ReactNative下一代引擎重构介绍

熊文源

京东中台-多端融合部　架构师

Geekbang. 极客邦科技 InfoQ

极客邦科技
会议推荐2019

自我介绍

工作经验

2016年加入京东，目前是技术中台多端融合平台架构师，Android端的技术架构负责人，有11年的Android端开发经验。

工作在京东

- JDReact Android端的平台搭建及架构设计
- JDFlutter Android端平台搭建及架构设计
- 新技术预研及培训

1

京东ARES平台简介

- ✓ 简单介绍JDReact框架及ARES平台

2

ReactNative现状分析

- ✓ 简单分享实际开发过程中的问题
- ✓ 京东是如何做性能优化的

3

ReactNative重构介绍

- ✓ 架构重构引入的新架构介绍
- ✓ 组件社区化介绍

4

如何应对此次升级

- ✓ 原生端组件和API影响
- ✓ React业务影响

京东ARES平台简介


原生端
Android
iOS
Web端
PC端
M页
类小程序端
百度小程序
快应用
微信小程序
京东ARES开放组件库（支持RN、Flutter、H5、微信小程序多端）
业务组件
统一登陆
购物车
收银台
业务跳转
用户消息
...
原生组件
地图能力
传感器
视频能力
...
界面组件
轮播图
下拉刷新
图片
对话框
进度条
...
京东ARES多端融合引擎
JDReact引擎 (ReactNative)
H5转换引擎
小程序转换引擎
JDFlutter引擎
京东ARES - 支撑系统
APM监控
数据监控
容灾降级
线上发布
京东ARES – 管理平台
接入申请
权限管理
发布审批
开发者控制台

ReactNative问题分析

1

启动性能

- ✓ 原生API/组件
- ✓ 业务包大小
- ✓ API需遍历调用

2

渲染性能

- ✓ 异步的UI渲染
- ✓ JSBridge限制
- ✓ 组件复用回收
- ✓ 动画渲染性能

3

手势

- ✓ 系统横向、竖向手势组件事件冲突
- ✓ Guest滑动不流畅

4

API统一性

- ✓ Andorid、iOS组件、API参数和调用方式不一致
- ✓ WEB端API无法对齐

5

组件问题

- ✓ 组件问题修复较慢
- ✓ 新组件更新慢

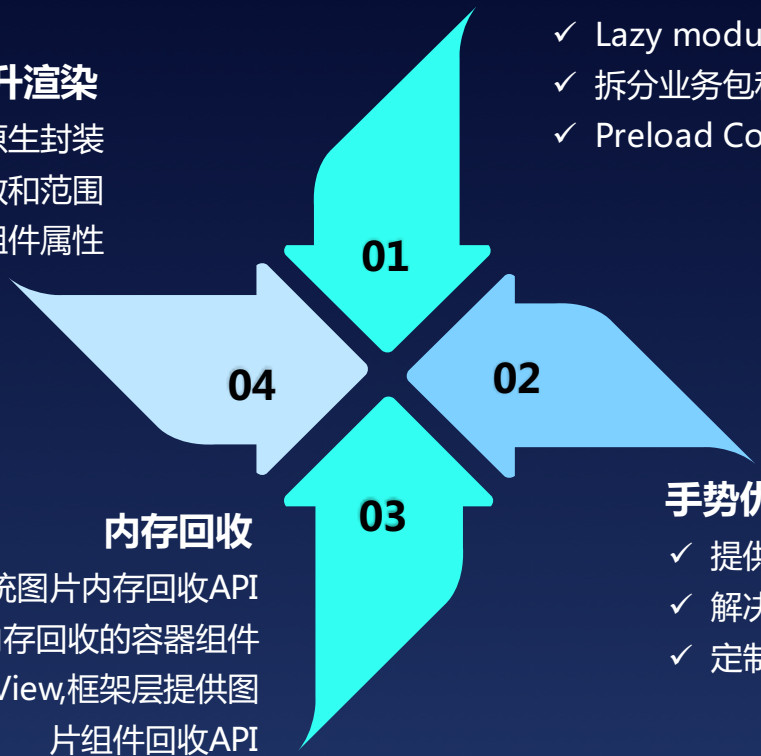
京东是如何做性能优化的

加载性能优化

- ✓ Lazy module、Lazy Require改造
- ✓ 拆分业务包和框架Common包
- ✓ Preload Common或者业务包

提升渲染

- ✓ 对于交互频繁的组件，采用原生封装
- ✓ 业务层减少render次数和范围
- ✓ 减少不必要的组件属性



手势优化

- ✓ 提供View层手势拦截API
- ✓ 解决多个ScrollView嵌套手势问题
- ✓ 定制手势处理组件

内存回收

- ✓ 定制系统图片内存回收API
- ✓ 新增内存回收的容器组件
- ✓ 优化FlatList、ScrollView,框架层提供图片组件回收API

ReactNative问题分析

目前解决方案的局限性：

- ✓ 依赖原生端定制的组件和API越来越多，导致原生端原来越重，维护成本升高(三个平台)
- ✓ 即使采用FlatList等回收功能组件，仍然存在渲染问题
- ✓ 手势冲突的解决依赖特殊的业务场景，无法通用
- ✓ 低端设备上还是无法达到原生体验，特别是Android平台
- ✓ Android平台兼容性问题太多

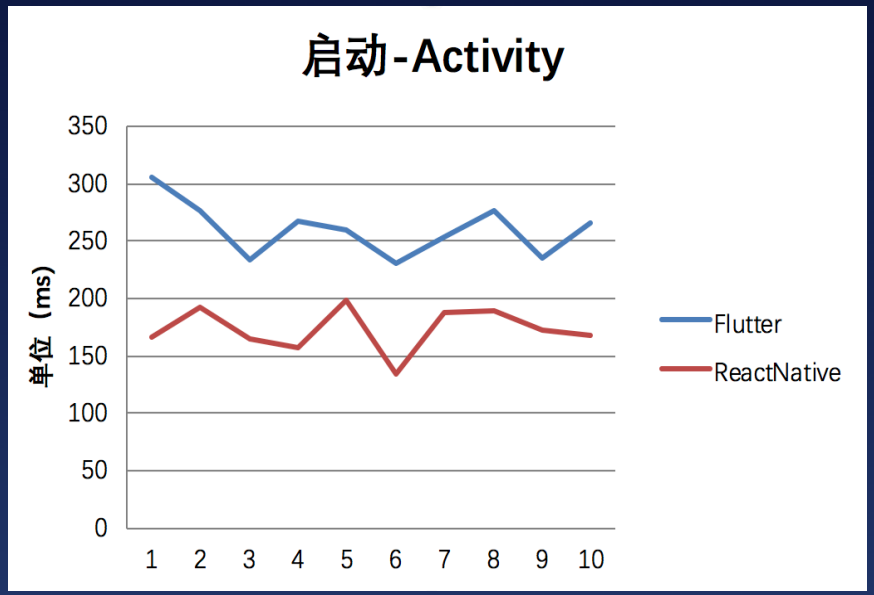
ReactNative问题分析

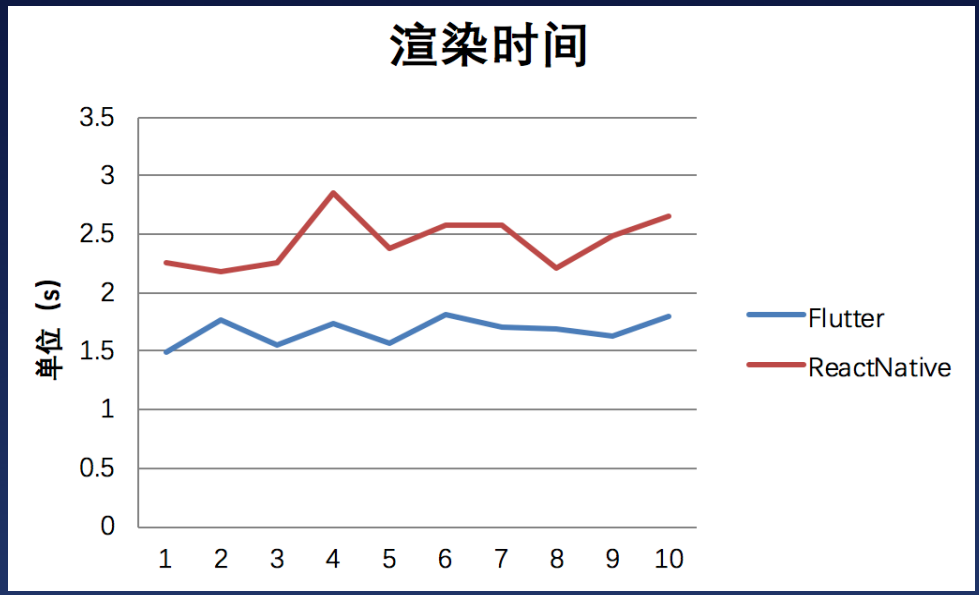
技术领域分析：

- ✓ 要做好ReactNative开发，需要对Android、iOS系统有所了解，比如优化、内存、调试
- ✓ ReactNative还没有完全实现对两个平台跨平台抽象，许多功能都要多个平台混合开发
- ✓ 大的原生端和ReactNative升级，需要做大量的适配

性能对比

在启动性能上React Native稍微优于Flutter，但渲染方面明显不如Flutter，这就是整体框架的瓶颈，无法从业务层级做太大的改进。



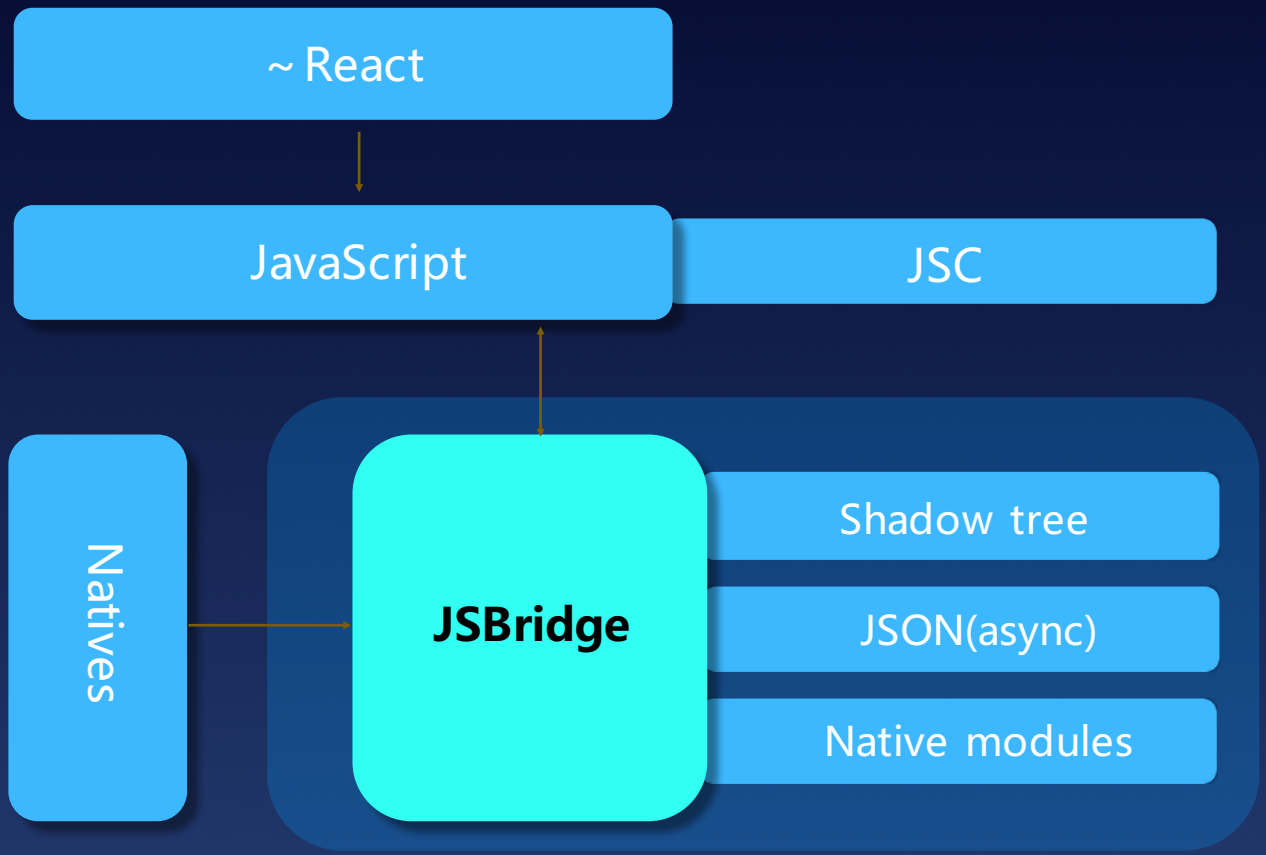


为什么还要选择ReactNative

- ✓ 大部分前端人员更适应JavaScript和web开发
- ✓ 工程化非常简单，跨端方案也提高了开发和版本迭代效率
- ✓ 社区大量开源组件和API的支持
- ✓ 强大的热更新能力，让上线和问题修复更及时，同时有助于客户端瘦身
- ✓ 官方持续更新的版本，不断提升了体验和功能，同时也意识到现有架构的瓶颈

ReactNative历史版本

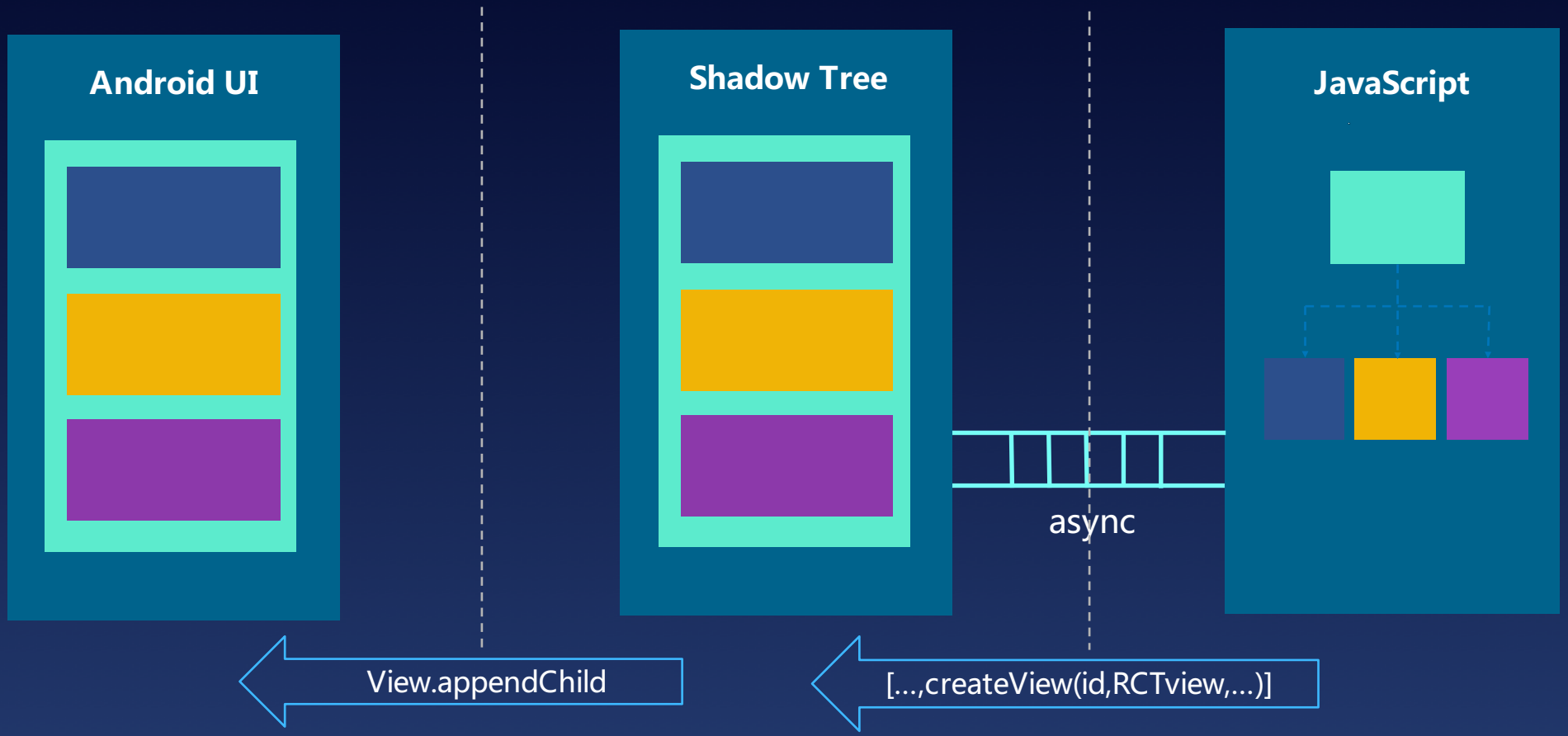
ReactNative瓶颈



现有的问题：

- ✓ 业务启动依赖JSBridge加载
- ✓ 所有调用为异步操作
- ✓ 所有JS和原生端交互采用的JSON数据
- ✓ JSBridge的消息通道容易阻塞

ReactNative瓶颈

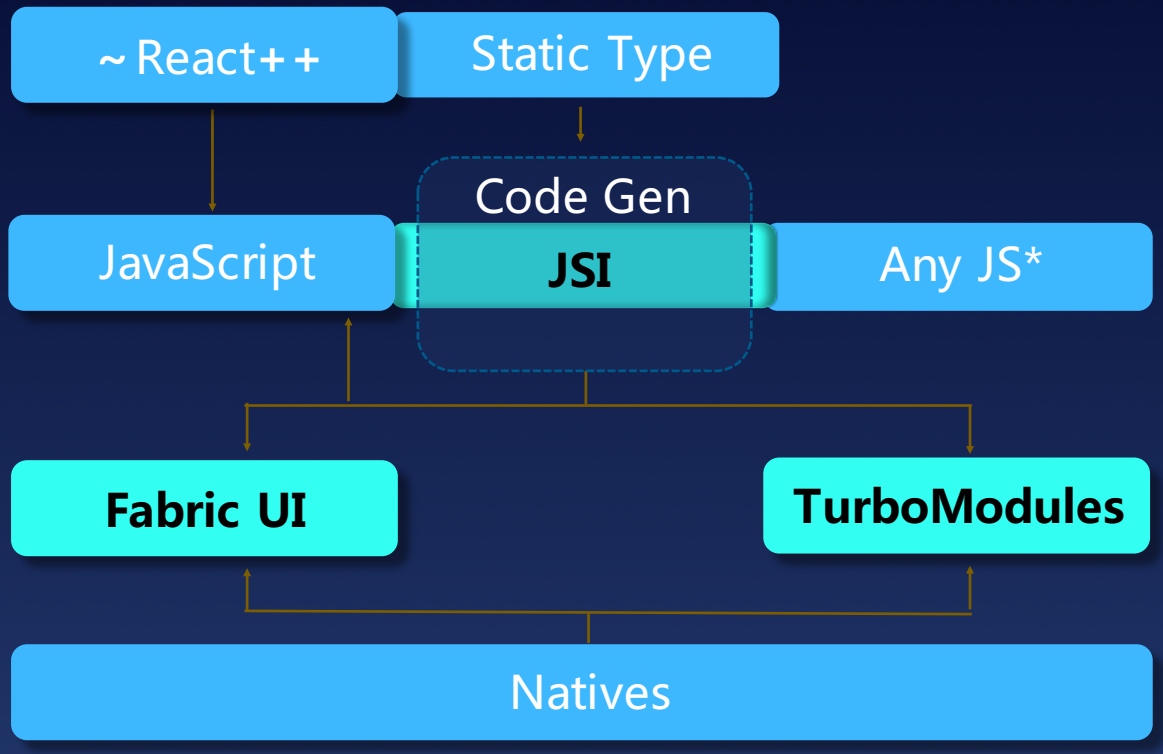


ReactNative重构计划

Facebook在2018年6月官方宣布了大规模重构React Native的计划及重构路线图。目的是为了让React Native更加轻量化、更适应混合开发，接近甚至达到原生的体验。

- ✓ 改变线程模式。UI 更新不再同时需要在三个不同的线程上触发执行，而是可以在任意线程上同步调用 JavaScript 进行优先更新，同时将低优先级工作推出主线程，以便保持对 UI 的响应。
- ✓ 引入异步渲染能力。允许多个渲染并简化异步数据处理。
- ✓ 简化JSBridge，让它更快、更轻量。
- ✓ 社区化、轻量化ReactNative引擎

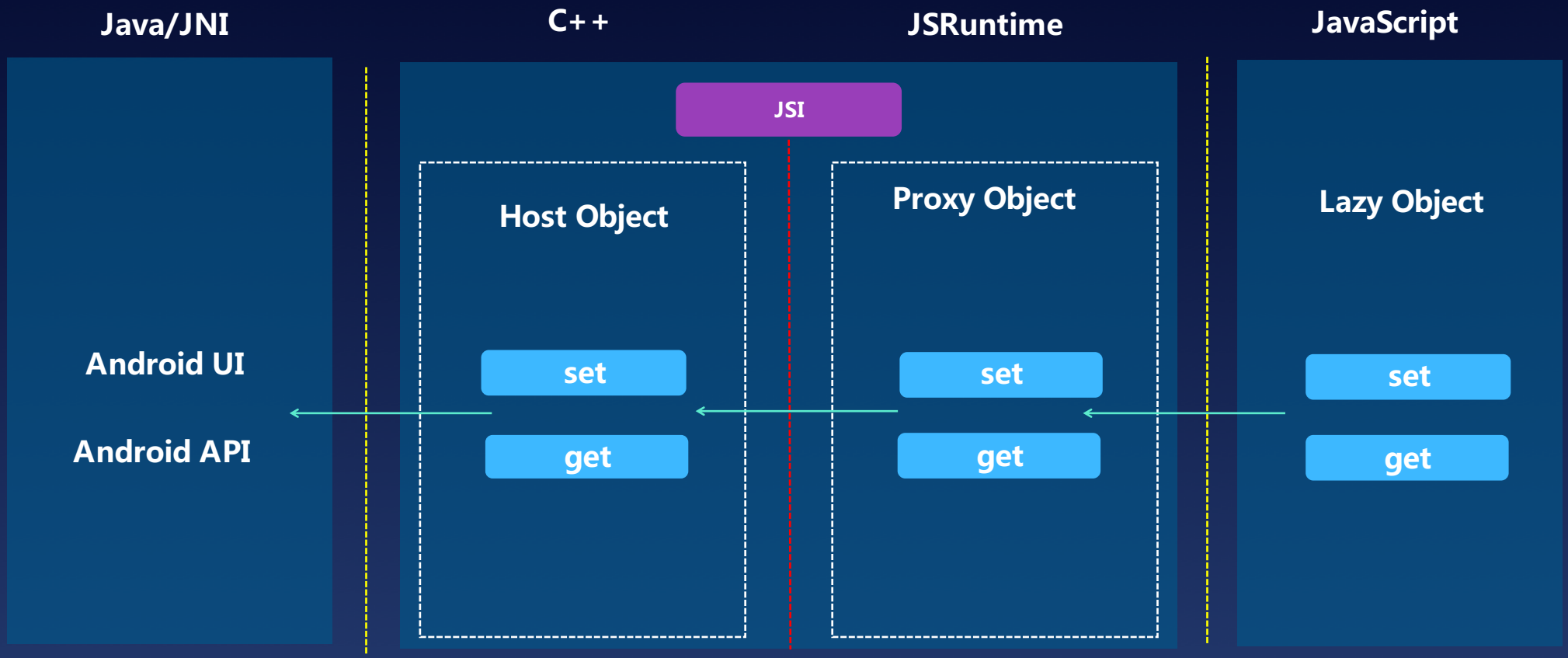
ReactNative重构



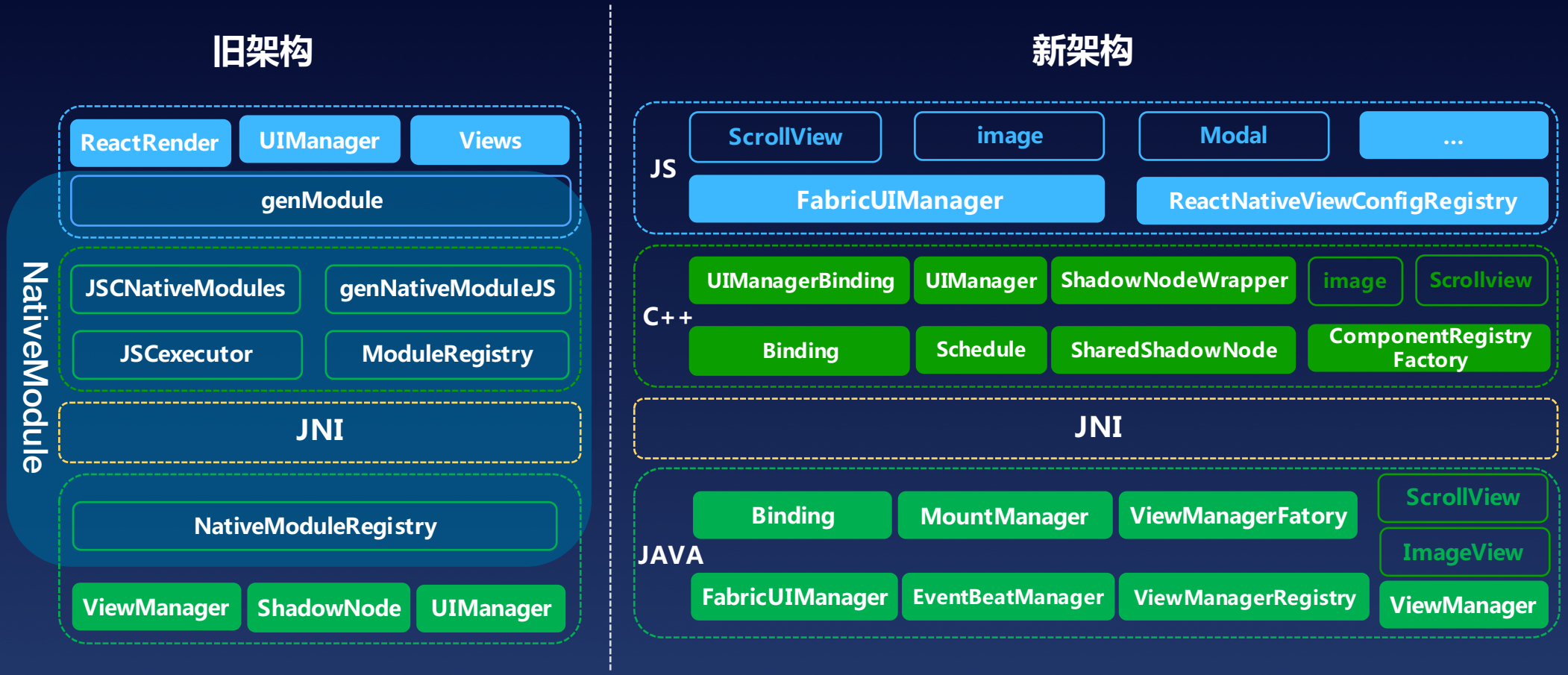
新架构的变化：

- ✓ **Fabric**：UI渲染框架，替换了原有的UIManager
- ✓ **TurboModule**：API框架，用于JS到原生端的API调用，替换了原有的API机制
- ✓ **JSI**：JS引擎上抽象的一层通用层，通过他可以灵活的适配替换JS引擎，同时能支持在JS中对C++host的引用，直接调用API
- ✓ **Code Gen**：工具，用于生成Fabric和TurboModule 的JSI的框架代码，另外也加入JS静态代码检查功能

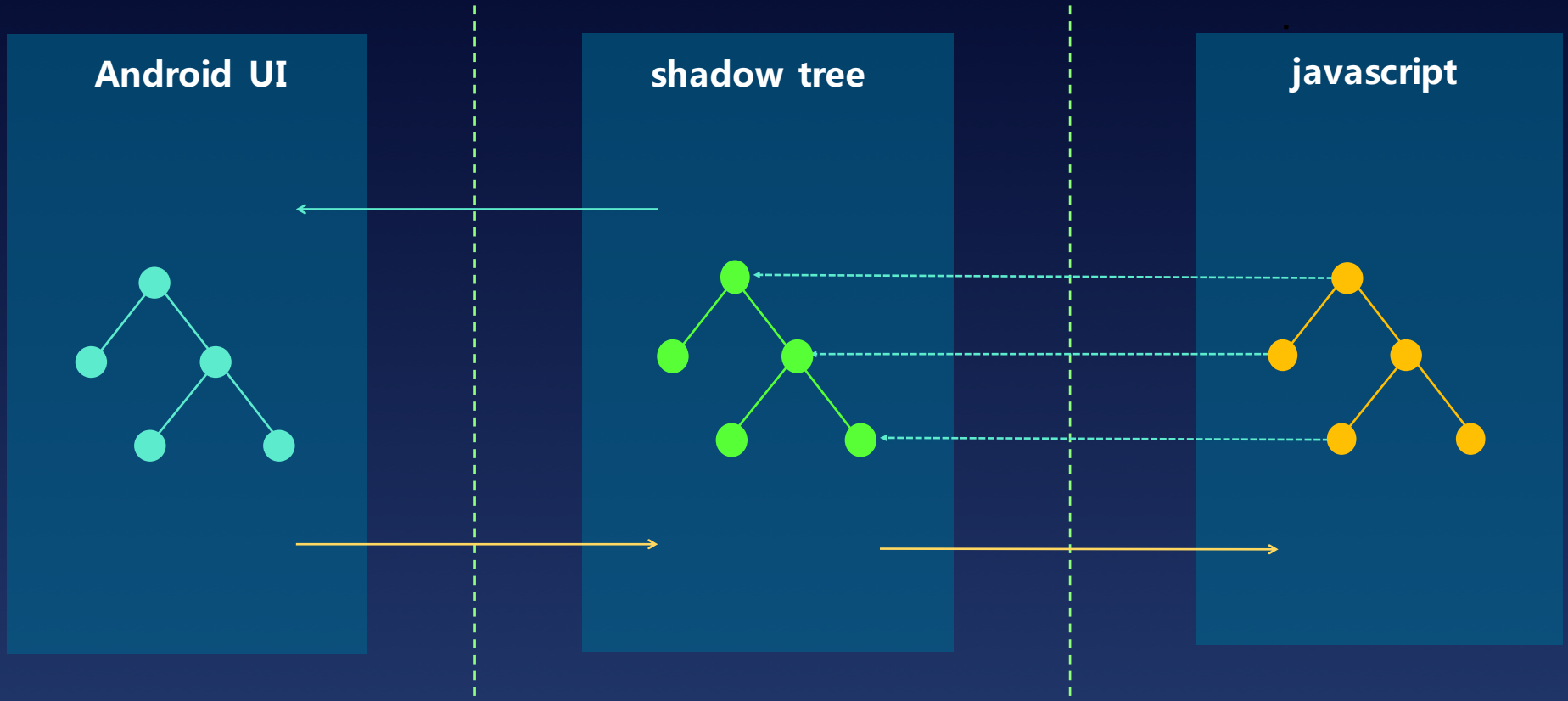
ReactNative重构-JSI



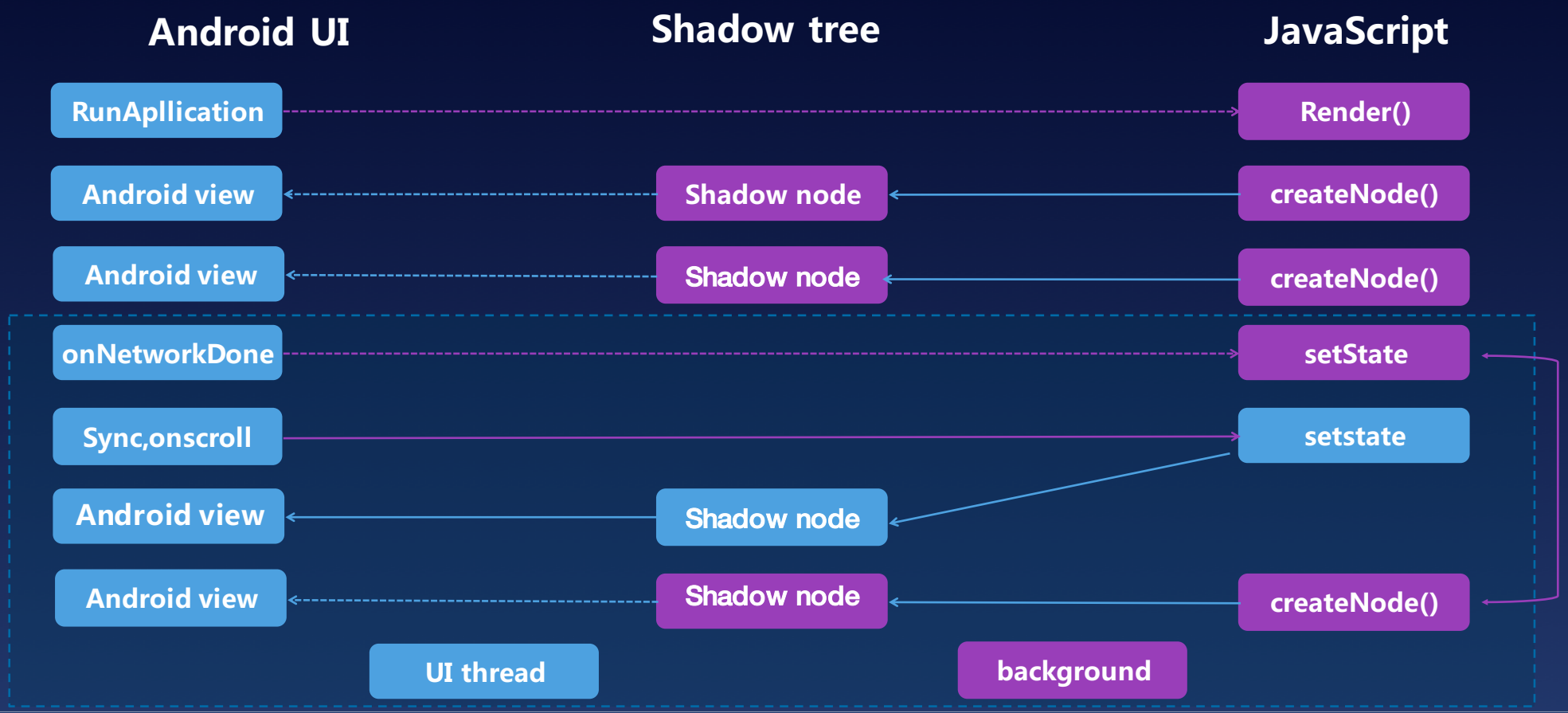
ReactNative重构-Fabric



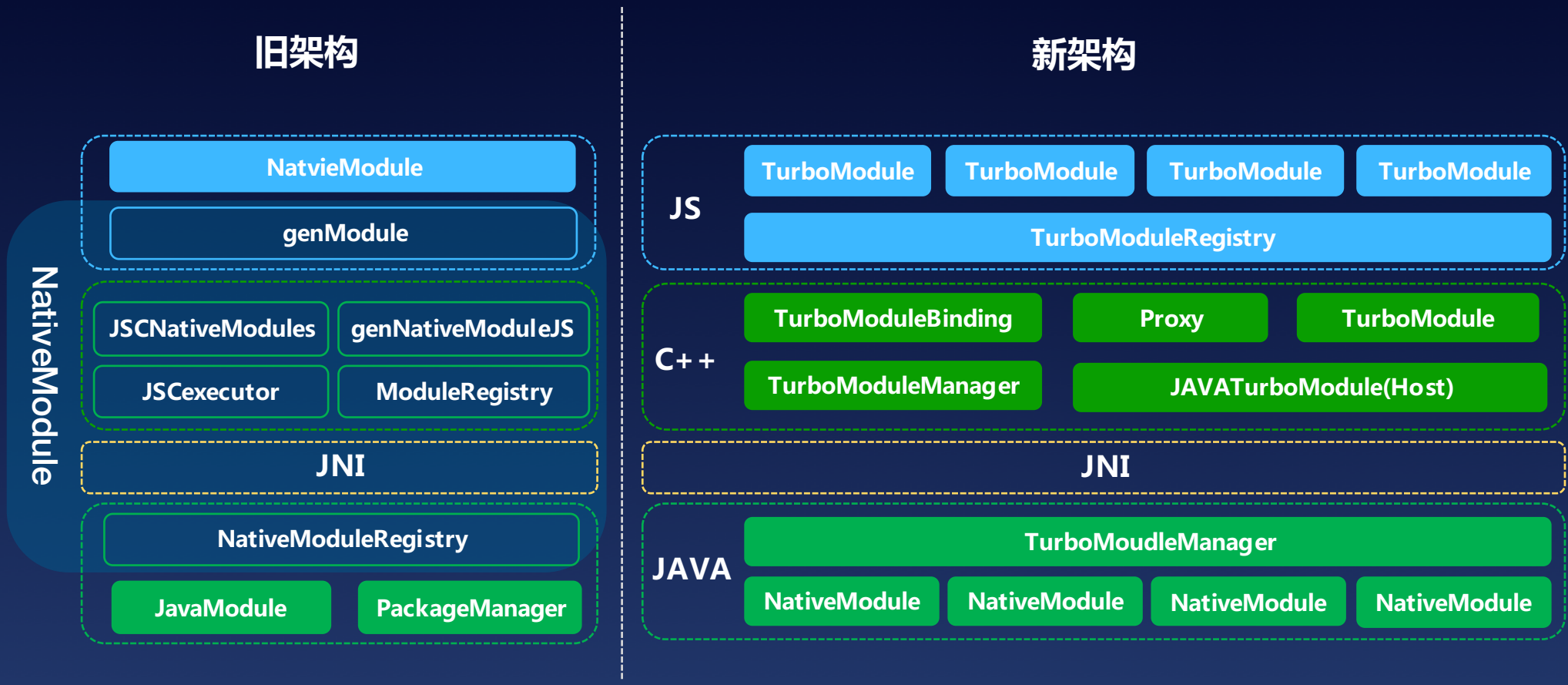
ReactNative重构-Fabric



ReactNative Fabric渲染流程



ReactNative重构-TurboModule

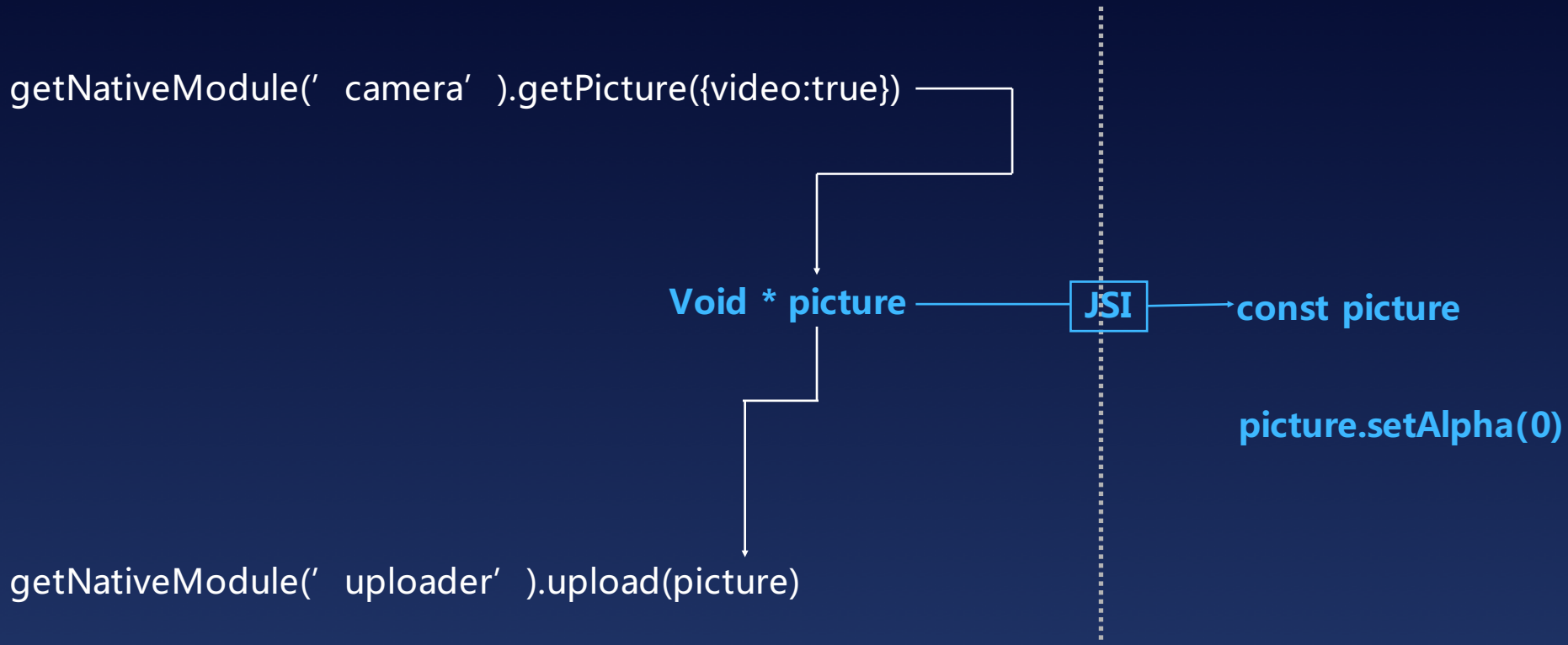


ReactNative重构- TurboModule

```
getNativeModule(‘ camera’ ).getPicture({video:true})


//C++
Camera:HostObject{
   Value get(propName){
      switch(propName){
         Case ’ getPictures’ :
         //Call Android/iOS API
         return JavaCameraManager.tackPic();
      }
   }
}
```

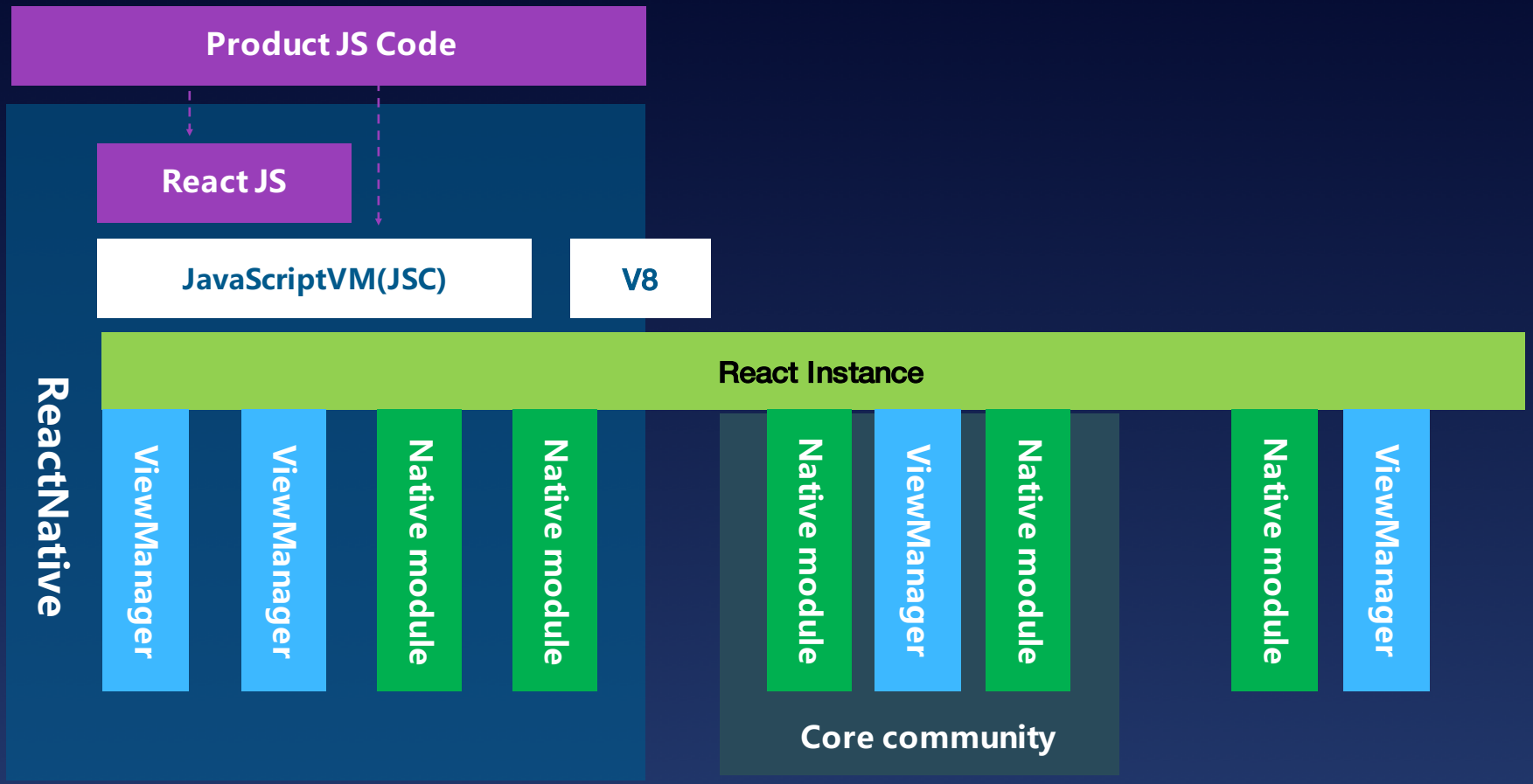
ReactNative重构- TurboModule



ReactNative重构-总结

- ✓ 增加了JSI object数据结构类型
- ✓ 相比原有的架构，增加C++层的组件和API封装
- ✓ UI组件shadow tree从JAVA层移到了C++层
- ✓ 不再依赖JSBridge

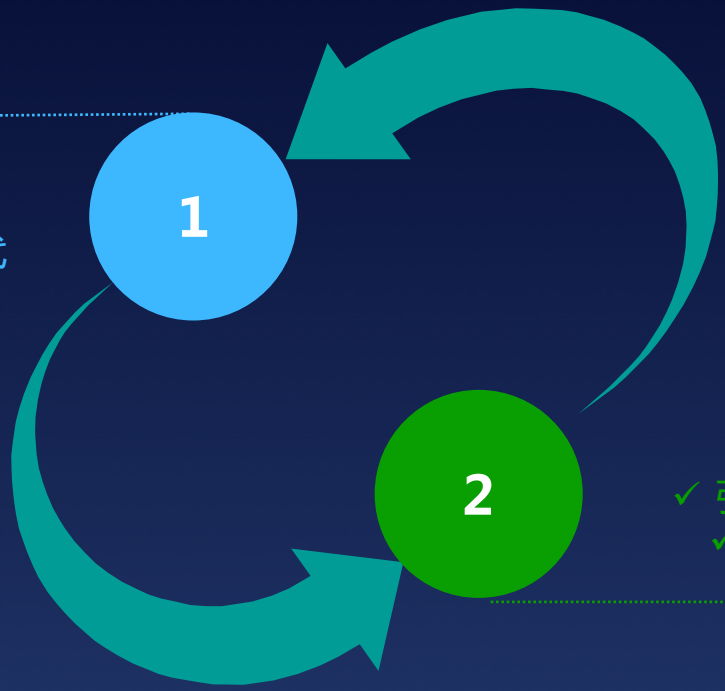
ReactNative组件社区化



ReactNative组件社区化好处

版本迭代变快

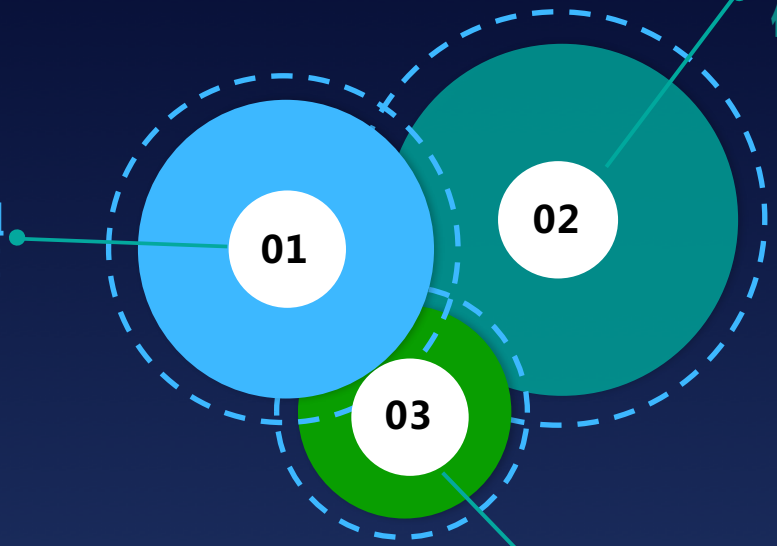
- ✓ Facebook只用维护少量的核心组件及引擎
- ✓ 社区维护提高了组件迭代速度，避免了重复开发



- ✓ 引擎与大量非核心组件解耦
- ✓ 删除了大量不需要的模块

轻量化

架构重构对开发者的影响



自定义组件、API

- ✓ 原生端UI组件封装方式变化
- ✓ 保留了原有的UI框架
- ✓ 废弃了LazyPackage

系统组件、API

- ✓ 原有的系统组件API不变
- ✓ 社区化后，需到社区同步组件
- ✓ 增加了组件集成和版本控制难度
- ✓ 组件和API的安全性影响

业务开发者

- ✓ 业务大部分业务代码不变
- ✓ 使用原生组件或API调用方式变化
- ✓ 增加同步和异步渲染

如何应对

- ✓ 新的架构在github master分支，了解新旧架构的区别，特别是组件和API。
- ✓ 新架构对C++、JNI的编码能力要求较高，需熟悉相关语言。
- ✓ 新架构的基础是JSI架构，需了解其工作原理
- ✓ 新框架仍保留了旧架构的功能，可以考虑分步骤完成架构改造。
- ✓ 组件社区化后，需了解组件的依赖和集成方式。
- ✓ 对于旧组件和API改造， Facebook提供code gen工具， 只需开发者做代码整合。

THANKS

GMTC
全球大前端技术大会