小程序性能优化

腾讯/微信事业群/开放平台基础部 赵景晨

个人经历

2018.4

加入腾讯

2018 - 至今

微信小程序基础框架与性能优化

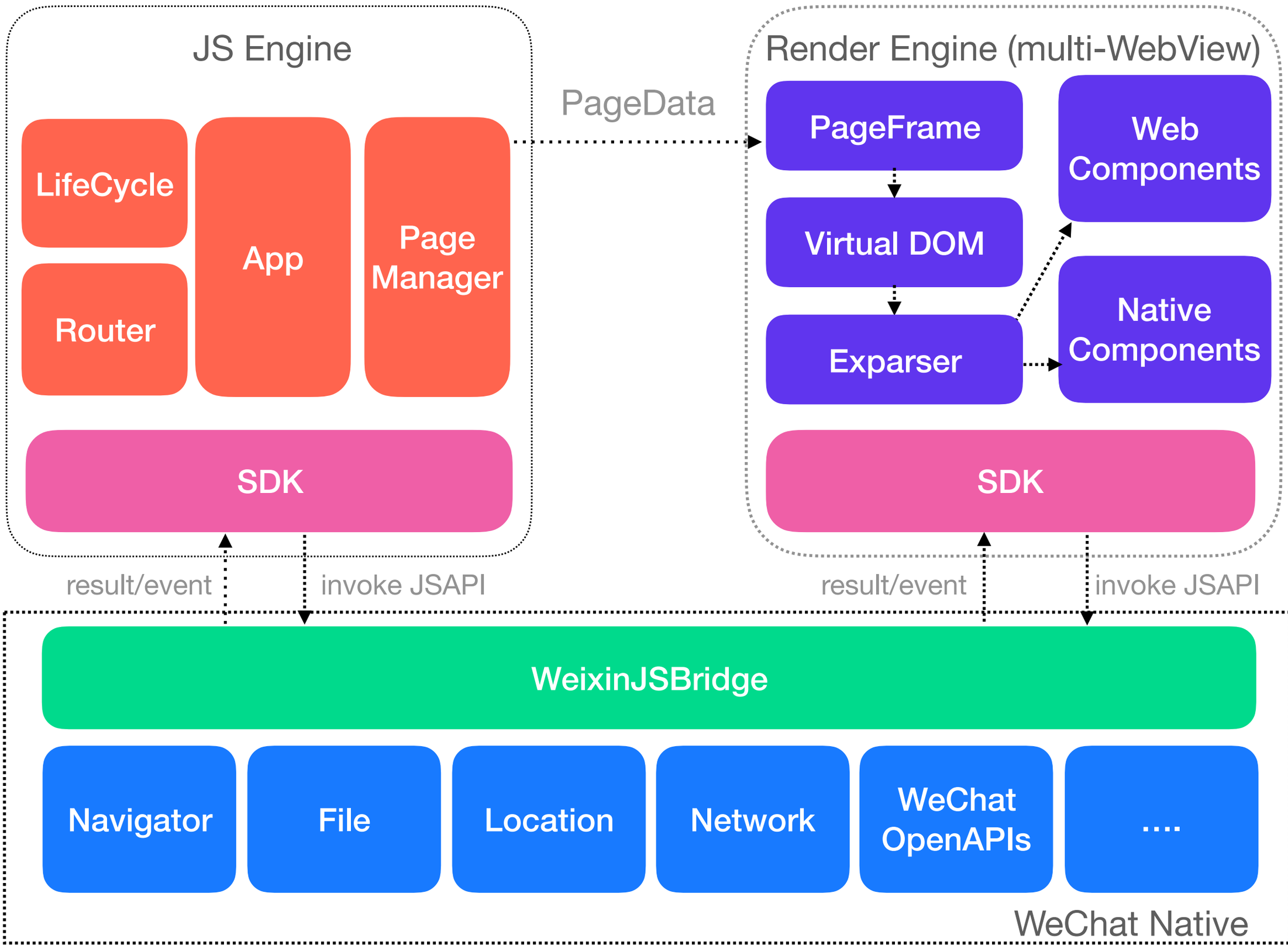
内容大纲

- 基础架构
- 数据传输
- 启动速度
- 优化经验

内容大纲

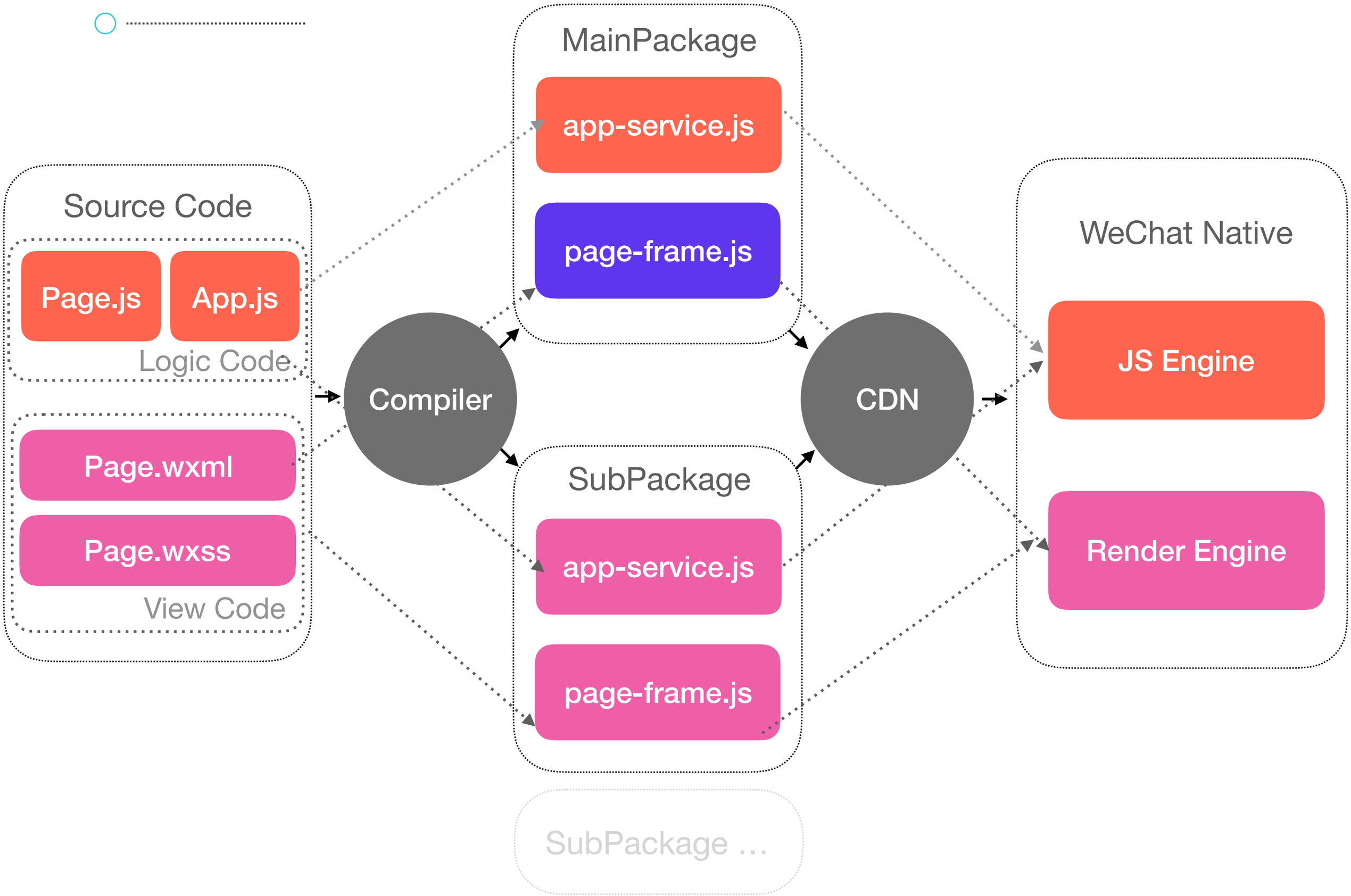
- **基础架构**
- 数据传输
- 启动速度
- 优化经验

基础架构 静态架构


JS Engine
LifeCycle
Router
App
Page Manager
SDK
PageData
Render Engine (multi-WebView)
PageFrame
Virtual DOM
Exparser
Web Components
Native Components
SDK
result/event
invoke JSAPI
result/event
invoke JSAPI
WeixinJSBridge
Navigator
File
Location
Network
WeChat OpenAPIs
....
WeChat Native

基础架构

动态架构

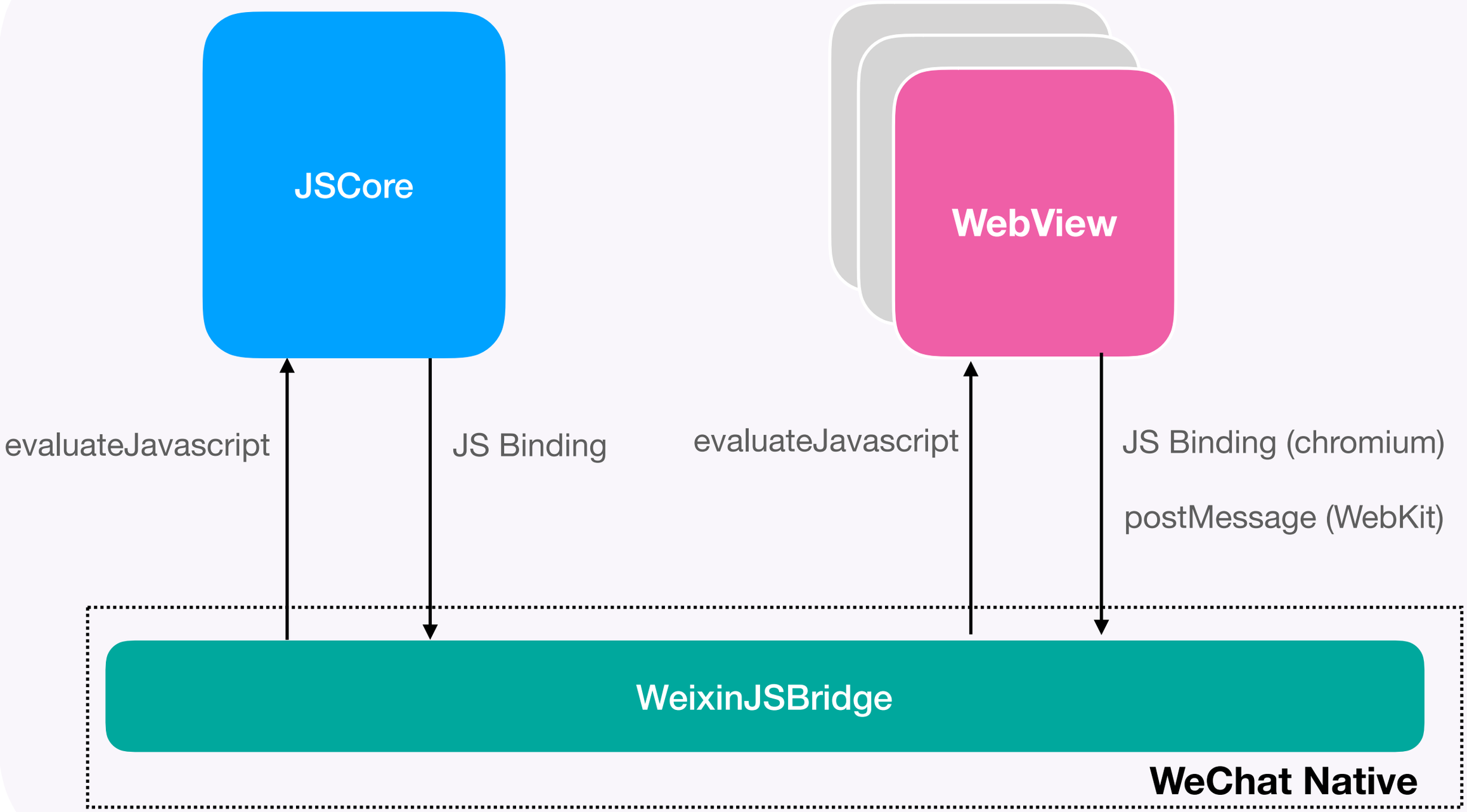

MainPackage
app-service.js
page-frame.js
Source Code
Page.js
App.js
Logic Code
Page.wxml
Page.wxss
View Code
Compiler
CDN
WeChat Native
JS Engine
Render Engine
SubPackage
app-service.js
page-frame.js
SubPackage ...

内容大纲

- 基础架构
- **数据传输**
- 启动速度
- 优化经验

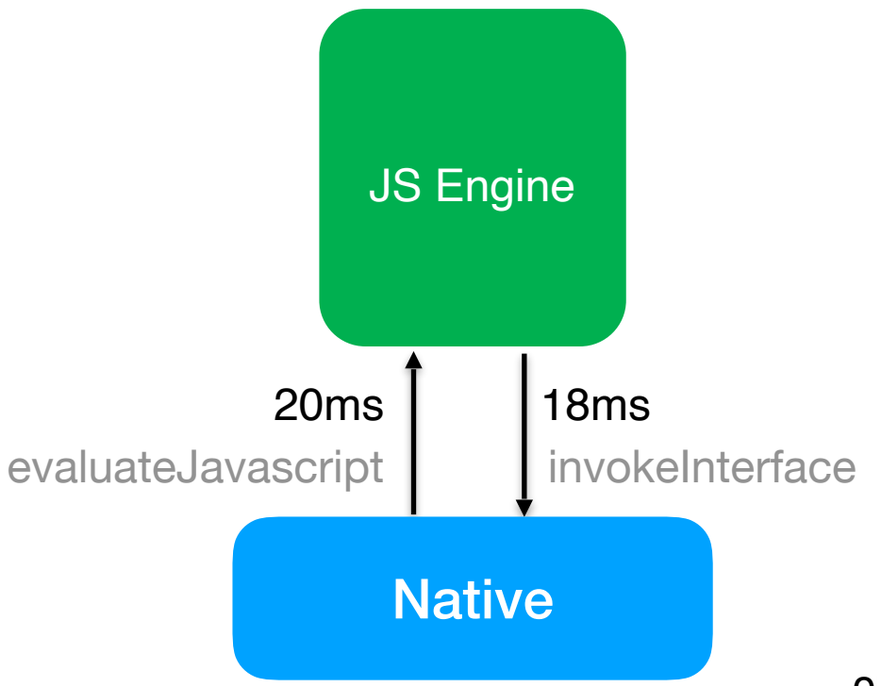
数据传输 基于字符串

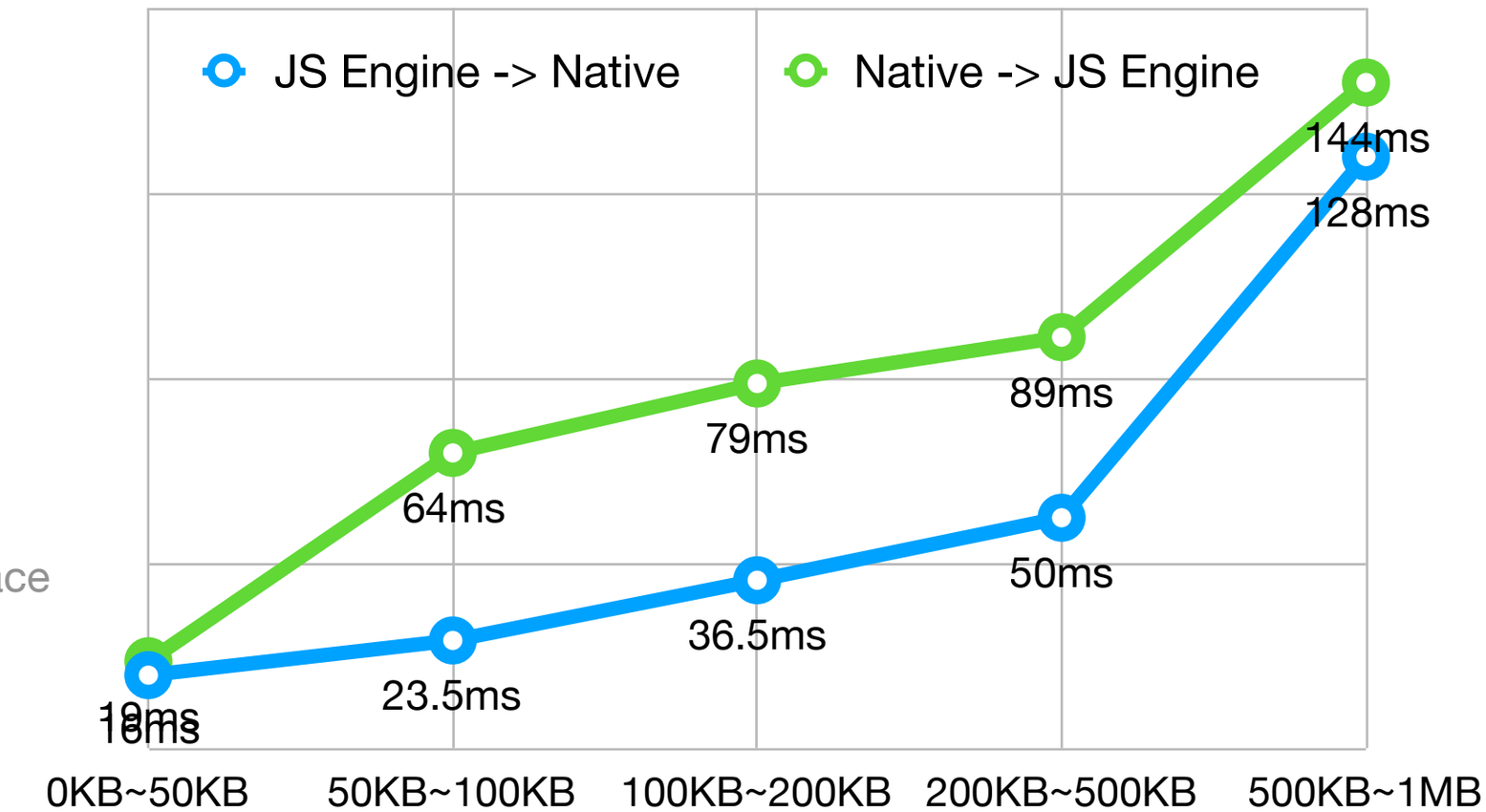
业界主流方案



数据传输

基于字符串传输的性能问题





现网不同大小数据的传输速度数据

现状

- 传输速度慢
- 不适合传输大数据

需求

- 实时录音识别要求低延迟
- 文件系统要求传输大数据

数据传输　如何解决？共享内存

可否不传输 String，而是传输内存数据呢

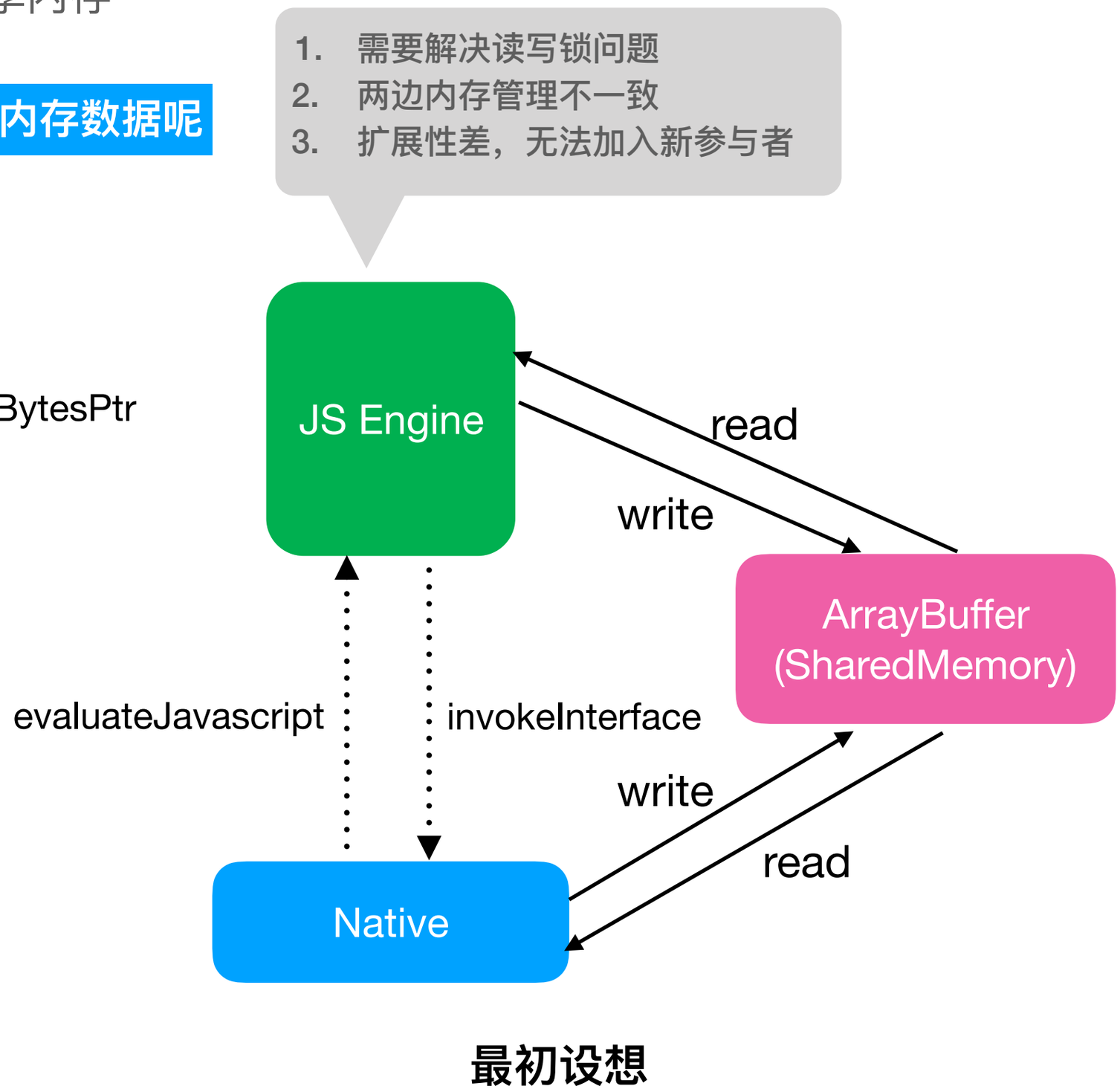
可行，基于两个基础 API

iOS：

iOS 10 及以上 JSObjectGetArrayBufferBytesPtr

Android：

V8 可基于一块内存创建 ArrayBuffer



最初设想

数据传输　NativeBuffer ——基于共享内存的数据传输

最后方案

1. 内存复制取回各自环境使用，解决两者内存管理、读写锁问题
2. NativeBuffer 即取即销毁 & 缓冲区大小限制处理异常情况
3. 缓冲池方式可接纳更多参与者，例如 Worker

JS Engine

1. copy data to NativeBuffer

2. return id #0

3.
```
wx.writeFile({
  bufferId: 0,
  ....
})
```

4. request NativeBuffer #0

5. copy data to Native

Native

NativeBuffer Pool

NativeBuffer#0

NativeBuffer#1

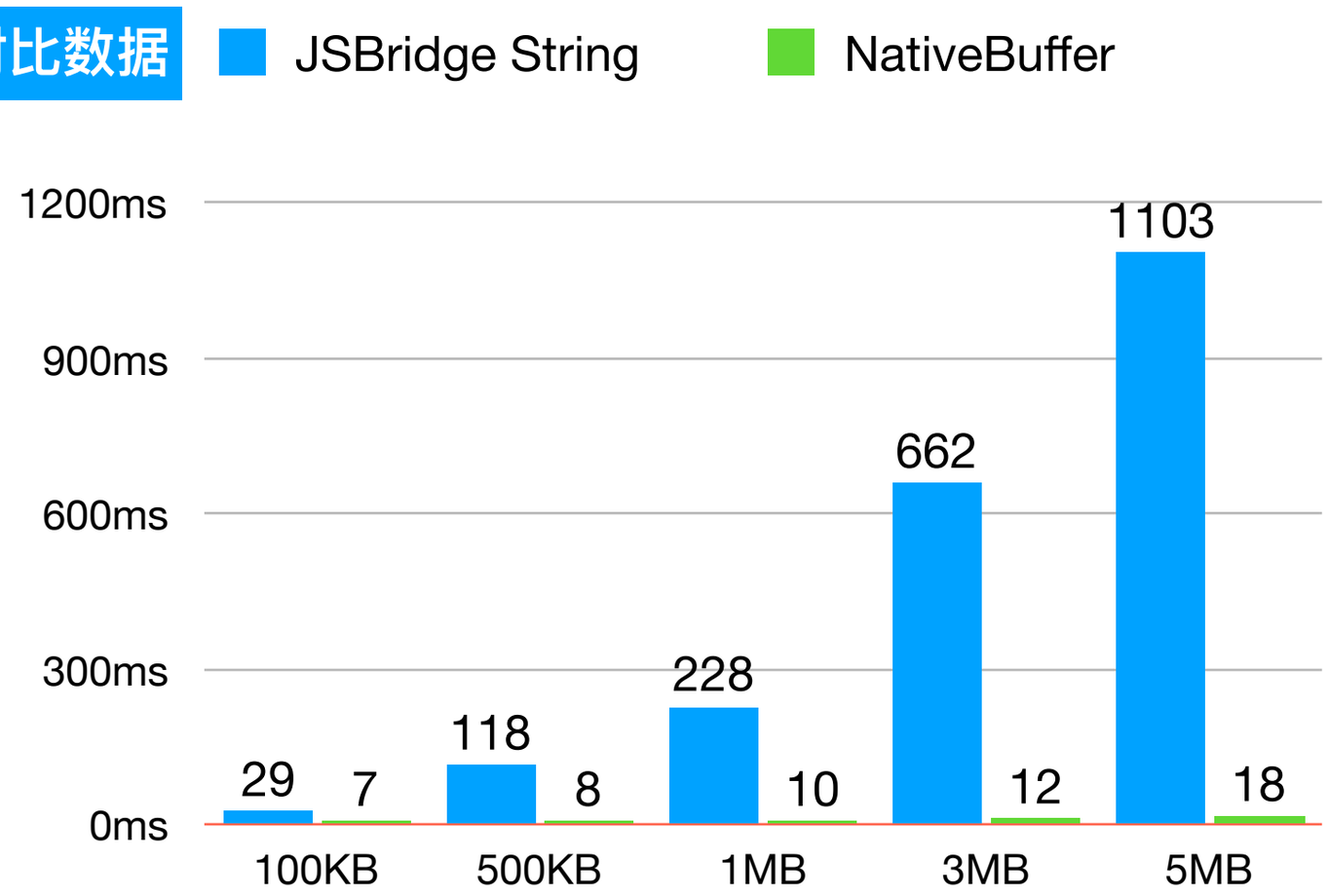
NativeBuffer#…

生产者 - 消费者模式

JSWorker　能加入更多参与者

数据传输 基于字符串 V.S. 基于共享内存

对比数据

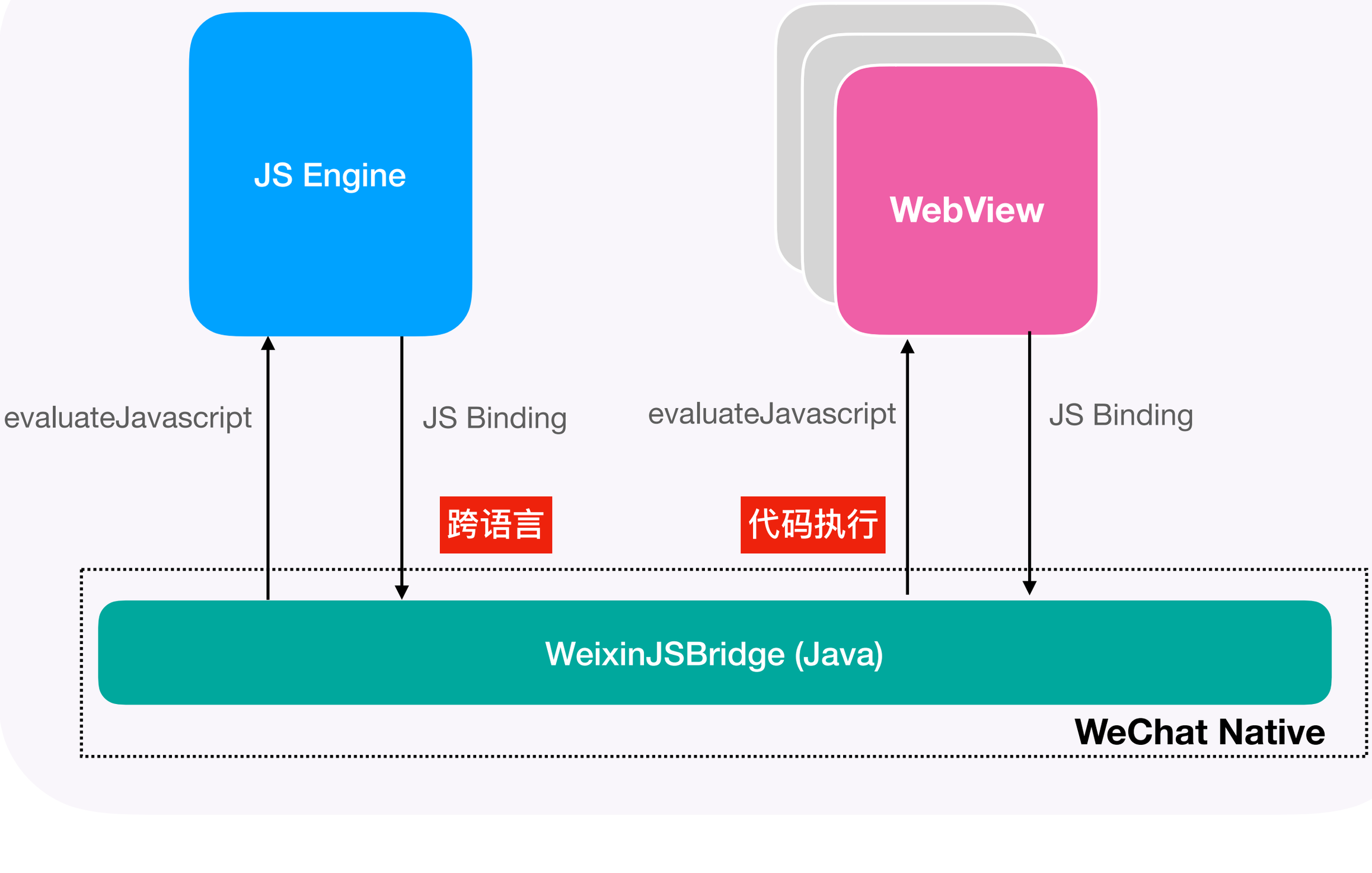


实际应用

- 小程序小游戏 WebSocket、Request
- 小程序小游戏文件系统
- 小程序 NFC 功能
- 小程序录音功能

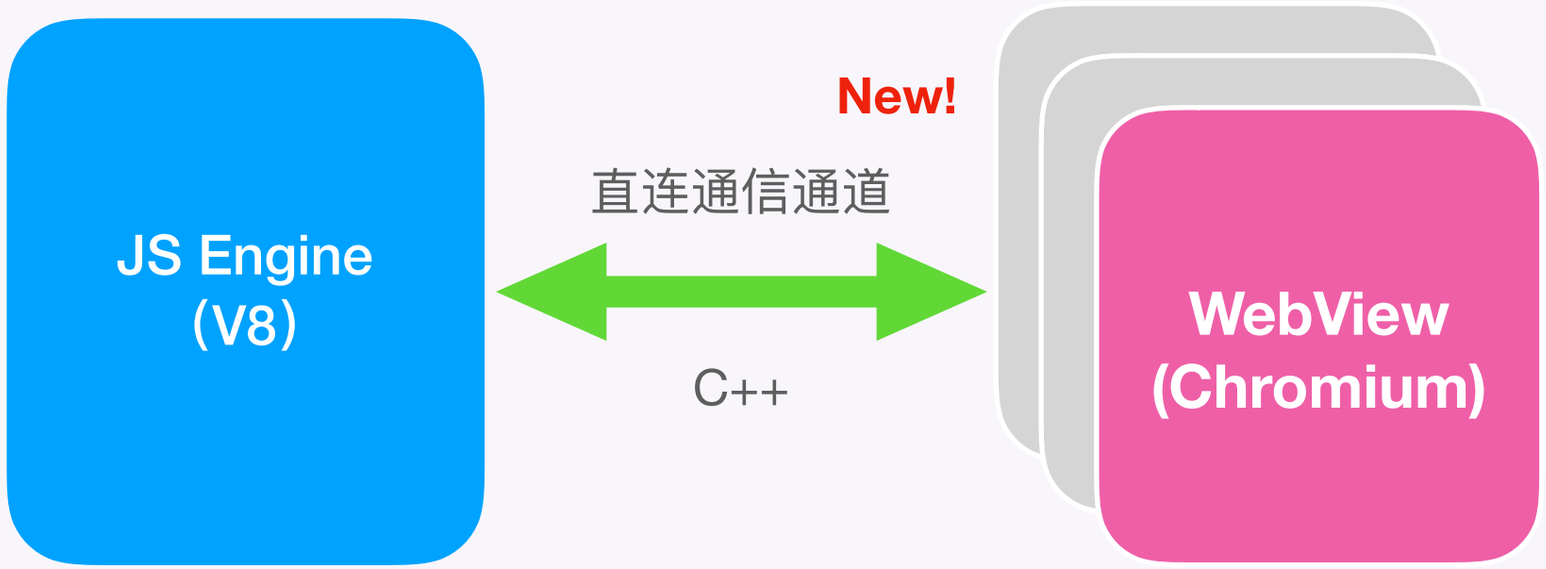
数据传输 JS Engine <=> WebView 通信

JS Engine 和 WebView 间通信需要客户端中转



数据传输　JS Engine <=> WebView 通信

C++ 直连！（目前仅支持安卓）



~~跨语言~~　　~~代码执行~~

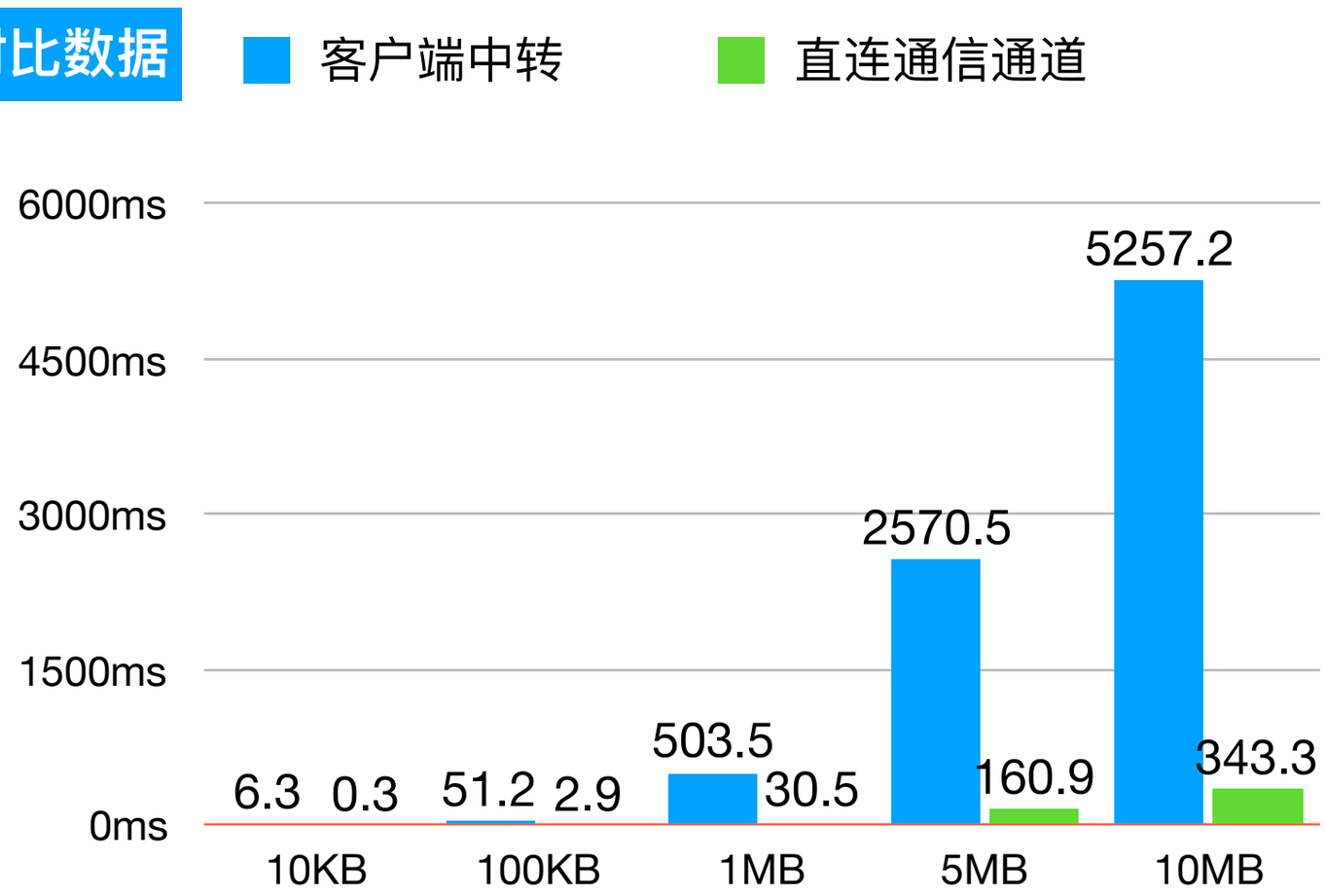
WeixinJSBridge (Java)

WeChat Native

数据传输

客户端中转 V.S. 直连通信通道

对比数据



主要应用

- setData
- 事件转发

内容大纲

- 基础架构
- 数据传输
- **启动速度**
- 优化经验

启动速度 现状

Android 启动速度令人望而却步

Android 平均 4.8 秒 iOS 平均 1.7 秒

启动速度 现状

小程序广告点击转化率较低

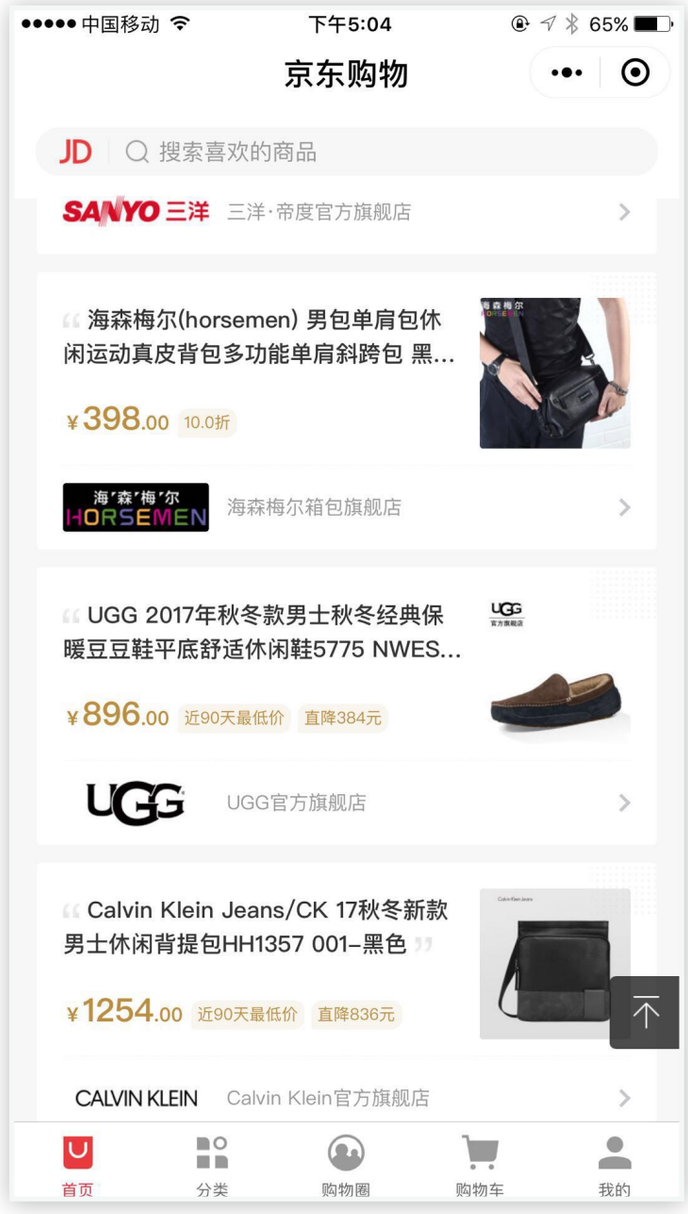
当前启动速度影响业务方发展

启动速度 现状

小程序业务发展要求更大代码包容量

更大代码包意味着更长首次启动耗时

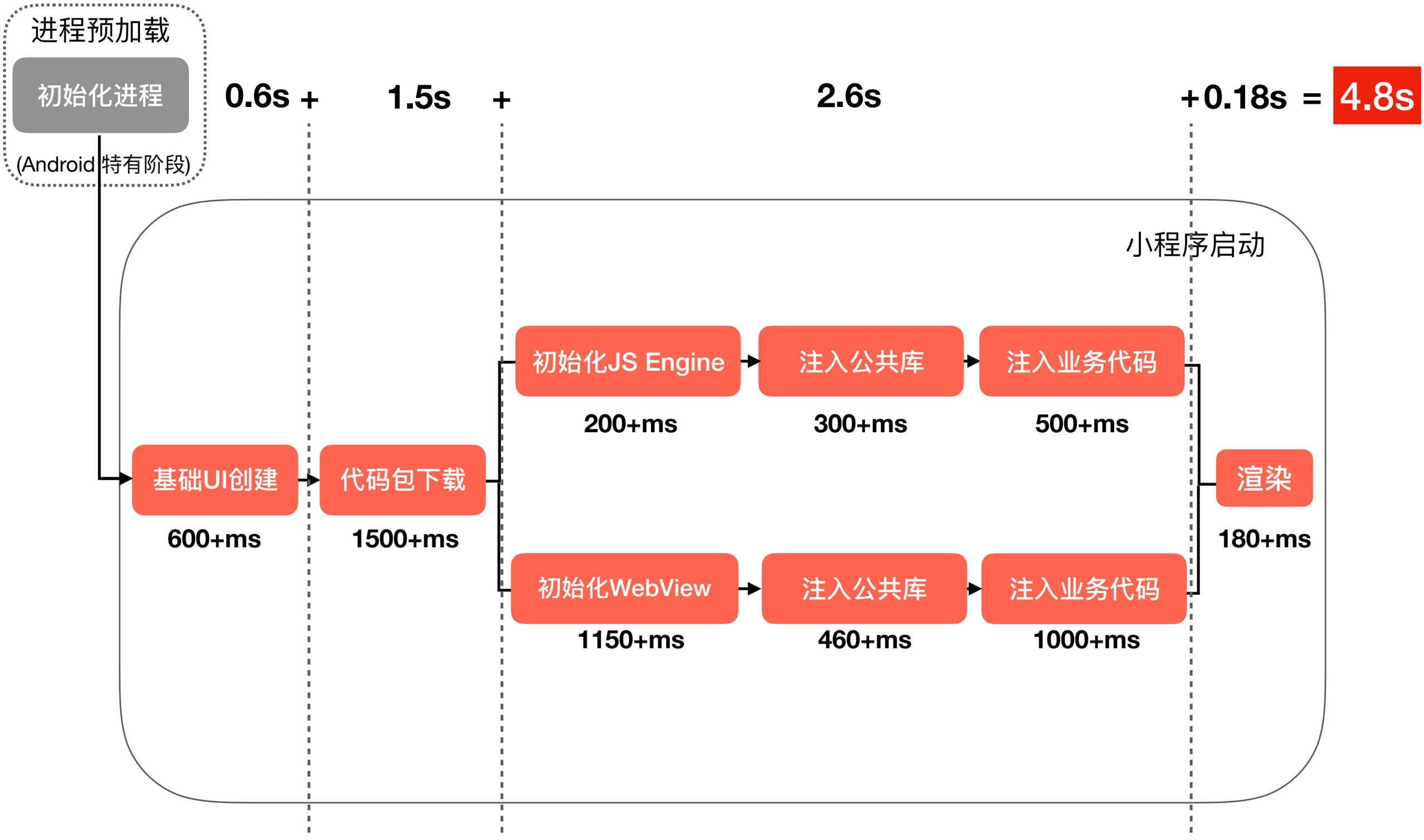
如何在启动性能和使用体验上进行权衡?



启动速度

问题分析：如何优化平台自身启动速度

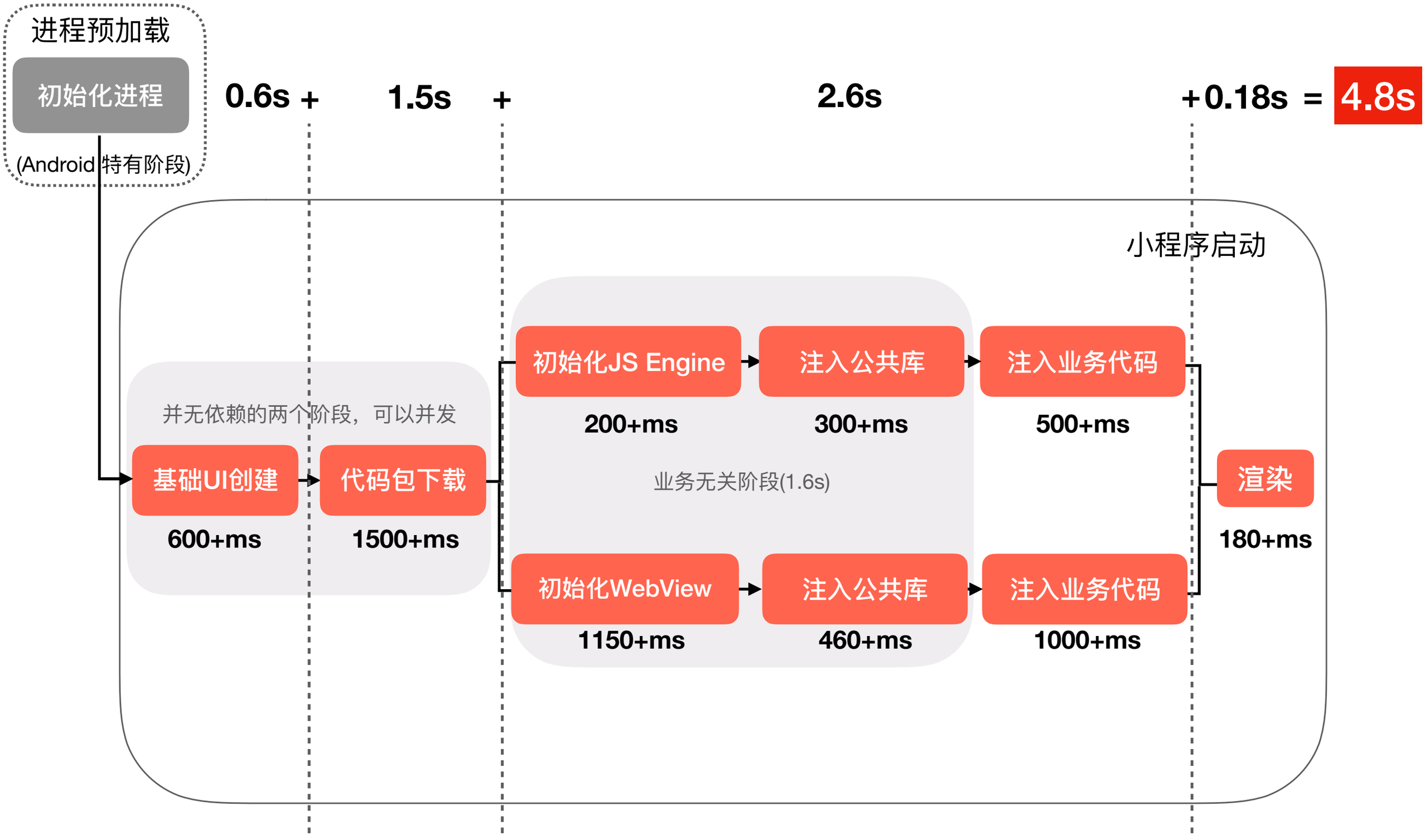
小程序启动分阶段耗时分析



启动速度

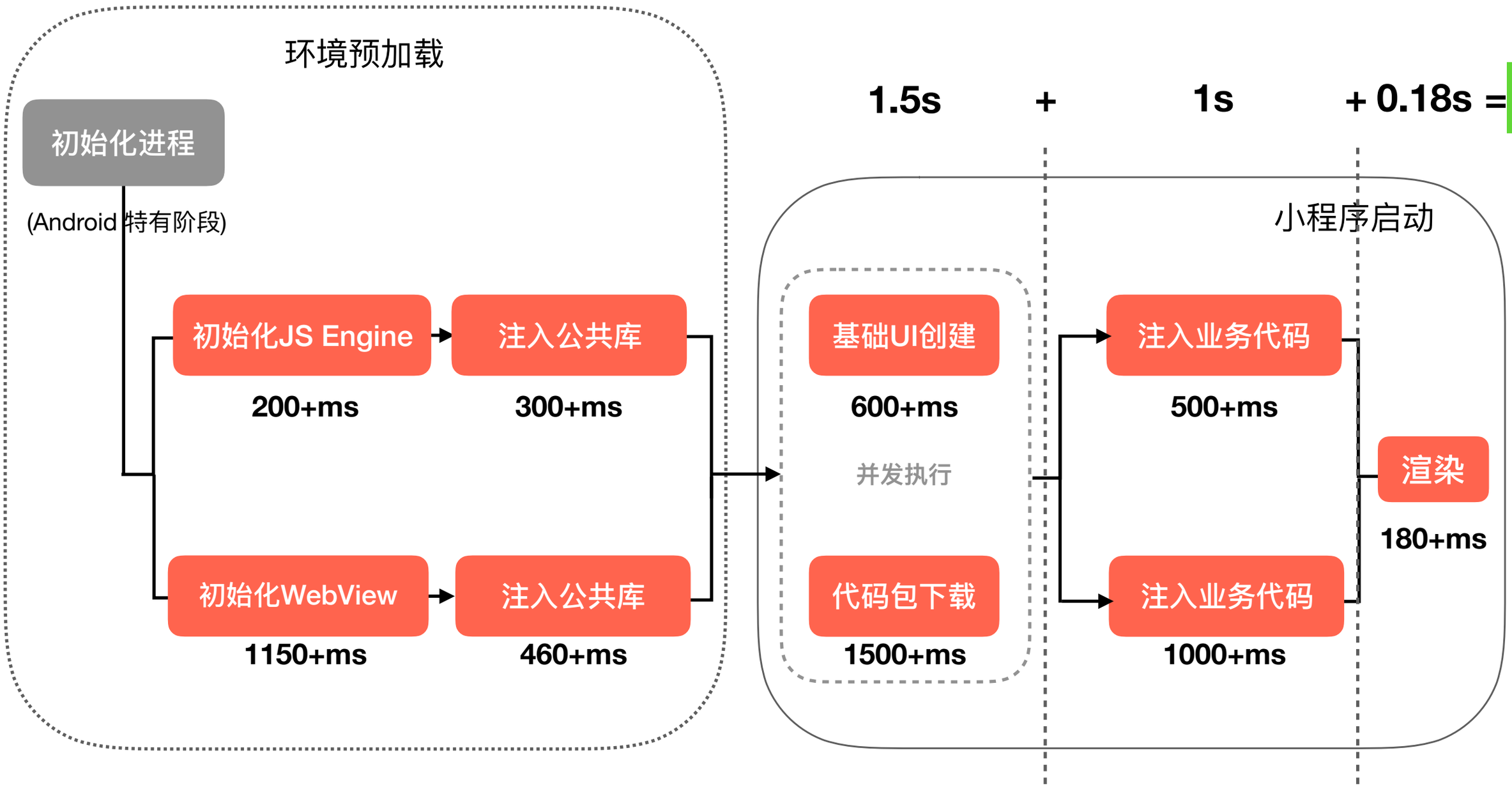
问题分析：如何优化平台自身启动速度

小程序启动分阶段耗时分析



启动速度

解决方案：**业务无关阶段预加载、无依赖阶段并发**



启动速度

解决方案：业务无关阶段预加载、无依赖阶段并发

难点：需要大幅修改小程序现有架构

- 历史原因 PageFrame、前端公共库耦合业务信息内容

PageFrame

```
<html lang="zh-CN">
  <head>…</head>
  <body>
    <WXML/>
  </body>
</html>
```

小程序 WXML 代码

PageFrame 耦合业务代码

前端公共库

```
// 与业务无关的逻辑
function init() {
  // ...
}

// 业务相关分支逻辑
function initByApp() {
  // ...
}
```

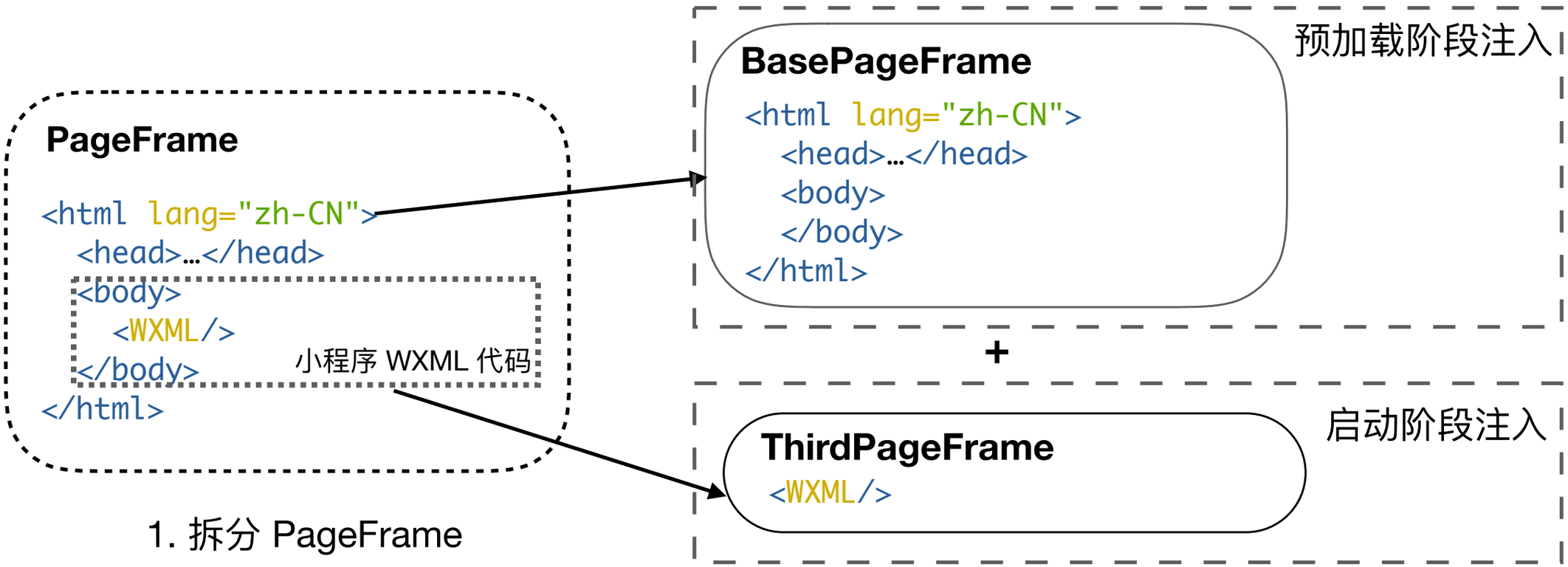
公共库耦合业务分支逻辑

预加载阶段没有业务相关信息

启动速度

解决方案：业务无关阶段预加载、无依赖阶段并发

怎么解决



1. 拆分 PageFrame

启动速度

解决方案：业务无关阶段预加载、无依赖阶段并发

怎么解决

前端公共库

```
// 与业务无关的逻辑
function init() {
  // ...
}

// 业务相关分支逻辑
function initByAppType() {
  // ...
}
```

2. 拆分公共库公共/业务逻辑

预加载阶段执行

公共逻辑

```
// 与业务无关的逻辑
function init() {
  // ...
}
```

+

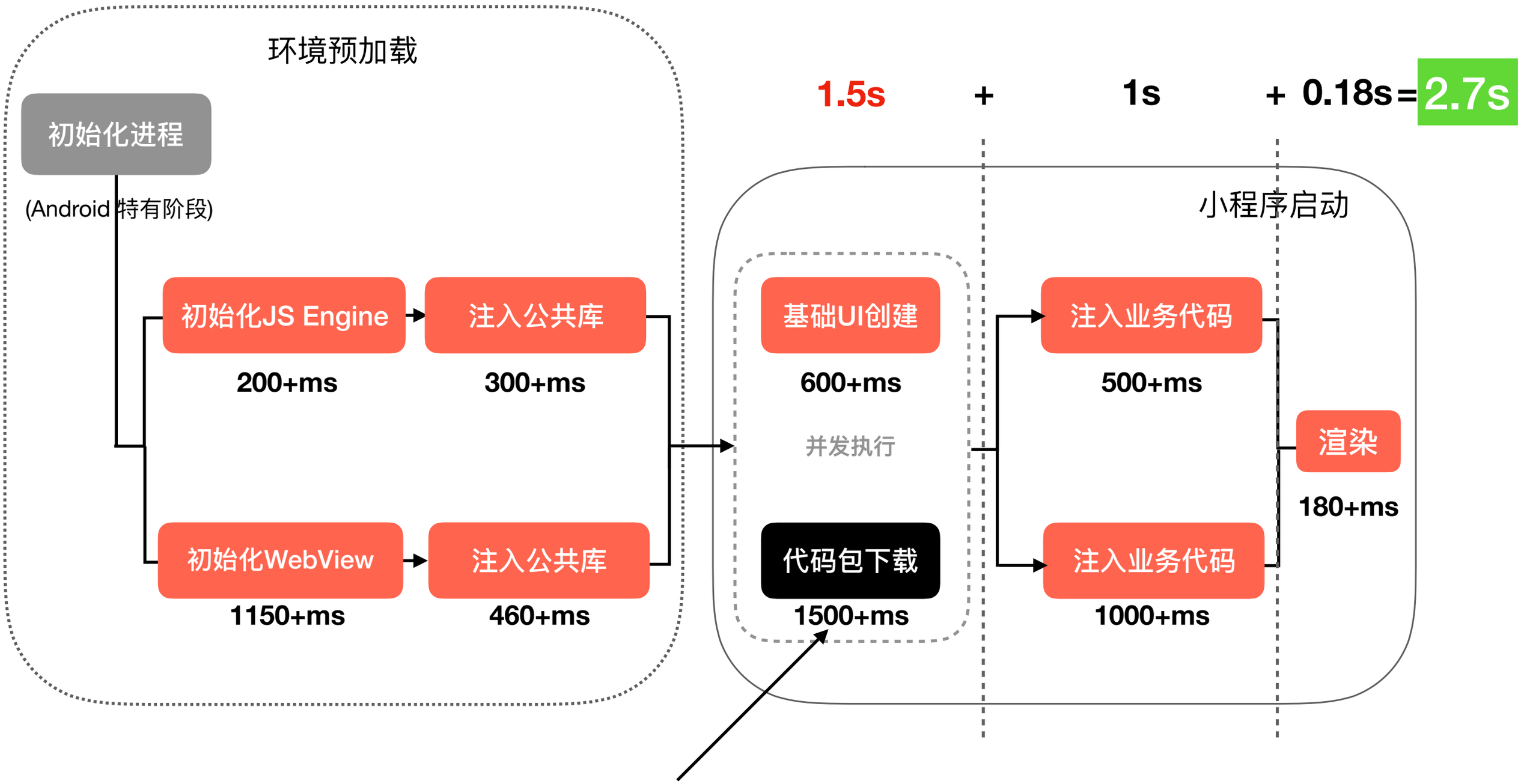
启动阶段执行

业务逻辑

```
// 业务相关分支逻辑
function initByApp() {
  // ...
}
```

启动速度

问题分析：如何减少代码包下载对启动耗时的影响

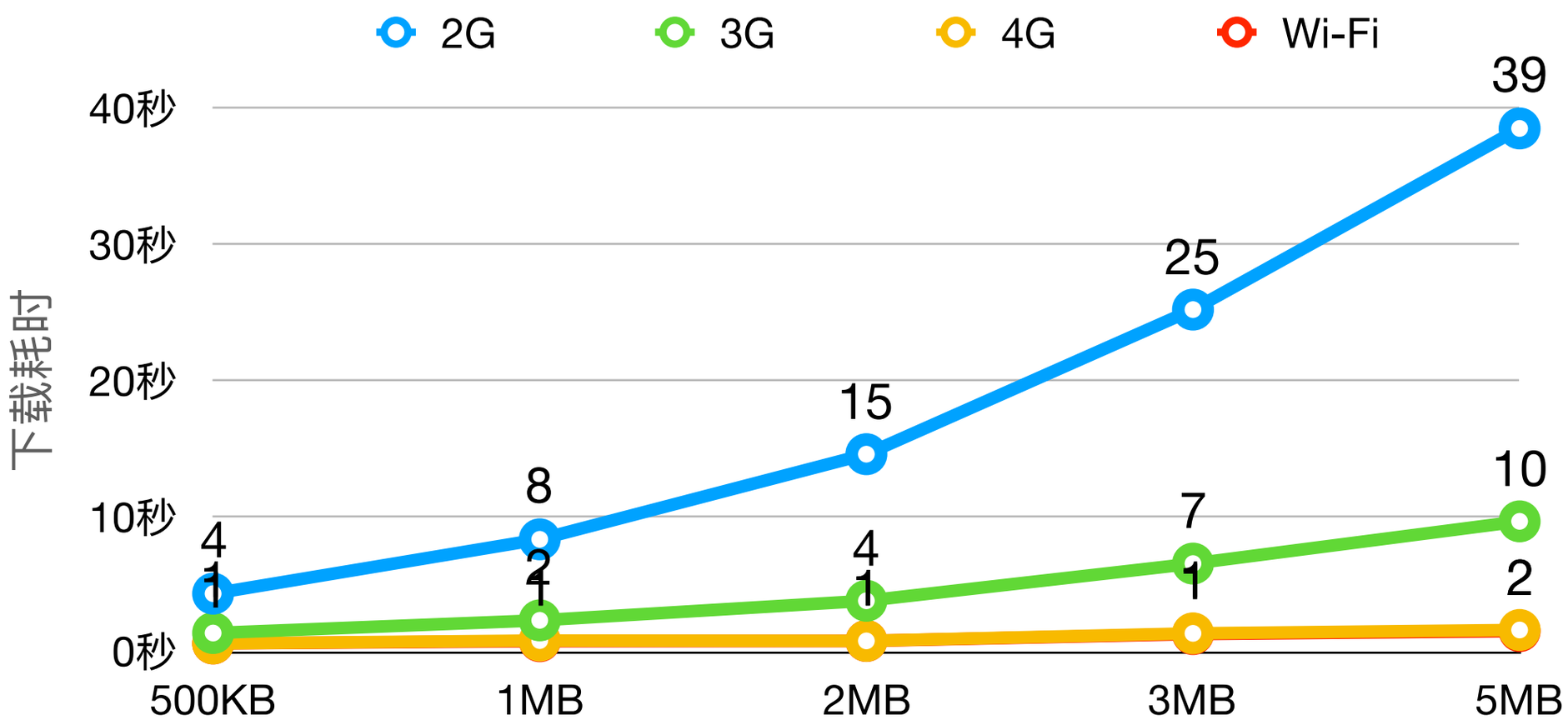


如何减少代码包大小对启动耗时的影响?

启动速度

问题分析：如何减少代码包下载对启动耗时的影响

不同代码包的下载速度



数据结论

- 代码包越大、下载耗时越长
- 2G、3G 网络更受代码包大小影响

启动速度

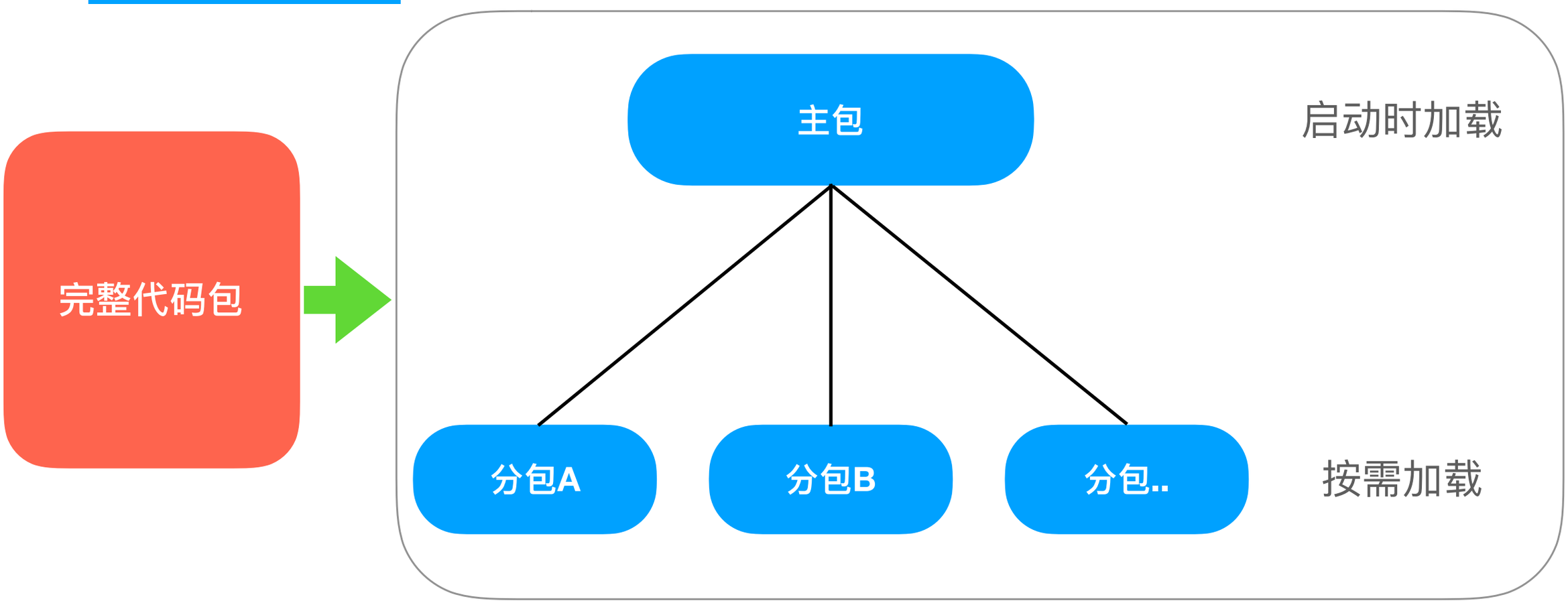
问题分析：如何减少代码包下载对启动耗时的影响

优化方案

- **方案一：使用 zstd 压缩算法替代 Gzip**
  - 对比 Gzip 优化约 19%（线上数据）
  - 不能解决下载耗时随包大小增加问题
- **方案二：使用 QUIC 协议**
  - 对比 HTTPS 优化 50%（实验数据）
  - 不能解决下载耗时随包大小增加问题
- **方案三：分包加载**
  - 限制启动包大小，解决耗时随包大小增加问题
  - 没有提升下载速度

启动速度　解决方案：分包加载

什么是分包加载?

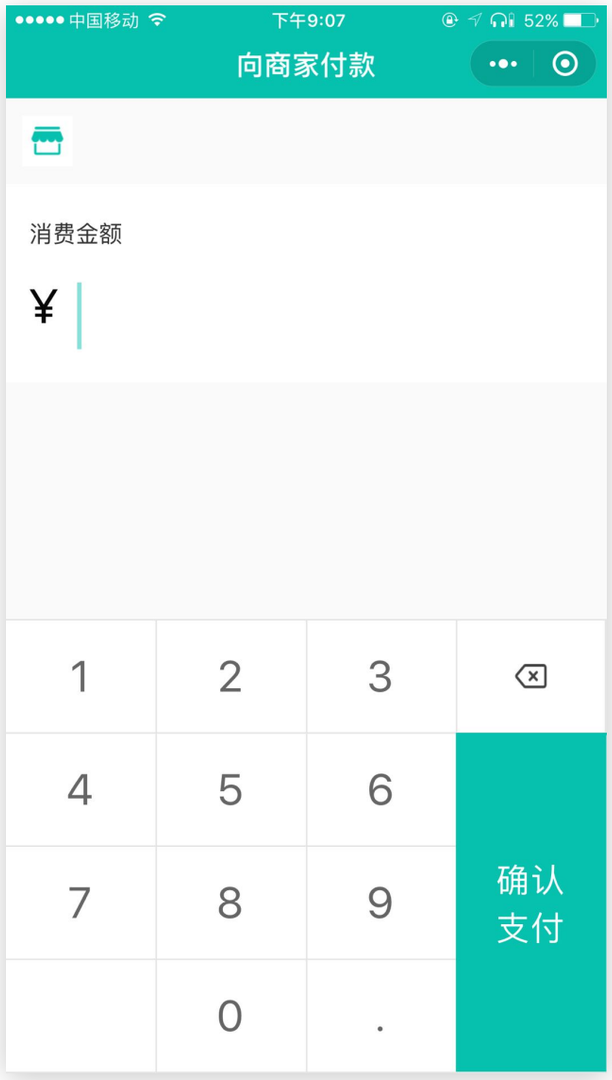


- 将小程序部分页面放入分包，主包只保留最常用页面
- 启动时只加载主包，使用时按需下载分包

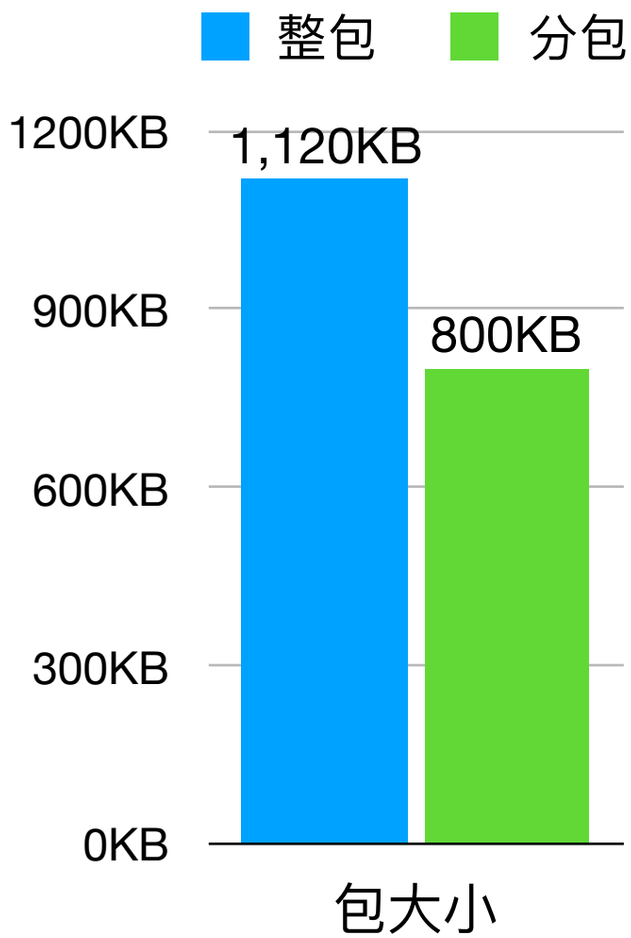
降低启动时代码包下载和代码注入耗时

启动速度　解决方案：分包加载

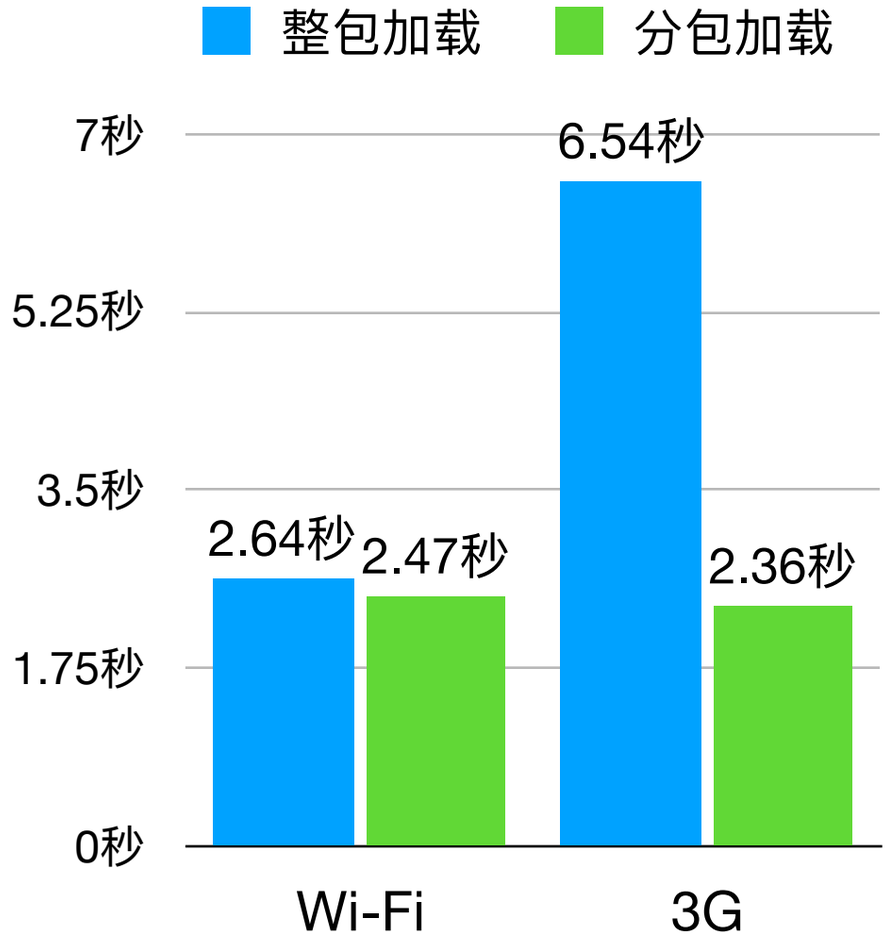
分包加载效果



美团扫码支付使用分包



包大小变化



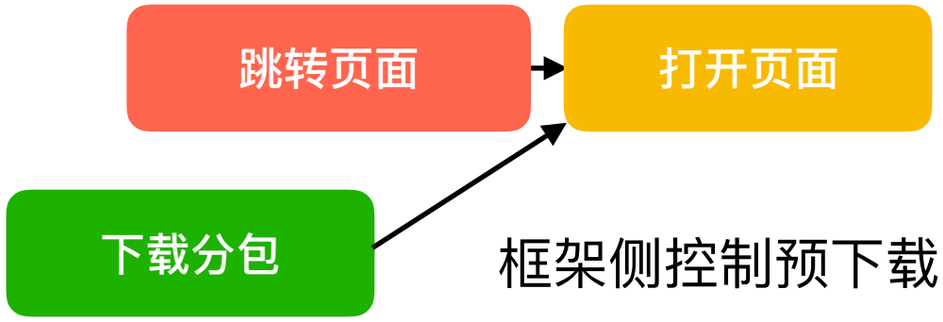
首次启动耗时对比

启动速度

解决方案：分包预下载

问题分析：首次打开分包页面时，因下载注入分包造成跳转产生延迟

解决方案：根据开发者配置，提前下载用户可能使用的分包

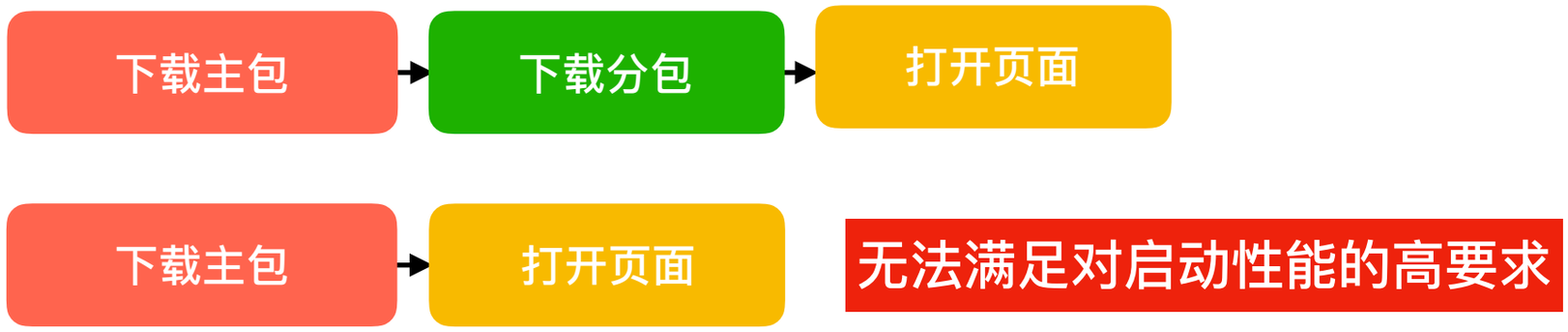


框架侧控制预下载的分包数量、大小、时机，防止开发者滥用

启动速度　解决方案：独立分包

问题分析：业务复杂，分包之后主包大小依然很大。
页面功能独立性强，对启动速度要求高

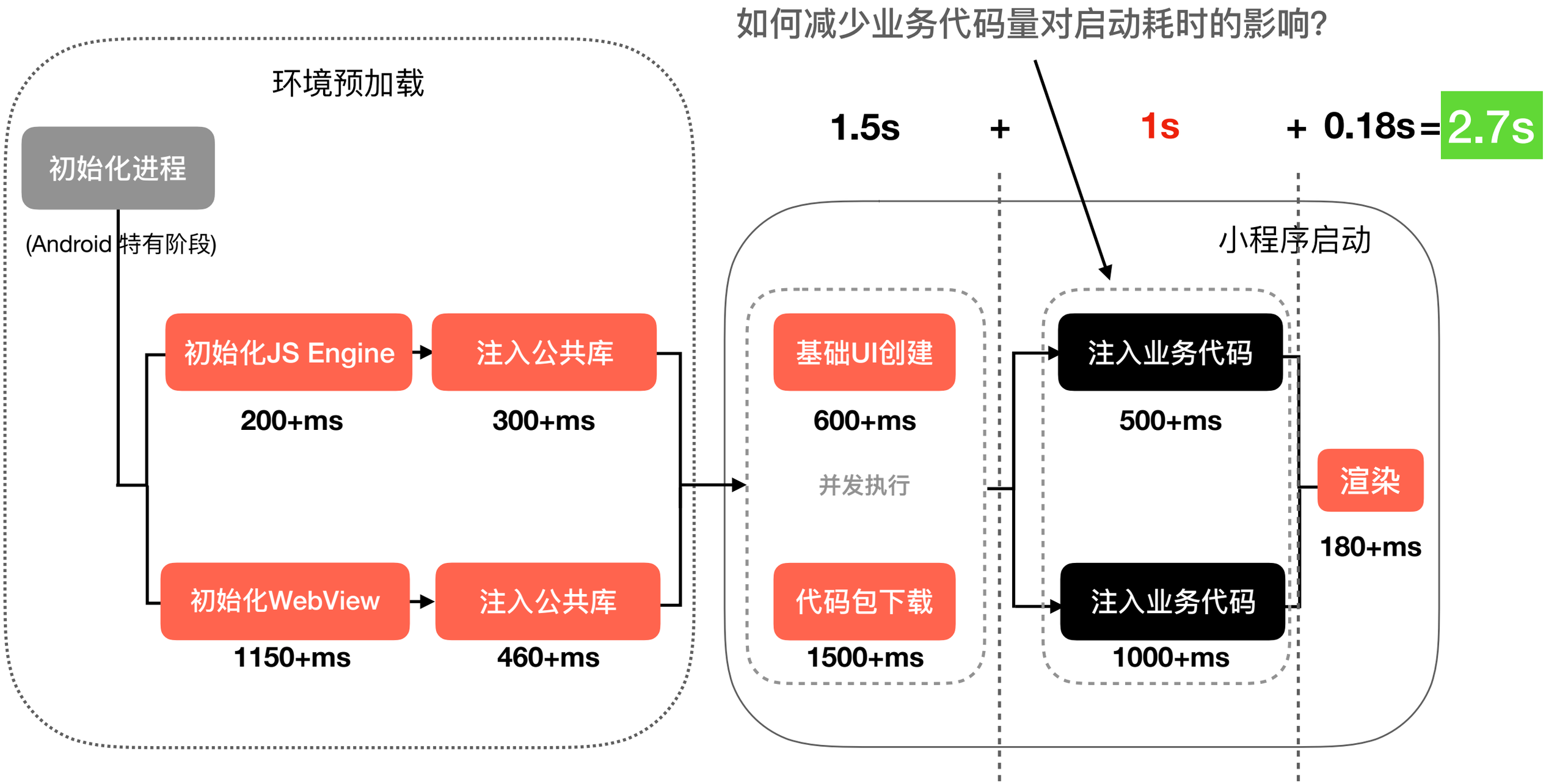
- 广告落地页
- 活动推广页
- 分享商品详情页
- 支付页面



解决方案：不依赖主包，可以独立下载和运行的特殊分包类型

启动速度

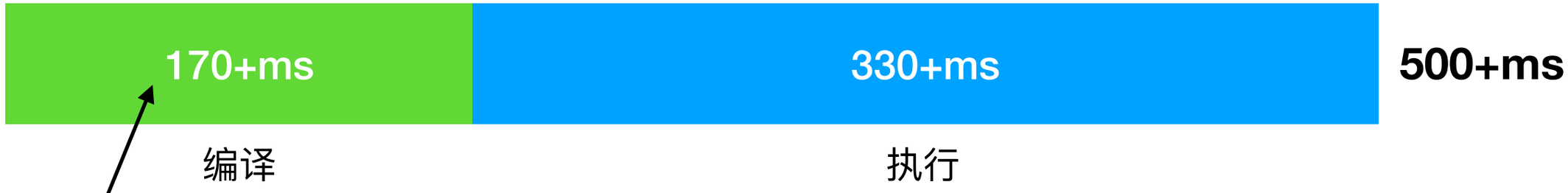
问题分析：如何减少代码注入对启动耗时的影响



启动速度　问题分析：代码注入分阶段耗时分析

代码注入耗时分析

自选股开发者代码 (314KB) 注入耗时分布



启动速度　解决方案：V8 Code Caching

V8 Code Caching

自选股 app-service.js(314KB) 注入耗时分布

Monday, July 27, 2015

Code caching

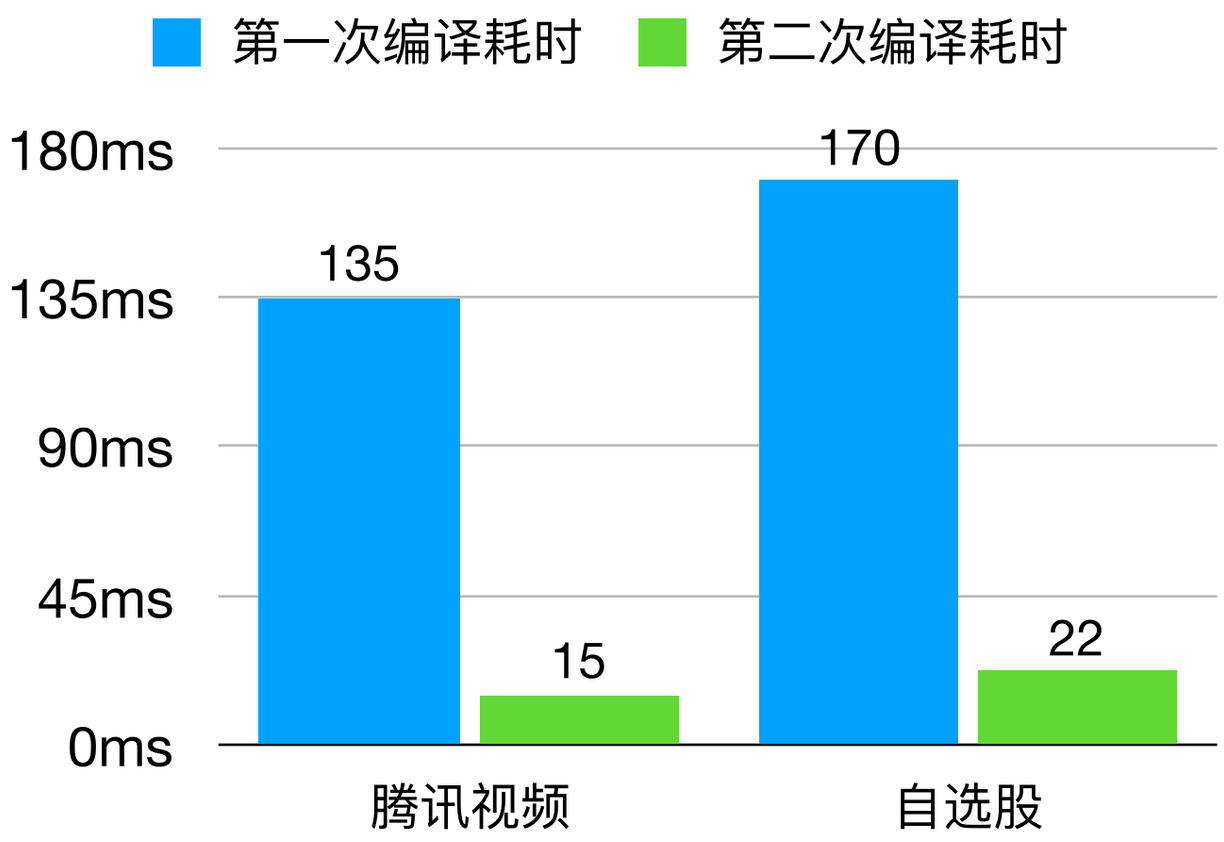
V8 uses just-in-time compilation (JIT) to execute Javascript code. This means that immediately prior to running a script, it has to be parsed and compiled - which can cause considerable overhead. As we announced recently, code caching is a technique that lessens this overhead. When a script is compiled for the first time, cache data is produced and stored. The next time V8 needs to compile the same script, even in a different V8 instance, it can use the cache data to recreate the compilation result instead of compiling from scratch. As a result the script is executed much sooner.

可行，V8 Code Caching 支持将代码结果缓存下来 √

启动速度　解决方案：V8 Code Caching

优化效果

自选股 app-service.js(314KB) 注入耗时分布（命中Cache）



启动速度 整体优化效果

优化前

视频长度：5 秒

优化后

视频长度：3 秒

测试机器：Nexus 5（13 年低端机）

启动速度 小结

优化 平台自身问题

- 运行环境预加载，流程并发解耦

满足 业务发展需求

- 分包加载、分包预下载、独立分包、V8 Code Caching，在不影响启动速度前提下满足业务发展

指引 帮助业务优化

- 提供分析工具、优化指引文档给到开发者

内容大纲

- 基础架构
- 数据传输
- 启动速度
- 优化经验

优化经验 提前设计

架构设计时需要重点关注性能

- 性能是功能和体验的保障，必须提前进行规划
- 架构定型后再进行性能优化的成本将十分高昂
- 为项目设置 performance budget

优化经验　发现性能问题

数据监控 + 性能测试

- 完善的数据上报、分析、监控链路
- 版本发布时需要进行必要的性能测试
- 通过 benchmark 或 profiling 发现性能瓶颈

优化经验　关注性能需求

收集和评估反馈

- 业务发展对性能提出新的挑战
- 业务发展可能引起新的性能问题

优化经验 解决性能问题

核心方法论

- 能提前做的提前做
- 提升流程的并行度
- 控制流程每一步的时间
    - 善用缓存
    - 按需加载
    - 选择更高效的算法和实现
- 在性能和其他因素（如灵活性、开发成本等）之间进行权衡
- 注意性能优化手段可能产生的副作用

THANKS