

# Weex架构简介和性能优化

技术创新，变革未来





weex

Docs

Showcase

Download

 Alibaba Cloud

 Apply for access

# weex

A framework for building Mobile cross-platform UI

Getting Started

Apply for access



## Lightweight

low footprint , simple  
syntax , and easy to use



## Extendable

abundant build-in components ,  
extendable apis , various events



## High Performance

load fast , render fast ,  
better experience

<https://github.com/alibaba/weex>



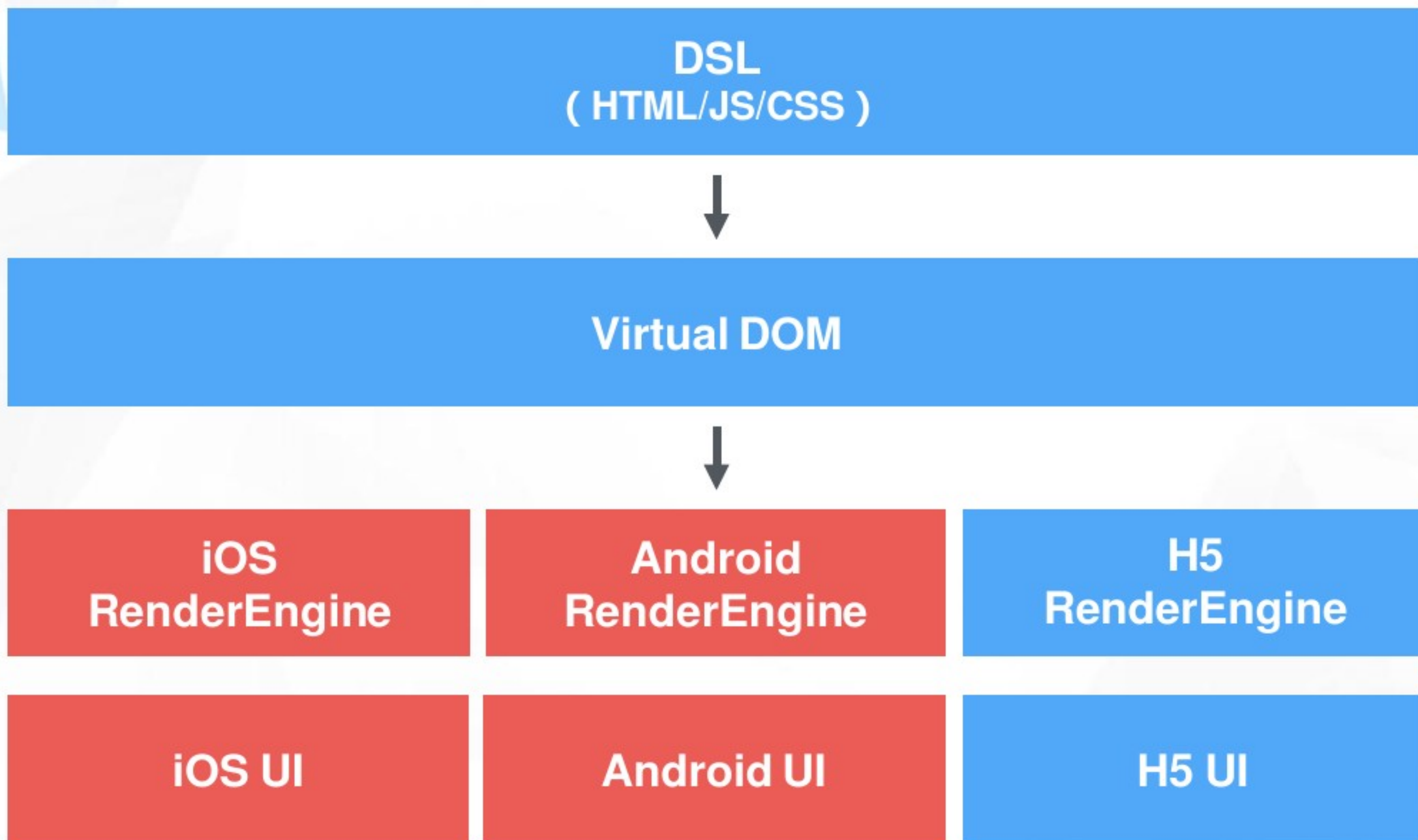
# 目录

- 1、Weex框架简介
- 2、Weex性能优化实践
- 3、最佳实践

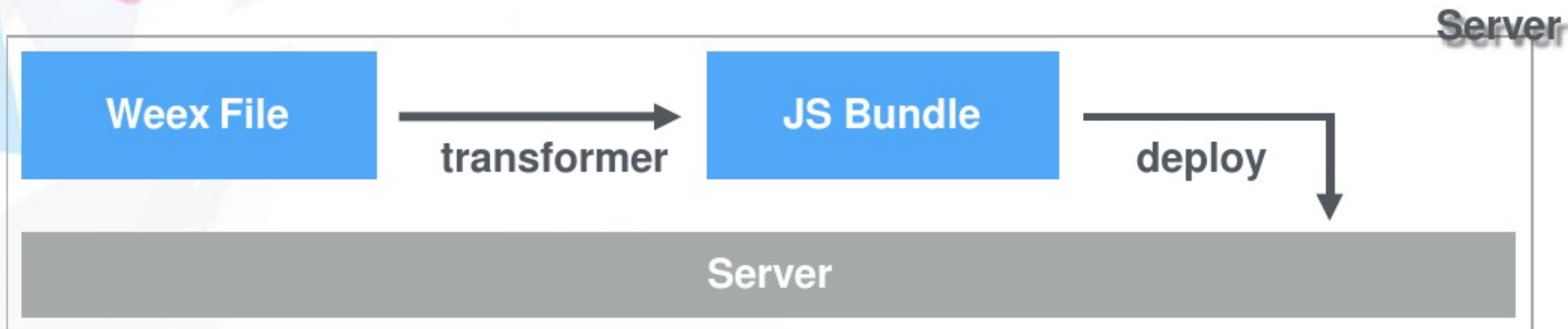


# Weex框架简介

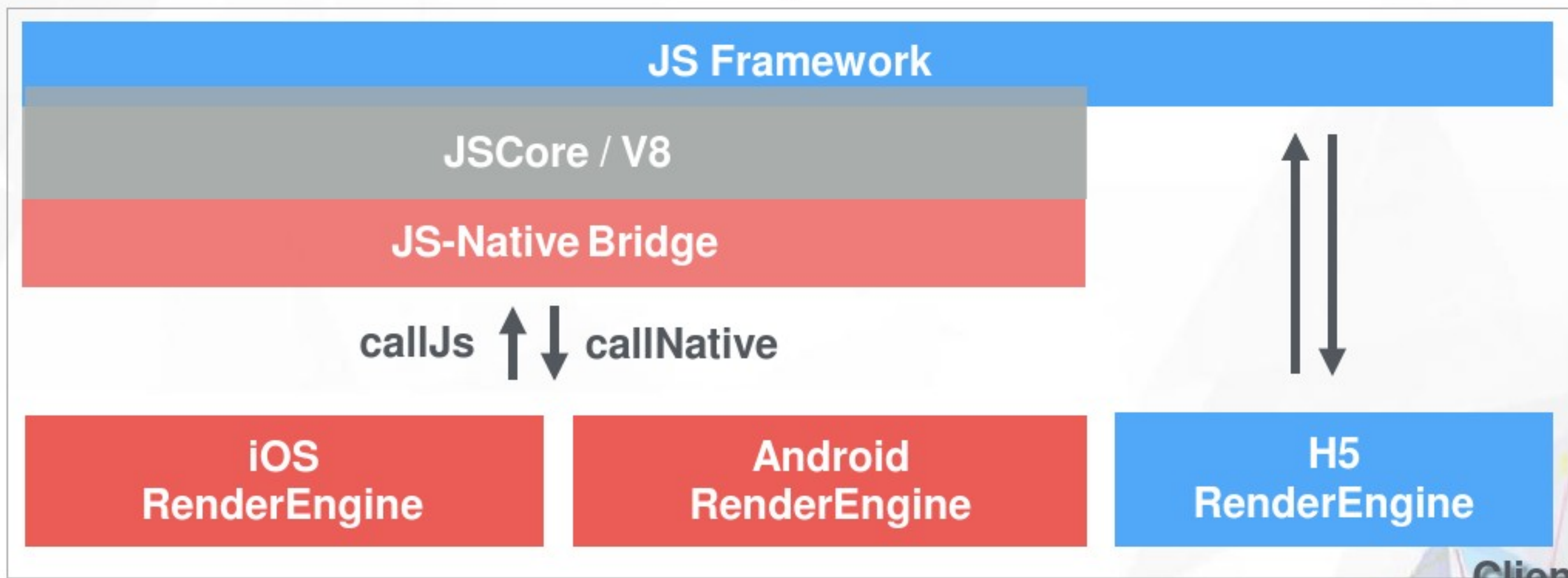
# 简明架构图



# 简明架构图



↓ JS



Client



```
<template>
  <div class="hello" onclick="clickHandler">
    <text>{{message0}}</text>
    <text>{{message1}}</text>
  </div>
</template>
```

```
<style>
  .hello {
    flex-direction: row;
  }
</style>
```

```
<script>
  module.exports = {
    data: {
      message0: 'Hello',
      message1: ' World.'
    },
    methods: {
      clickHandler: function() {}
    }
  }
</script>
```

DSL

— — — →

— — — →

— — — →

```
module.exports.template = {
  "type": "div",
  "classList": [
    "hello"
  ],
  "events": {
    "click": "clickHandler"
  },
  "children": [
    {"type": "text"...},
    {"type": "text"...}
  ]
};

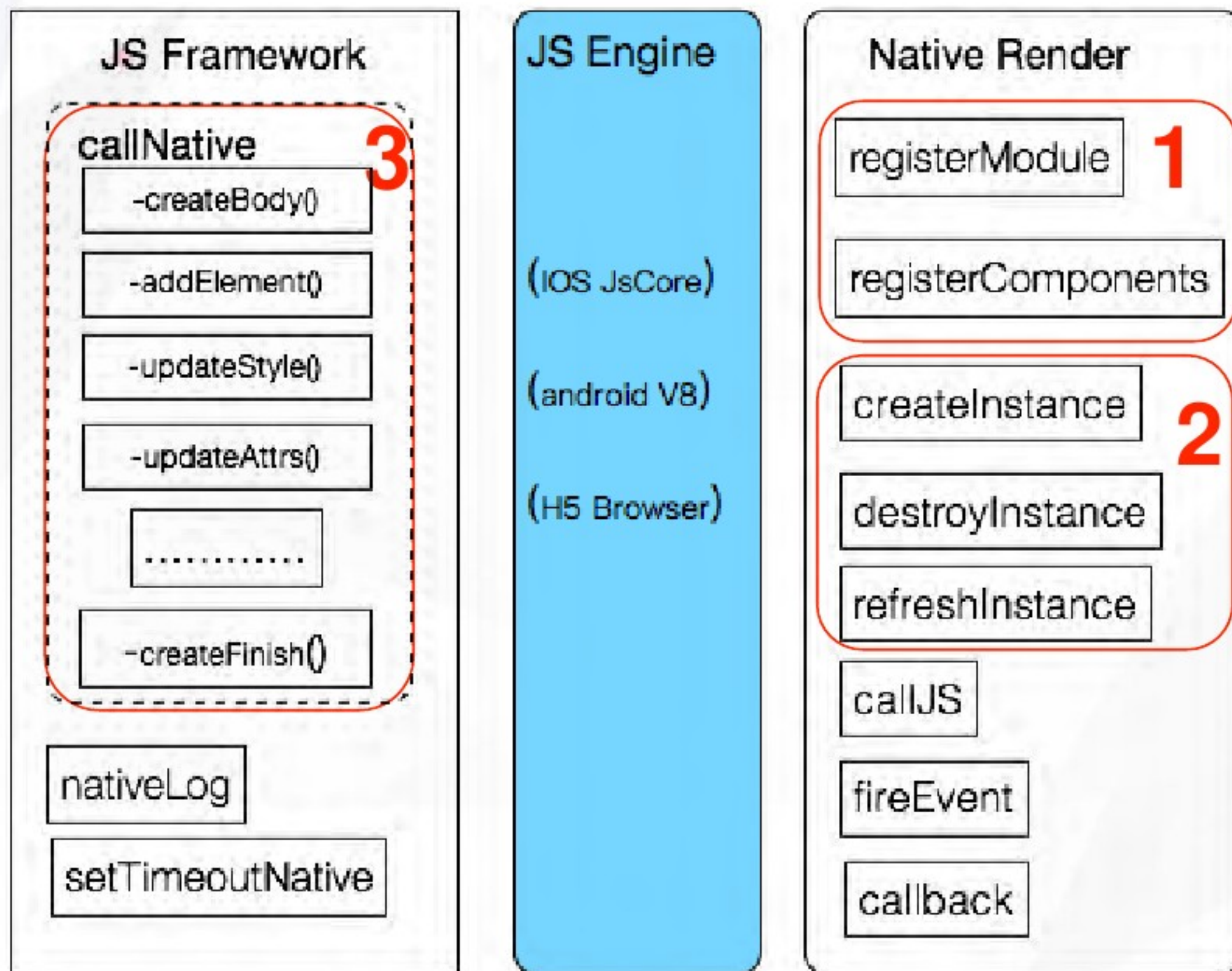
module.exports.style = {
  "hello": {
    "flexDirection": "row"
  }
};

module.exports = {
  data: function() {
    return {
      message0: 'Hello',
      message1: ' World.'
    }
  },
  methods: {
    clickHandler: function() {}
  }
};
```

JS-Bundle



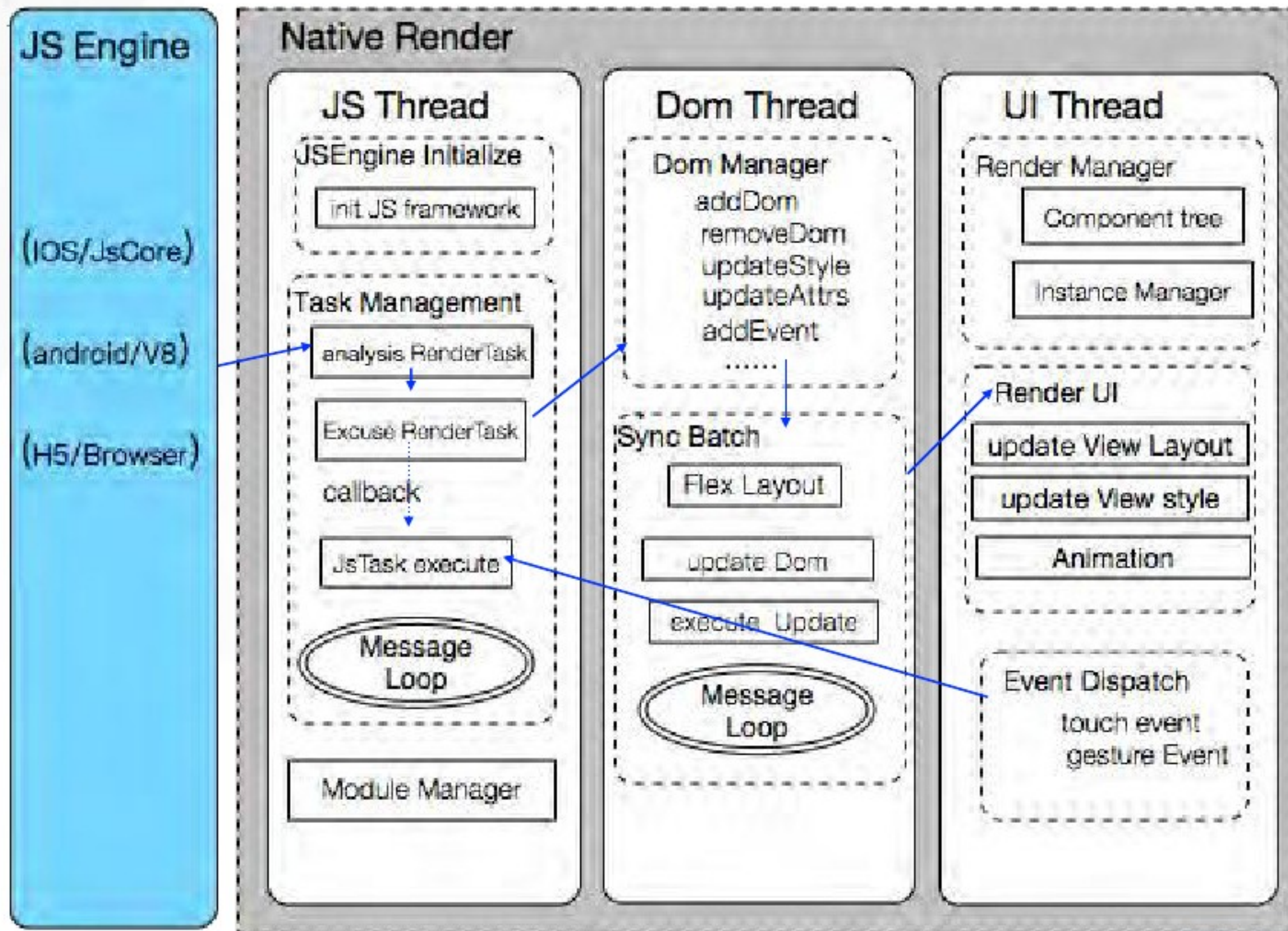
# JS&Native 通信协议





# Native Render

```
call Native >>>> instanceId:1, tasks:  
[{"module":"dom","method":"addElement","arg  
s":["_root",{"ref":"186","type":"div","attr":  
{,"style":{}},-1]], callback:185  
  
call JS >>>> instanceId:5 tasks:[{"args":  
["185",{"}], "method":"callback"}]
```





# 性能优化实践





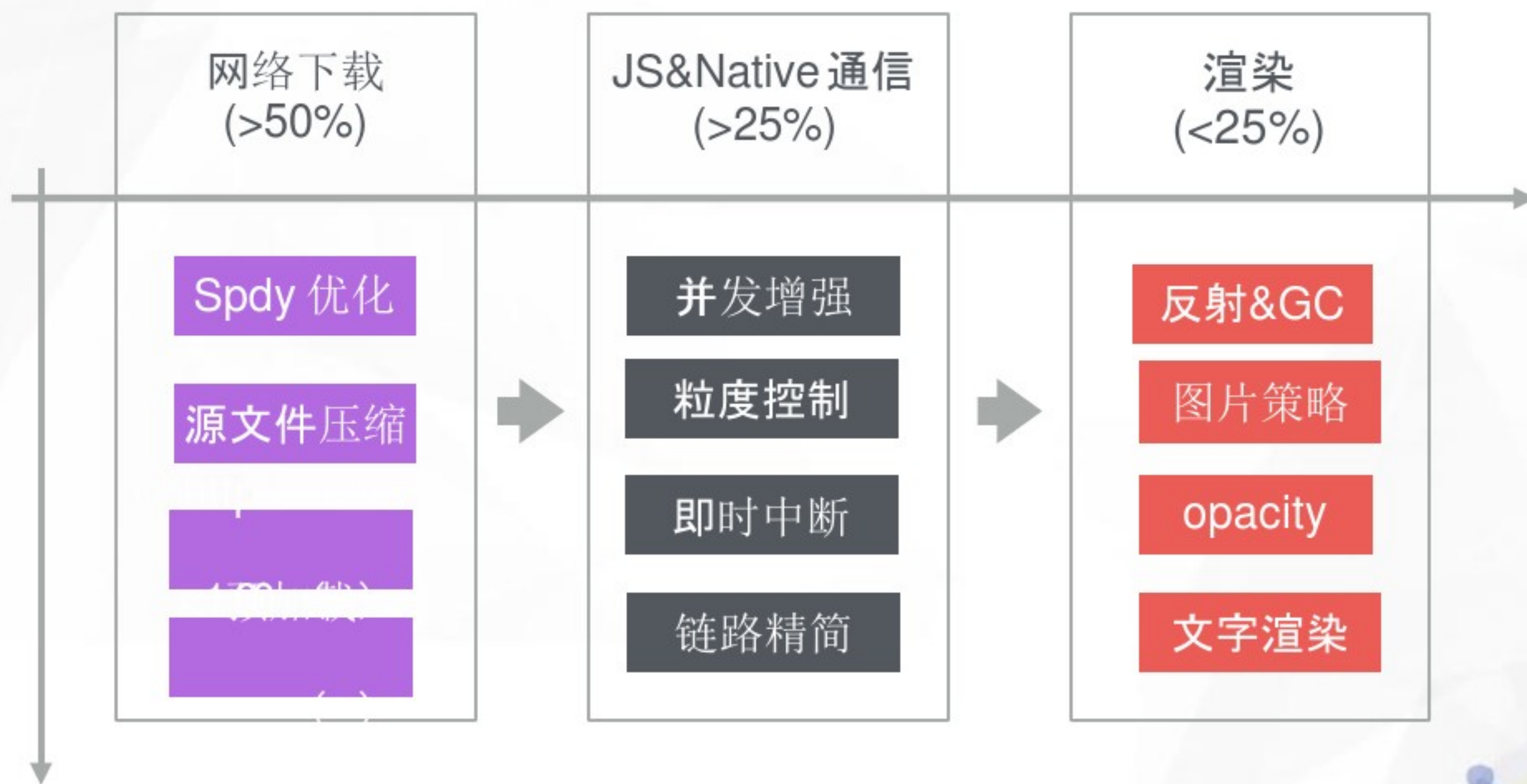
# 网络+渲染：秒开

# 问题与挑战

- 1、JS bundle下载慢（压缩后60k, NetworkTime>800ms）
- 2、JS和Native通信效率低，拖慢加载时间
- 3、长页面VDom渲染时间慢，占首屏时间40%左右
- 4、JS Thread过于繁忙（Json解析&反射调用）
- 5、JS Task无法抢占执行，导致新页面打开慢
- 6、复杂页面滚动帧率低的问题



# Weex TimeLine

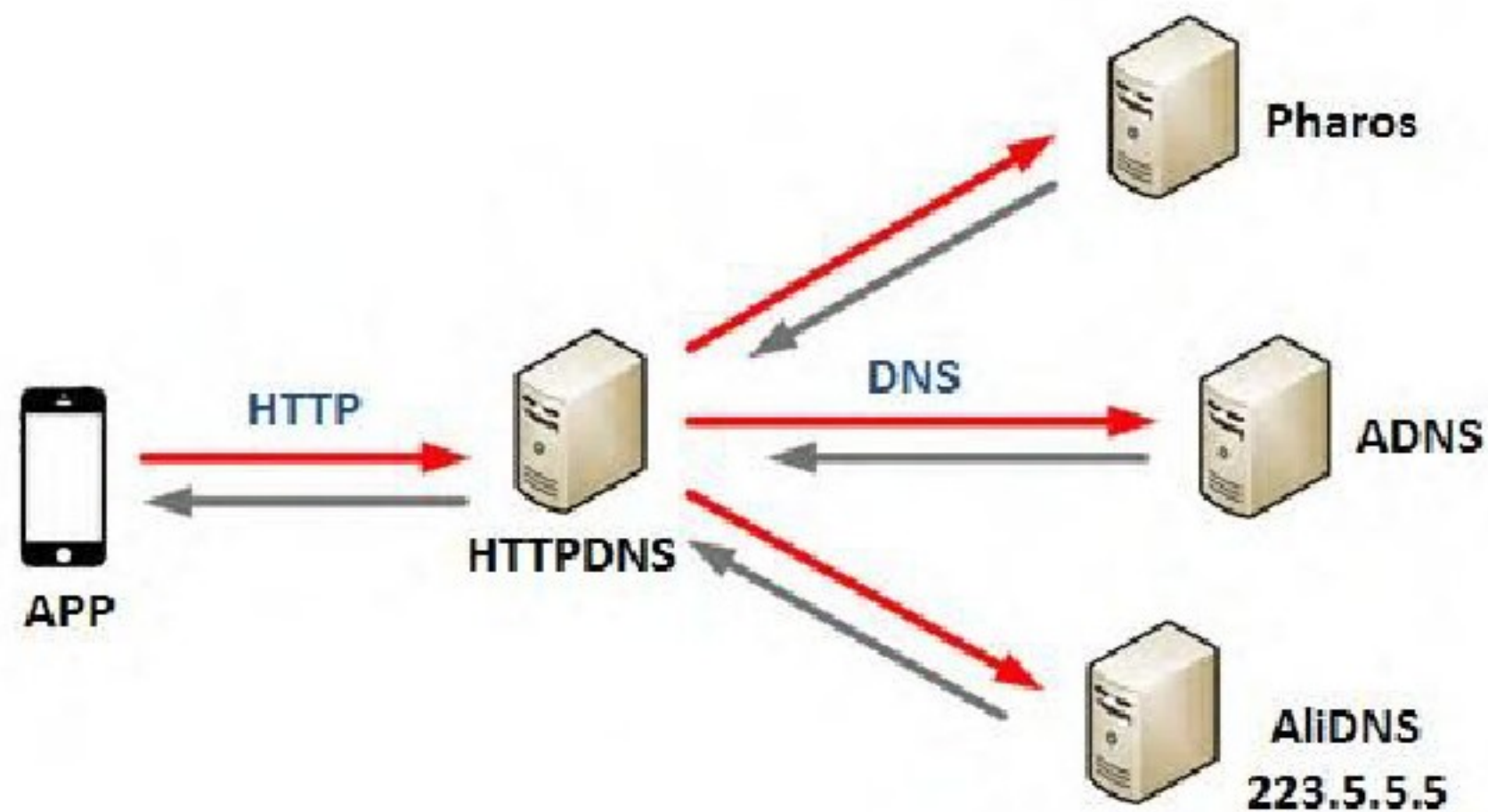




# 网络下载优化

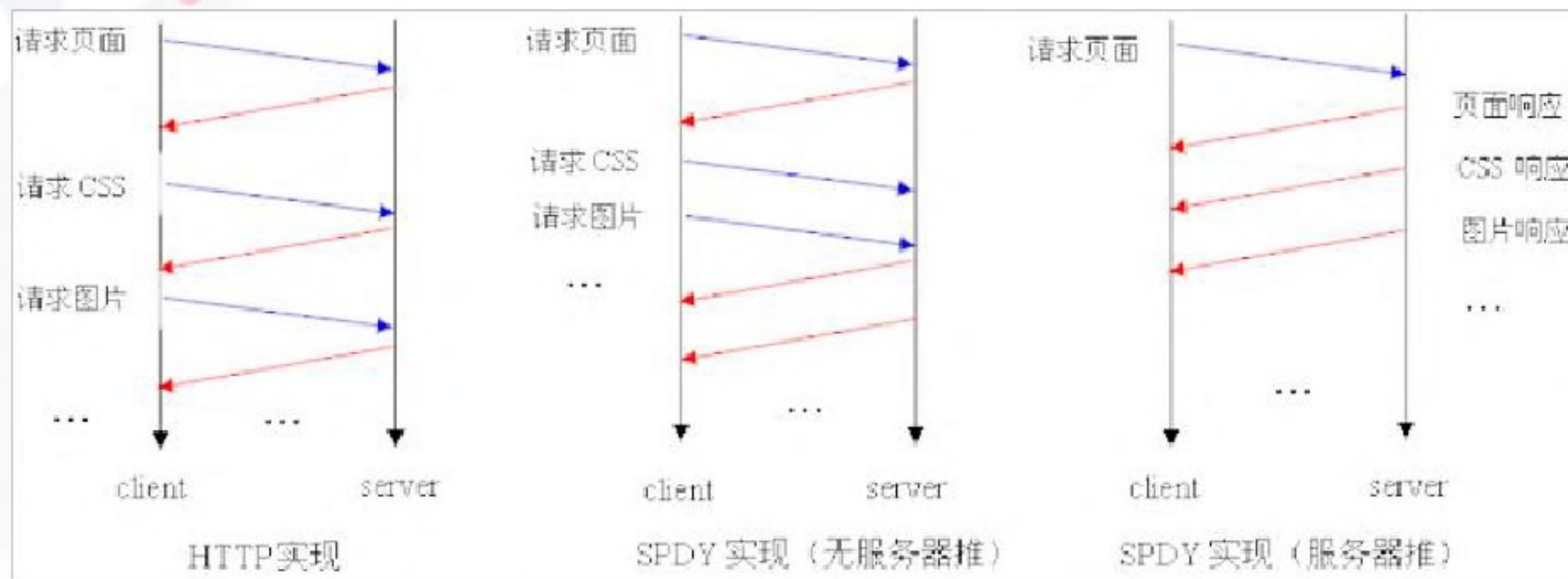


# HTTP-DNS



1. 防止域名劫持
2. 更精准的调度，更快的建连
3. 额外的域名相关信息

# SPDY



1. 多路复用请求优化
2. 服务器推送技术
3. SPDY压缩http头



# 网络优化

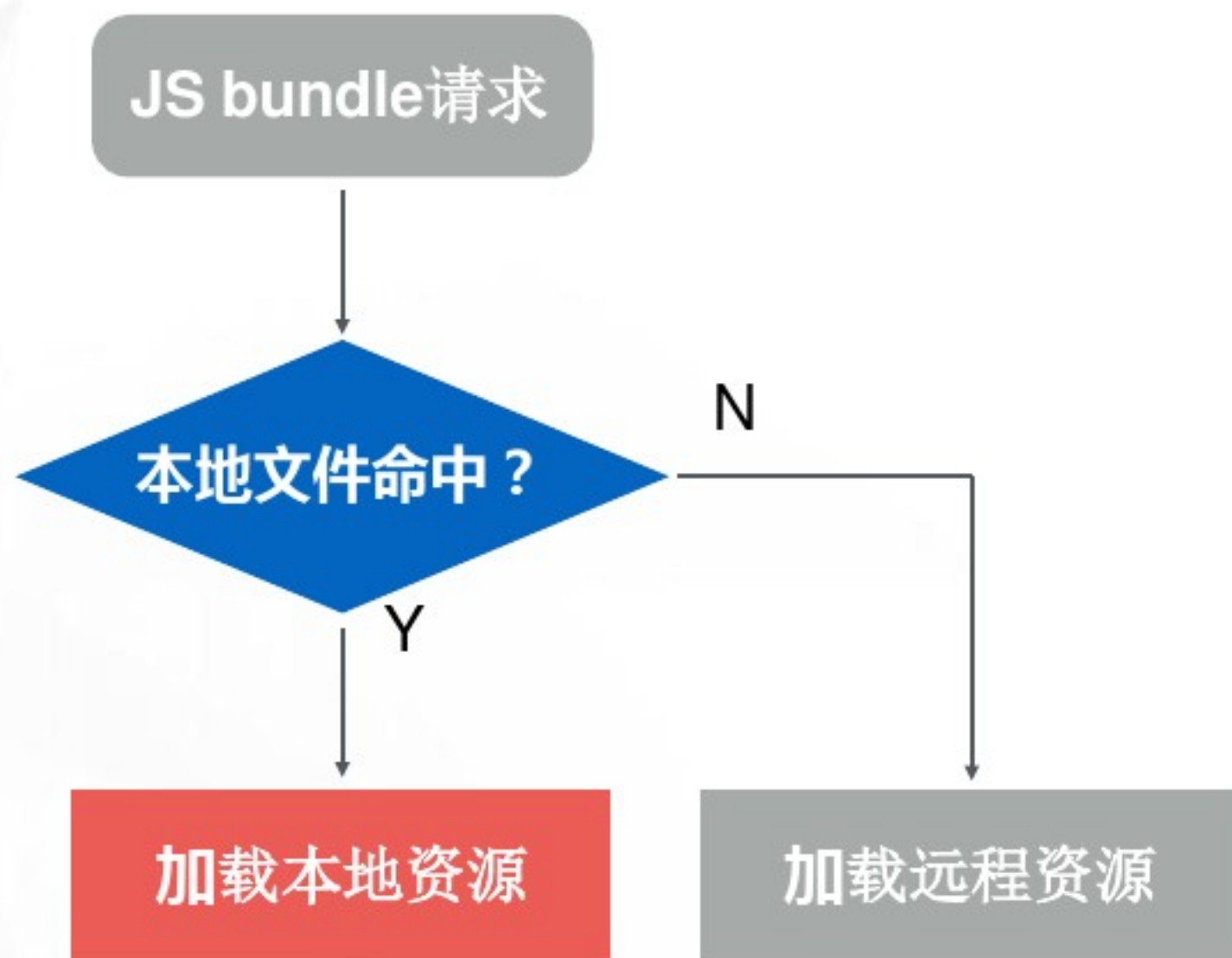
## 网络问题：

- 1、优化前状况：iOS 280~450ms Android 700~900ms
- 2、业务JS文件过大，gzip压缩后（60~80k）
- 3、域名没有收敛
- 4、没有配置Spdy协议
- 5、没有httpCache, 200请求

## 线上性能数据对比：

日期	App版本	页面名称	连接类型	样本数	networkTime	样本数占比(本页)
2016-10-09	6.0.0	DetailWeexFragment	cache	106.69	106.69	0.64%
2016-10-09	6.0.0	DetailWeexFragment	http	13159.59	13159.59	0.01%
2016-10-09	6.0.0	DetailWeexFragment	https	10397.83	10397.83	1.38%
2016-10-09	6.0.0	DetailWeexFragment	packageApp	40.19	40.19	23.55%
2016-10-09	6.0.0	DetailWeexFragment	spdy	248.30	248.30	1.28%
2016-10-09	6.0.0	DetailWeexFragment	spdy_0rtt_cdn	266.83	266.83	73.14%

# 预加载



会场典型值	iOS	Android
走网络	400ms	850ms
预加载	<b>10ms</b>	<b>35ms</b>

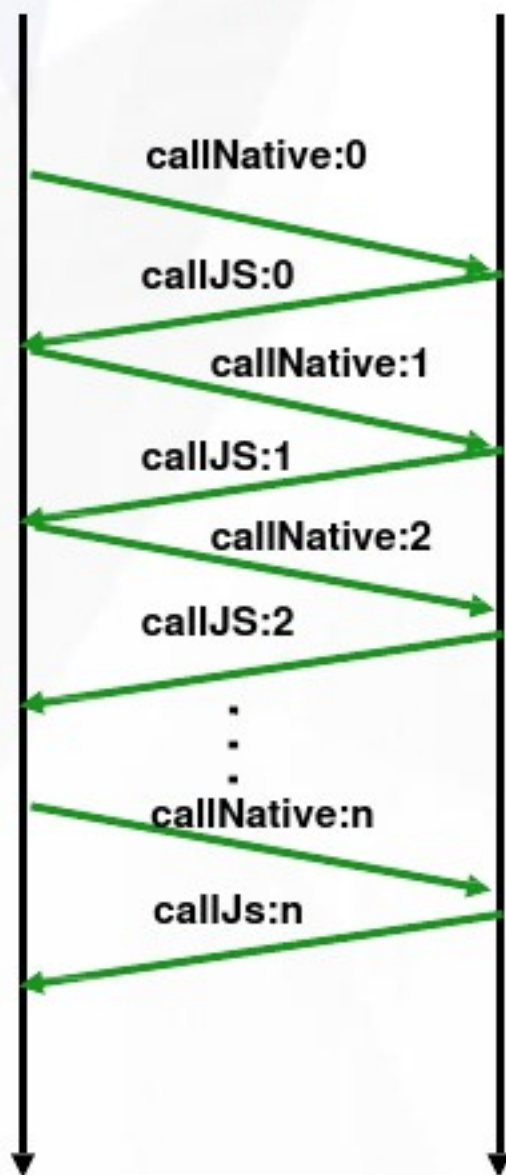




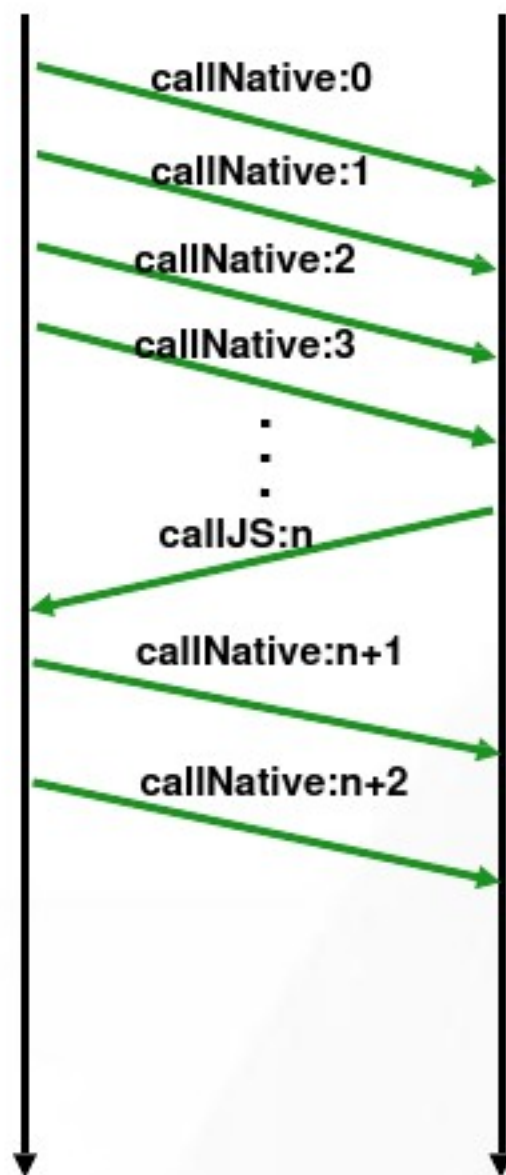
# Weex 通信机制

# 并发增强

JS Native



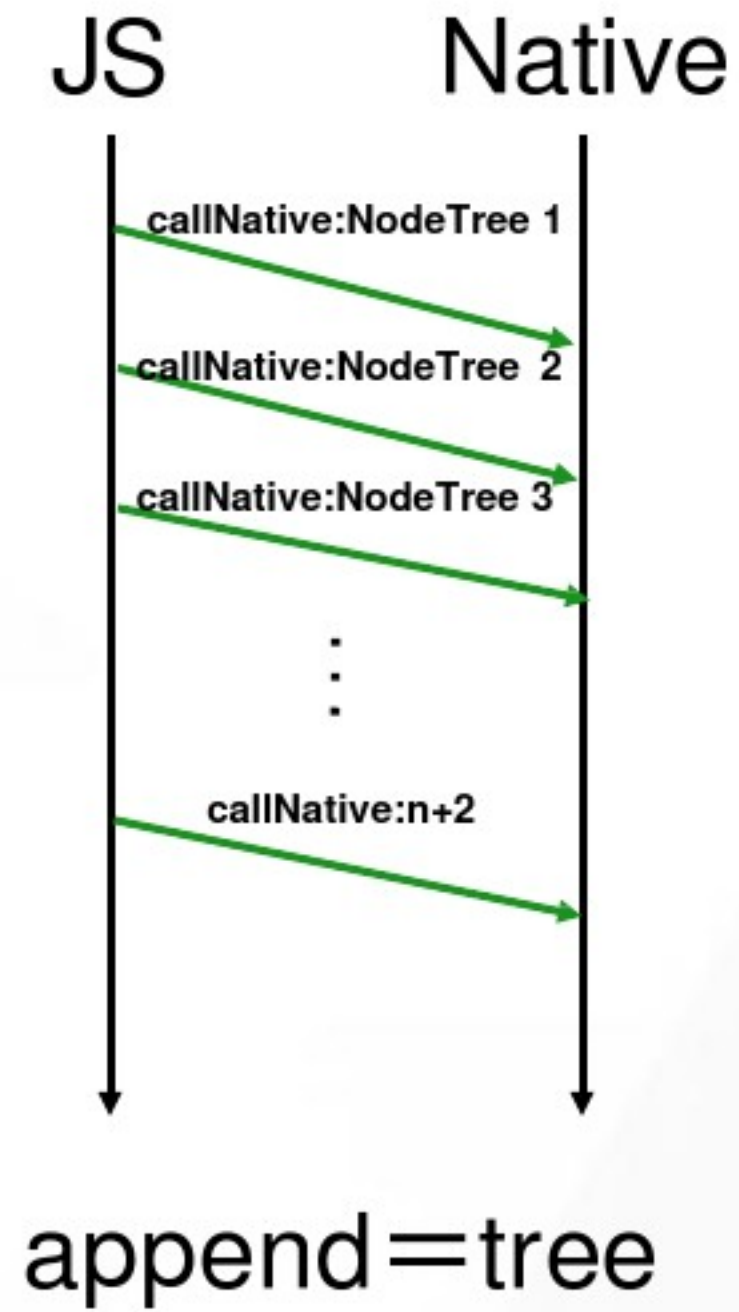
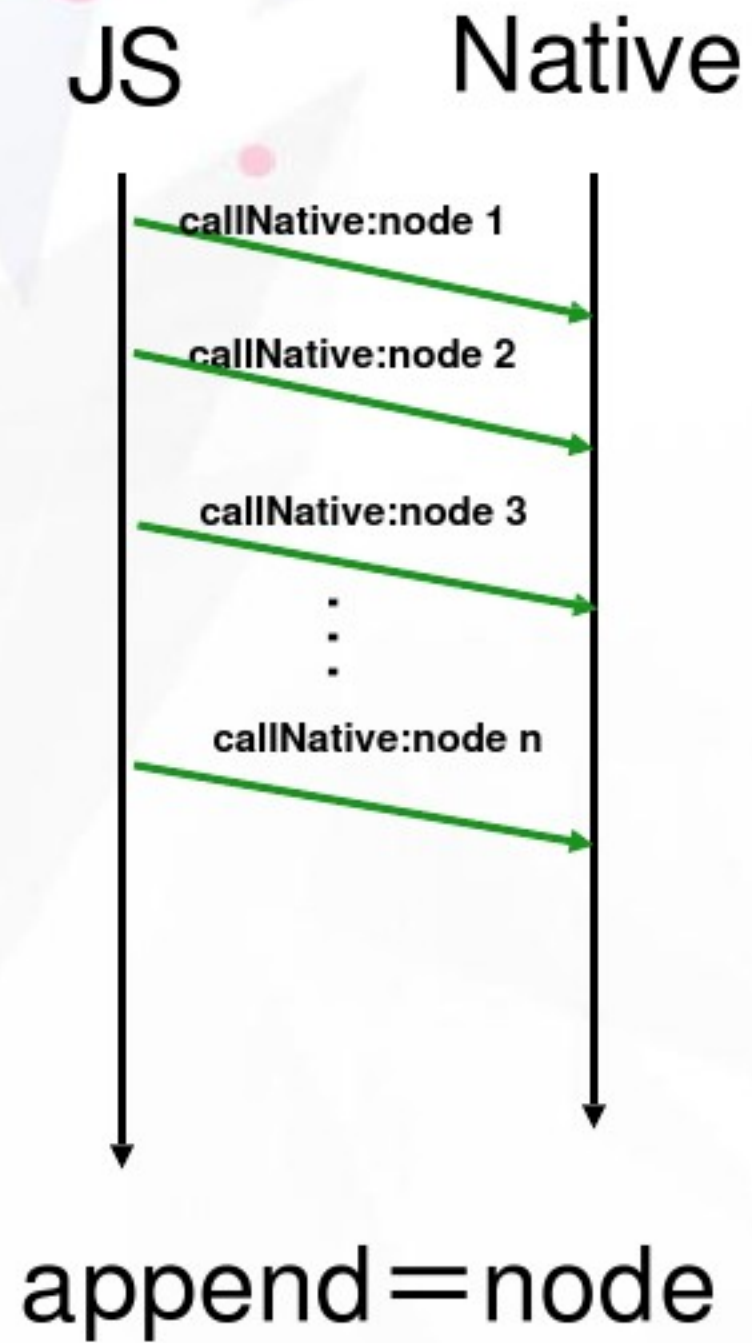
JS Native



压测页面加载性能：加载性能提升30%

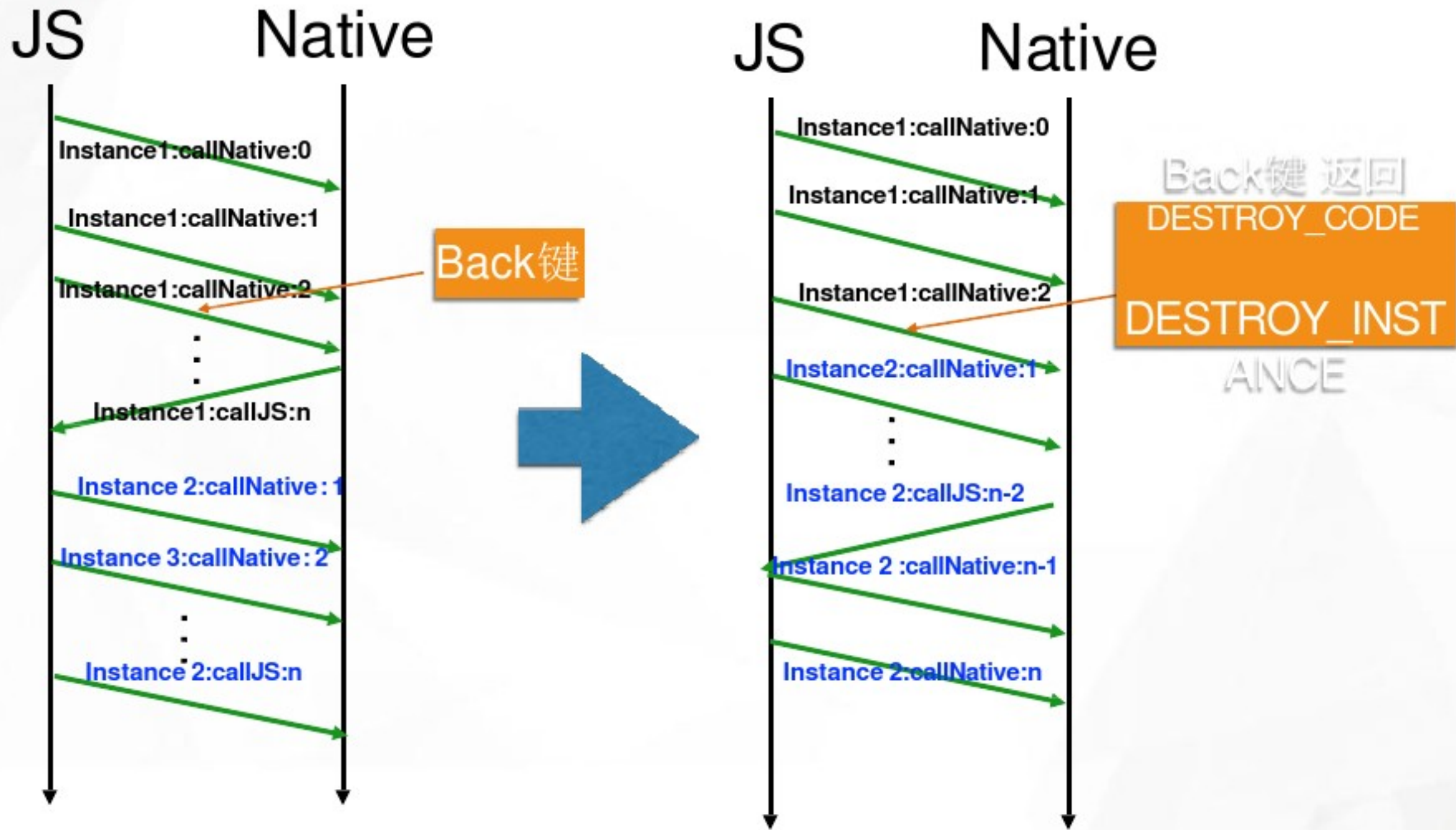


# 粒度控制



压测页面加载性能: append = tree 方式比 append = node 提升 10%

# 即时中断



解决长页面快速进出，无法中断的问题

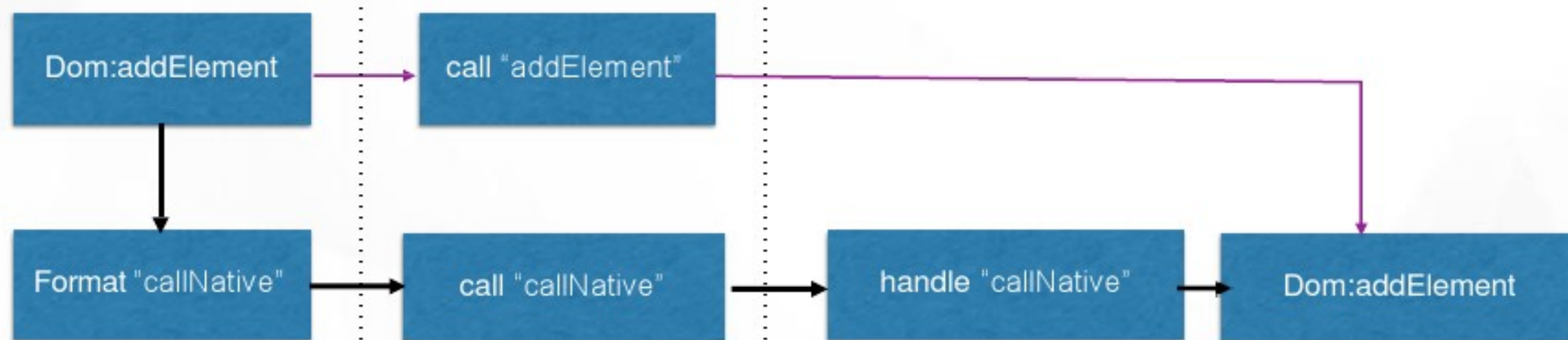


# 链路精简

JS Framework

JNI

Native Render



高频通信接口上浮，减少中间数据转化消耗，加载性能平均提升

# 渲染优化

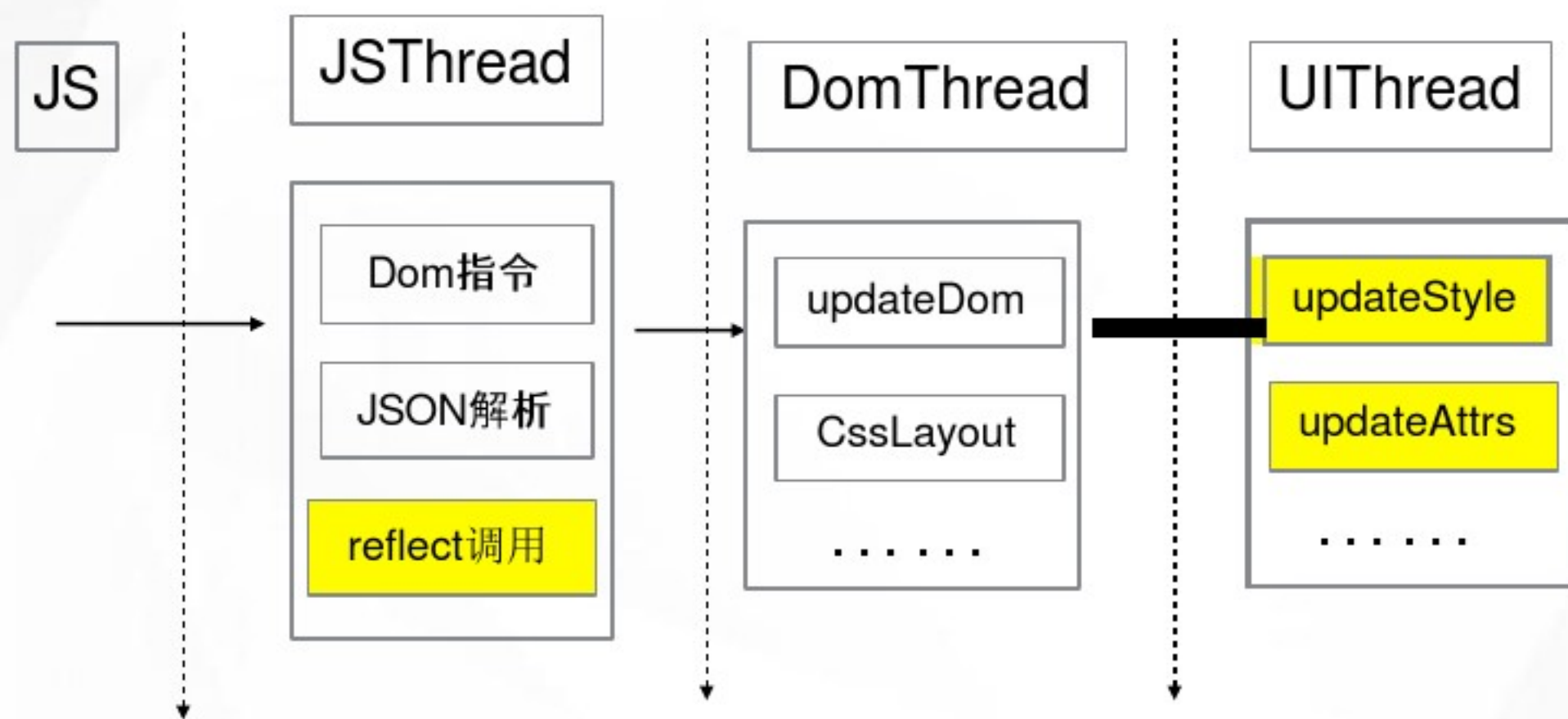




# 渲染优化

- 帧率优化
  - 图片加载策略（域名收敛、webp收敛、尺寸复用等）
  - 反射&GC优化（Android Device Monitor）
  - List最佳实践（一个List 一个10屏的Cell？）
  - opacity属性优化（大坑）
- **文字渲染**优化（截断、性能）

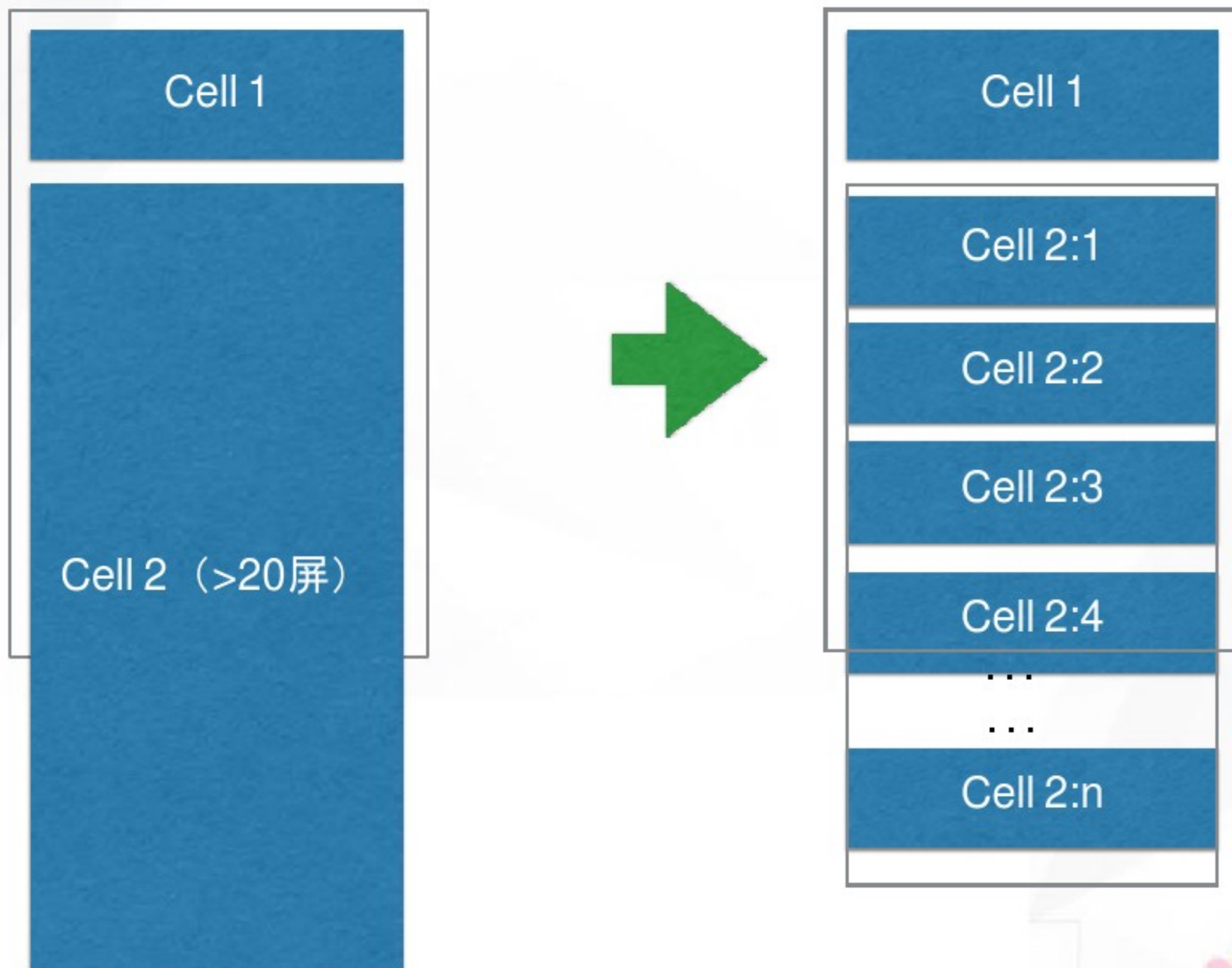
# 反射&GC优化



优化效果：加载时间提升10%，帧率提升5



# List最佳实践





# Opacity 属性问题



解决方案：setLayerType(View.LAYER\_TYPE\_HARDWARE,null);  
优化效果：38帧->55帧



# Text优化

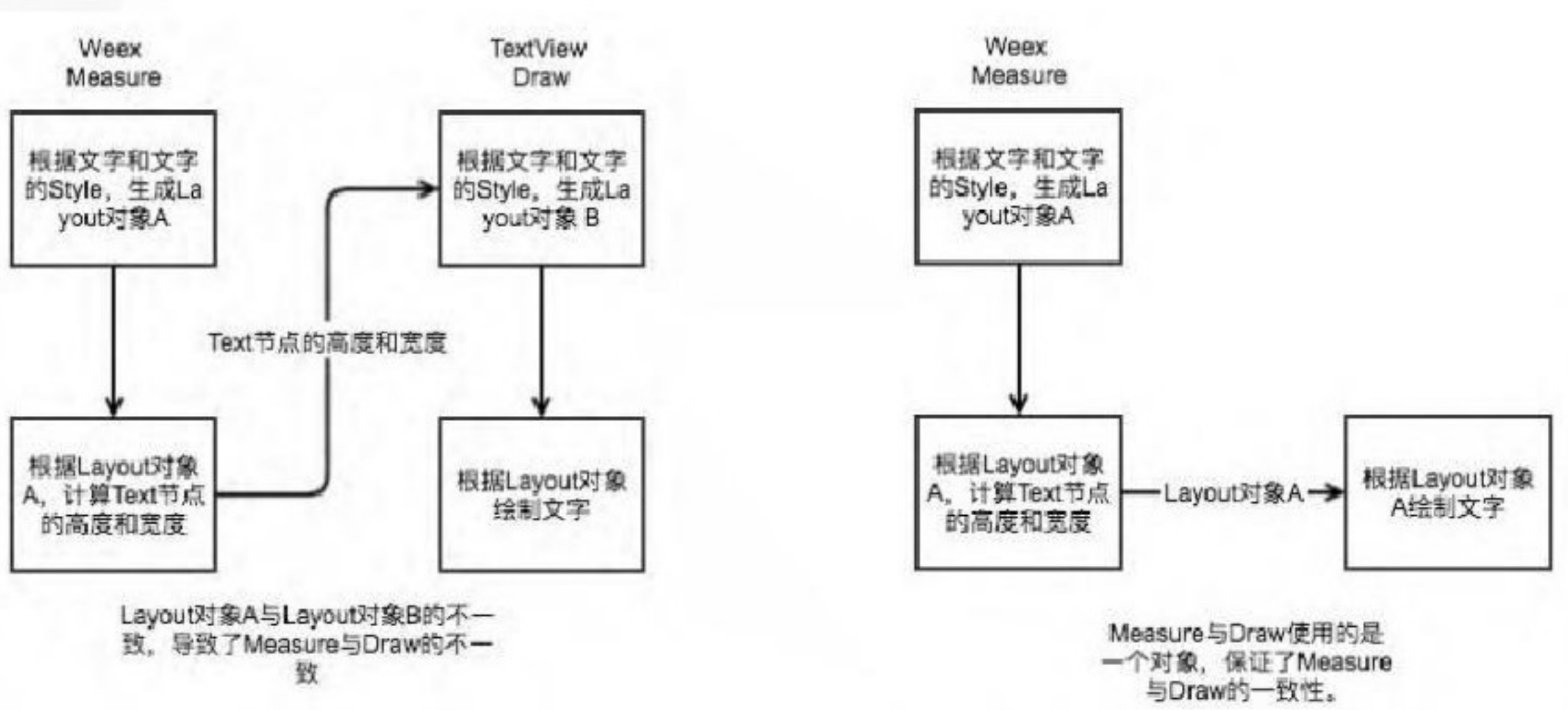
优化前(文字被截断)



优化后(正常渲染)



# Text优化



Measure和Draw使用了同一个对象, 避免了额外的对象创建, 预计提升10%左右