WEEX Conf

Weex 技术演进

隐风(Adam Feng)
饮源(Steve Zheng)

2017 Weex 的业务规模

	2016	2017
会场	99%主分会场	100%主分会场+100%店铺承接页
导购业务	少量	100%(有好货、每日好店、淘金币、人群导购等)
核心链路	NAN	支付完成、店铺等
基础业务	NAN	我的淘宝、足迹、收藏夹等
集团内App	<10个	30+（手机淘宝、手机天猫、飞猪、UC浏览器、千牛、钉钉、虾米音乐、优酷、闲鱼、阿里云、Lazada、国际站买家版、淘票票专业版、裹裹侠、一淘、开语、阿里健康、阿里数据、阿里卖家、淘必中等）
集团外App	<10个	不完全统计：极客时间、企鹅电竞、全民竞猜、盛大、Paytm、贝贝、点我达、陌陌、分期乐、饿了么、富途证券、今日头条、众安保险、尚妆网、蛙趣等

2017 Weex 的技术挑战

业务复杂度的不断攀升带来技术上的挑战

交互体验升级

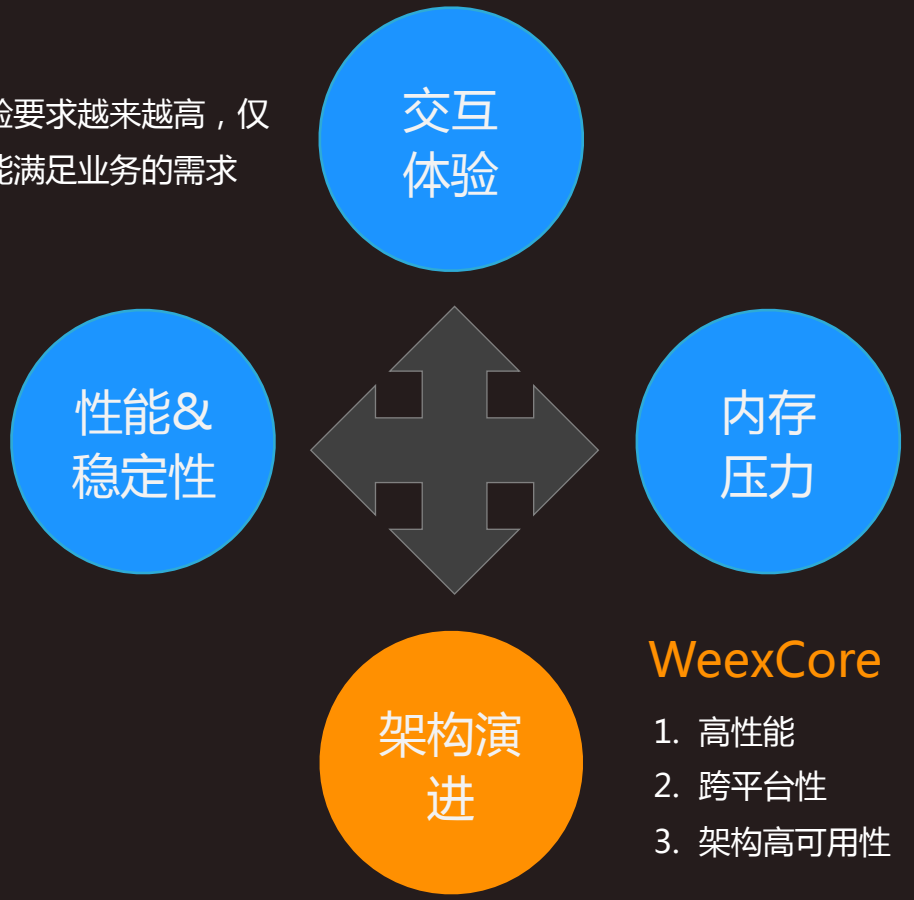
业务对动画、手势等交互体验要求越来越高，仅仅提供页面展示能力远远不能满足业务的需求

内核引擎升级

JS引擎升级 、 JSC独立进程、瘦身

Sandbox沙箱机制等

全新的Layout engine



Weex RecycleList

全链路Weex + H5后内存压力陡增，很容易导致 App crash 率升高。

WeexCore

1. 高性能
2. 跨平台性
3. 架构高可用性

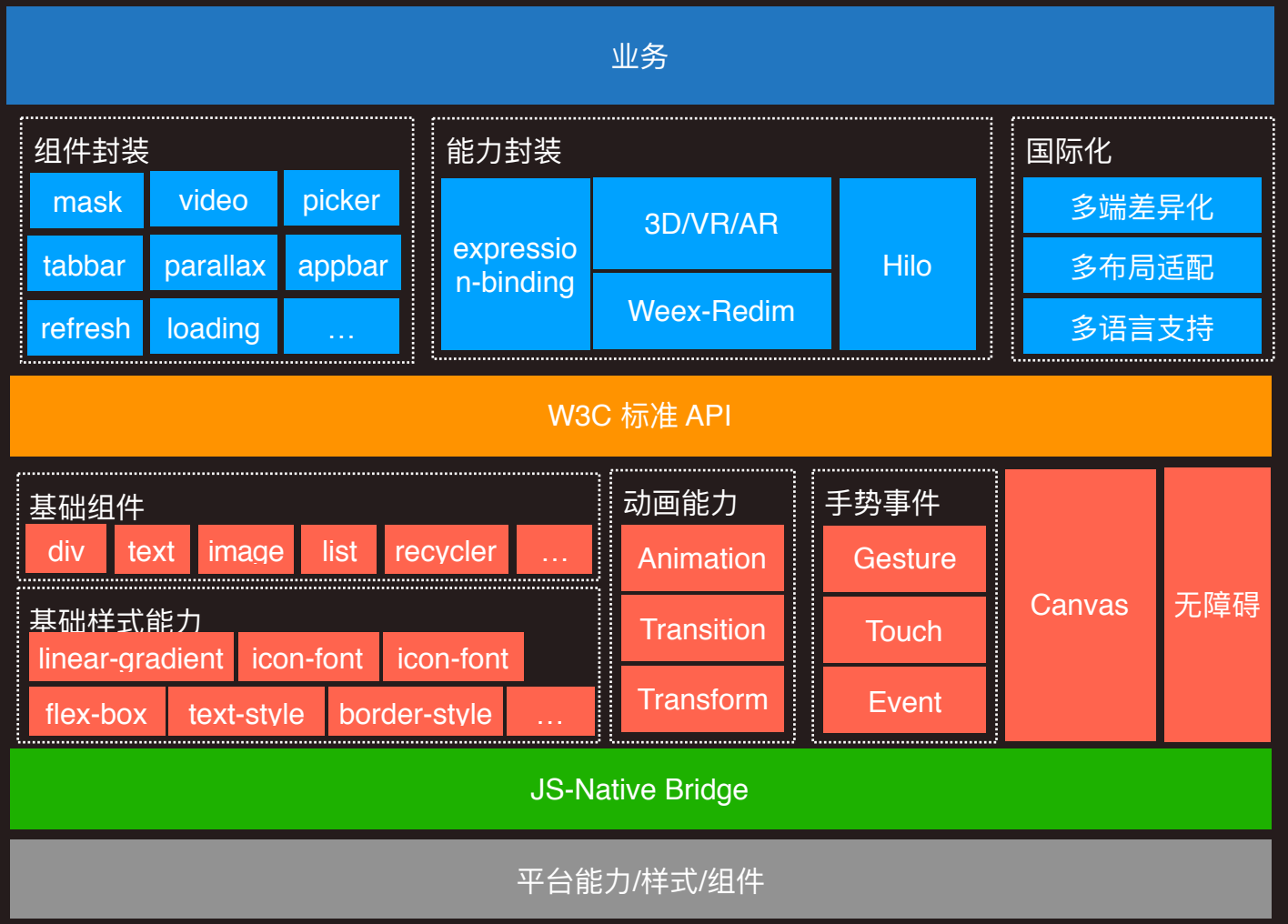
交互体验升级

Weex 视觉交互演进

从具象到抽象的不断探索

Weex 视觉交互演进

- **指导原则**
  - **Weex 最大的优势和价值在于其 native 的能力， 我们要最大化利用 native 的特性， 而不是去模仿它**
- **扩展原则**
  - 从下至上
  - JS封装优先
  - 允许多端差异化
- **核心抓手**
  - 视觉/交互
  - 虚拟/互动
  - 国际化

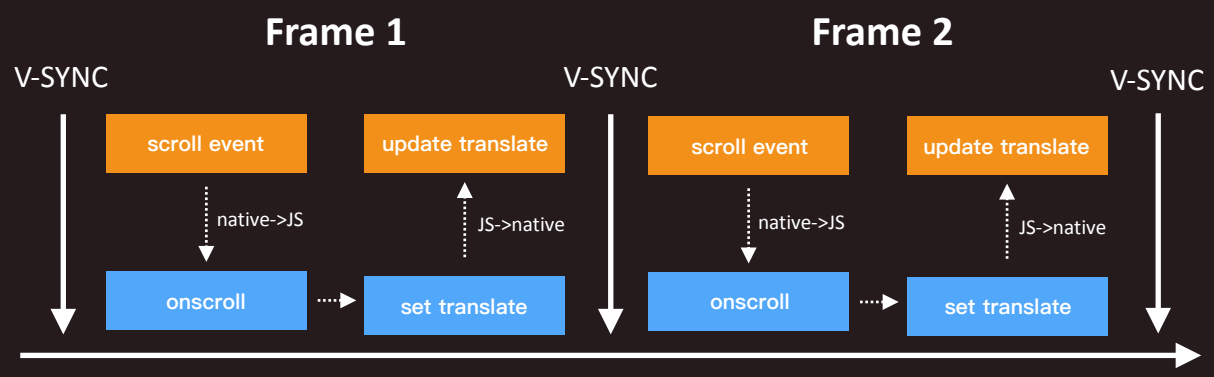


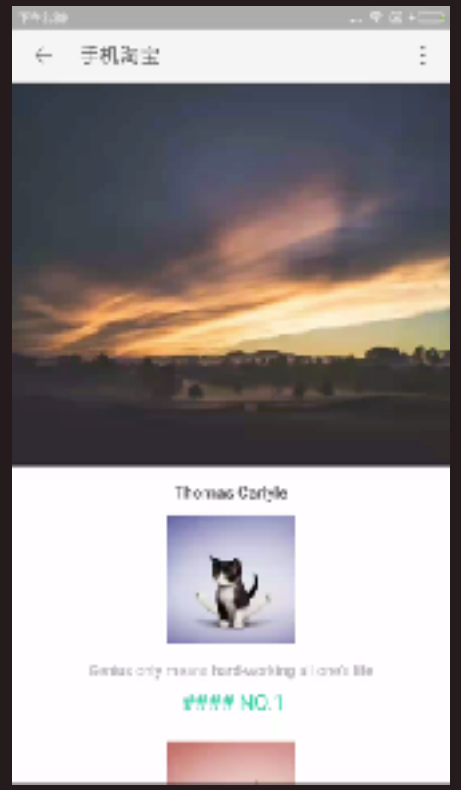
什么是好的交互体验？

- 加载速度很快，秒开
- 滚动流畅，60FPS
- 持续又快速地响应

最大的挑战

- JS 和 Native 的通信时延
  - 减少通信次数
  - 优化单次通信耗时

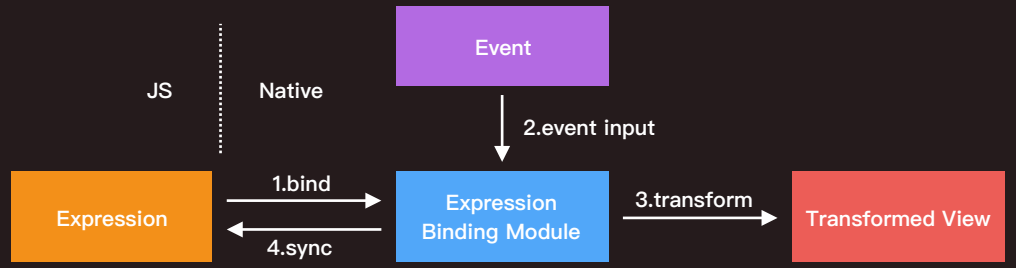




WEEX

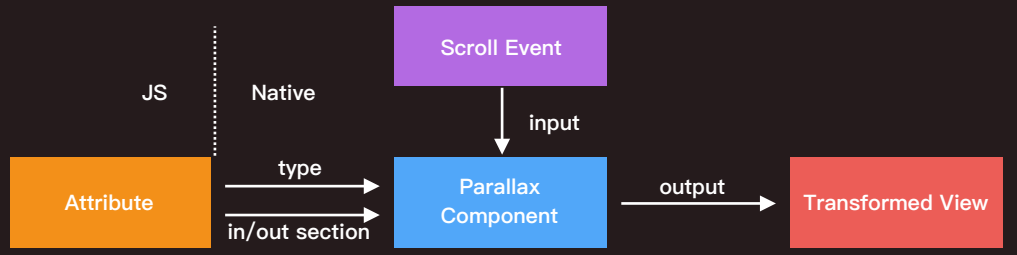
减少通信次数

Expression binding

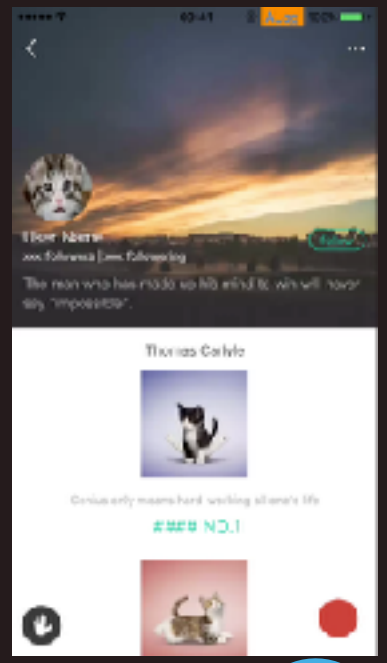



```
var binding = weex.requireModule('binding')
var result = binding.bind({
    eventType:'pan', // pan | scroll | timing | orientation
    exitExpression: {
        transformed: "",
        origin: ""
    },
    props: [{
        element:foo,
        property:'transform.translateX',
        expression:{
            transformed:expression_x_transformed,
            origin:expression_x_origin
        }},
        ...]
  },function(e){
})
```

Parallax

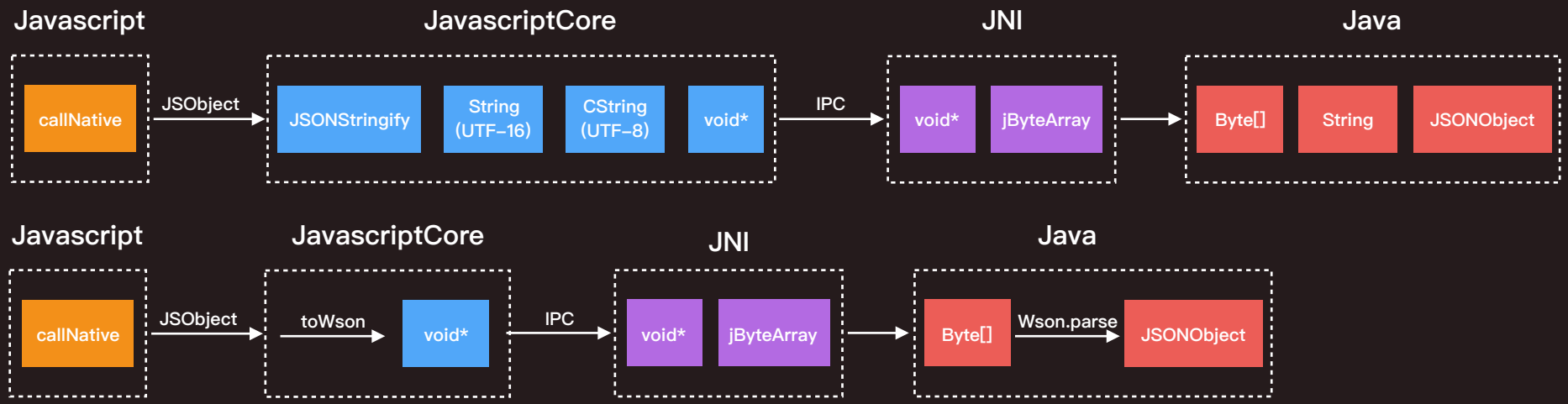



```
<parallax class="header"
  :transform="[{
    type: 'translate',
    in:[0,500],
    out:[0,0,0,-250] // [x1,y1,x2,y2]
  },{
    type: 'scale',
    in:[-150,0],
    out:[1.3,1.3,1,1]  // [x1,y1,x2,y2]
  }]">
  <image ref="source" class="image" src="...."></image>
</parallax>
<list class="list" ref="list">
  <cell></cell>
  ...
</list>
```

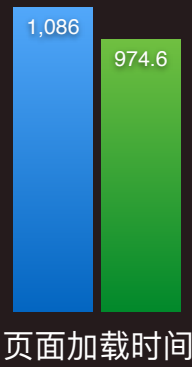


WEEX

优化单次通信耗时



type	signature	format
number int	i'	signature + varint
number double	d'	signature + 8 byte (big endian)
number float	F'	signature + 4 byte (big endian)
string	s'	signature + var length + bytes(unicoder utf-16)
null	0'	signature
boolean	t' or 'f'	signature
array	['	signature + var length + elements
map	{'	signature + var size + key, value, key, value



WEEX

列表渲染架构升级

Weex 列表组件的演进

不断攀登性能高峰

2015.11

Scroller

优势：

- 完成从JS DOM树到View树的转换

挑战：

- 需要等待 JS 解析完所有节点
- 一次性创建过多View，阻塞 UI 线程

2016.5

Scroller+渐进式渲染

优势：

- node 和 tree 模式的灵活搭配
- 最小粒度渲染
- 解析完单个DOM后可以立即显示

挑战：

- 渲染了不可见区域
- 内存占用极高

2016.11

List

优势：

- 使用UITableView/RecyclerView
- 只渲染可见区域
- View 内存回收复用

挑战：

- 无法做 View 结构复用
- 内存占用和 native 列表还是有不小的差距
- 低端机滚动帧率差

2017

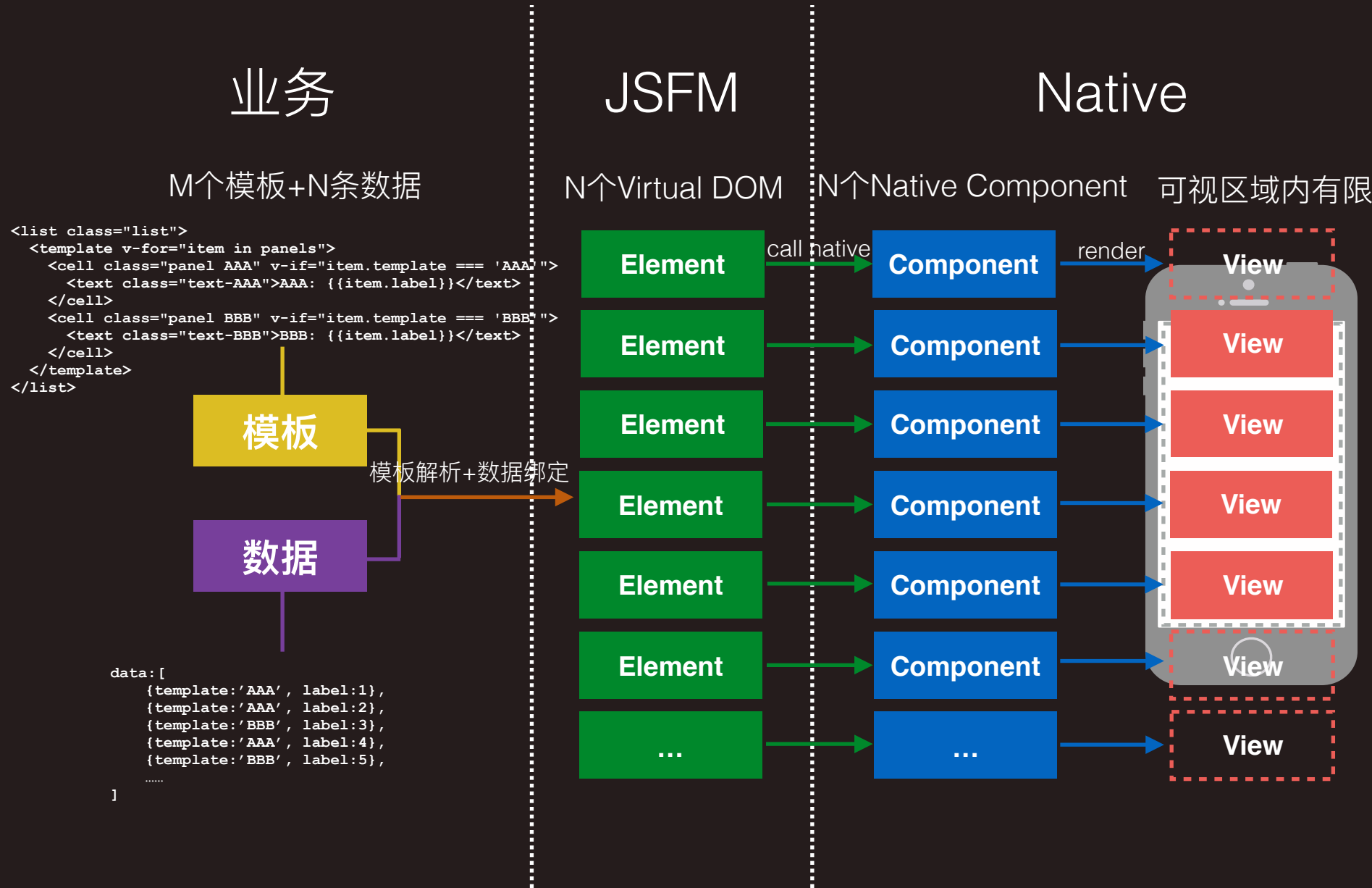
?

真正接近native性能的列表组件？

WEEX

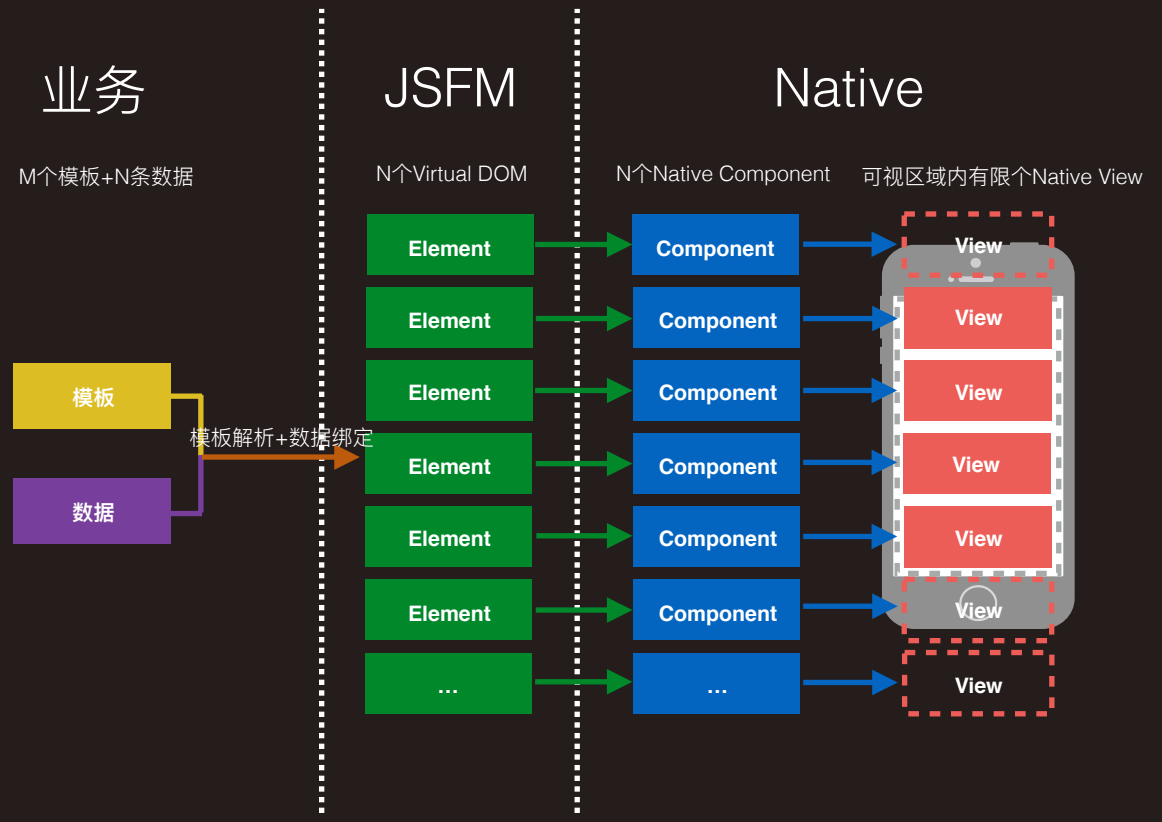

Native List
Weex Developers
Weex List

老的List设计


业务
JSFM
Native
M个模板+N条数据
N个Virtual DOM
N个Native Component
可视区域内有限个Native View
<list class="list">
<template v-for="item in panels">
<cell class="panel AAA" v-if="item.template === 'AAA'">
<text class="text-AAA">AAA: {{item.label}}</text>
</cell>
<cell class="panel BBB" v-if="item.template === 'BBB'">
<text class="text-BBB">BBB: {{item.label}}</text>
</cell>
</template>
</list>
模板
数据
模板解析+数据绑定
data:[
{template:'AAA', label:1},
{template:'AAA', label:2},
{template:'BBB', label:3},
{template:'AAA', label:4},
{template:'BBB', label:5},
……
]
Element
Element
Element
Element
Element
...
call native
Component
Component
Component
Component
Component
...
render
View
View
View
View
View
View

老的List设计

- 为什么做不到 View 结构的复用？
- 为什么内存占用高？
- 为什么帧率低？



真正的复用

Cell A

Cell B

Cell B

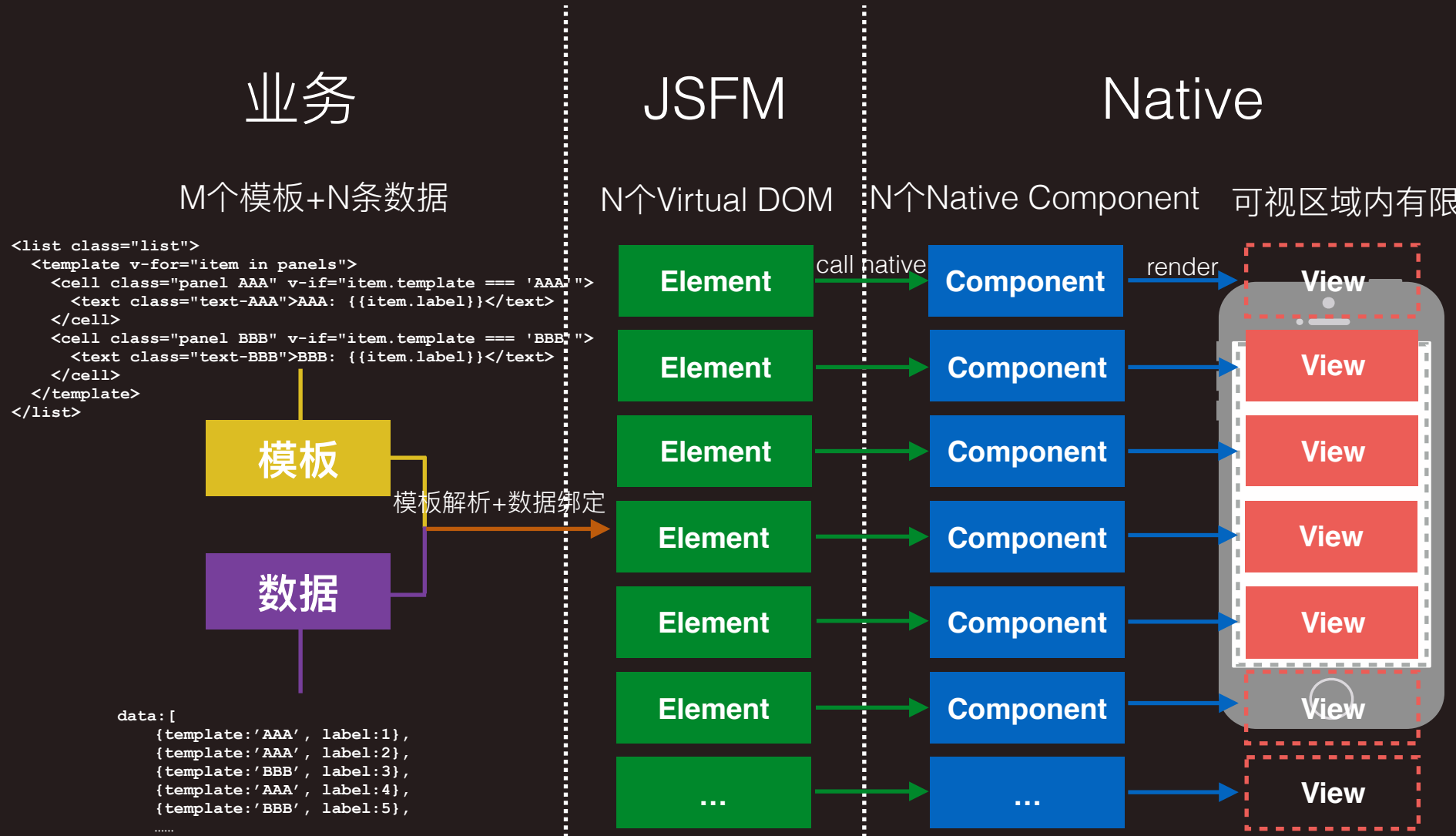
Cell B

数据 =

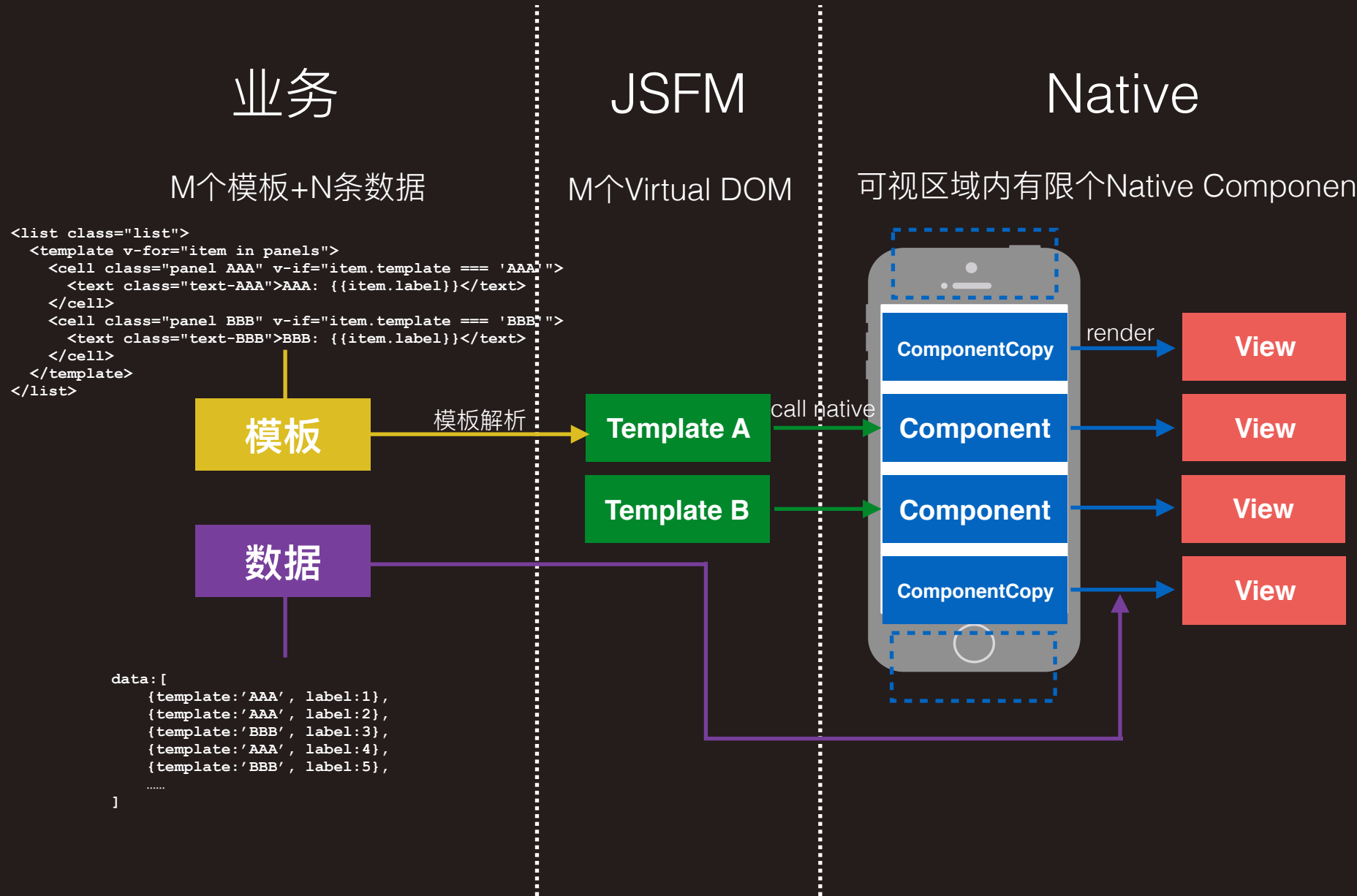
Cell模板 =

WEEX

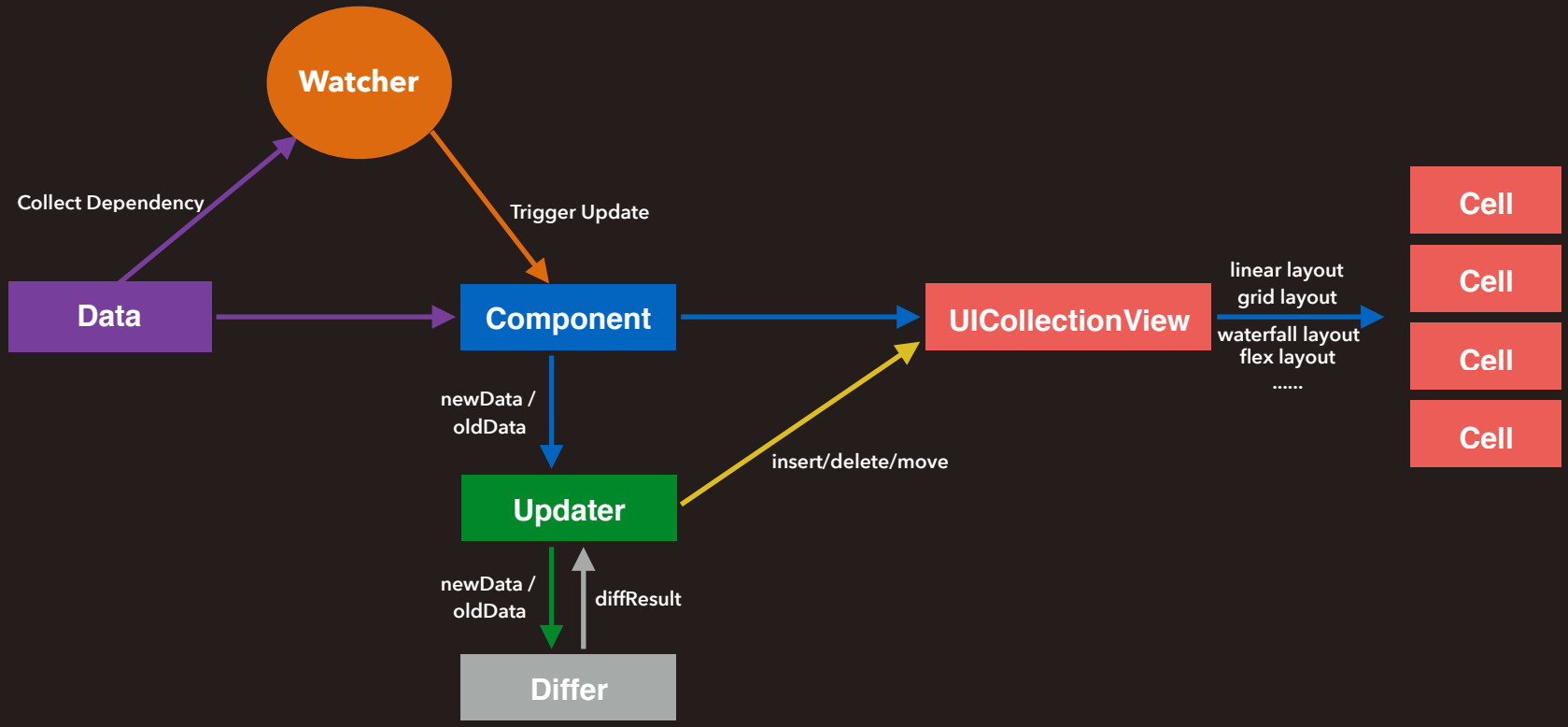
老的List设计


业务
JSFM
Native
M个模板+N条数据
N个Virtual DOM
N个Native Component
可视区域内有限个Native View
<list class="list">
<template v-for="item in panels">
<cell class="panel AAA" v-if="item.template === 'AAA'">
<text class="text-AAA">AAA: {{item.label}}</text>
</cell>
<cell class="panel BBB" v-if="item.template === 'BBB'">
<text class="text-BBB">BBB: {{item.label}}</text>
</cell>
</template>
</list>
模板
数据
模板解析+数据绑定
data:[
{template:'AAA', label:1},
{template:'AAA', label:2},
{template:'BBB', label:3},
{template:'AAA', label:4},
{template:'BBB', label:5},
……
]
Element
Element
Element
Element
Element
...
call native
Component
Component
Component
Component
Component
...
render
View
View
View
View
View
View

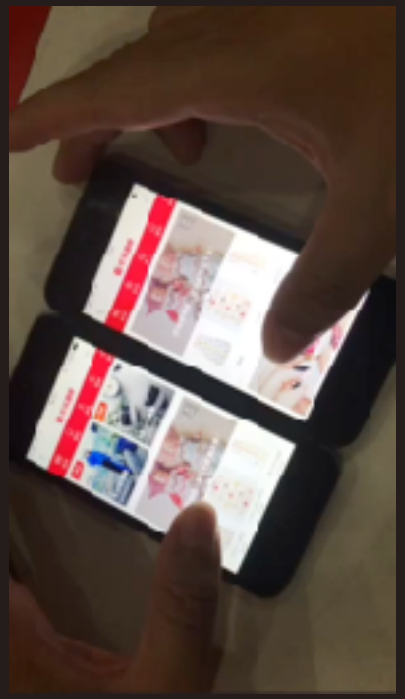
新的List设计


业务
JSFM
Native
M个模板+N条数据
M个Virtual DOM
可视区域内有限个Native Component+View
<list class="list">
<template v-for="item in panels">
<cell class="panel AAA" v-if="item.template === 'AAA'">
<text class="text-AAA">AAA: {{item.label}}</text>
</cell>
<cell class="panel BBB" v-if="item.template === 'BBB'">
<text class="text-BBB">BBB: {{item.label}}</text>
</cell>
</template>
</list>
模板
模板解析
Template A
Template B
call native
数据
data:[
{template:'AAA', label:1},
{template:'AAA', label:2},
{template:'BBB', label:3},
{template:'AAA', label:4},
{template:'BBB', label:5},
……
]
ComponentCopy
Component
Component
ComponentCopy
render
View
View
View
View

新的List设计

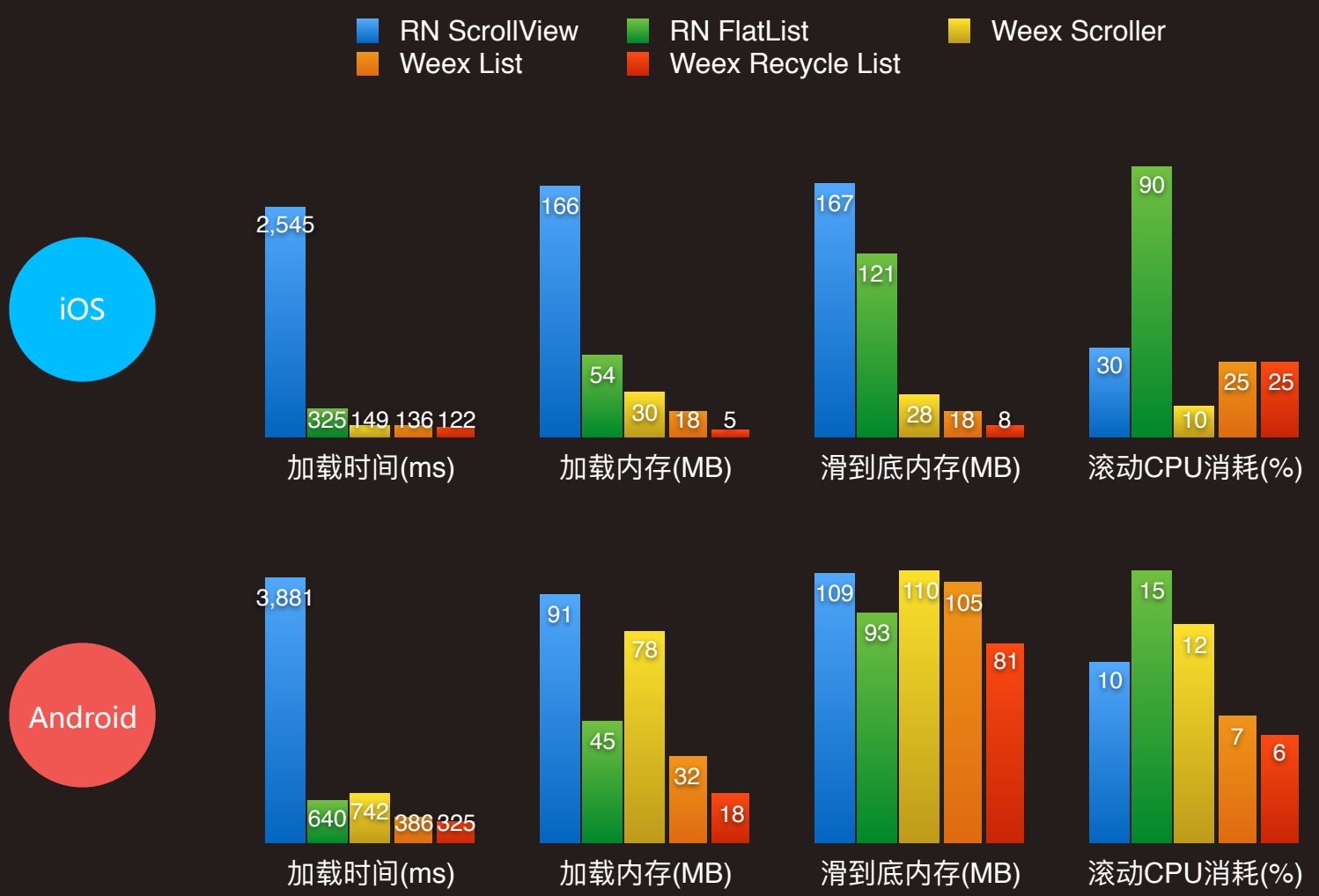

Watcher
Collect Dependency
Trigger Update
Data
Component
UICollectionView
linear layout
grid layout
waterfall layout
flex layout
.....
Cell
Cell
Cell
Cell
newData /
oldData
insert/delete/move
Updater
newData /
oldData
diffResult
Differ

性能对比



60屏数据

3000+节点



WEEX

内核升级

内核挑战

- 业务复杂度的提升，带来性能挑战；
- 业务场景覆盖的提升，带来稳定性的挑战；
- GCanvas 、AR/VR 等富交互组件对Weex 提出更高效的通信要求；
- V8 引擎 安全漏洞，ES6特性不支持问题；
- yoga 引擎存在证书风险

WEEX

Weex 内核演进

更快，更稳定，更安全，更小

1. 旧版本V8引擎 ->新版本Javascript引擎 ->
独立进程版本JSC -> Sandbox 机制 ->瘦身

JS 引擎

高性能，支持更多Flex属性

1. yoga 证书风险；
2. Layout 作为基础技术自主演进；

Layout 引擎

Weex Core

架构演进

1. 高性能
2. 跨平台性
3. 架构高可用性

WEEX

JS Engine 引擎

JS Engine 引擎

2017.3~2017.7

profile v8 & JSC
提升40%
ES6

2017.7~2017.10

独立进程
0.5%

2017.11~至今

安全漏洞
SandBox

2017.11~至今

编译工具链优化
裁剪

WEEX

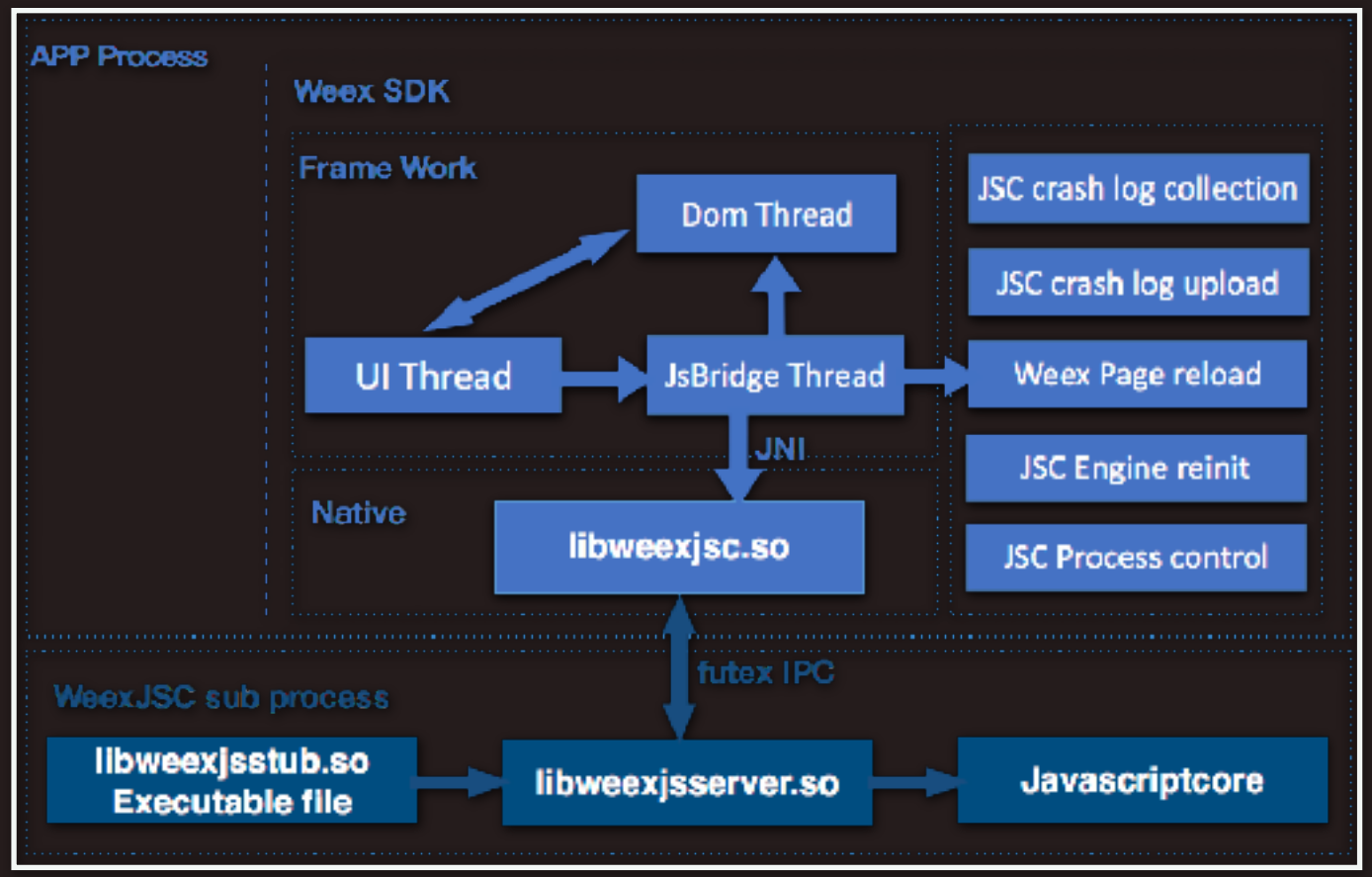
JS Engine — 独立进程化

挑战

- 资源竞争/内存挑战
- 碎片化 / 适配问题
- 稳定性

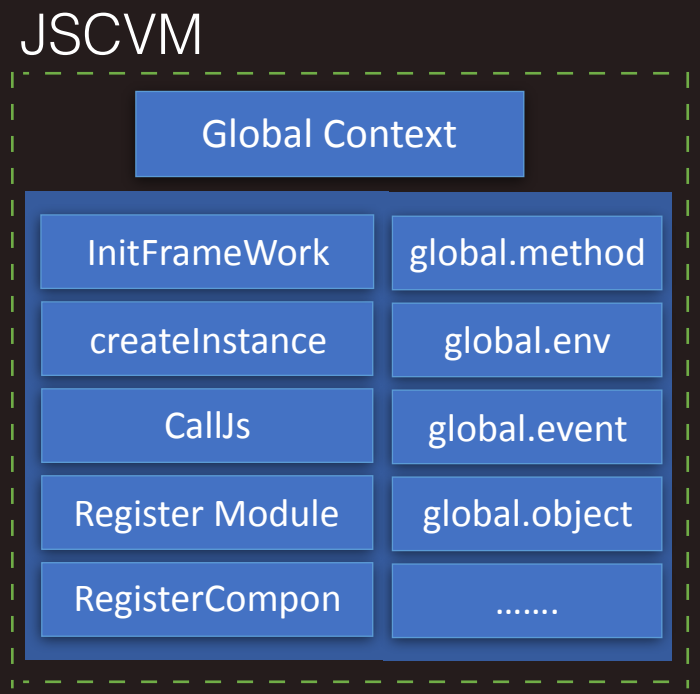
升级

- 主程序隔离性
- 并发性增强
- 自修复能力



JS Engine— SandBox机制

全局对象污染风险



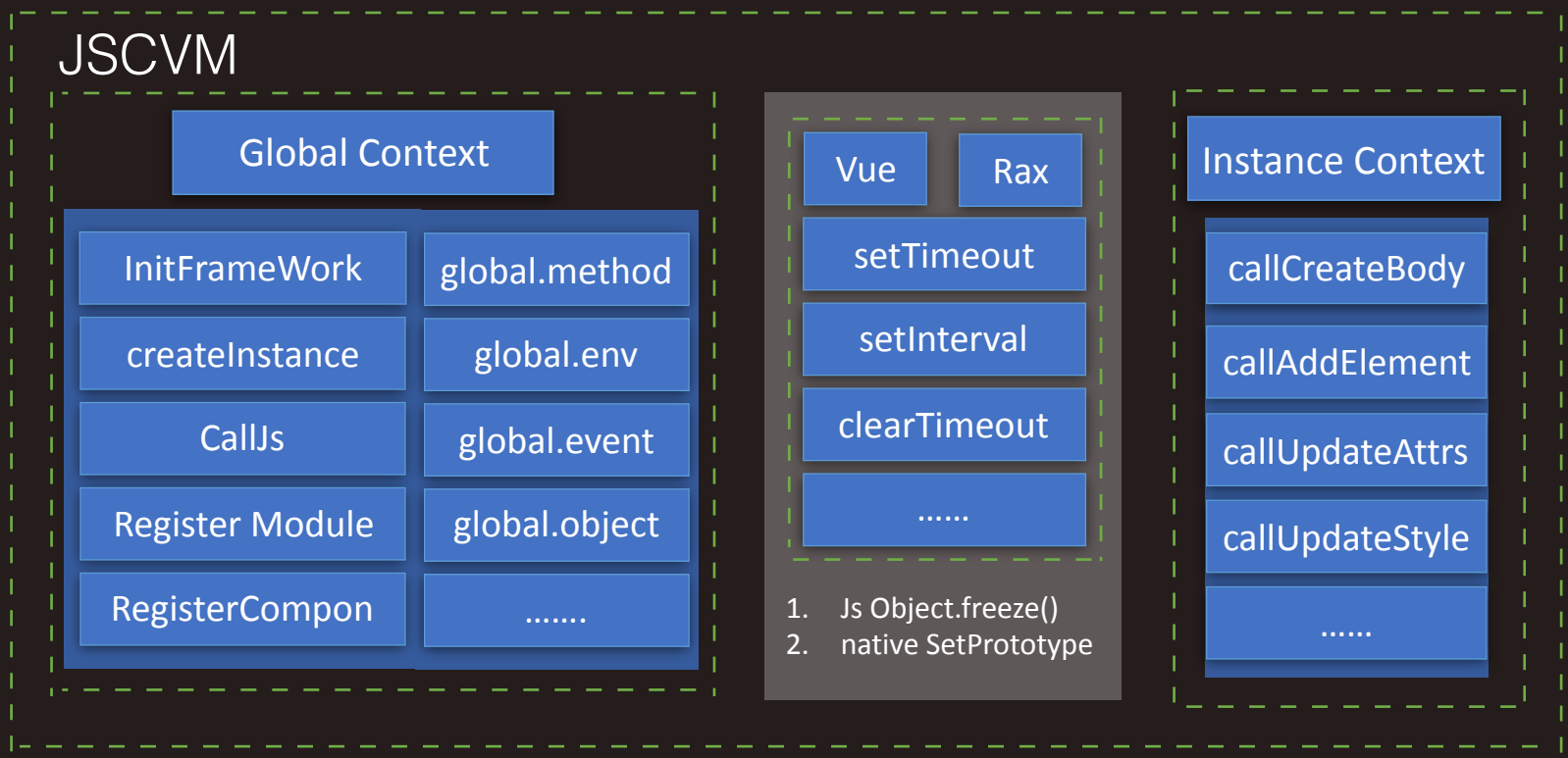
WEEX

JS Engine— SandBox机制

全局对象污染风险

↓

页面Context 隔离



WEEX

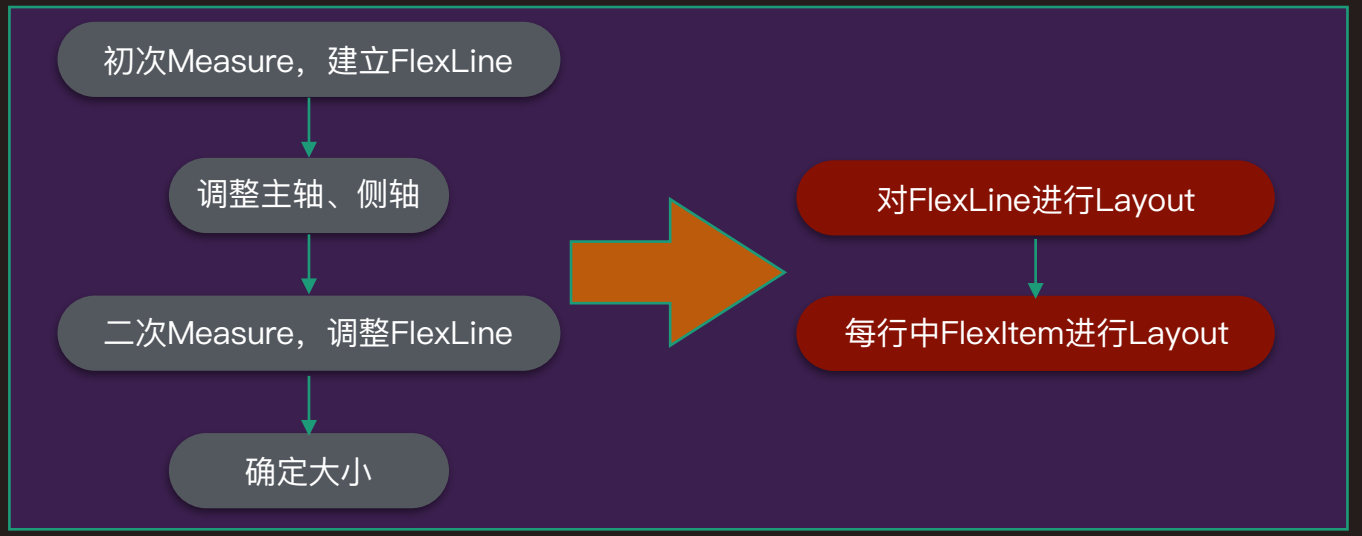
Layout Engine 引擎

Layout Engine 引擎

难点

- 复杂度: **2^34**种可能的组合
- 高频递归调度，性能要求高
- Layout算法下沉到C++层，统一iOS和Android

算法流程



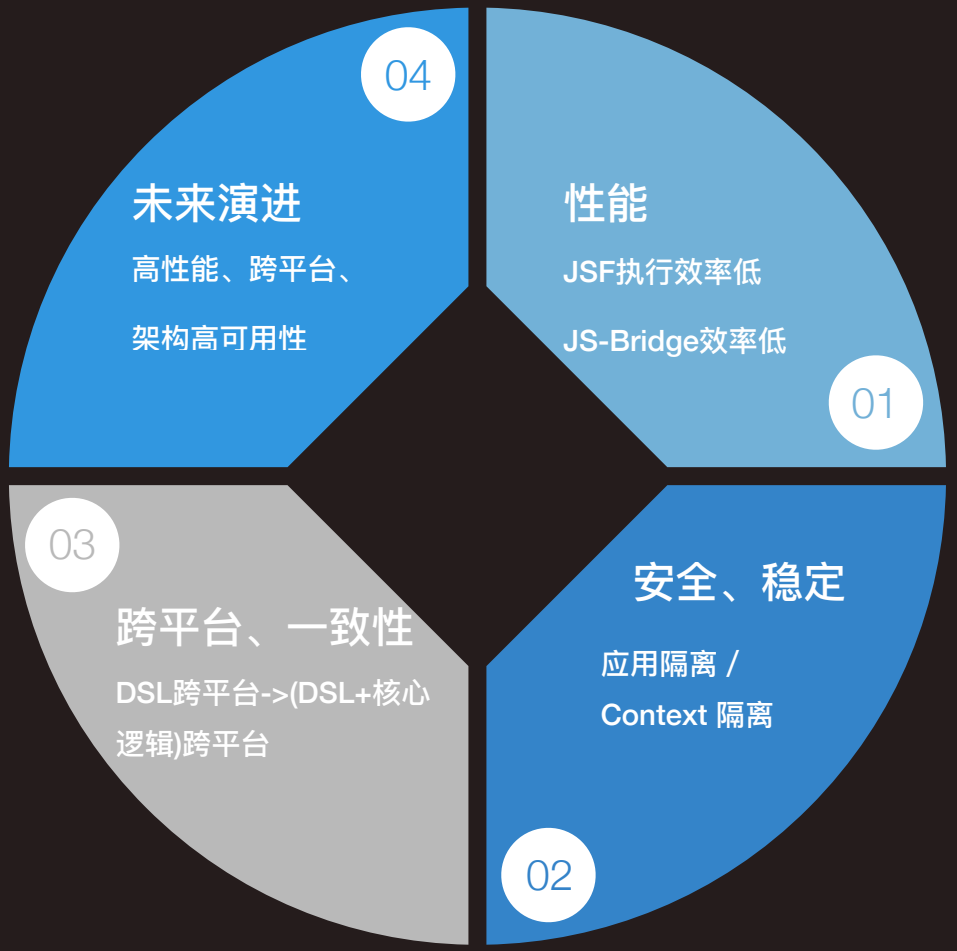
演进路径

WEEX

Why WeexCore

Why WeexCore

目前架构上无法逾越的瓶颈



性能

JSF执行效率低，不native化基本无解
JSBridge通信效率低，必须减少核心模块的跨Contex通信

稳定性、安全

主进程隔离 / Context 隔离
自修复能力

一致性

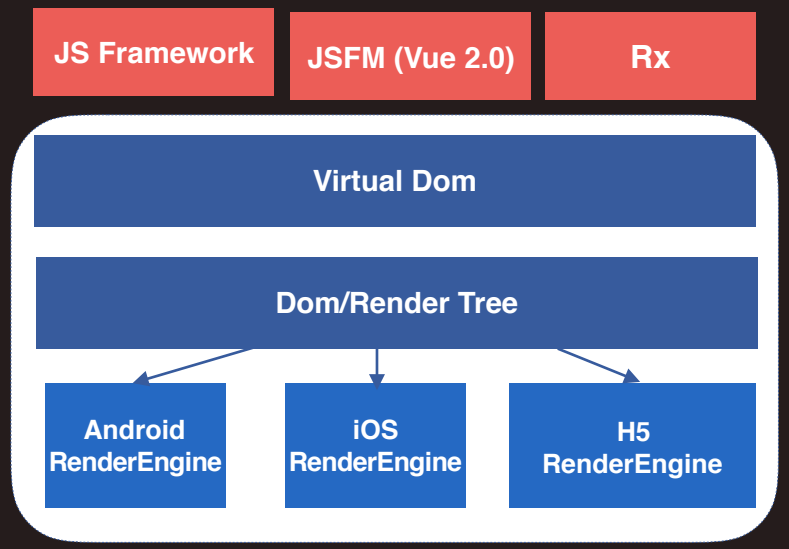
DSL一致，不能保证表现一致
将 Weex 核心逻辑统一到WeexCore，代码层面统一
多平台多核 > 多平台单核，聚焦在内核深度上做功夫

未来演进

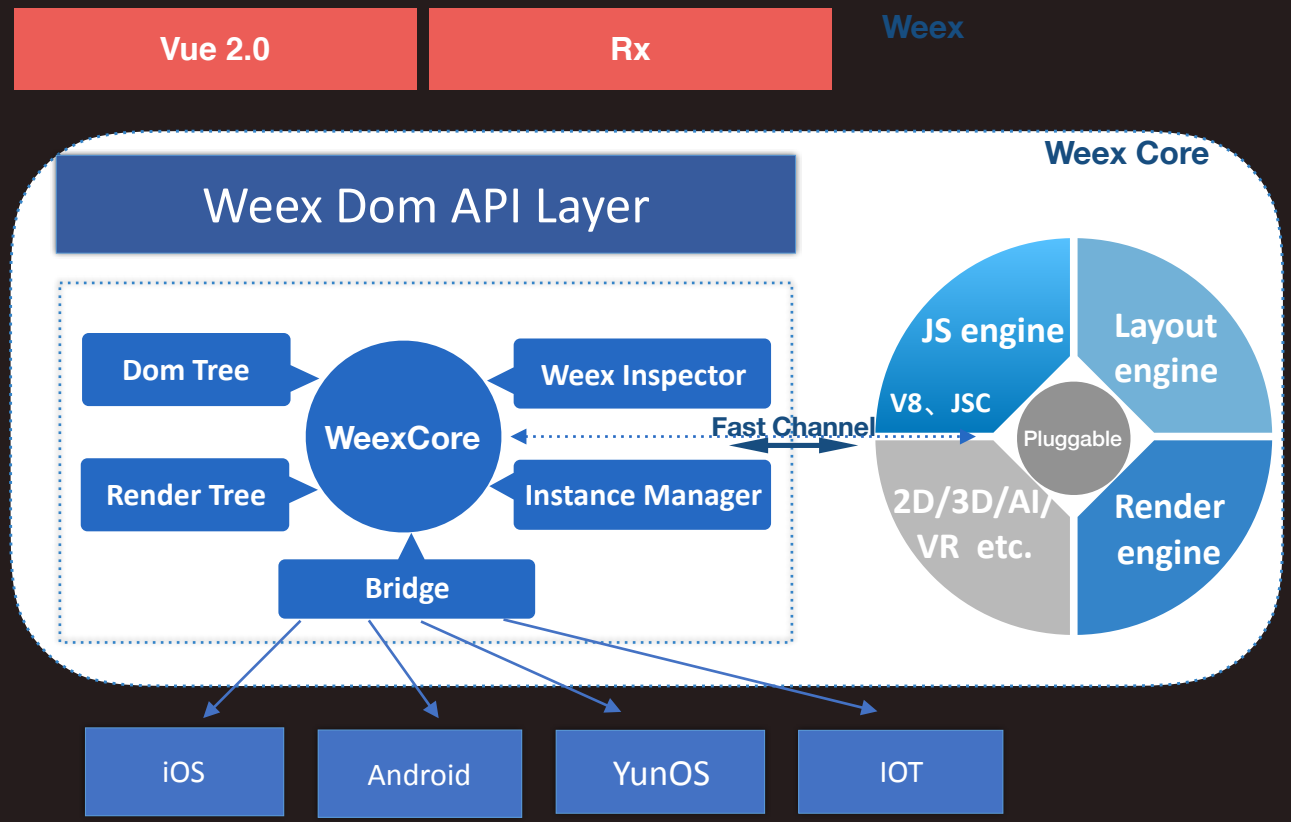
Weex 架构上无法很好支持Gcanvas、AR/VR、视频等复杂组件
如何构造高可用架构，实现基础能力的输出

WEEX

WeexCore 架构升级



Weex 旧架构



WeexCore 架构

架构升级

- 标准化Dom API
- JSFramework native 化
- Fast-Channel
- 架构高可用，通用模块可插拔
- 跨平台

https://weex.apache.org