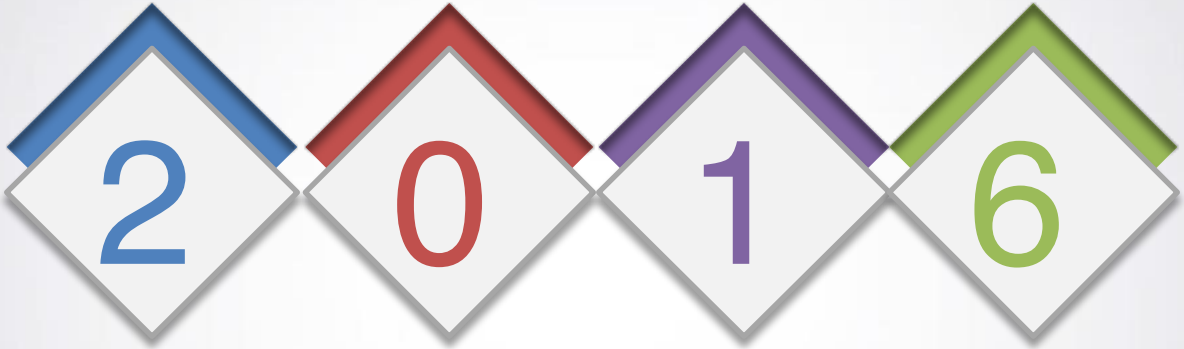


Android热修复系统设计

刘昭

目录

1 系统背景

- 线上Bug多
- HotFix不稳定
- 需要灰度发布

- 线上Bug多，信任度低；
- 目前采用HotFix不稳定，最后一道防线不牢靠；
- 需要灰度发布、灰度测试；

2 设计目标

- 功能
- 动态
- 灰度与监控
- 安全

功能完备、稳定；

动态配置；

可灰度；

完整的监控流程；

安全校验与安全模式；

设计目标

功能：

功能完整，免安装修复、灰度；
稳定、可用。

动态

根据版本；
可上、可下；
可配置；网络环境、立即生效等。

灰度

粗粒度的灰度验证；
精准的灰度验证。

监控

用户信息上报；
重要指标上报。

安全

安全校验；
Patch可清除、可回退；
安全模式、同步热修复等

其它

开发透明、无需改变编码习惯；
性能优秀、不影响App性能；

3 修复流程

Tinker+自己后端
参见：
http://www.processon.com/diagraming/582eb837e4b05594f5062922

4 系统挑战

- Patch包大
- Patch失效
- 进程易被Kill
- 其它

系统挑战

- Patch包大
  - DexDiff生成差异包，本地合成；
  - Wifi环境下载；
- 进程易被Kill
  - 并发dexopt、oat过程；
  - 单开Service进程；
  - 进程保活；
  - 重试机制；
- Patch失效
  - 版本可回退；
  - 安全模式；
- 其它
  - 灰度策略；
  - 监控；
  - 个性化；

5 实施方案

- 阶段一
- 阶段二
- 阶段三

第一阶段：核心在拥有能力；稳定的修复、版本灰度、功能灰度等基础能力，粗粒度的灰度策略、监控、安全模式、机型黑名单等。

第二阶段：核心在优化，精细化；提升修复质量与性能优化，完善全方位监控，可回退任意版本；客户端进程保活，细粒度App安全模式等。

第三阶段：核心在动态：与配置中心打通，个性化修复、灰度策略；A/BTest打通，细粒度的灰度策略，Push通道。

第一阶段：核心在功能，各项基础功能完备；

- 功能上：
  - 稳定的修复、灰度能力；
- 动态性上：
  - 可上、可下；
  - 可配置：wifi环境、所有网络下载？是否立即生效；
- 灰度上：
  - 粗粒度的灰度，所有用户限制人数灰度；
- 安全性上：
  - 粗粒度进程保活；
  - 可清除Patch包。可回退到发包版本；
  - 粗粒度的安全模式：自我修复、同步热修复；
  - 机型黑名单；
- 监控上：
  - 粗粒度重要指标上报。

第二阶段：核心在优化、精细化；

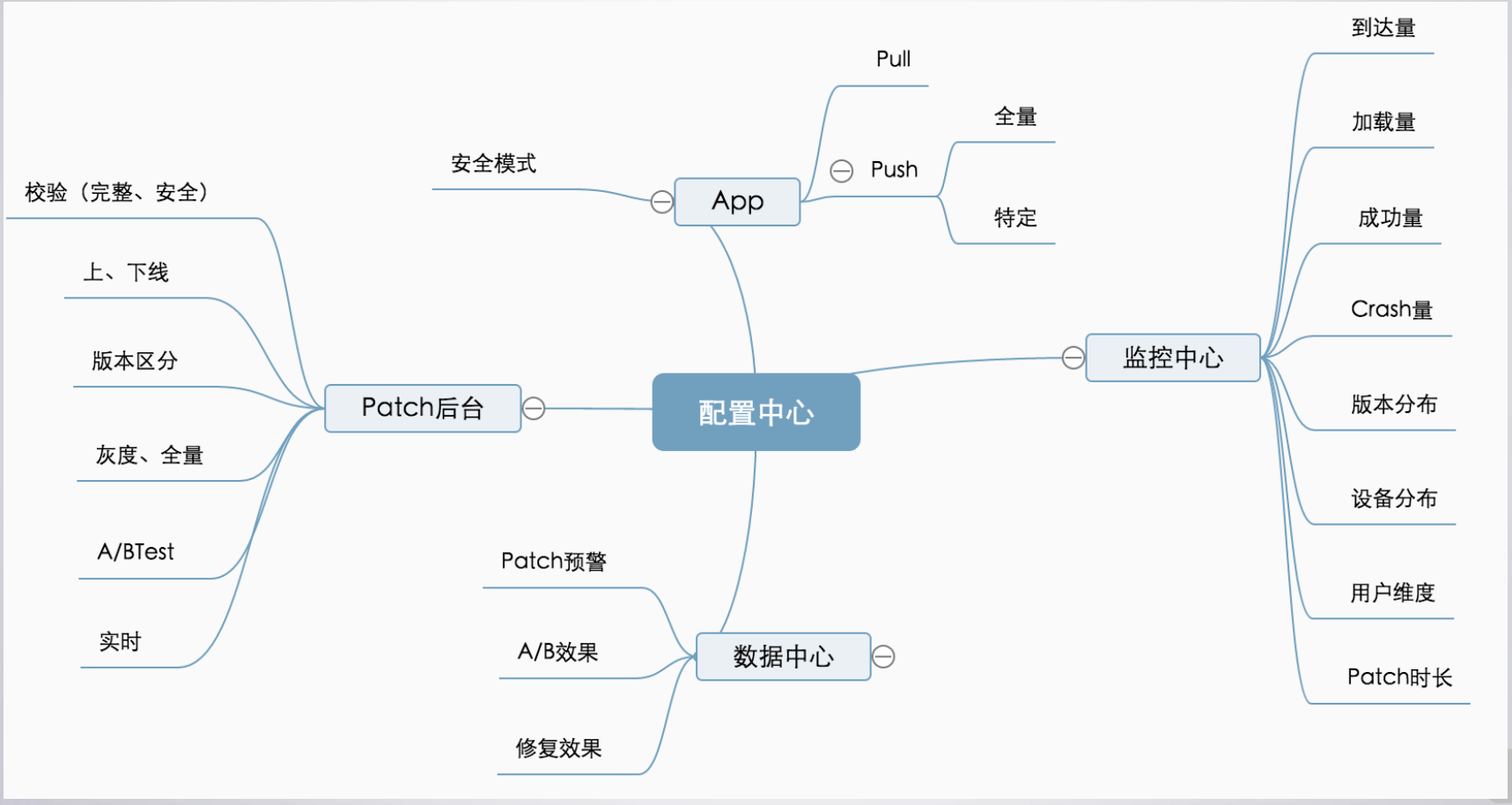
- 客户端优化：
  - 性能优化：解决内存占用高、卡顿问题，减少被Kill几率，提高成功率；
  - 进程保活：细粒度的进程保活，增加后台存活时长；
  - 安全模式：细粒度的安全模式，分等级分场景分维度自我修复；
  - Pull策略优化：增加触发场景；
  - 重试机制；下载成功随后过程失败。

- 后台优化：
  - 支持断点续传；
  - 全维度监控；下载、合并、释放、执行等过程；
  - 报表：生效过程转化率。

第三阶段：核心在精准；

- 细粒度灰度上线：针对特定人群（Crash过）之类；
- A/BTest打通：可使用灰度来做A/B；
- 添加Push通道，全部用户可达；
- Crash预警；

热修复系统


配置中心
Patch后台
校验（完整、安全）
上、下线
版本区分
灰度、全量
A/BTest
实时
App
安全模式
Pull
Push
全量
特定
数据中心
Patch预警
A/B效果
修复效果
监控中心
到达量
加载量
成功量
Crash量
版本分布
设备分布
用户维度
Patch时长

Thank You ！