GIAC

主办方：msup | ARCHNOTES 架構

GIAC

全球互联网架构大会

GLOBAL INTERNET ARCHITECTURE CONFERENCE

面向国际化业务的 Android 组件开发框架

张明庆 字节跳动 Android高级工程师

目录

国际化的挑战

国际化业务

150+ 国家和地区；TikTok 连续五个季度（App Store）下载量最高

臃肿的代码

多个产品

多套资源

百万行+代码

几十个依赖库

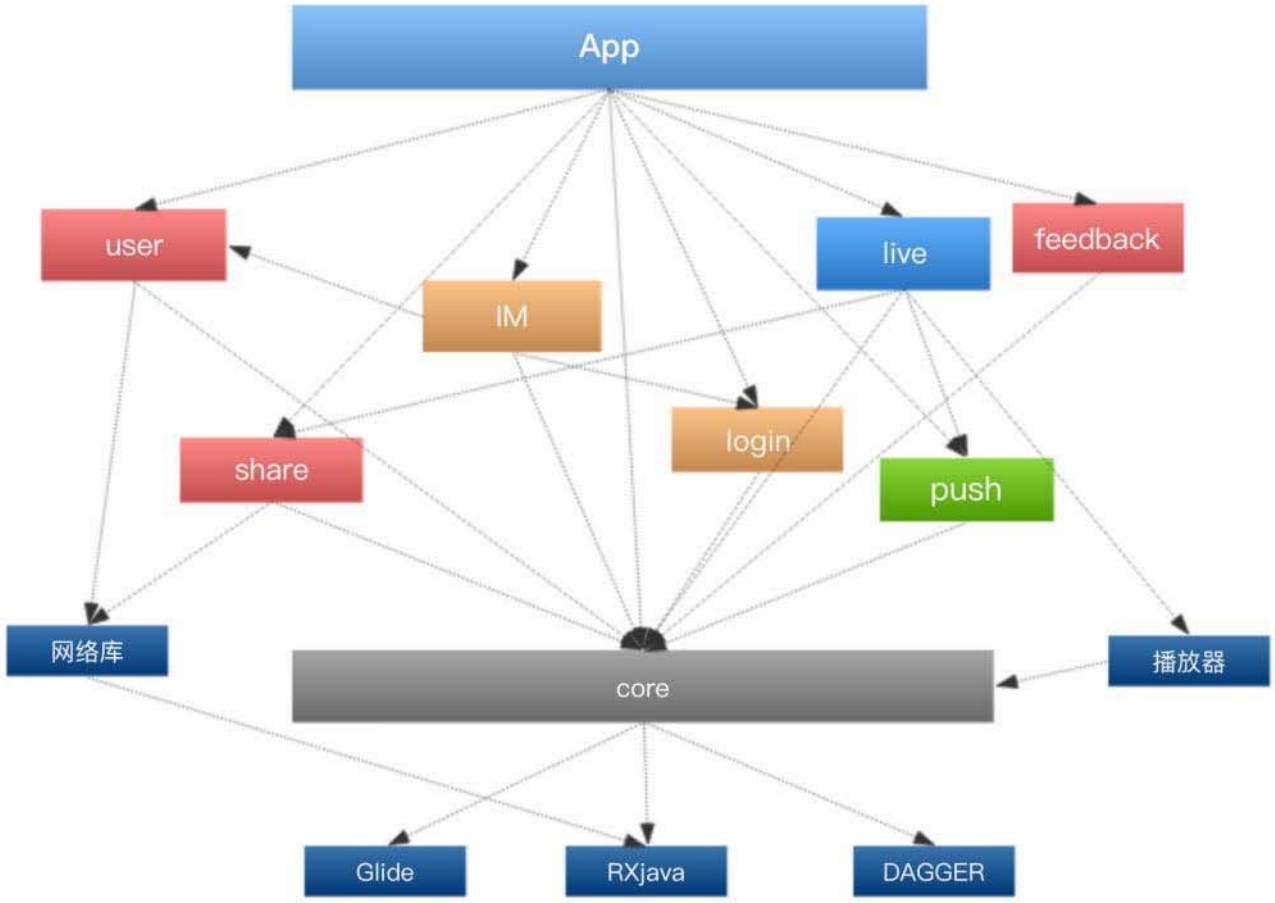
10+登录方式

N种支付方式

30+语言包

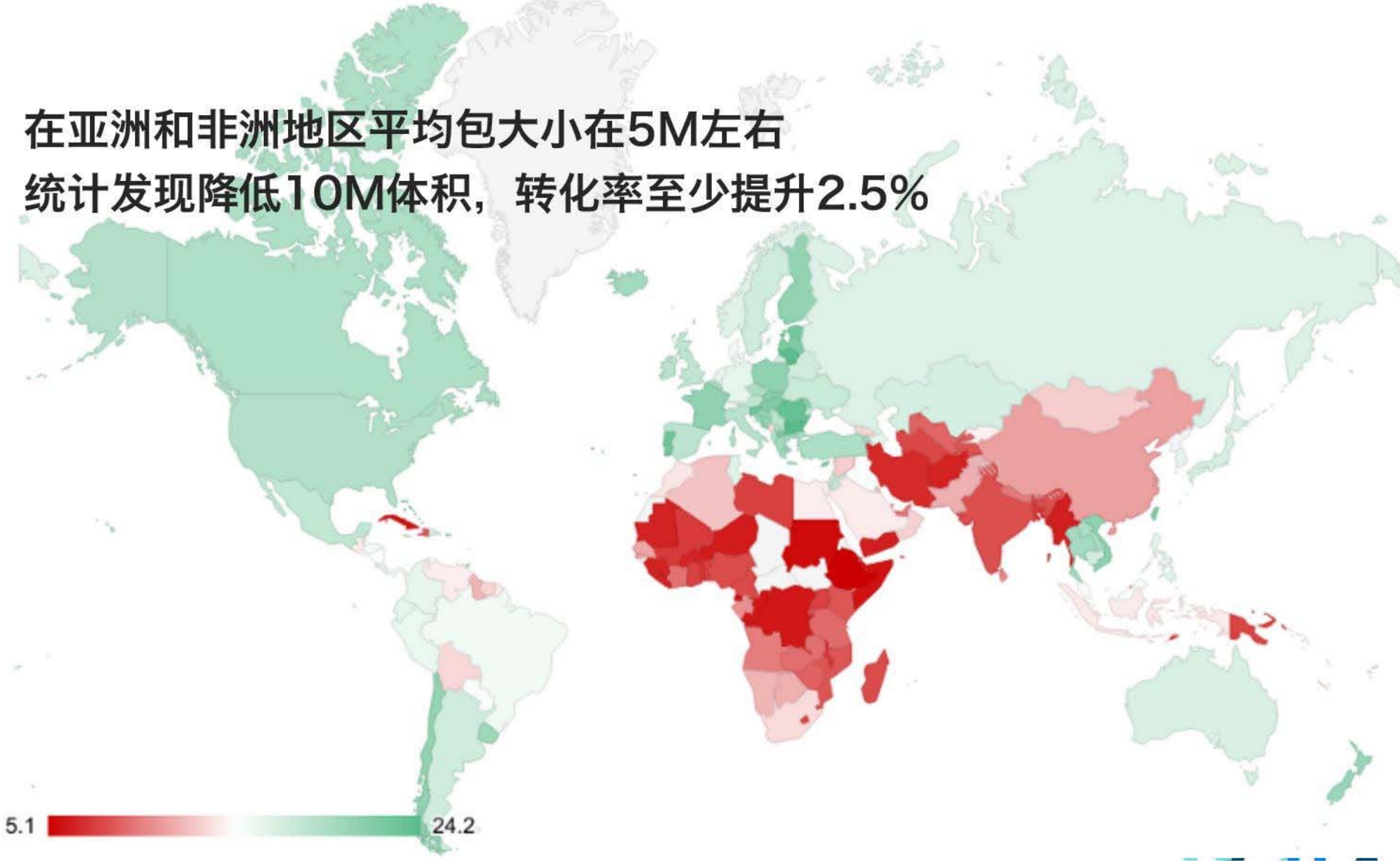
32/64位

混乱的依赖


App
user
IM
live
feedback
share
login
push
网络库
core
播放器
Glide
RXjava
DAGGER

世界不同地区 APK 大小

在亚洲和非洲地区平均包大小在5M左右
统计发现降低10M体积，转化率至少提升2.5%




错误传递
编译慢
网络差
资源包
服务于
火山
5M+
包体积
语言包
抖音
sdk
Video
Vigo
印度
10min
非洲
机型差
150
5%
icon
TikTok
南美
字节跳动
百万代码
so
蔓延
国家
依赖混乱
bug
登录方式
代码庞杂
代码臃肿
下降10M

整体架构图

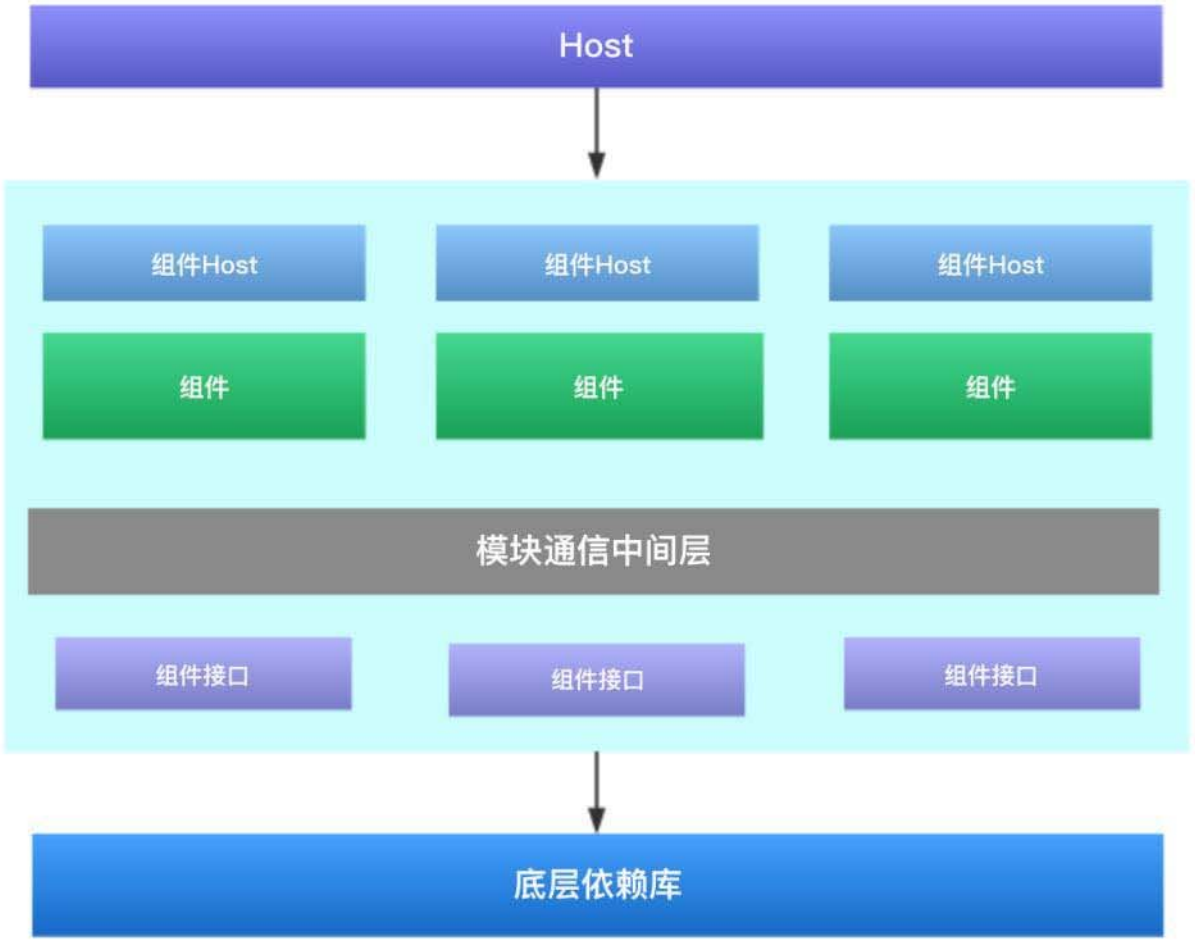

资源检查
仓库管理
依赖管理
代码检查
Host
组件Host
插件代理层
插件/AAB 适配层
组件
插件
Daymic Feature
底层依赖库
壳工程
组件模板
AS 插件
CI/CD

模块化拆分的新姿势

模块化的技术点

- 分层解耦
- 服务发现与注册
- UI跳转-路由表
- 单独编译与调试
- 组件集成-切换源码和 AAR
- …

组件化的架构图


Host
组件Host
组件Host
组件Host
组件
组件
组件
模块通信中间层
组件接口
组件接口
组件接口
底层依赖库

模块化拆分的新姿势

模块化通信

模块通信新姿势：　服务发现 -> 依赖注入

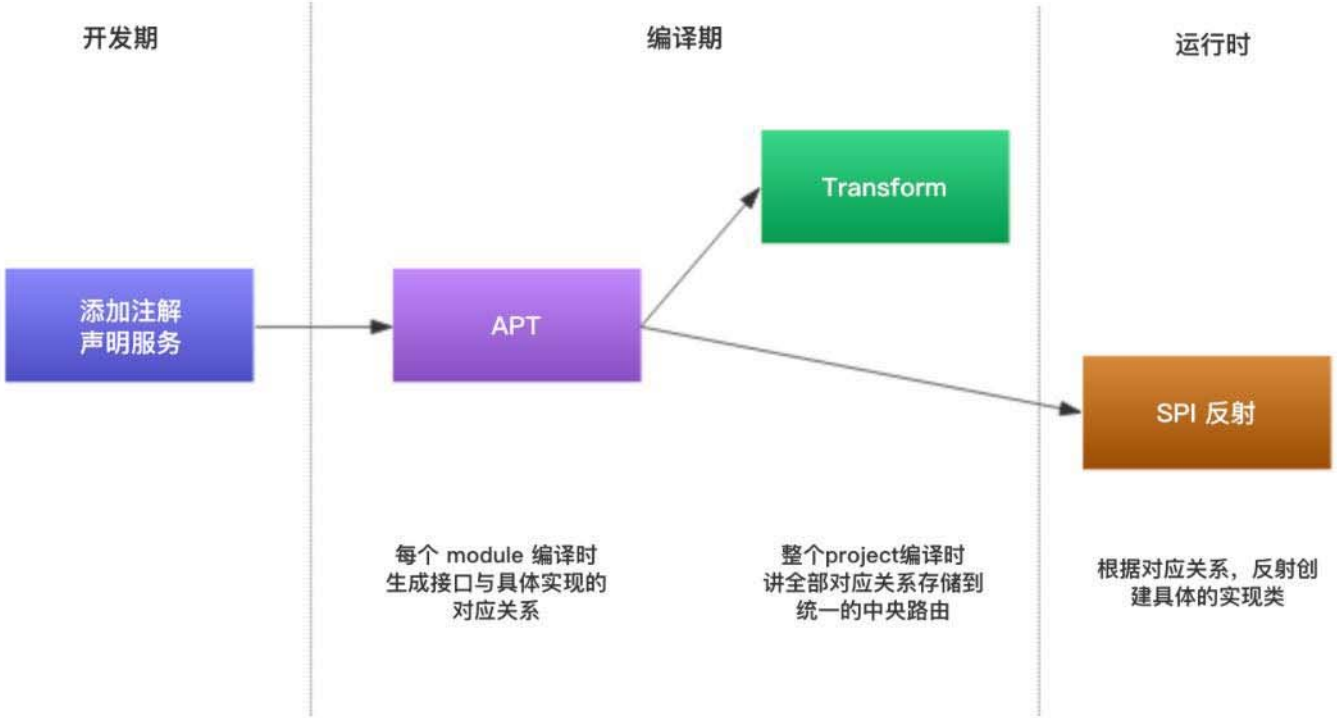
服务注册（Host）

```
1 CenterRouter.register(AccountService.class, new AccountServiceImpl());
2 ...
3 CenterRouter.register(LiveService.class, new LiveServiceImpl());
```

服务发现

```
1 CenterRouter.getService(AccountService.class).getUserId();
```

服务发现传统方式


开发期
编译期
运行时
添加注解
声明服务
APT
Transform
SPI 反射
每个 module 编译时
生成接口与具体实现的
对应关系
整个project编译时
讲全部对应关系存储到
统一的中央路由
根据对应关系，反射创
建具体的实现类

要解决的问题

1、服务发现应该是被动的

2、服务的依赖顺序

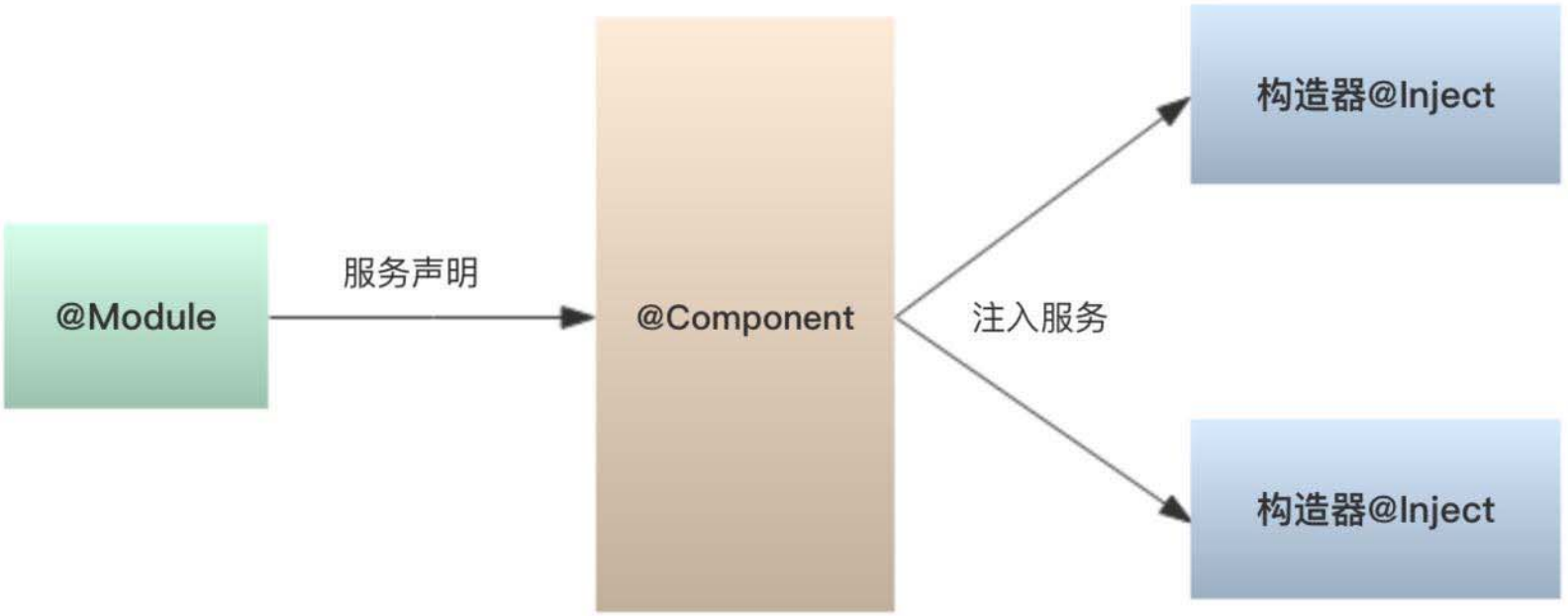
3、懒加载

4、无感知的注入

5、线程安全

6、编译耗时 or 运行时反射耗时

Dagger



Dagger

```
1 @Inject
2 public LoginService(AccountService account, LiveService live) {
3     ...
4 }
```

被动：使用方不关系谁提供服务，何时提供，甚至不知道中央路由的存在

依赖顺序：只需要在构造器中声明依赖就可以，不关心顺序

无感知的注入：基本无感知；即使对于 Activity/Fragment 也是如此

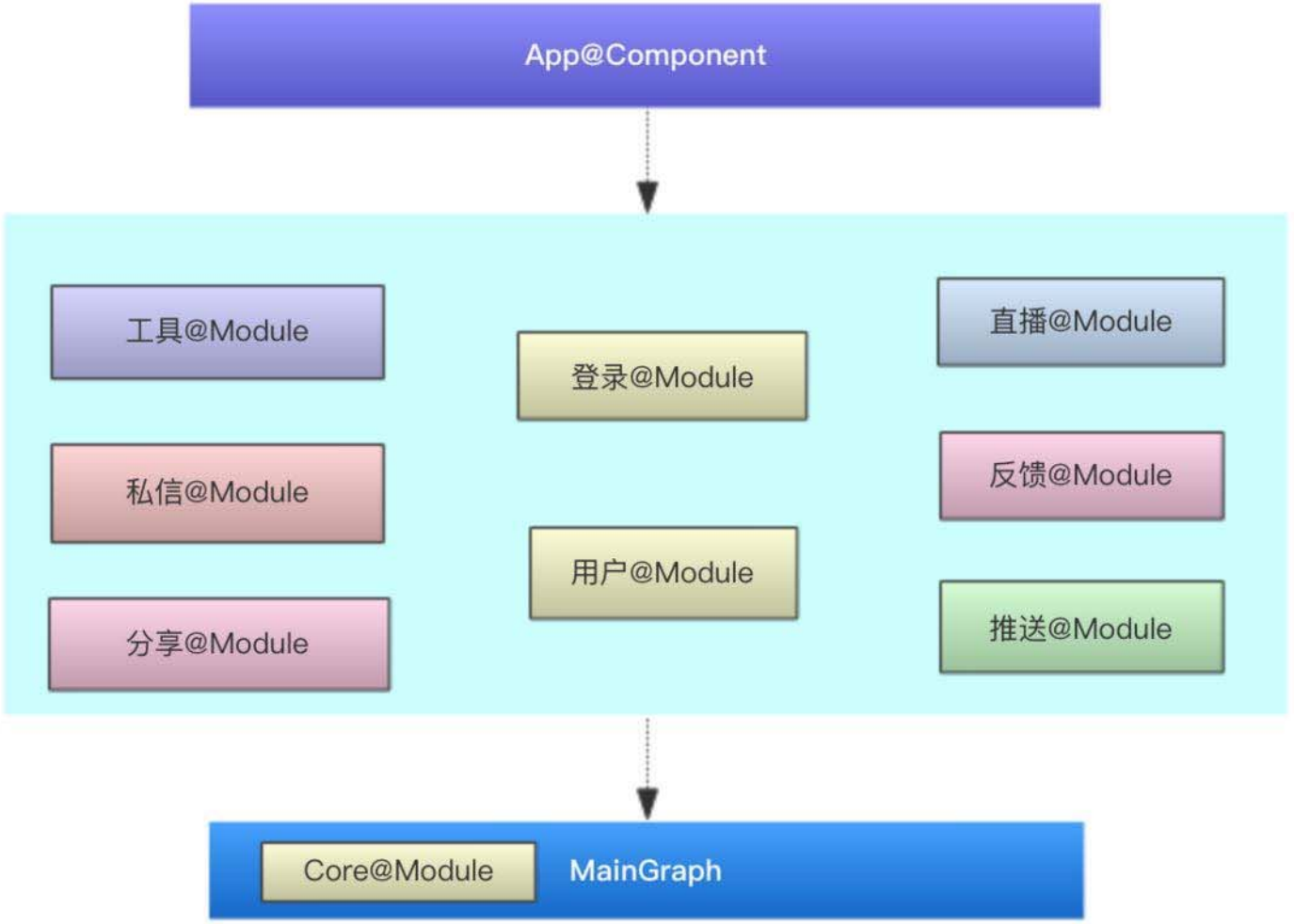
Dagger

```
1 @Inject
2 private Lazy<AccountService> account;
3 public void fun() {
4     account.get().xxxx();
5 }
```

线程安全和懒加载：天然支持

耗时：没有Transform 和反射

Dagger 依赖树结构



模块化拆分的新姿势

UI跳转-路由表

Intent跳转的N宗罪

```
1 Intent intent = new Intent(this, LoginActivity.class);
2 intent.putExtra("userName", "zhmqq");
3 intent.putExtra("channelCode", 2);
4 startActivity(intent);
```

有哪些问题：

- 直接引用LoginActivity.class，导致耦合
- 构建 Intent 和从 Intent 中获取参数都是胶水代码
- 不利于三端（Android/ios/H5）统一
- 传入参数的key硬编码，并且定义随意
- 传入参数没有类型校验
- 没有区分必传参数和非必传参数

Router跳转不是最优解

```
SmartRouter.buildRoute(context,"//login")
        .withParam("userName", "zhmqq")
        .withParam("channelCode", 2)
        .open();
```

还有哪些问题：

- ~~直接引用LoginActivity.class，导致耦合~~
- ~~构建 Intent 和从 Intent 中获取参数都是胶水代码（使用注解）~~
- ~~不利于三端（Android/ios/H5）统一~~
- 传入参数的key硬编码，并且定义随意
- 传入参数没有类型校验
- 没有区分必传参数和非必传参数

路由表-简洁可靠

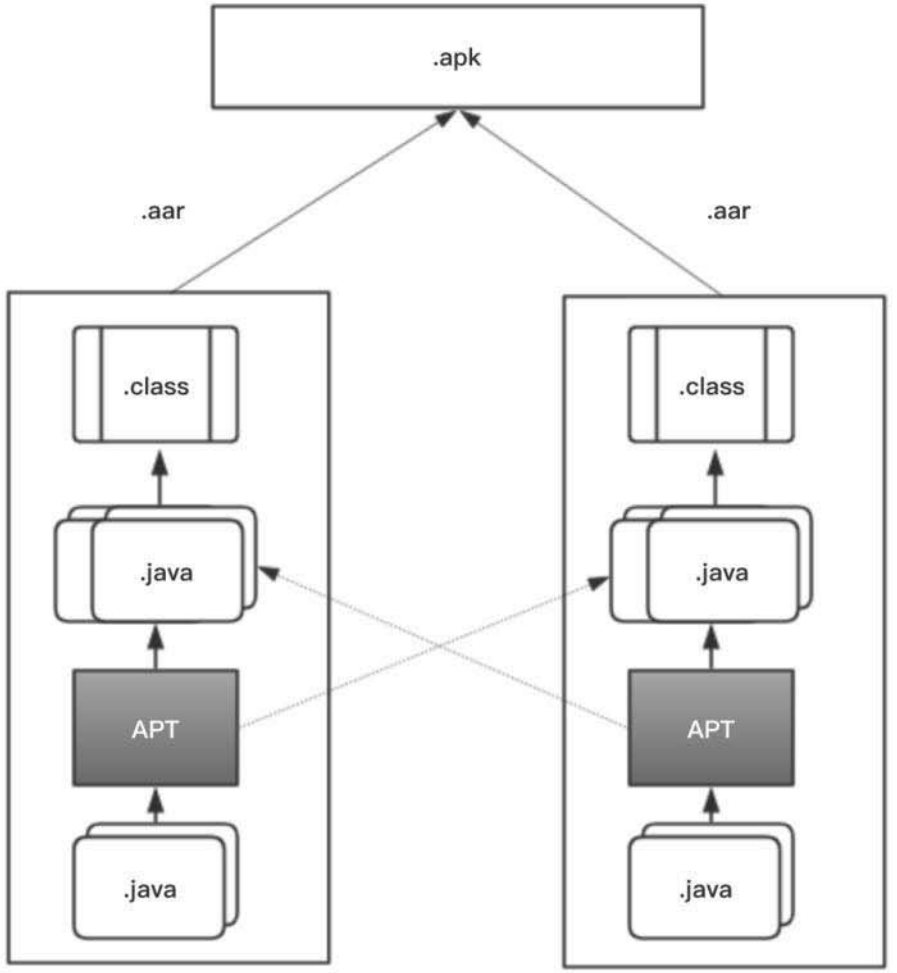
```
1 NavigatorAccount.login(context, "zhmqq")
2         .withChannelCode(2)
3         .open();
```

还有哪些问题：

- ~~直接引用LoginActivity.class，导致耦合~~
- ~~构建 Intent 和从 Intent 中获取参数都是胶水代码（使用注解）~~
- ~~不利于三端（Android/ios/H5）统一~~
- ~~传入参数的key硬编码，并且定义随意~~
- ~~传入参数没有类型校验~~
- ~~没有区分必传参数和非必传参数~~

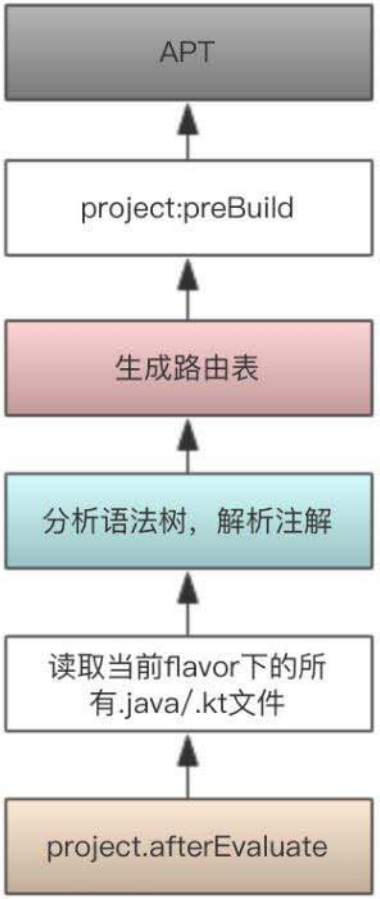
路由表的生成

- 路由表的生成依赖于注解的解析
- 但不能简单的使用 APT
- 并行编译，APT 需要所有类都存在



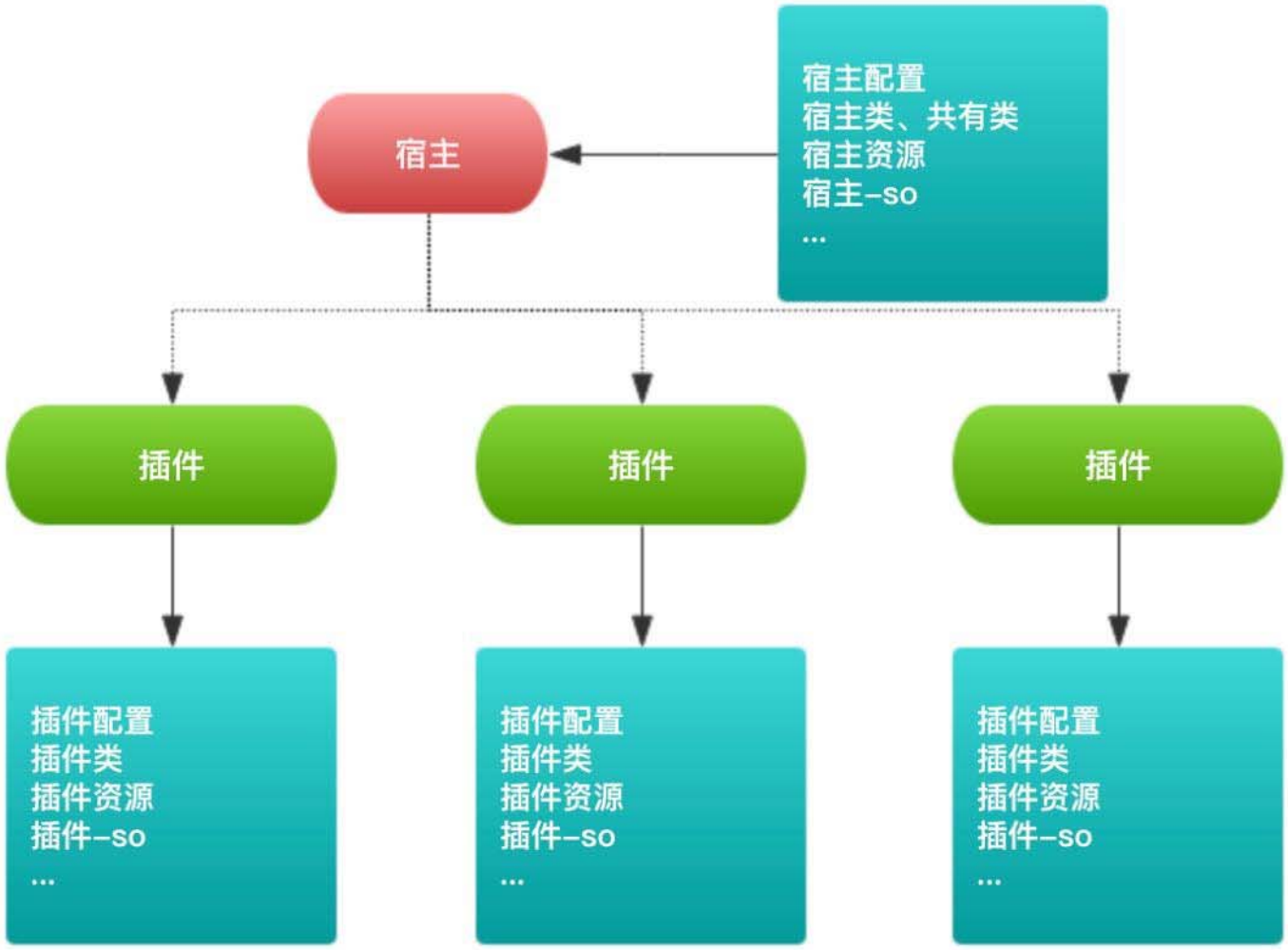
路由表的生成

- 将路由表的生成提前到每个 module config 完成之后
- 不要求编译通过
- 使用 TreePathScanner/KtFile 来解析 Java 和 Kotlin 类
- 使用 javapoet 生成路由表类

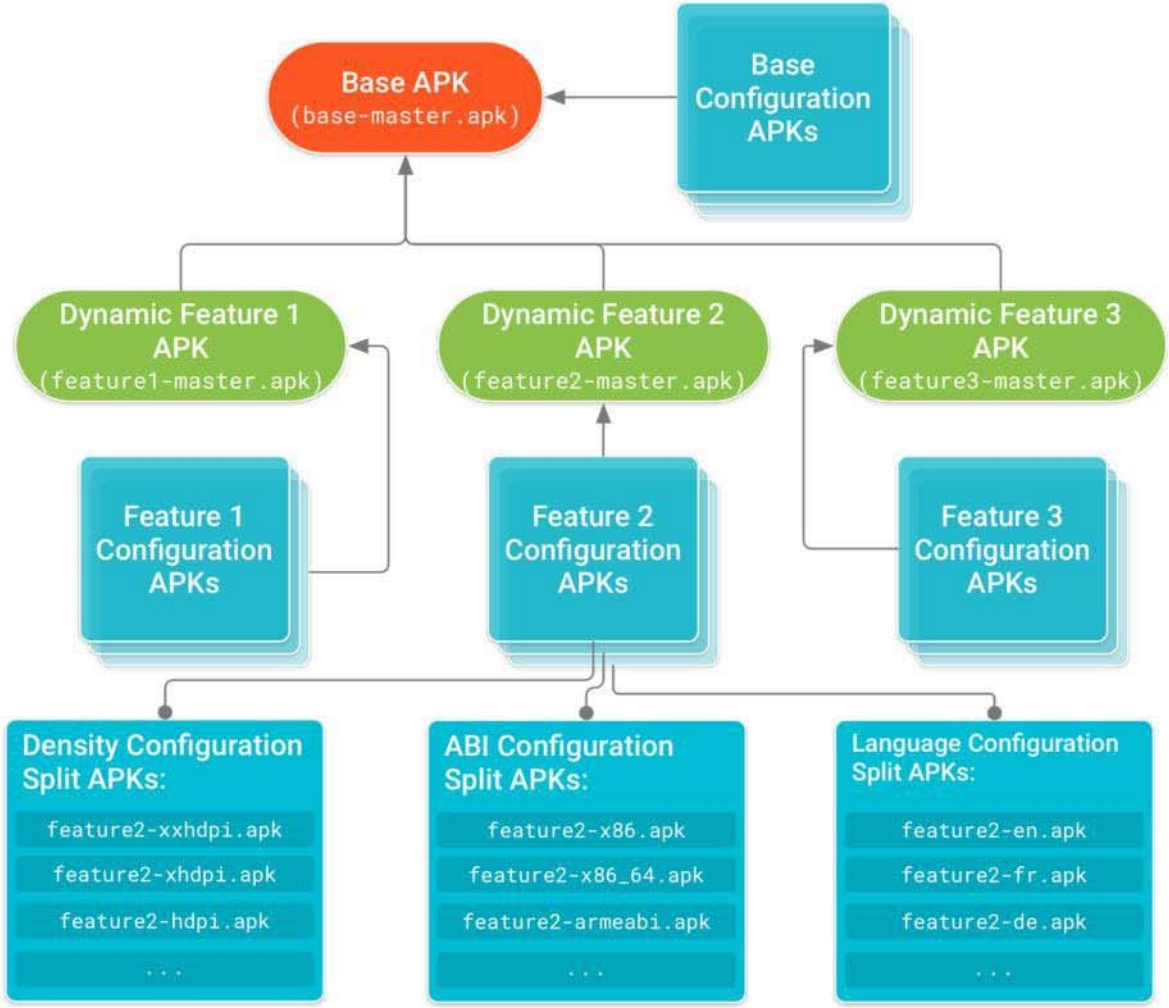


插件化与 Android App Bundle

国内-插件化

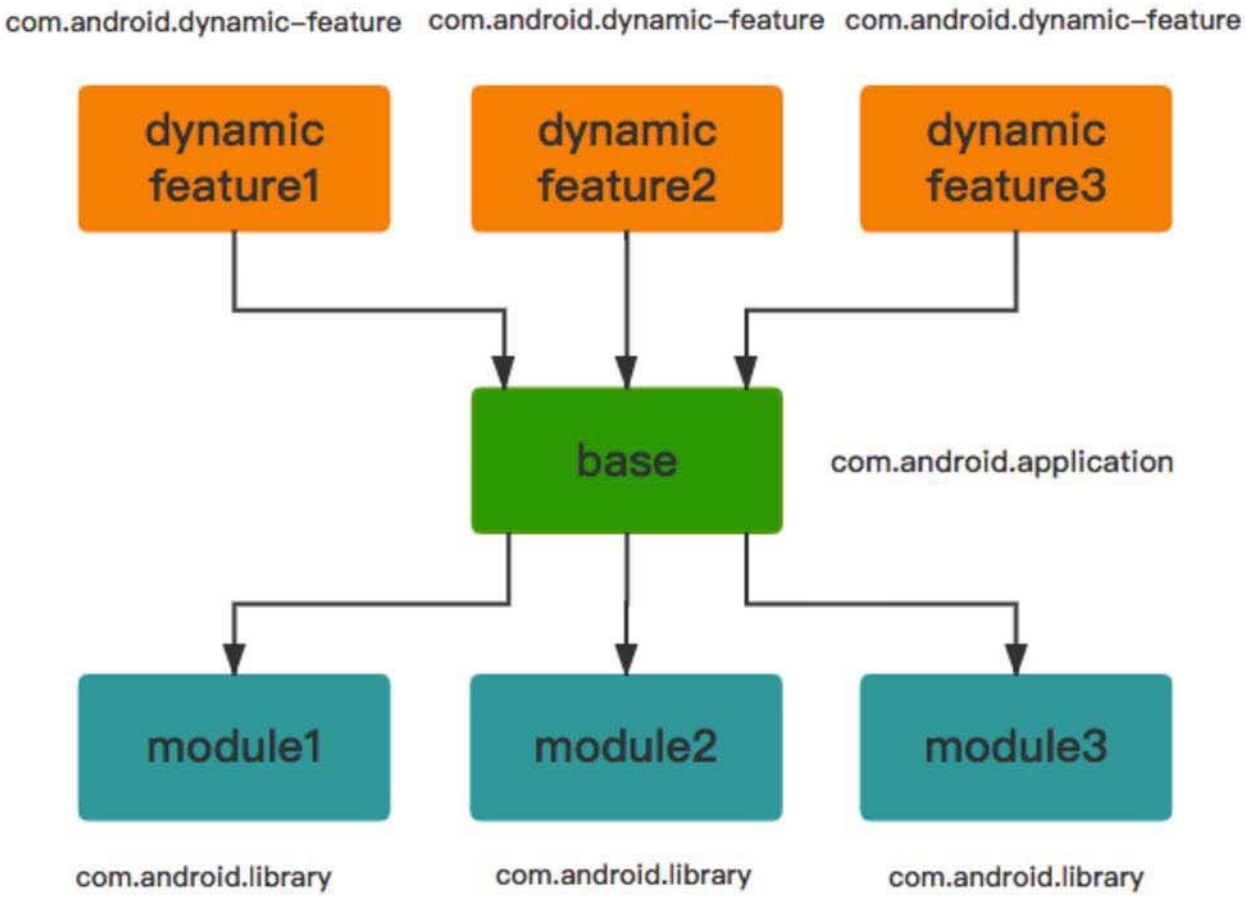

宿主
宿主配置
宿主类、共有类
宿主资源
宿主-so
...
插件
插件
插件
插件配置
插件类
插件资源
插件-so
...
插件配置
插件类
插件资源
插件-so
...
插件配置
插件类
插件资源
插件-so
...

App Bundle 是什么



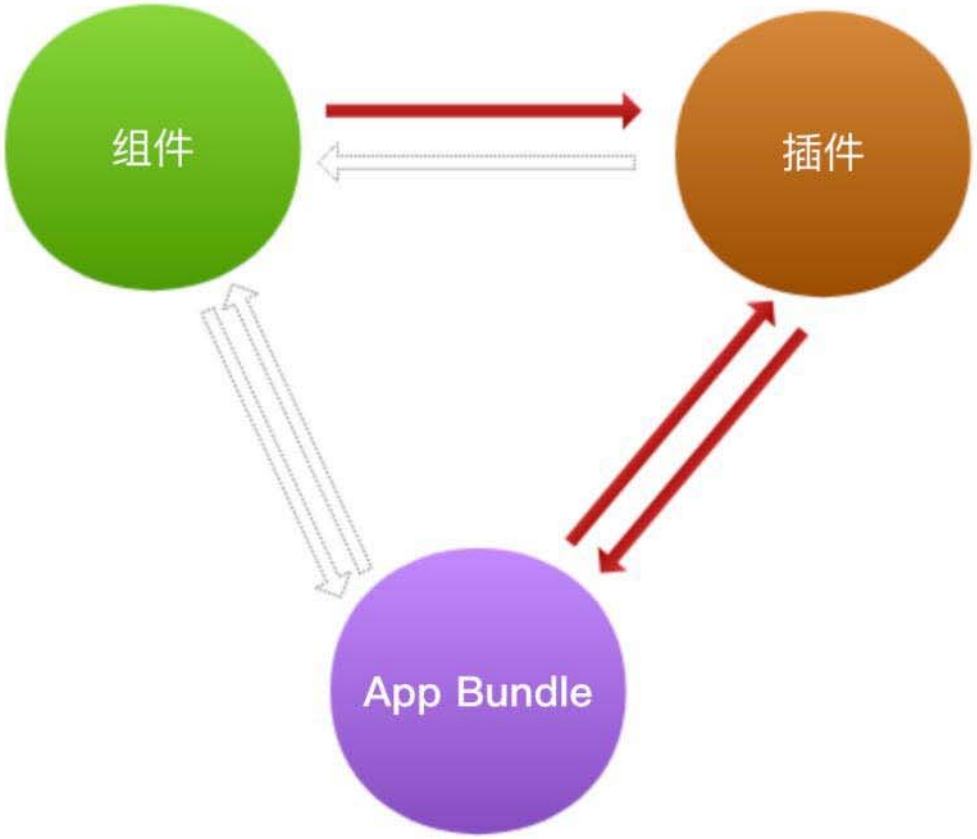
依赖关系

和组件化相反的依赖关系

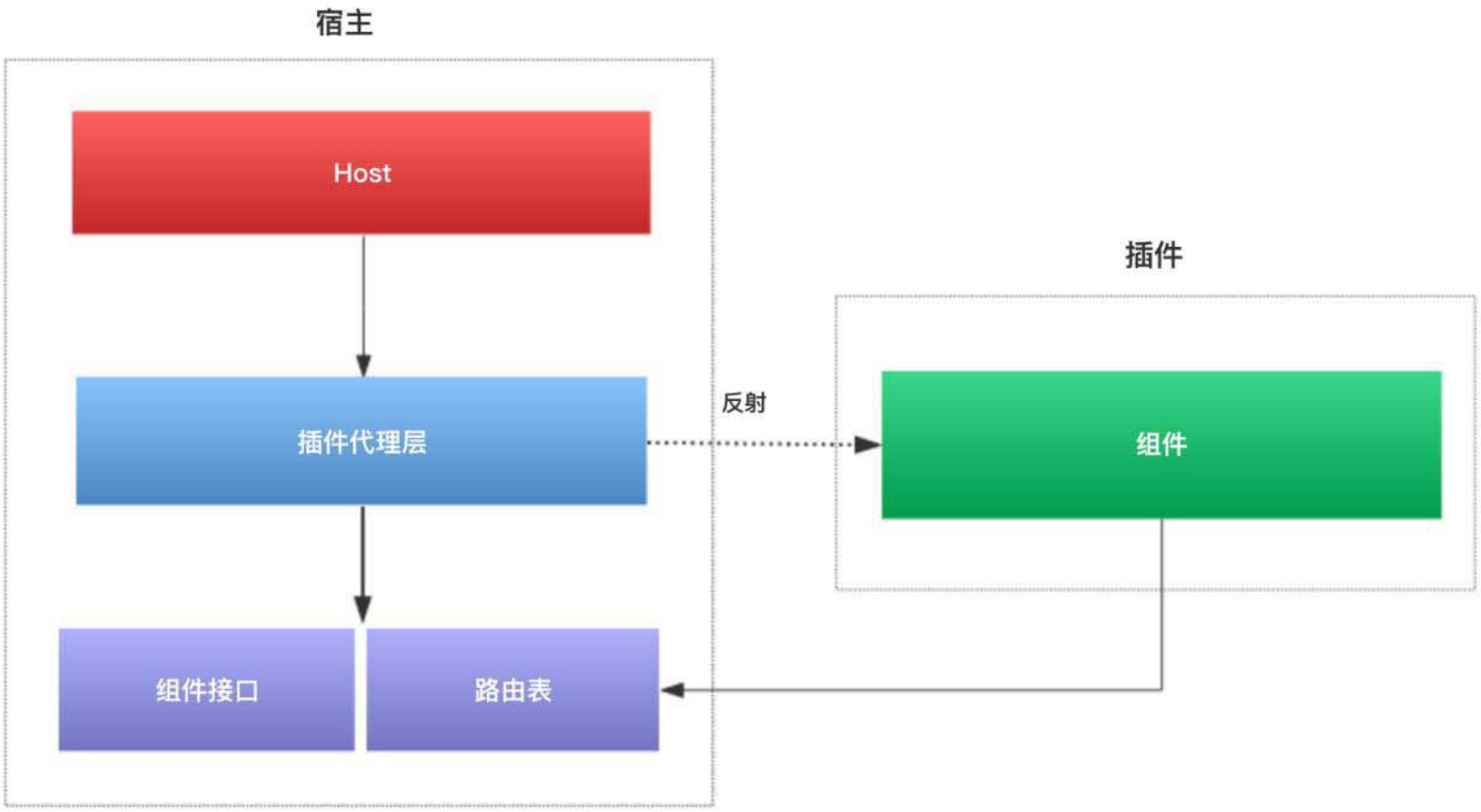


多种模式 无缝切换

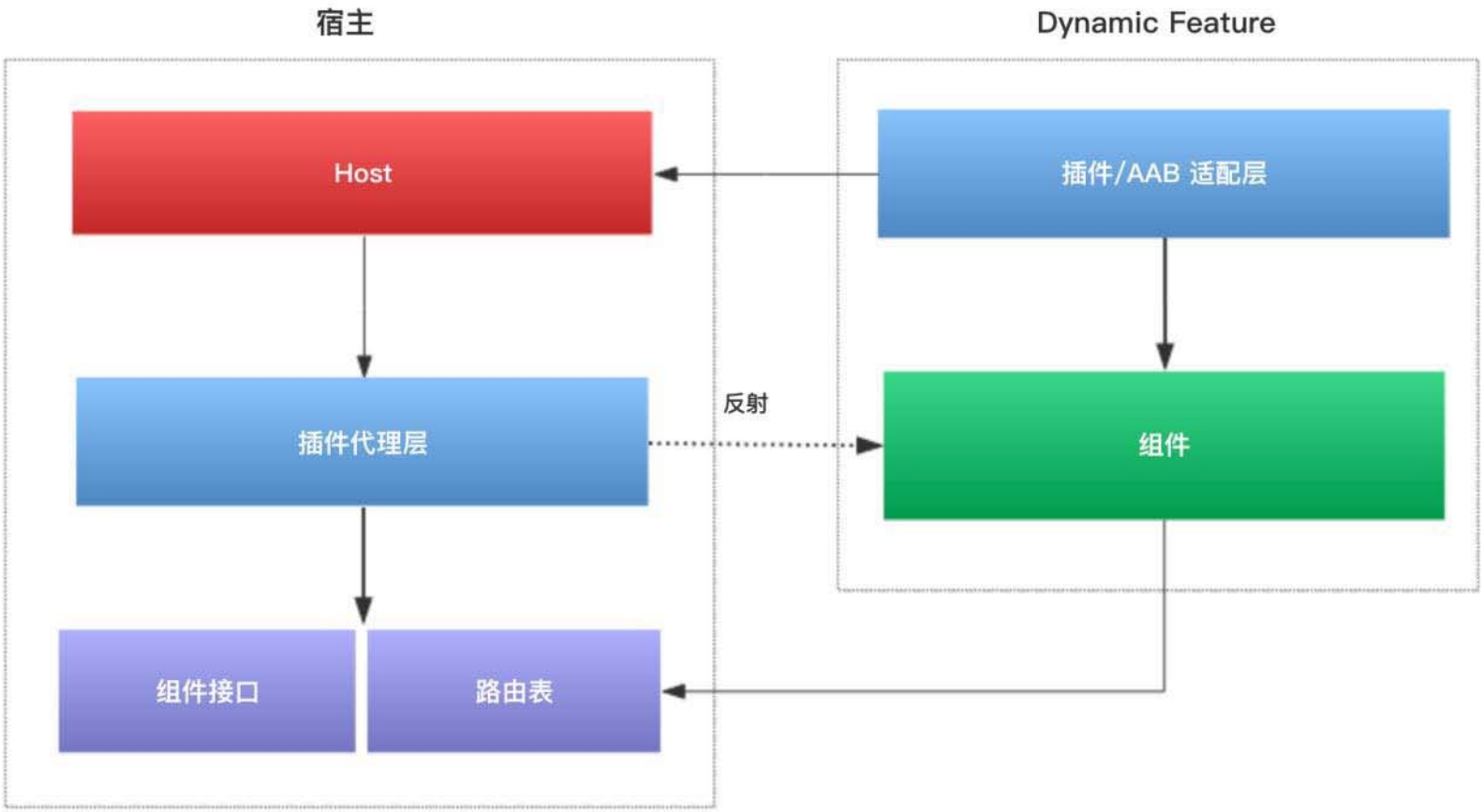
三种状态 切换



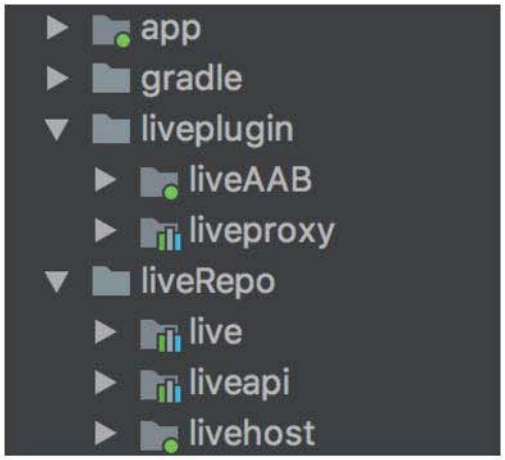
从组件到插件

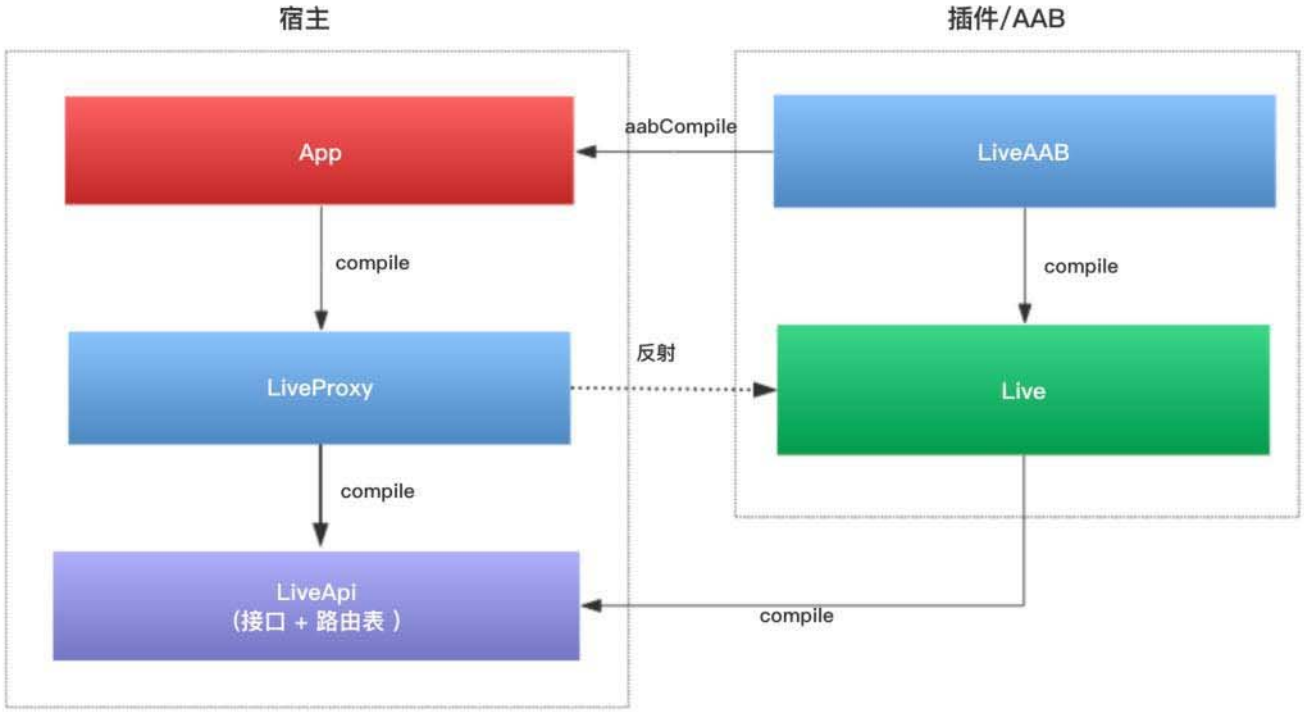


从插件到 App Bundle



举例


app
gradle
liveplugin
liveAAB
liveproxy
liveRepo
live
liveapi
livehost


宿主
插件/AAB
App
aabCompile
LiveAAB
compile
compile
LiveProxy
反射
Live
compile
LiveApi
（接口 + 路由表）
compile

统一状态接口

状态名称	插件	aab
UNKNOWN	未知状态	未知状态
PENDING	待下载	待下载
REQUIRES_USER_CONFIRMATION	无	插件超过10M需要用户确认
DOWNLOADING	下载中	下载中
DOWNLOADED	下载完成	下载完成
INSTALLING	安装中	安装中
INSTALLED	安装完成	安装完成
FAILED	失败	失败
CANCELING	无	取消中
CANCELED	无	取消完成

其他问题

资源混淆

- 修改 AndResGuard，支持 AAB格式

模拟测试

- 本地App mock GP 的交互，提供各种状态
- bundletool 本地出包测试

字节码处理

- minfyEnable 或者 multidex
- debug 下使用 buildSrc 同名替换

架构是一个生态

开发效率

- 壳工程
- 仓库管理
- 组件模板
- AS 插件

恶化预防

- 资源检查
- 代码检查
- 依赖管理
- CI/CD

总结

问题回顾

代码臃肿

- 服务于 150+ 国家，语言包、so、icon、登录、sdk
- 编译慢 10min+=

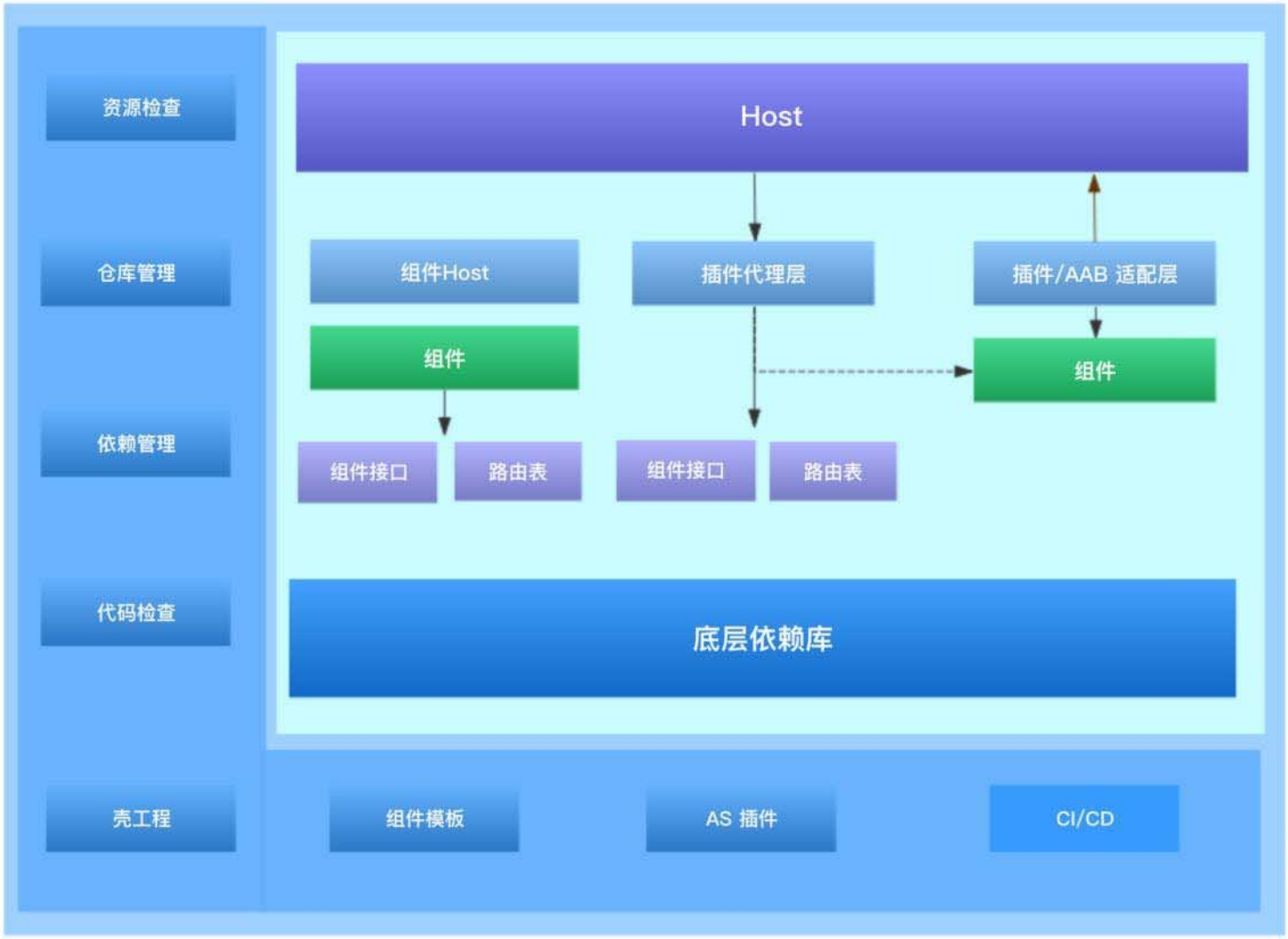
依赖混乱

- 动一发而牵全身，拖慢整个迭代进度
- 错误传递，bug 蔓延

包体积太大

- 印度、非洲、南美等国家网络差，平均 5M+
- 每下降10M，转化率提升 2.5% +

整体架构图


资源检查
仓库管理
依赖管理
代码检查
壳工程
Host
组件Host
插件代理层
插件/AAB 适配层
组件
组件
组件接口
路由表
组件接口
路由表
底层依赖库
组件模板
AS 插件
CI/CD

重点回顾

国际化的挑战 代码庞杂

模块化的原则 服务发现 **依赖注入** Dagger **路由表**

包体积 插件化 **Android App Bundle**

无感知的切换 适配层

中间状态 恶化检查 效率提升 **配套工程**

GIAC

主办方：msup | ARCHNOTES 架构

高高山顶立 深深海底行

站在山上做架构

深入海底去践行

反复的上山和下海才能最终成功

GIAC

主办方：msup | 架构 ARCHNOTES

用飞聊扫码，加我好友