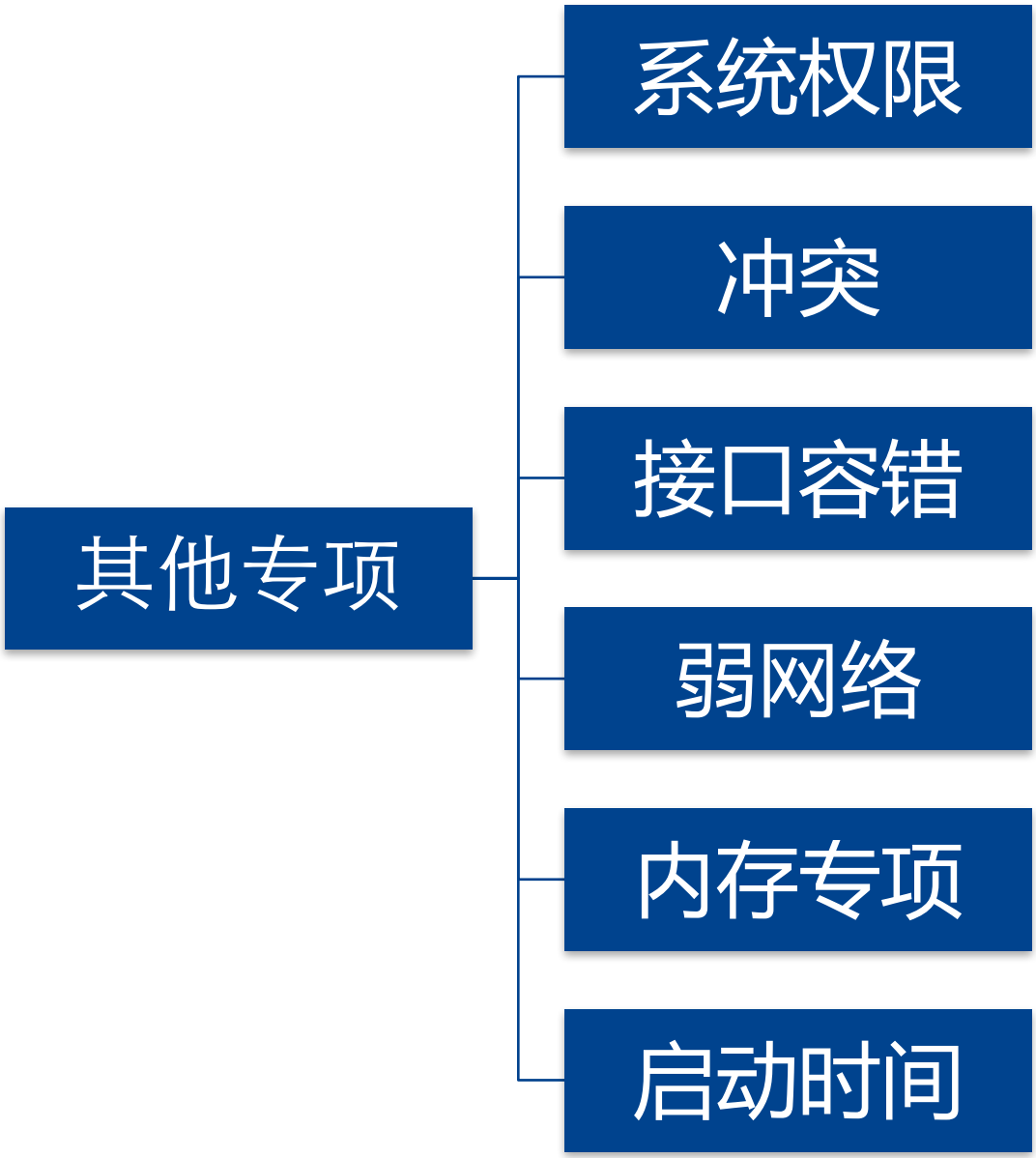
交互性测试-专项测试


交互类
指标
流畅度(FPS,JANKY)
响应时延
特征
直观感受
复杂难查
策略
性能监控
分流资源类性能
数据上报（卡上报，性能数据采集）
竞品测试

资源性测试-专项测试


资源类性能
指标
CPU (cpuusage,overload,thread cpu cost)
内存 (内存泄漏，内存常驻，OOM，GC)
网络(流量合理性，流量兜底策略，传输速度)
磁盘(文件I/O,数据库,磁盘空间占用,安装包)
电量(mAh,wakeup,condibon,网络操作频率)
特征
后知后觉
易查
策略
静态检查
动态检查
反哺交互类性能

其他专项测试列表？



系统权限 - 专项测试

- **目的**：验证需要调用系统相关权限时，客户端是否处理合理，友好的提示引导了用户

- **测试范围**：各模块中有涉及到调用系统权限相关的，都在确认范围内

- **测试方法**：在被测设备中安装有具备权限管理的app，分别设置允许权限、权限提醒、禁止权限后，对涉及到权限的功能模块进行逐一确认。如净化大师，360手机卫士等软件都可以管理权限

- **通过标准**：在允许权限设置下，功能一切正常。在提醒权限拒绝后，或设置禁止权限情况下，客户端需明确提示用户，而且不出现异常崩溃等问题。

「有钱用」- 系统权限 - 专项测试

权限 / 模块	结 果
拨打电话	不涉及该权限，功能正常
获取通信录	功能正常
获取通话记录	功能正常
定位	功能正常
手机识别码	不涉及该权限，功能正常
打开摄像头	功能正常
读取设备信息	功能正常
录音	不涉及该权限，功能正常
打开移动网络	功能正常
打开无线网络	功能正常
打开蓝牙	不涉及该权限，功能正常

冲突 - 专项测试

- **目的**：验证在模块操作过程中触发一些常见打断，客户端对于事件优先级的处理是否影响了功能

- **测试方法**：在被测app操作过程中触发预先准备好的事件。确认被触发的事件是否可以正常出现，以及事件发生后之前的状态是否继续在执行，最后确认事件结束后状态的现状是否正常。

- **通过标准**：系统事件可以正常触发，app中的各状态表现正常。3种情况的预期现象是否符合常规、合理为标准。

「有钱用 」- 冲突 - 专项测试

状态列表 事件列表	还款计划时	认证时	扫描身份证 人脸识别时
来电呼入	正常	正常	正常
网络切换	正常	正常	正常
短信通知	正常	正常	正常
push消息	正常	正常	正常
闹钟	正常	正常	正常
USB插入	正常	正常	正常
挂起	正常	正常	正常
程序历史记录启动	正常	正常	正常
长按关机键	正常	正常	正常
自动横屏	横屏模式已关闭	横屏模式已关闭	横屏模式已关闭
锁屏待机	正常	正常	正常

接口兼容 - 专项测试

- **目的**：验证服务端返回异常或不合理数据情况下，客户端对于异常数据的处理是否合理

- **测试范围**：各模块常用接口定义**3-5**个，可以满足服务端下发数据异常情况下，客户端的处理正常，可友好提示用户

- **测试方法**：使用Fiddler或Charles工具模拟服务端返回的数据，修改返回值使之满足以下定义的场景，观察客户端的情况

- **通过标准**：接口超时及异常都能正常有合理提示或交互，可以引导或提醒用户。反之不符合预期或体验不好的情况都不通过。

「有钱用」- 接口容错 - 专项测试

模块	场景	容错场景	测试结果	问题描述或备注	问题ID
「有钱用」取现	xxx接口	code：-1 code: 1 code 3 数组对象长度为0：[] 数组对象：null 字符串类型字段：“” 字符串类型字段：null 关键字段缺失	通过	已解决问题： code：3踢出登陆后一直提示无反应问题已经解决	
	xxx接口	code：-1 code: 1 code 3 数组对象长度为0：[] 数组对象：null 字符串类型字段：“” 字符串类型字段：null 关键字段缺失	通过	已解决问题： 数组对象长度为0时，无友好提示问题已经解决	

弱网络- 专项测试

- **目的：**验证在网络不佳情况下，网络逻辑处理是否合理，客户端是否有友好提示

- **测试范围：**各模块常用接口定义**3-5**个，可以满足网络不佳情况下超时提示或重试机制，断网情况下可友好提示用户
- **测试方法：**（1）网络超时或重试
- （2）断网情况，或者进入到无网络环境中……

- **通过标准：**接口超时及异常都能正常有合理提示或交互，可以引导或提醒用户。反之不符合预期或体验不好的情况都不通过。

「有钱用」- 弱网络 - 专项测试

模块	网络模式	场景	测试结果	问题描述或备注	问题ID
「有钱用」取现／还款	2G 3G 4G Wifi 无网络 无网络到有网络有网络到无网络 wifi->3G/4G 3G/4G->wifi	Home（主页）	通过	已解决问题： 请求失败后无提示或重试按钮问题	

内存-专项测试

- **目的**：验证客户端在操作后，模块对应功能是否正常释放了内存，是否有常驻占用内存的资源，以保证客户端的流畅性及稳定性

- **测试范围**：主要模块的各个页面或功能点，都需要确认内存消耗情况

- **测试方法**：对被测模块操作一定场景或一些步骤后，可以用DDms中的Dump内存工具，再使用Memory Analyzer tool工具进行具体分析

- **通过标准**：通过Mat工具分析后，对可疑的问题或已确定的问题提交到禅道系统进行记录，确认修复时间点，修复后再次验证无问题后通过。

「有钱用」 - 内存消耗 - 专项测试

模块	是否有内存泄漏	是否有大对象常驻内存	备注
「有钱用」借款	否	否	调用5个Activity界面连续操作100次后，dump内存heap后进行分析，对包含.*yqy.*的类进行分析
「有钱用」借款			
.....			

- 1、指定界面实例个数，界面打个实例数量规则。界面循环打开，导致界面多个实例。排名最高的崩溃是页面都是多开导致的（该原因占总内存溢出中的60%）

- 2、界面内存占用过大，暂定标准：2M；界面打开后本身占用内存过大，当其它页面发生内存泄露导致不足时，打开大内存开销的页面，多数会产生内存溢出错误。

- 3、内存未释放，暂定标准：1M；界面再开再次关闭后，还有内存未释放掉，可能存在内存泄露，需要重点分析。

- 4、界面使用了较大引导图、背景图，String字符串等。

启动时间-专项测试

- **测试范围**：按不同设备，不同系统记录启动时间

- **测试方法**：记录从桌面启动app进入到首页过程所需要的时间。三次或者多次后求平均时间

- **通过标准**：目标是3秒之内可以启动，如不在范围内提出期望目标进行优化

「有钱用」启动时间 - 专项测试

	设备1（Android 4.4.2,1280*720）				设备2（Android 5.0.1,1920*1080）				设备3（Android 6.0.0,480*800）			
确认场景	第一次	第二次	第三次	平均时间（秒）	第一次	第二次	第三次	平均时间（秒）	第一次	第二次	第三次	平均时间（秒）
首次启动												
...												

How

怎么做专项测试？

How 怎样做专项测试？

什么阶段去做？

每个阶段做什么？

做到什么颗粒度？

怎么才算完成了？

How 怎样做专项测试？

发现问题

例如启动速度，内存消耗…. . 重点测试场景的覆盖和数据的量化。

由此我们发现内存占用变高，我们的启动速度变慢了。

定位分析

我们发现问题，下一步就是定位问题。例如我们发现内存变大时取内存快照，卡顿时取堆栈。这些都是分析定位的能力。

解决问题

作为测试本身职责更多不在于解决问题，但是我们可以帮助开发让解决问题的能力组件化静态化；达到以点带面的效果。

什么阶段去做？

根据项目资源配比「人员，时间，测试目的」，可以在功能测试完成之后，或者回归过程中开始介入专项测试。也可在功能模块测试中穿插部分专项测试项.

每个阶段去做？

根据团队，对项目的测试目的，项目进行阶段等等。不同的时间段都是不同的策略。

做到什么颗粒度？

在项目迭代中持续去做，直到功能完成100%就可以做一轮完整专项测试。结合业务重要性，功能重要性，代码来一起评估专项测试通过标准。我们不可能每次都去做一次专项。

怎样才算完成？

达到在特定的测试范围内，达到测试目的即可。

参考「常规专项测试技术评审标准」

常规专项测试技术评审标准

测试维度	测试内容
客户端性能	内存占用:针对不同客户端，实际占用的物理内存PSS消耗合理; CPU占用:不同应用场景下，CPU占用合理; FPS数值:核心业务FPS无明显波动，稳定在一定数值以上; 流量消耗:以一定时间内的流量消耗为判断标准; APK大小:安装包过大建议使用CDN资源
适配兼容性	TOP100机器适配，适配不通过的机器用户占比不超过一定比例 Crash率不超过一定比例 适配专项测试期间无遗留crash问题
弱网络状况	应用核心功能正常 合理断线重连机制 合理异常提示
安全防护	核心逻辑校验 敏感信息保护 代码混淆
服务器性能	针对服务器的技术规范要求，比如:容灾处理、过载保护要求、防雪崩机制等，全部测试通过; 单机的综合场景测试，必须满足性能基线要求，并且各事务单位时间内可基本处理成功; 服务进程无重启，无内存泄漏的情况下长时间运行，各事务基本可处理成功;

Tool

那些实用的工具？

GT

1. 可对APP进行快速的性能测试(CPU、内存、流量、电量、帧率/流畅度等等)
2. 开发日志的查看、Crash日志查看、网络数据包的抓取、APP内部参数的调试、真机代码耗时统计。
3. GT支持iOS和Android两个手机平台

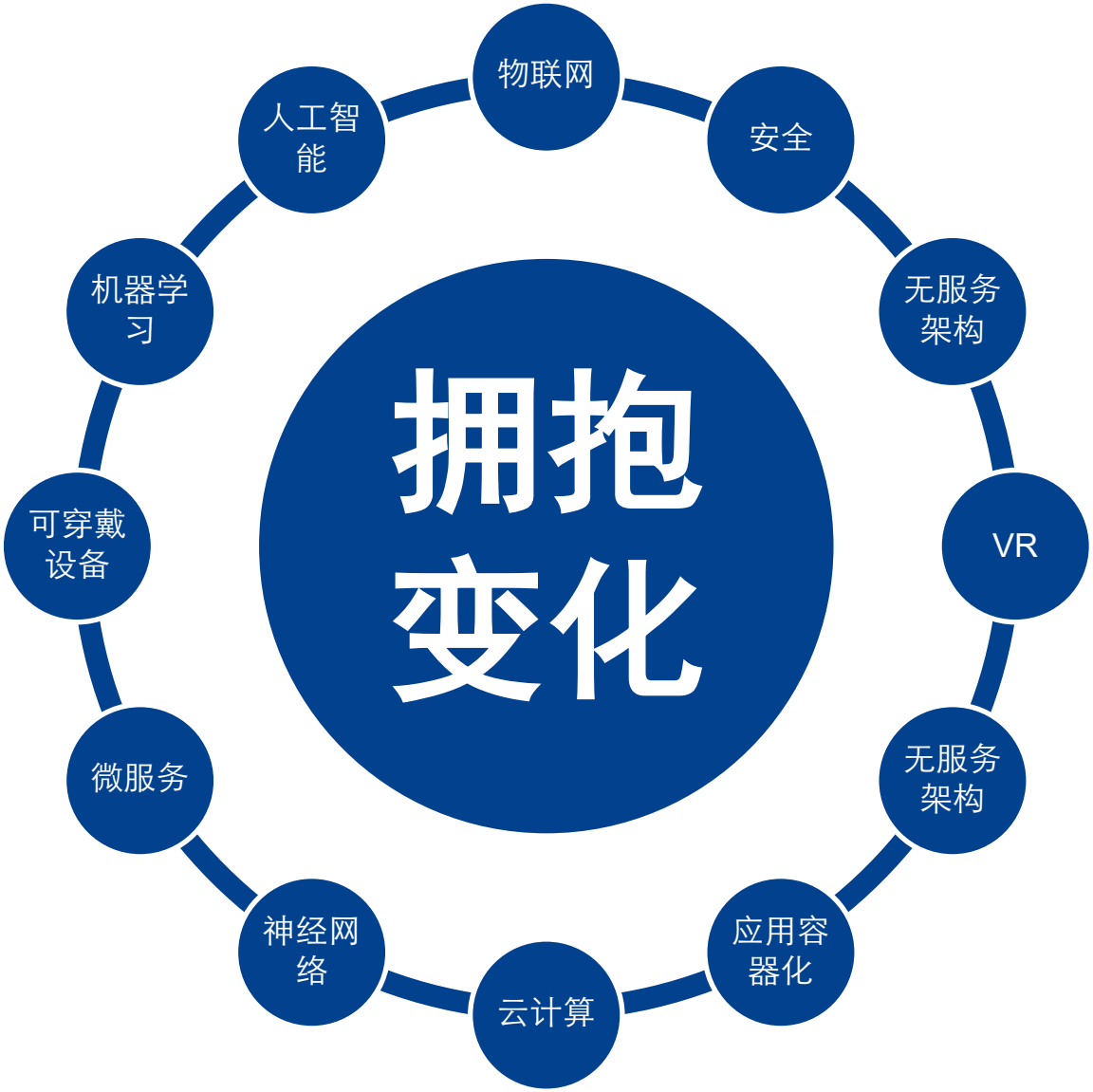
静态代码扫描工具

1. 四个主流的扫描工具：coverity、infer、clang、oclint

	学习成本	准确率	可扩展性	接入Ci难度	是否开源
Coverity	低	高	低	低	商业
Clang	中	中	高	中	开源
Infer	中	中	中	高	开源
Oclint	中	低	中	中	开源

2. 准确率：coverity > infer >clang > oclint;

还有很多工具，
还有很多挑战……


拥抱变化
物联网
安全
无服务架构
VR
无服务架构
应用容器化
云计算
神经网络
微服务
可穿戴设备
机器学习
人工智能

谢 谢！