架构迎接未来变化

IAS2017 • NANJING

爱游戏SDK业务技术栈演进 (Android)

炫彩互动网络科技有限公司
韩玮

00 Part Zero
业务场景

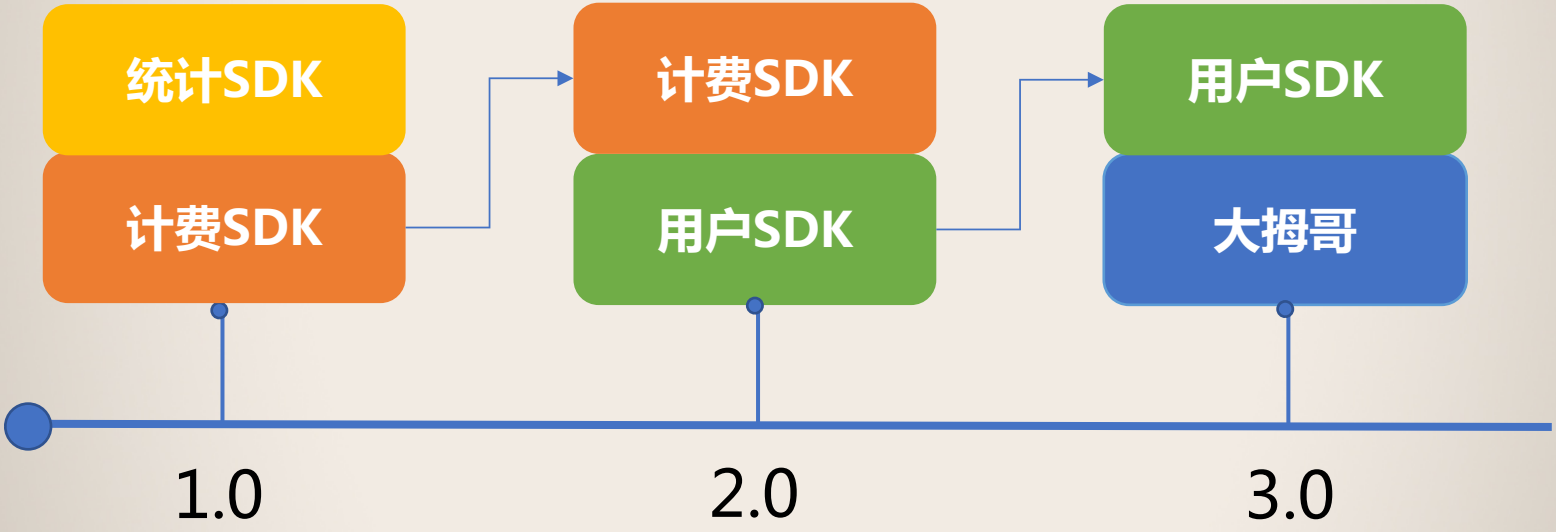
01 Part One
SDK 1.0——传统式

02 Part Two
SDK 2.0——动态化

03 Part Three
SDK 3.0——插件化

04 Part Four
一些思考

0.0 4种业务场景



00 Part Zero
业务场景

01 Part One
SDK 1.0——传统式

02 Part Two
SDK 2.0——动态化

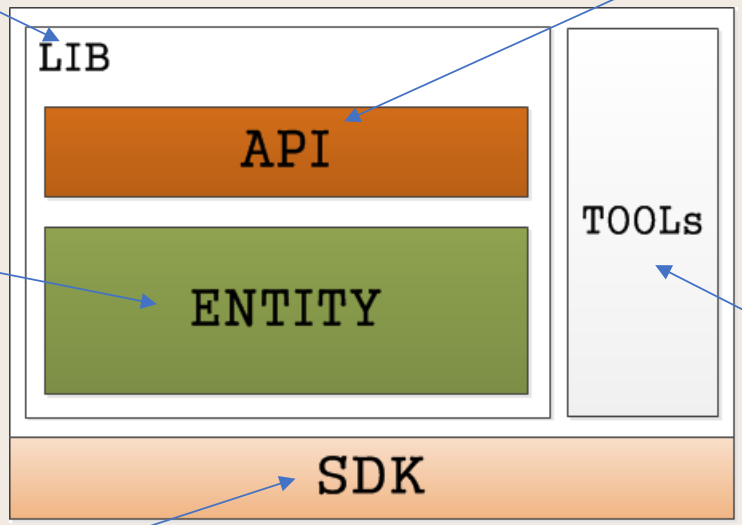
03 Part Three
SDK 3.0——插件化

04 Part Four
一些思考

1.1 传统型SDK—满足基本需求

库文件，Android平台
一般为.jar, .so, .a等

应用编程接口，Android平台一般为
Java方法定义和C/C++的头定义提供



SDK能力的具体
程序实现

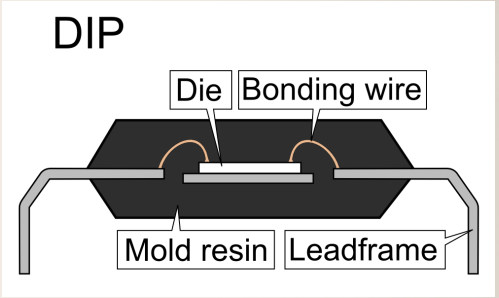
辅助工具，如文档，
脚本，IDE，插件等

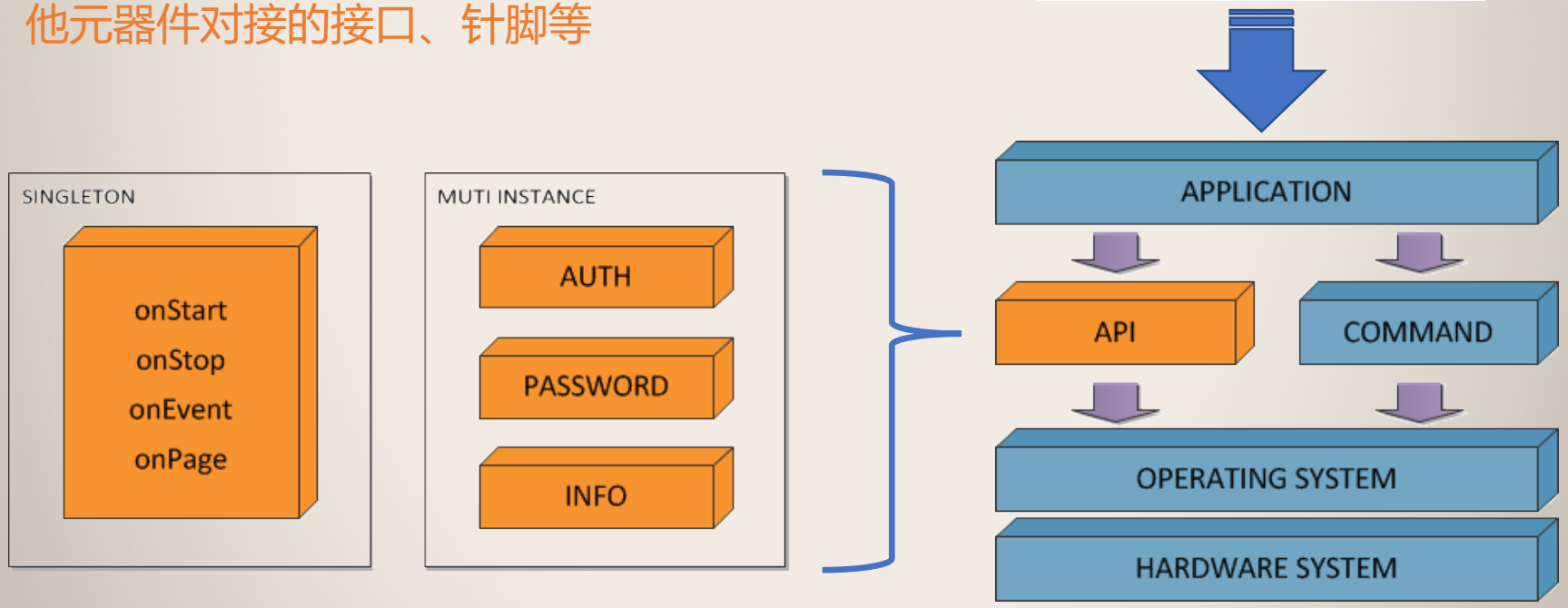
软件开发套件，一般包含程序实体(LIB)
和开发工具(TOOLs)两部分.

1.2 SDK的核心—能力的封装

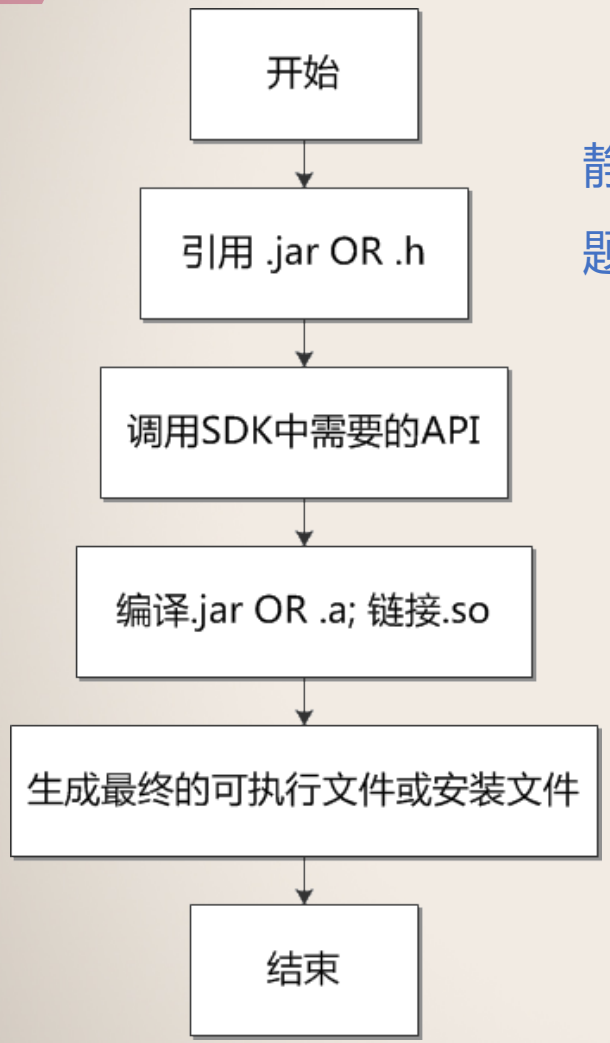
封装(Package)

封装一词源于半导体设计，电子原件按照一定的规则预留出和其他元器件对接的接口、针脚等





1.3 传统型SDK的局限



静态集成，输出移动端的安装包之后，如果发现问题或者迭代更新版本后只能更换SDK后重新打包

安全性，较差，原始代码容易易于被反编译

总结一句就是：傻白甜

00 Part Zero
业务场景

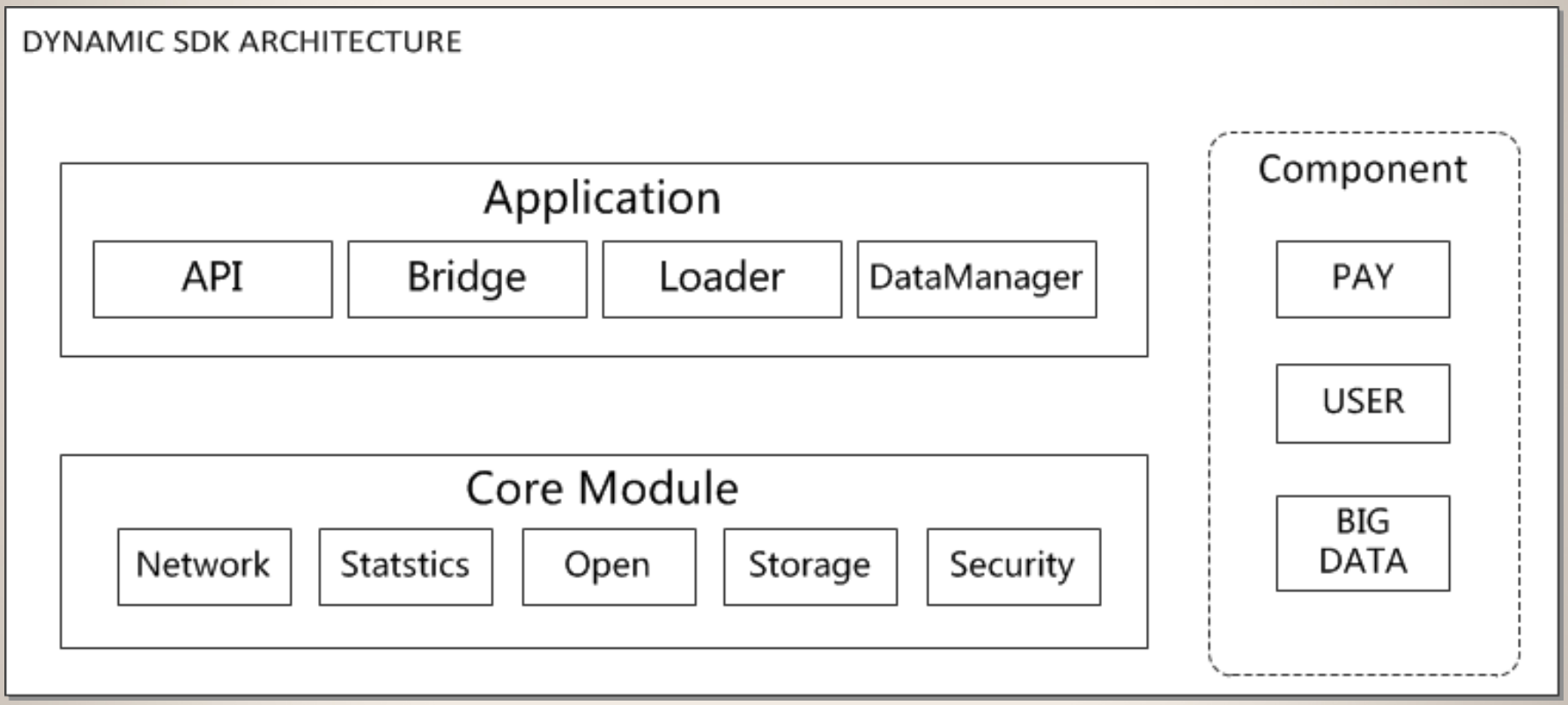
01 Part One
SDK 1.0——传统式

02 Part Two
SDK 2.0——动态化

03 Part Three
SDK 3.0——插件化

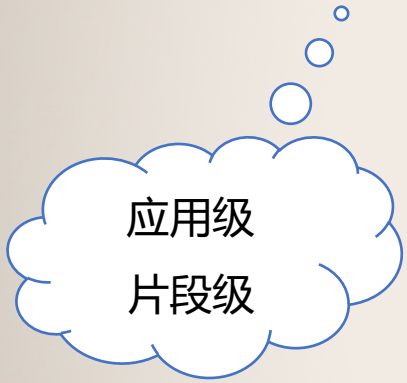
04 Part Four
一些思考

2.1 动态化SDK—安全、轻巧

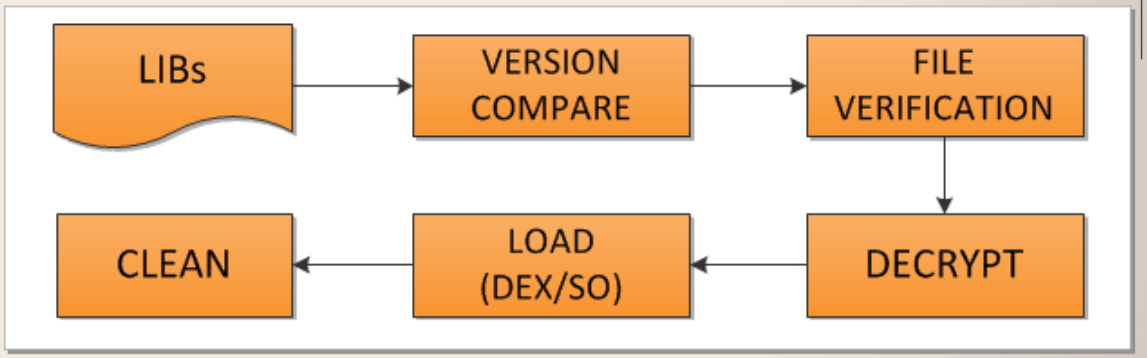


2.1 动态化SDK—安全、轻巧

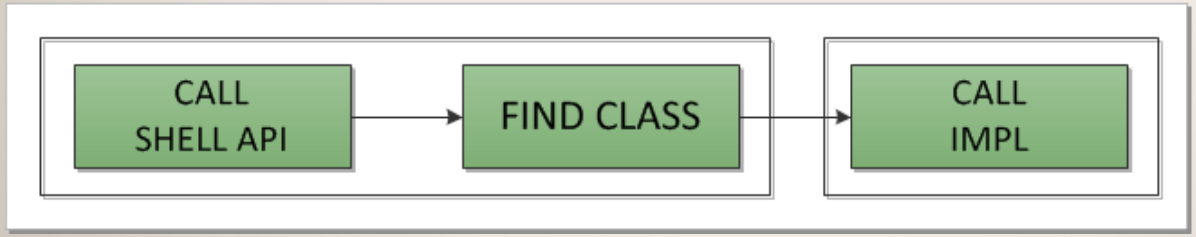
DYNAMIC SDK LOADER: INIT



DYNAMIC SDK LOADER: LOAD



DYNAMIC SDK LOADER: CALL



2.2 安全是第一考虑

针对安全性问题

- Shell90%由So实现，Java部门只有一些入口方法；(理论和现实)
- 动态加载，对核心代码的加固成为了可能
- 代码混淆：C++函数宏替换；Java 代码随机异或加密
- 静态字符串的安全存储：存储在某个伪装文件中

2.3 热更新是关键词

1. 需求：新功能开发 OR 修复BUG
2. 开发新功能 OR 修复BUG
3. 后台上线发布
4. APP 端生效
5. 失败回滚机制

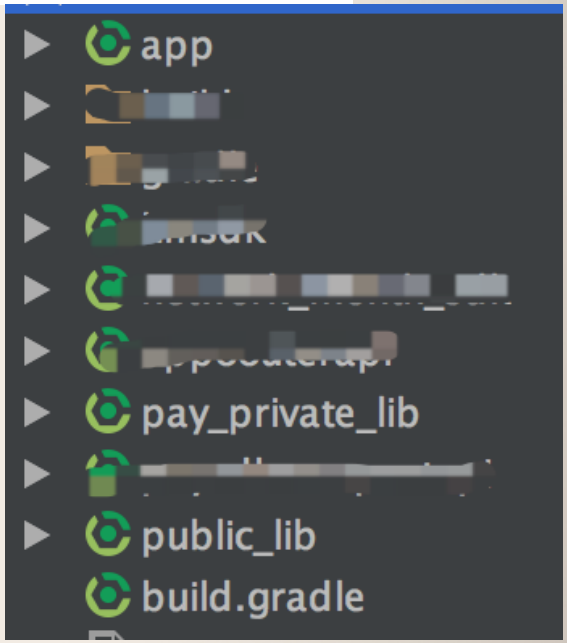
热更新可用于功能新增、BUG修复、AB测试、业务管控等

2.4 动态式SDK的局限性

```
public static int getLayout(String idName, Context context) {
    return getResById(idName, "layout", context);
}
```

资源

代码



- 要遵循一定的开发规范
- UI开发较为麻烦，无法使用所见即所得方式
- 四大组件只设计一个入口，导致无法自由增加组件(非机制原因)

00 Part Zero
业务场景

01 Part One
SDK 1.0——传统式

02 Part Two
SDK 2.0——动态化

03 Part Three
SDK 3.0——插件化

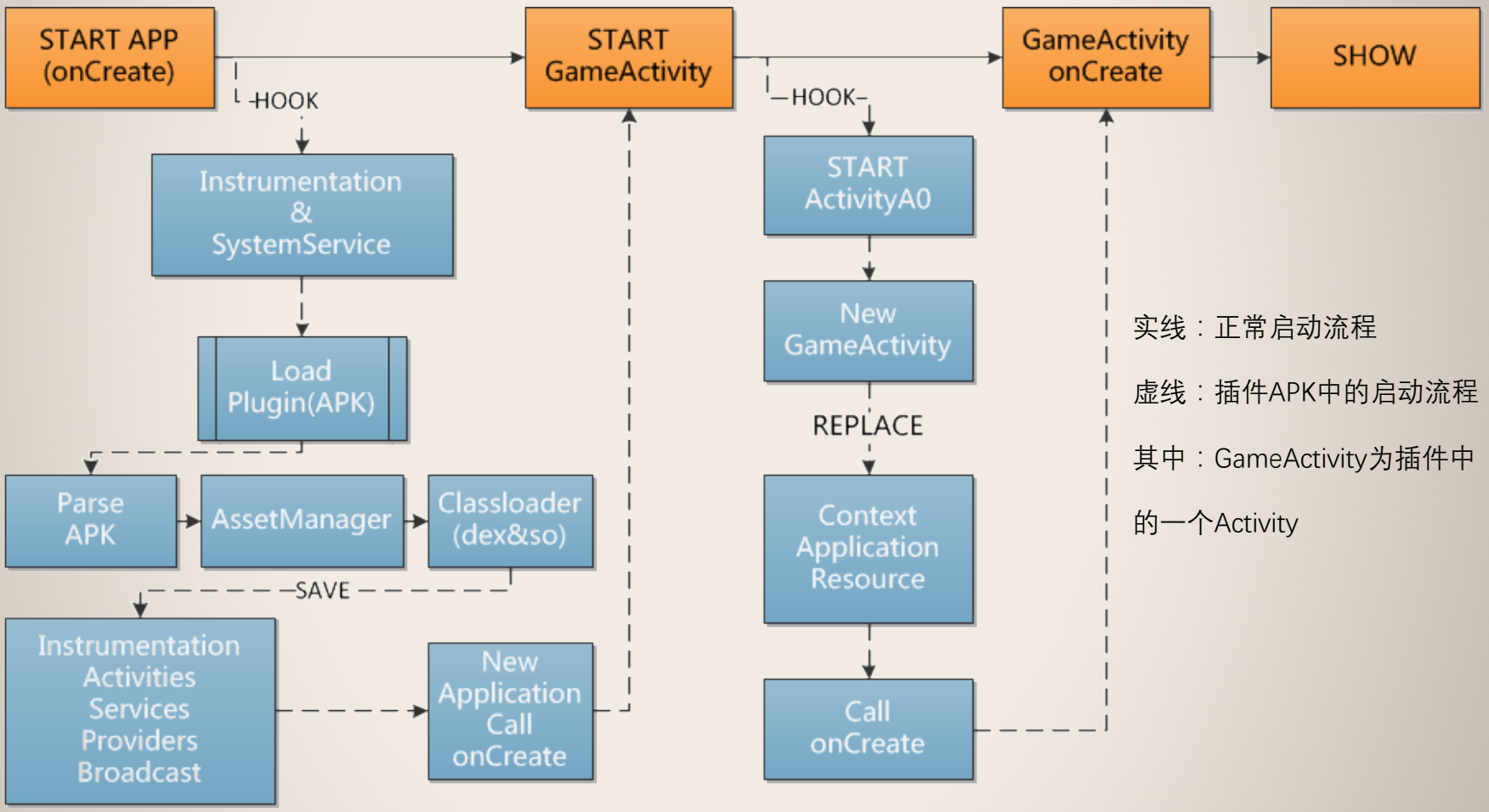
04 Part Four
一些思考

3.1 插件化SDK—以不变应万变

针对动态化SDK的局限性

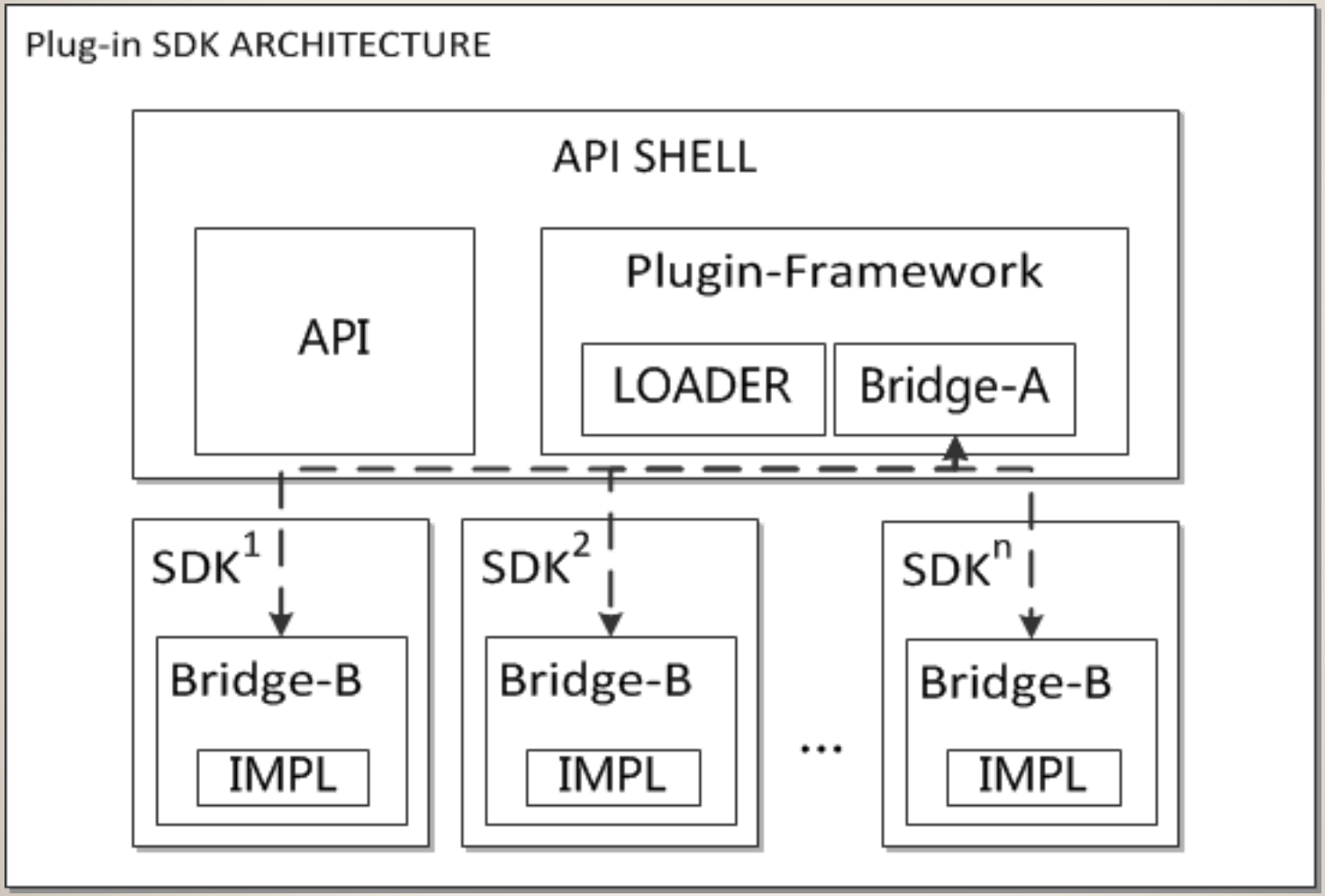
- SDK实质上就是APK
- 开发SDK的过程与开发APK的过程一致
- 核心问题
  - 资源管理
  - 组件管理
  - 通信

3.1 插件化SDK—以不变应万变



Plug-in SDK RUNNING PROCESS

3.1 插件化SDK—以不变应万变



3.2 插件化方案

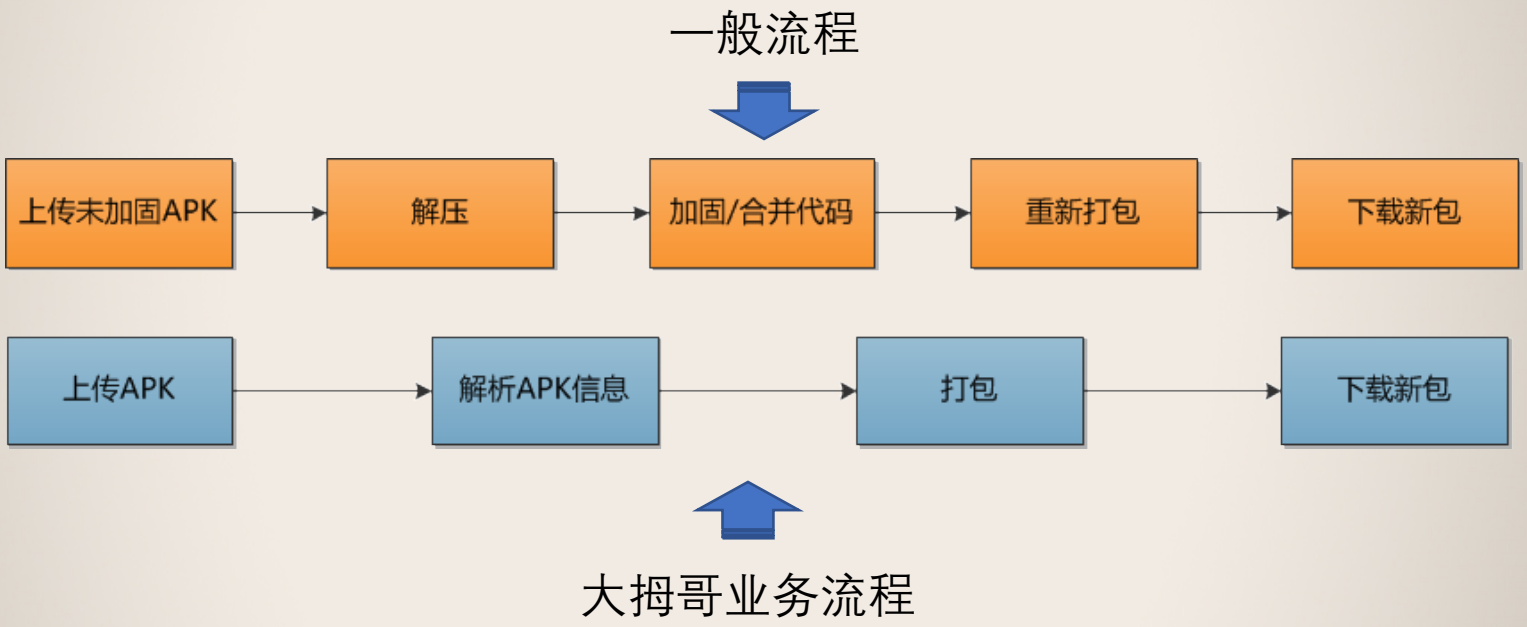
开源项目

早期，维护阶段：
DroidPlugin
Android-Plugin-Framework
dynamic-load-apk
AndroidDynamicLoader
DynamicAPK
……

推荐，活跃：
RePlugin(360)
VirtualAPK(滴滴出行)
Atlas(阿里)

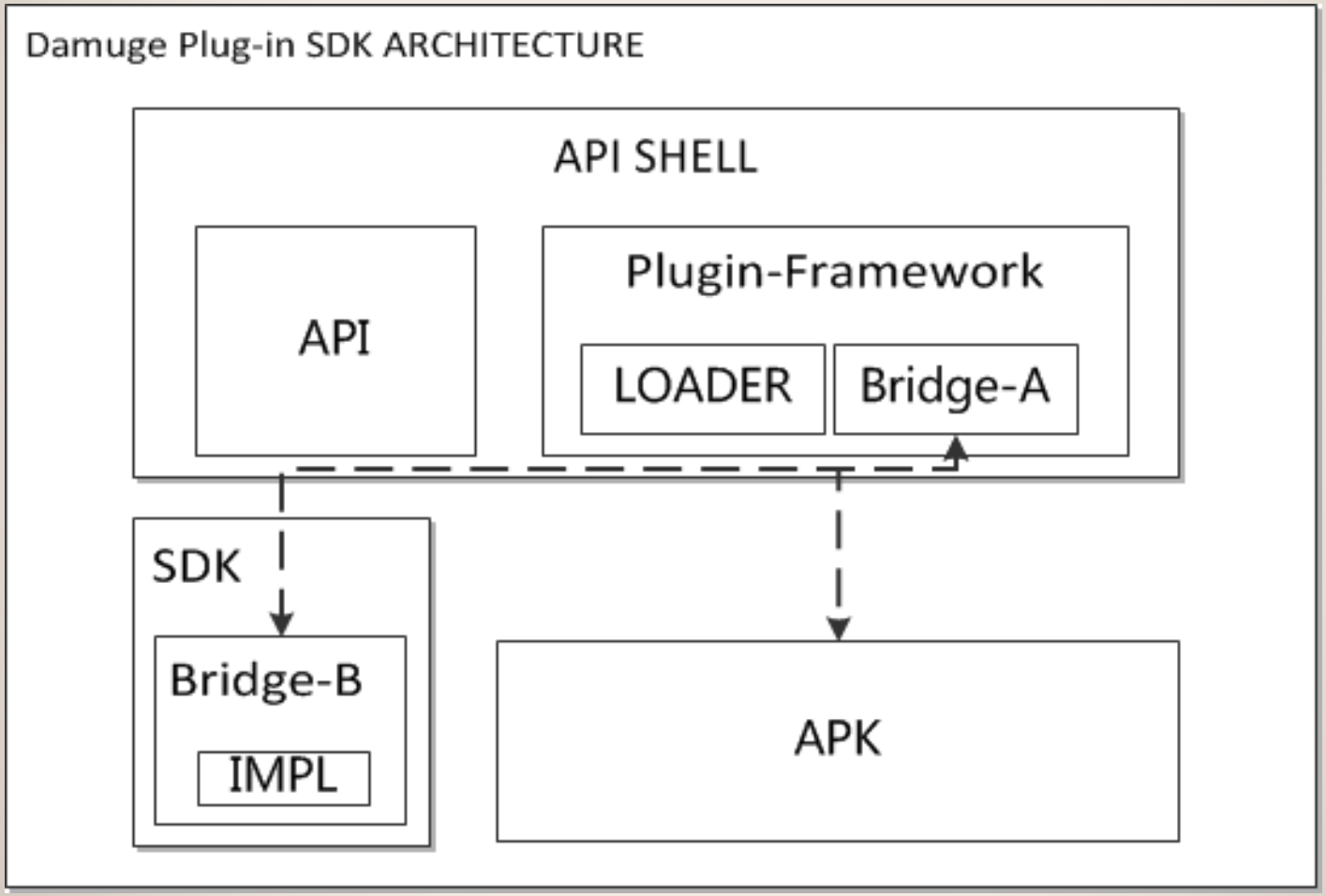
3.3 反向SDK接入—去API化

加固类



服务类

3.3 反向SDK接入—去API化



3.4 完美、万金油是不存在的

理想与现实的差距

- 兼容性问题
  - 设备兼容
  - Android版本兼容
  - APK功能性兼容
  - 第三方SDK兼容
- 通信问题
- 数据共享问题

- 挑选合适的业务线
- 不断增强兼容性(改代码)
- 不断增强兼容性(改代码)
- 不断增强兼容性(改代码)
- 不断增强兼容性(改代码)
- 弃坑

00 Part Zero
业务场景

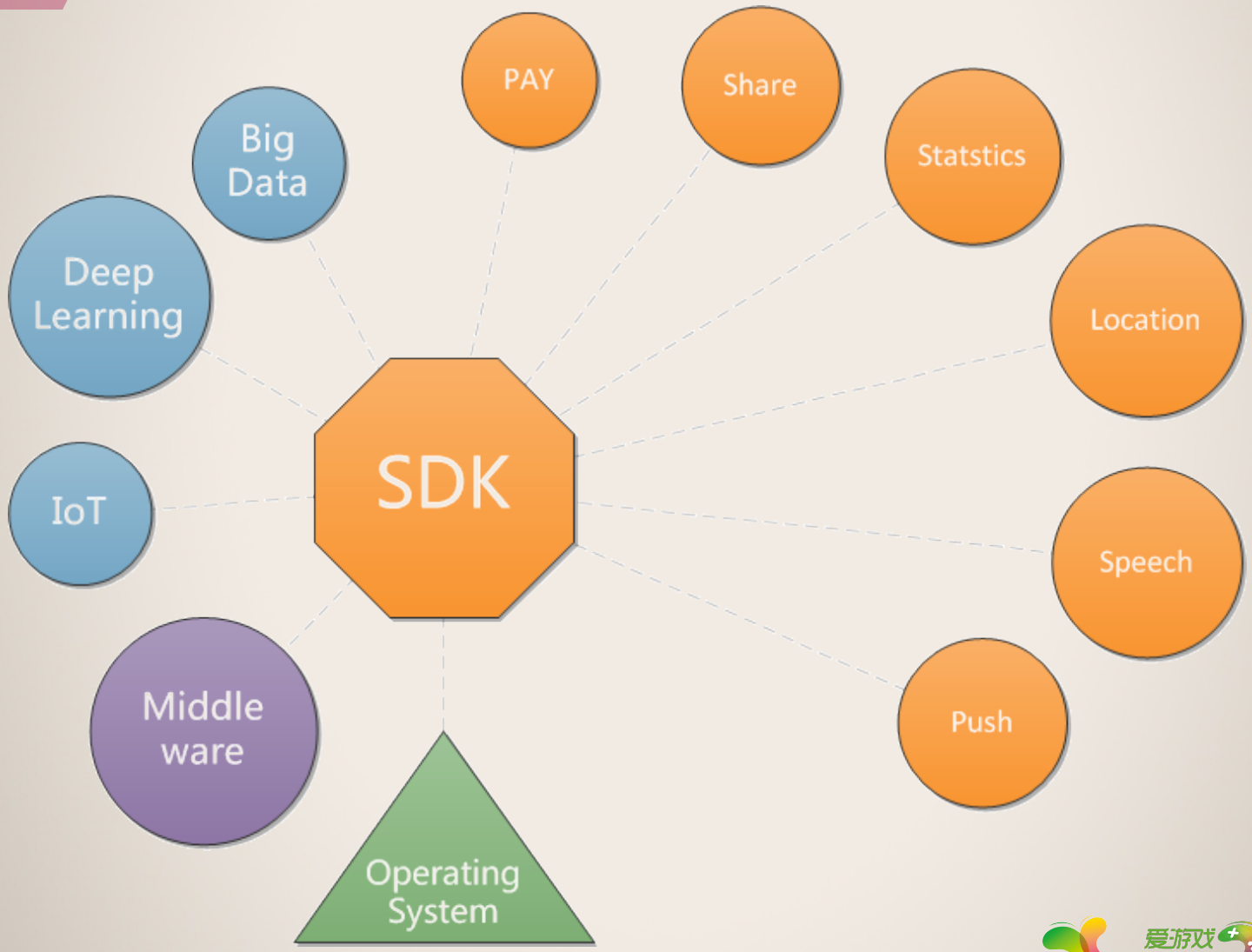
01 Part One
SDK 1.0——传统式

02 Part Two
SDK 2.0——动态化

03 Part Three
SDK 3.0——插件化

04 Part Four
一些思考

4.1 SDK的产品会越来越丰富



4.2 可定制化是趋势之一

静态定制 .01

- 勾选下载
- 按需取用

动态定制 .02

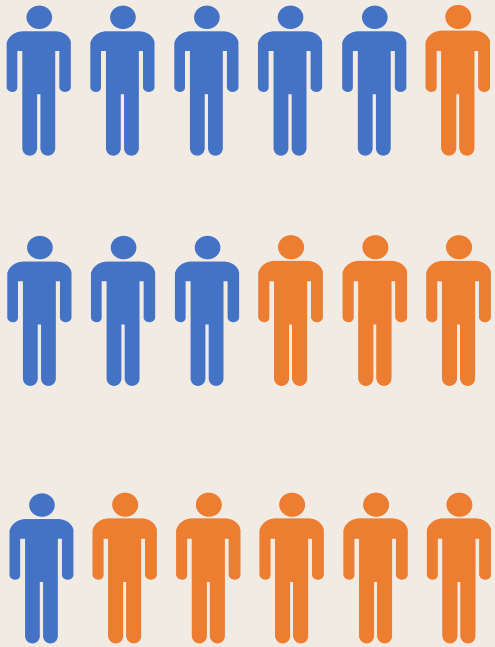
- 微服务
- 动态控制

4.3 开发更复杂、集成更简单

可定制
API进一步简化
定制的编译工具
强大的OneStep系统

可升级
API简化
安全性增强

能用就行



一行代码搞定

文档详细
API简单好用
被破解的风险降低

文档欠缺
兼容性太差
稳定性得不到保证
不同语言的代码接入方法不同

4.4 跨平台化开发

React Native　　H5　　C/C++

谢谢