



UC-Weex优化实践之路

阿里巴巴移动事业群 龚攀峰

2017.09.02

移动端开发痛点

发版节奏缓慢

客户端发版
流程重、成本高

版本覆盖缓慢

客户端动态能力
弱覆盖慢

多端研发成本高

同一业务多端发
带来高成本

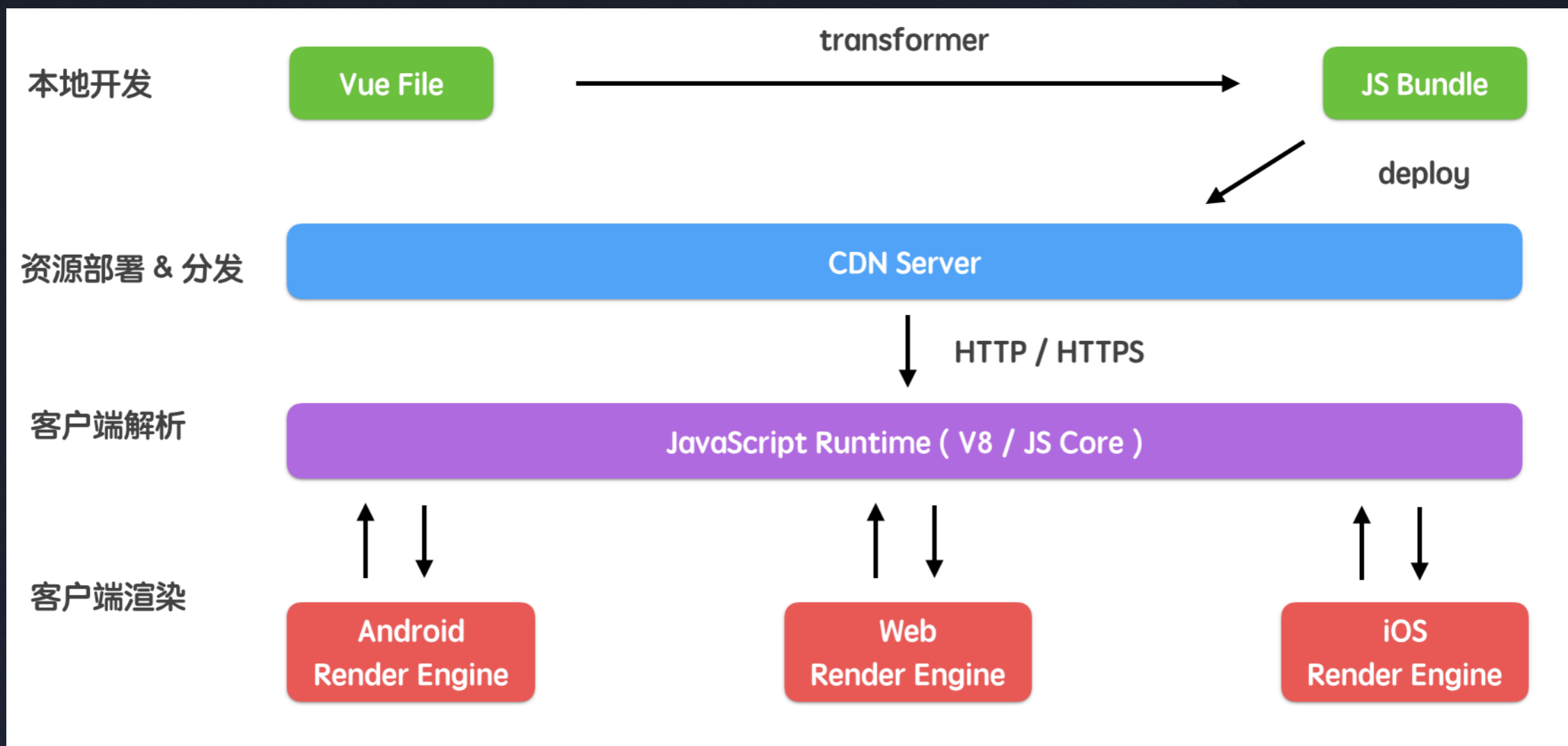
期望方向

- 统一技术栈，降低开发成本
- H5发布更新能力，原生native体验

技术选型

评估点	React Native	Weex (1.0 & 2.0)
	相似点	
JS - Native 通信机制	JS Bridge	
布局系统	Flexbox 子集	
开发语言	JavaScript	
调试手段	通过 Chrome 断点 / 输出日志, 支持 live reload	
发布方式	服务端下发 JS bundle 到客户端	
	差异点	
DSL	React & JSX	1.0: Base on Vue、2.0: Powerd by Vue
推崇框架	Redux & Immutable	1.0: 无、2.0: Vuex
环境配置	自带打包工具	基于 webpack 的 cli
框架稳定性	频繁更新, 向后可能不兼容	小步更新, 相对稳定
开发成本	一次学习, 多端编写	一次编写, 多端运行
学习成本	一揽子技术方案, 学习路线陡峭	贴近 Web 技术栈语法, 上手简单

Weex技术架构图



核心方向是美好的，现状能力却无法支撑

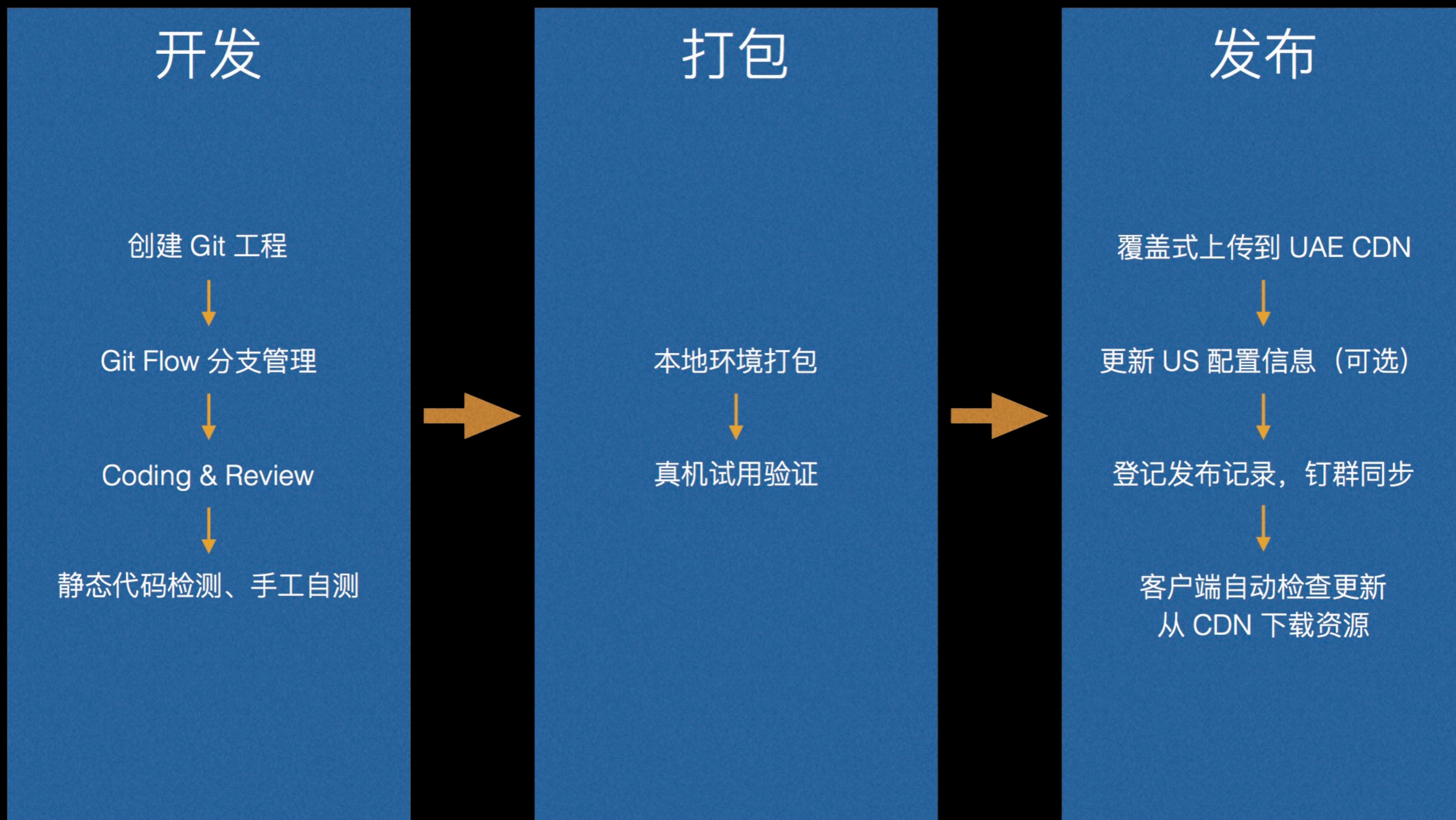
拓宽Weex能力边界，完善生态

● 配套工程化解决方案

● 核心性能持续优化

● 业务服务能力提升

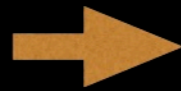
Weex JS 资源发布链路现状



问题与缺失

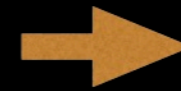
开发

- * 缺少更多样的质量保障手段
如：单元测试、自动化测试



打包

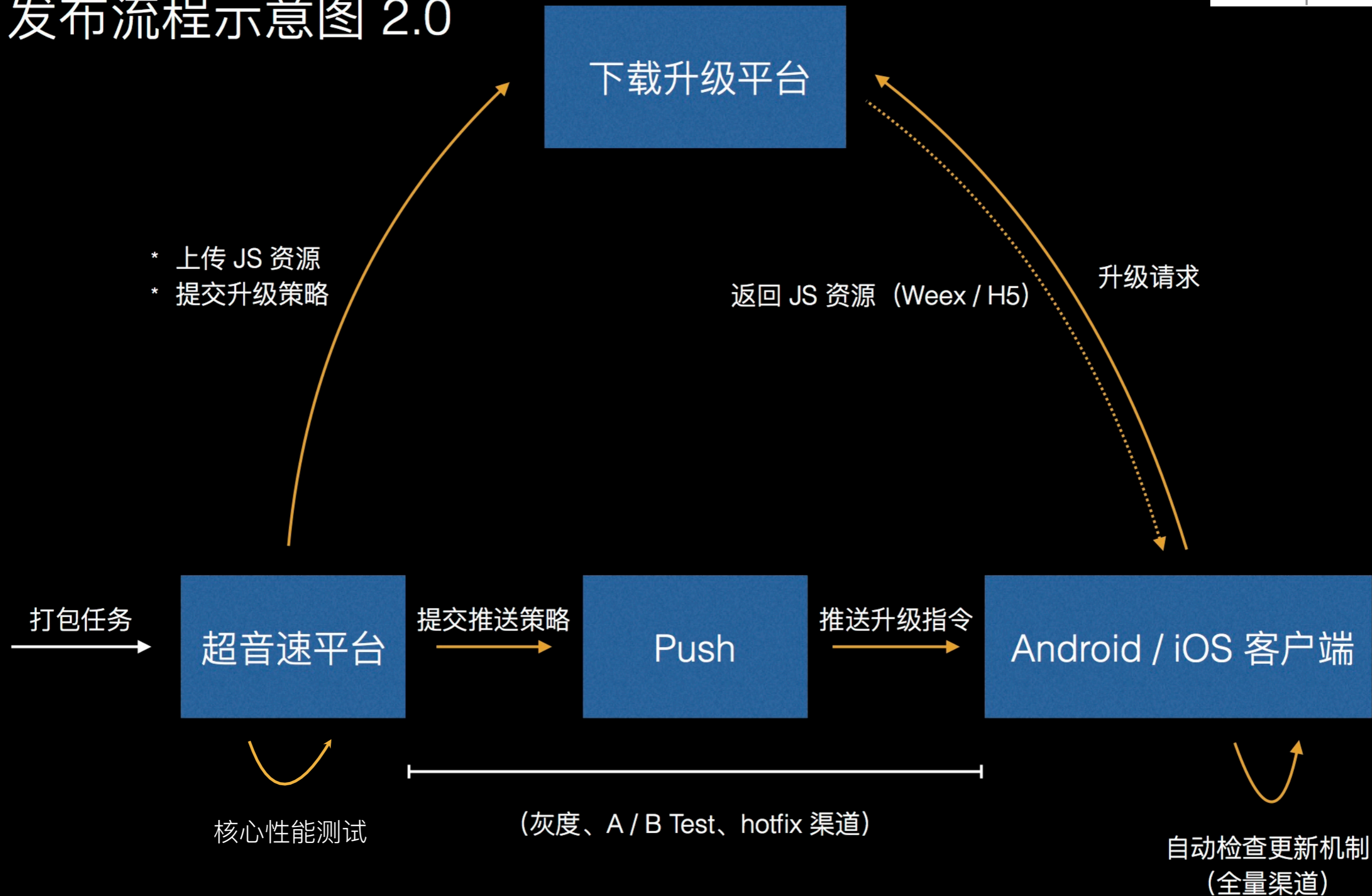
- * 打包约束相对较少，无权限控制，无事前校验，无打包记录，易诱发安全问题



发布

- * 人肉登记发布记录，操作成本高，追溯问题效率低下
- * **灰度成本高，且不具备画像能力**
- * **不具备控量发布能力**
- * 不具备 AB Test 能力
- * 回滚手段简单（重新覆盖 UAE 资源）
- * **发布过程缺乏监控与度量**

发布流程示意图 2.0



数据情况

- 以某个垂直类项目为例，开发到上线时间提升1倍以上
- 上线前：内部3个业务，支撑维护困难
- 上线后：服务内部10个以上垂直业务，周发布近30次
- 日覆盖率85%+

配套工程化解决方案

核心性能持续优化

业务服务能力提升

Framework瘦身之路

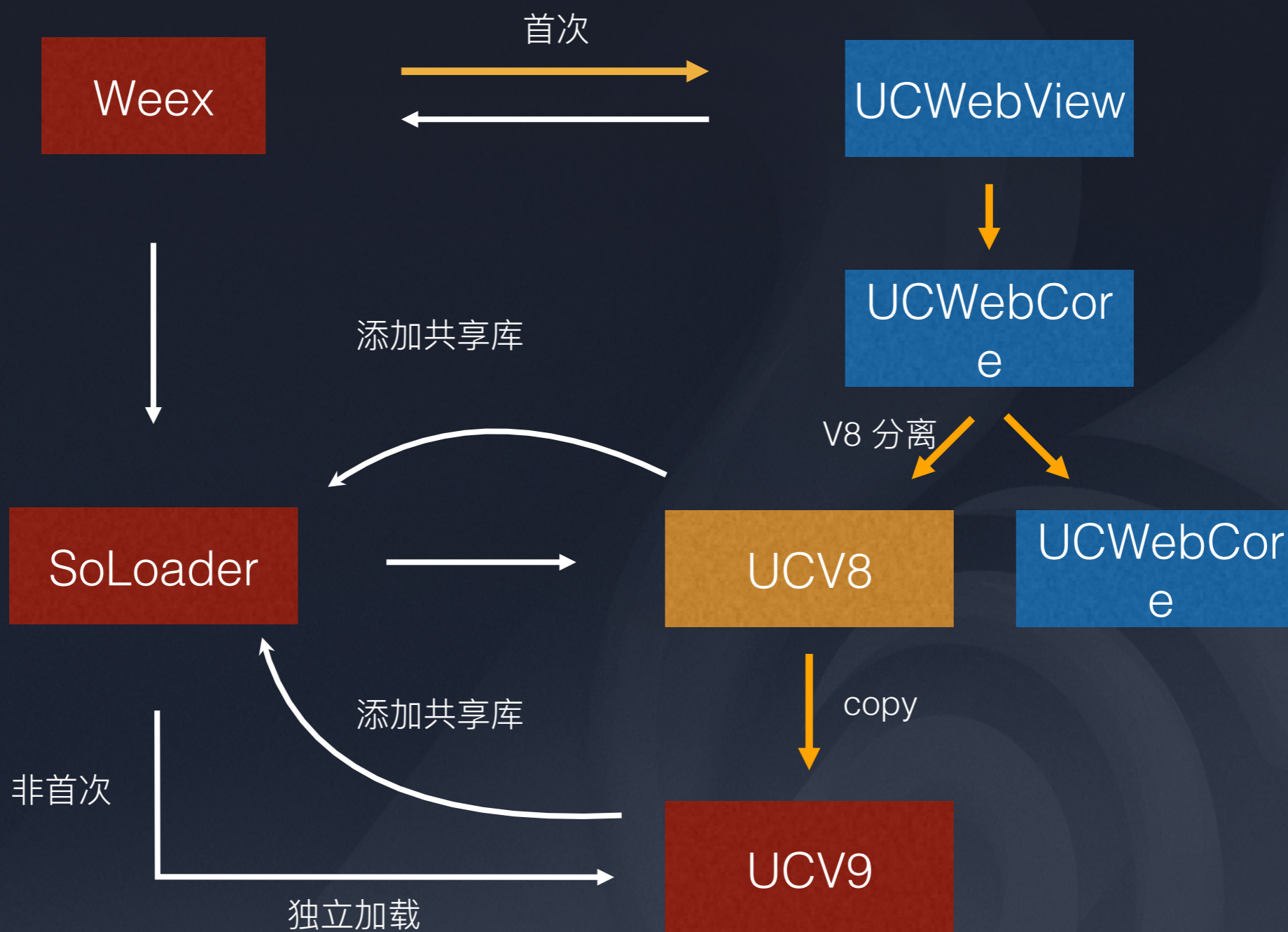
常规
裁剪

- 针对V4、V7进行裁剪

V8共享
技术

- 抛弃自带引擎
- 分离UC浏览器V8进行共享

V8共享方案核心流程



结果

- 接入Size从3M+降至500K
- 依赖UC内核初始化，不利于维护且影响首次启动时长
- 非UC系产品无法使用，方案推广有限

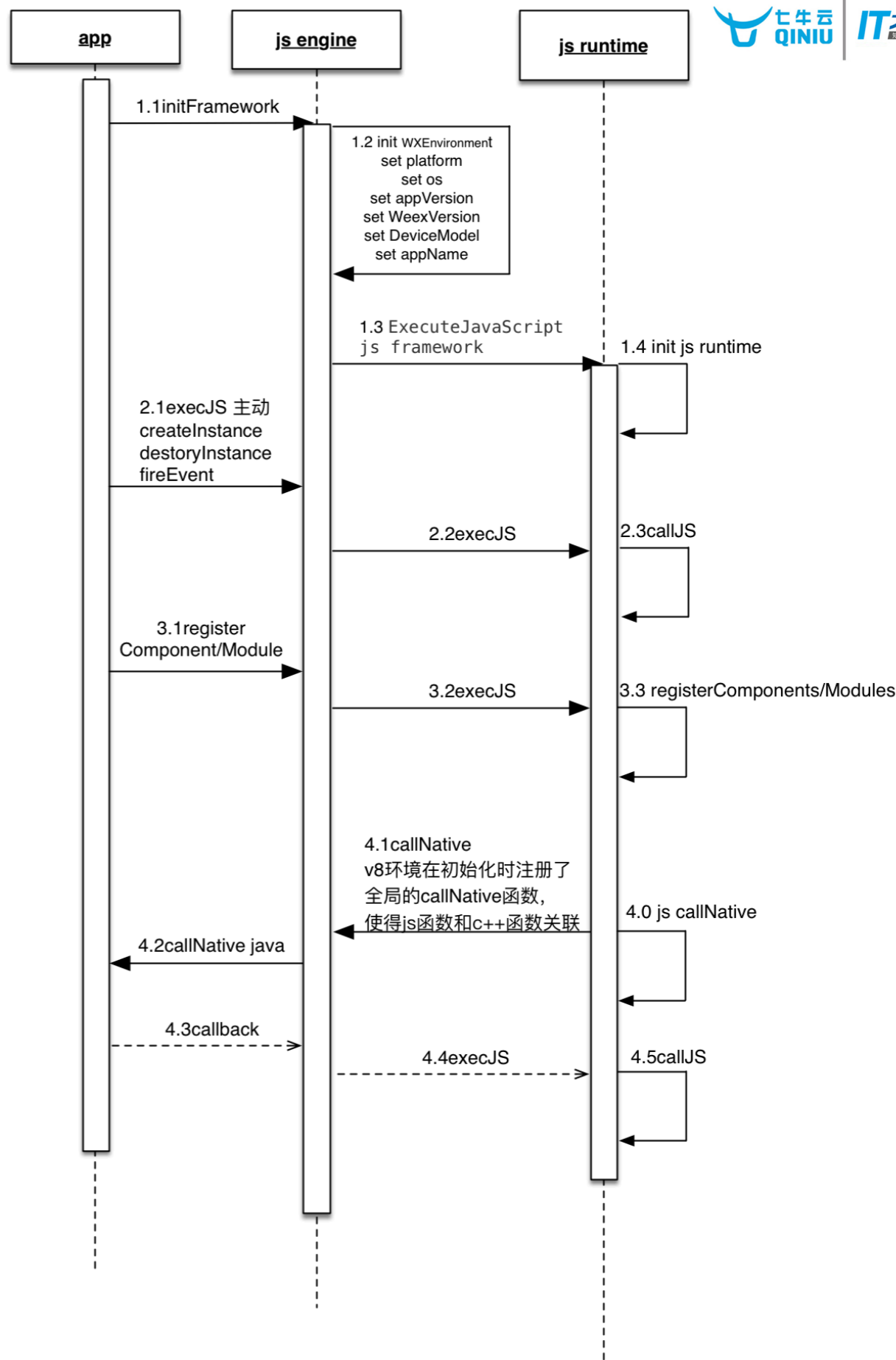
系统WebView作为Js引擎的可行性

限制

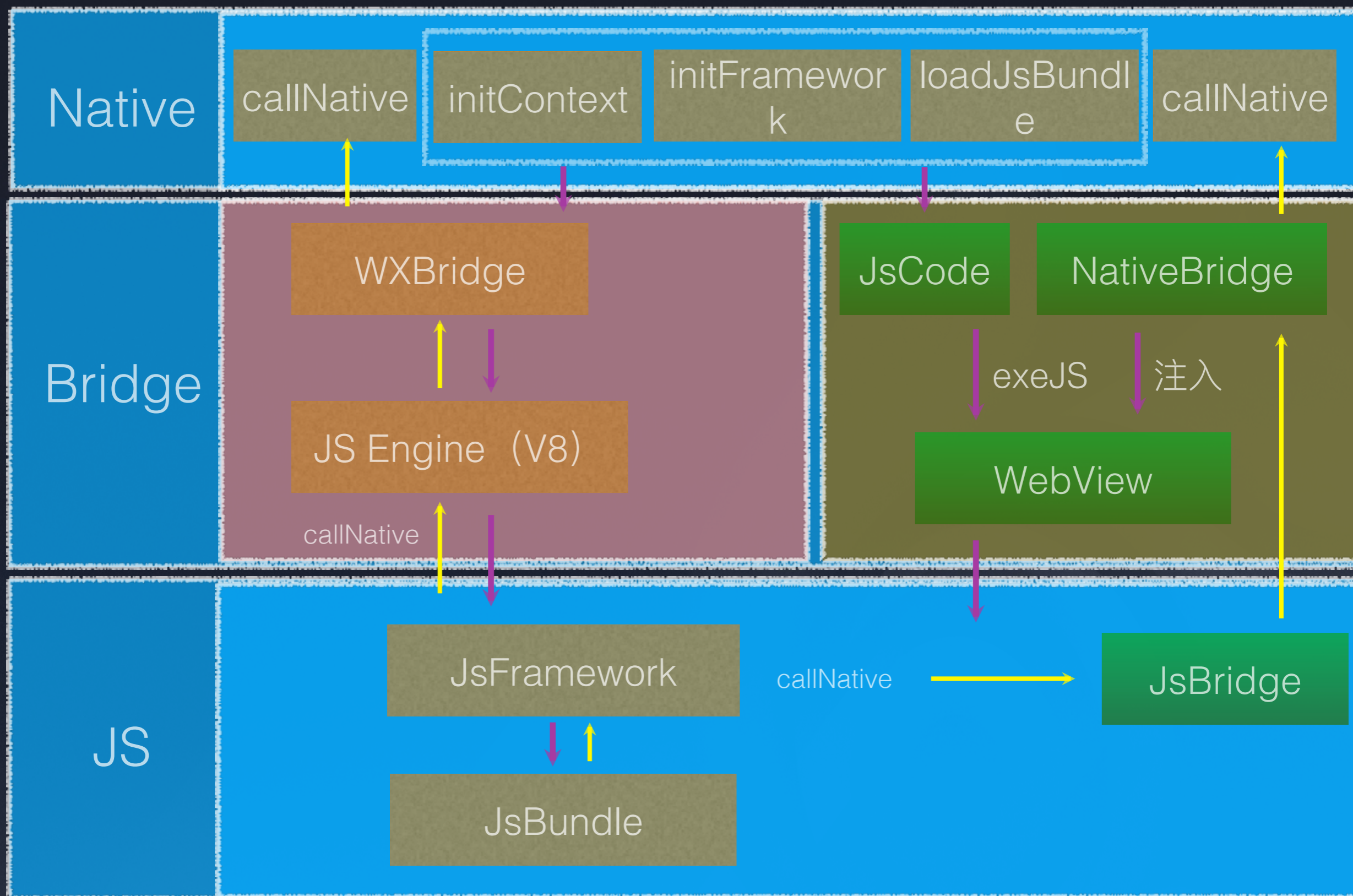
- 无法直接使用WebView的Js引擎
- WebView对外接口能力有限
- Android不同ROM兼容性

引擎关键流程分析

- JNI端通过V8构建上下文
- 加载JS Framework初始化
- 加载JsBundle创建页面
- 构建Native-JS通信通道



核心解决方案



加载JsFramework

```

jint Java_com_taobao_weex_bridge_WXBridge_initFramework(JNIEnv *env,
                                                         jobject object, jstring script,
                                                         jobject params) {
    .....
    WXEnvironment = v8::ObjectTemplate::New();
    //ObjectTemplate, 可以将 C++ 中的对象暴露给脚本环境, 因此脚本环境中就有了WXEnvironment这个变量
    jclass c_params = env->GetObjectClass(params); //这里的params就是java层传过来的对象
    .....设置params中的参数
    WXEnvironment->Set("platform", jString2V8String(env, (jstring) platform));
    WXEnvironment->Set("osVersion", jString2V8String(env, (jstring) osVersion));
    WXEnvironment->Set("appVersion", jString2V8String(env, (jstring) appVersion));
    WXEnvironment->Set("weexVersion", jString2V8String(env, (jstring) weexVersion));
    .....
    V8context = CreateShellContext();
    //ExecuteJavaScript 真正会是执行我们的js framework script脚本, 因此有了我们的js framework的run time环境
    const char* scriptStr = (env)->GetStringUTFChars(script, NULL);
    if(scriptStr == NULL || !ExecuteJavaScript(globalIsolate, v8::String::New(scriptStr), true)){
        return false;
    }
    env->ReleaseStringUTFChars(script, scriptStr);
    env->DeleteLocalRef(script);
    .....
    return true;
}
    
```

WebView -> JS

JNI -> V8 -> JS

```

@Override
public int initFramework(String framework, WXParams params) {
    String initJS = "";
    try {
        initJS = WXFileUtils.loadAsset("initJSFM.js", WXEnvironment.getApplication());
    } catch (Exception e) {
    }

    if (StringUtils.isEmpty(initJS)) {
        return 0;
    }

    final StringBuilder jsfm = new StringBuilder(initJS).append("\n")
        .append("var WXEnvironment = ").append(params.toJsonString()).append(";\n")
        .append(framework);
    return mWebViewJSEngine.initFramework(jsfm.toString());
}
    
```

JS - Native通道

```
v8::Persistent<v8::Context> CreateShellContext() {
    // Create a template for the global object.
    v8::Handle<v8::ObjectTemplate> global = v8::ObjectTemplate::New();
    // Bind the global 'callNative' function to the C++ callNative.
    global->Set(v8::String::New("callNative"), v8::FunctionTemplate::New(callNative));
    // Bind the global 'setTimeoutNative' function to the C++ setTimeoutNative.
    global->Set(v8::String::New("setTimeoutNative"), v8::FunctionTemplate::New(setTimeoutNative));
    // Bind the global 'nativeLog' function to the C++ Print callback.
    global->Set(v8::String::New("nativeLog"), v8::FunctionTemplate::New(nativeLog));
    // Bind the global 'WXEnvironment' Object.
    global->Set(v8::String::New("WXEnvironment"), WXEnvironment);
    return v8::Context::New(NULL, global);
}
```



```
@JavascriptInterface
public int callNative(String instanceId, String tasks, String callback) {
    return this.mWXBridge.callNative(instanceId, tasks, callback);
}

@JavascriptInterface
public int callAddElement(String instanceId, String ref, String dom, String index,
    return this.mWXBridge.callAddElement(instanceId, ref, dom, index, callback);
}

@JavascriptInterface
public Object callNativeModule(String instanceId, String module, String method, Str
    return this.mWXBridge.callNativeModule(instanceId, module, method, arguments !=
}

@JavascriptInterface
public void callNativeComponent(String instanceId, String componentRef, String meth
    this.mWXBridge.callNativeComponent(instanceId, componentRef, method, arguments
}
```

NativeBridge

```
function callNative(a0, a1, a2) {
    bridge.callNative(a0, JSON.stringify(a1), a2);
}

function callNativeModule(a0, a1, a2, a3, a4) {
    bridge.callNativeModule(a0, a1, a2, JSON.stringify(a3), JSON.stringify(a4));
}

function callNativeComponent(a0, a1, a2, a3, a4) {
    bridge.callNativeComponent(a0, a1, a2, JSON.stringify(a3), JSON.stringify(a4));
}

function callAddElement(a0, a1, a2, a3, a4) {
    bridge.callAddElement(a0, a1, JSON.stringify(a2), a3, a4);
}
```

JSBridge

其它挑战

分类	结果	备注
兼容性	ROM 4.0 -7.0	1.引擎特性支持 2.4.4以下ROM兼容性
调试能力	支持	对接远程调试协议
异常捕获	支持	顶层JSFM进行捕获

结果

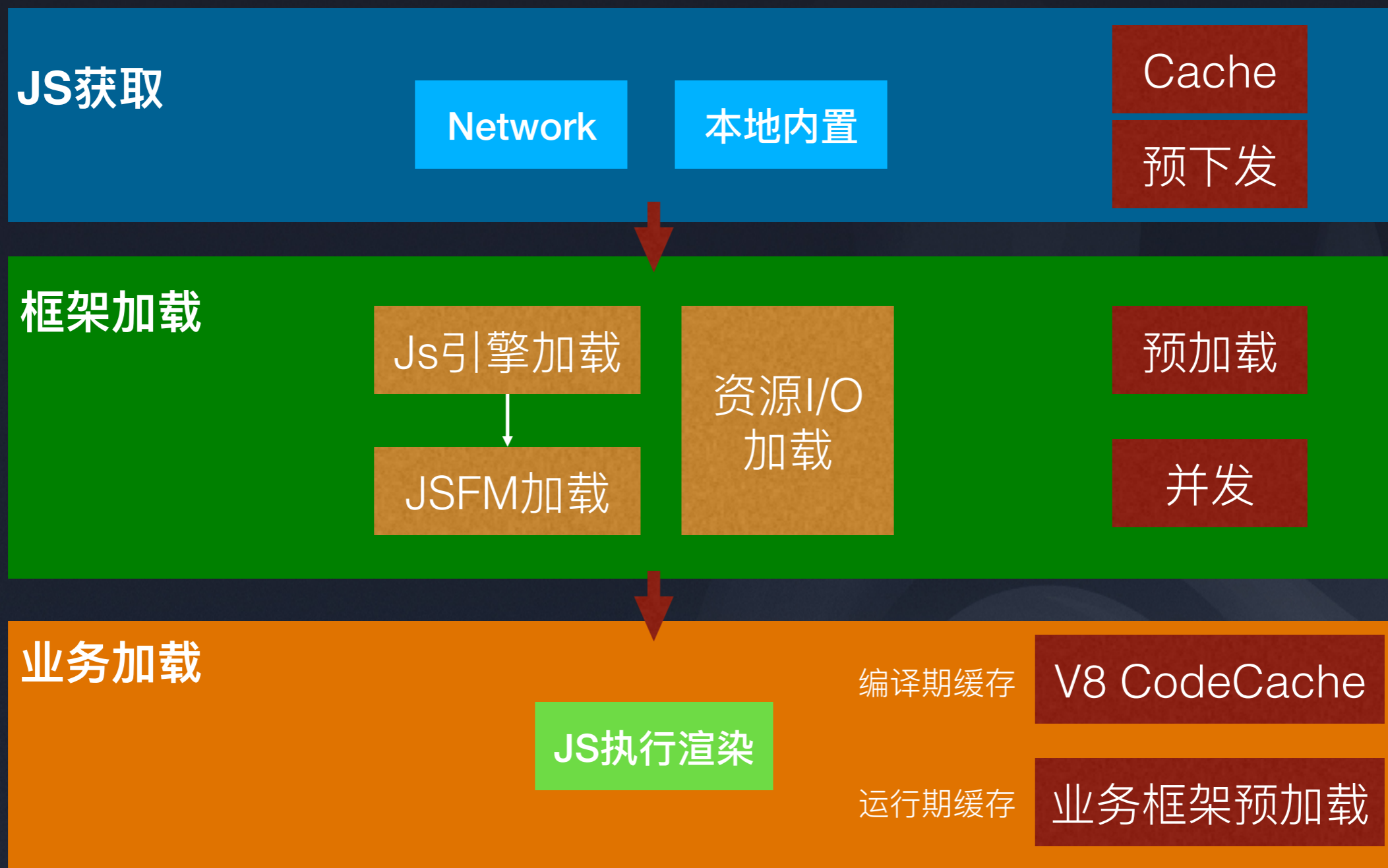
- 完全丢弃独立引擎，接入size降至500K
- 所有App都可以无依赖接入使用
- 稳定性受系统WebView控制，存在隐患

方案继续优化



- 一套大幅度Framework瘦身方案
- 其它依赖Js引擎的动态框架都可以借鉴

Framework优化：真实秒开率



数据：秒开率40%+ → 80%+

Framework优化：真实秒开率

后续

- JS Bundle size瘦身
- 业务拆分及模块化

● 配套工程化解决方案

● 核心性能持续优化

● 业务服务能力提升

垂直类型业务面临巨大挑战

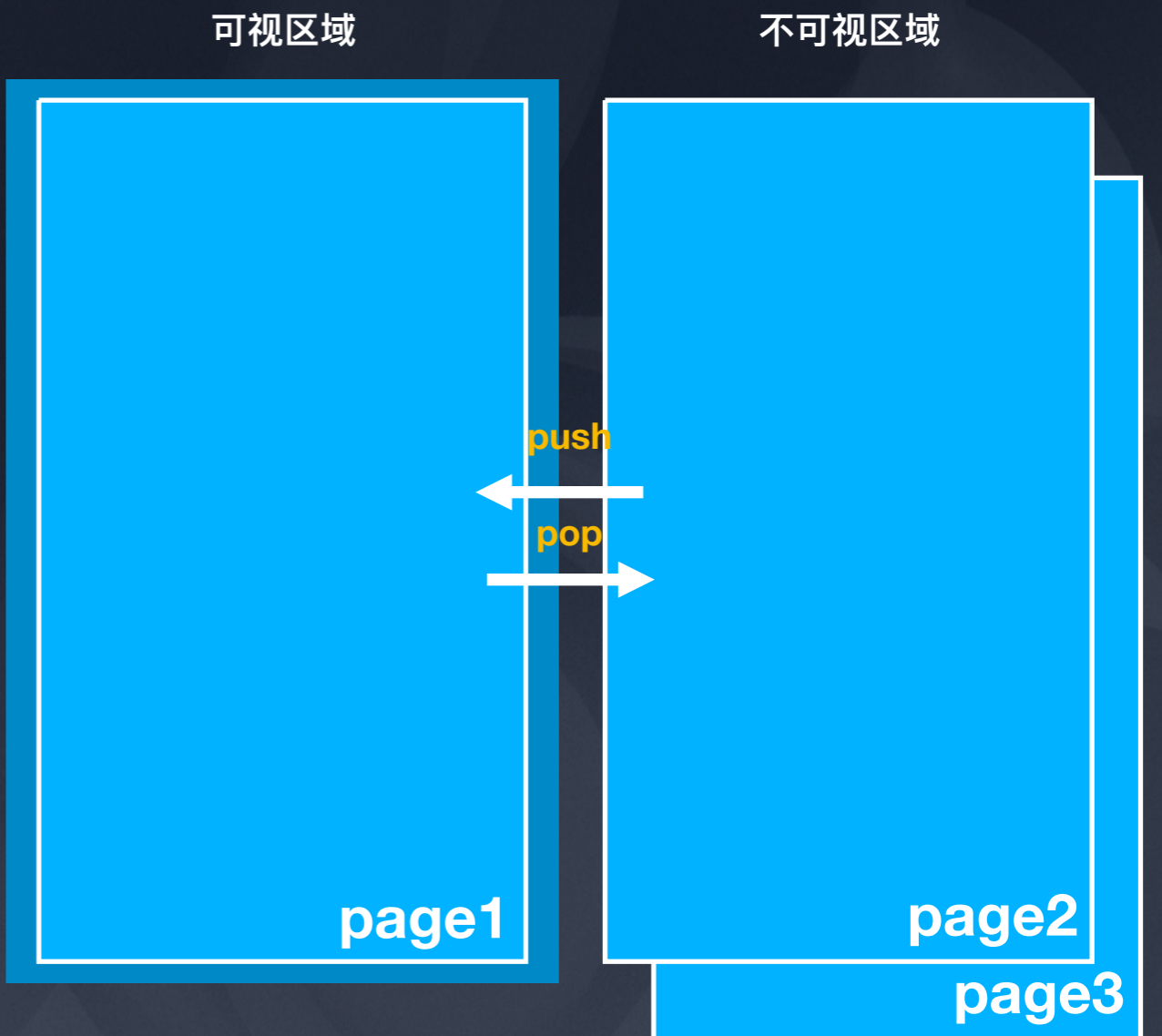
单页面路由解法

1.0 路由探索	想法	尝试基于当前 DSL 特性在 JS 侧实现	优点	不依赖 Native, 实现简单
	方案	节点层级覆盖 + 绝对定位 + 过渡动画	缺点	路由顺序固定, 无法随意切换
				透明模式皮肤下不可用
				无法实现跟手动画

```

1 <template>
2   <div class="root">
3     <div class="page default" id="page1"></div>
4     <div class="page" id="page2"></div>
5     <div class="page" id="page3"></div>
6   </div>
7 </template>
8
9 <style>
10  .page { position: absolute; left: 100%; }
11  .default { left: 0; }
12 </style>
13
14 <script>
15 export function created () {
16   /* 页面集合 */
17   const pages = ['page1', 'page2', 'page3']
18   /* 历史堆栈 */
19   const historyStack = []
20   /* 初始化路由 */
21   const router = new Router(pages, historyStack)
22   /* 切换路由 */
23   router.push('page2')
24 }
25 </script>

```



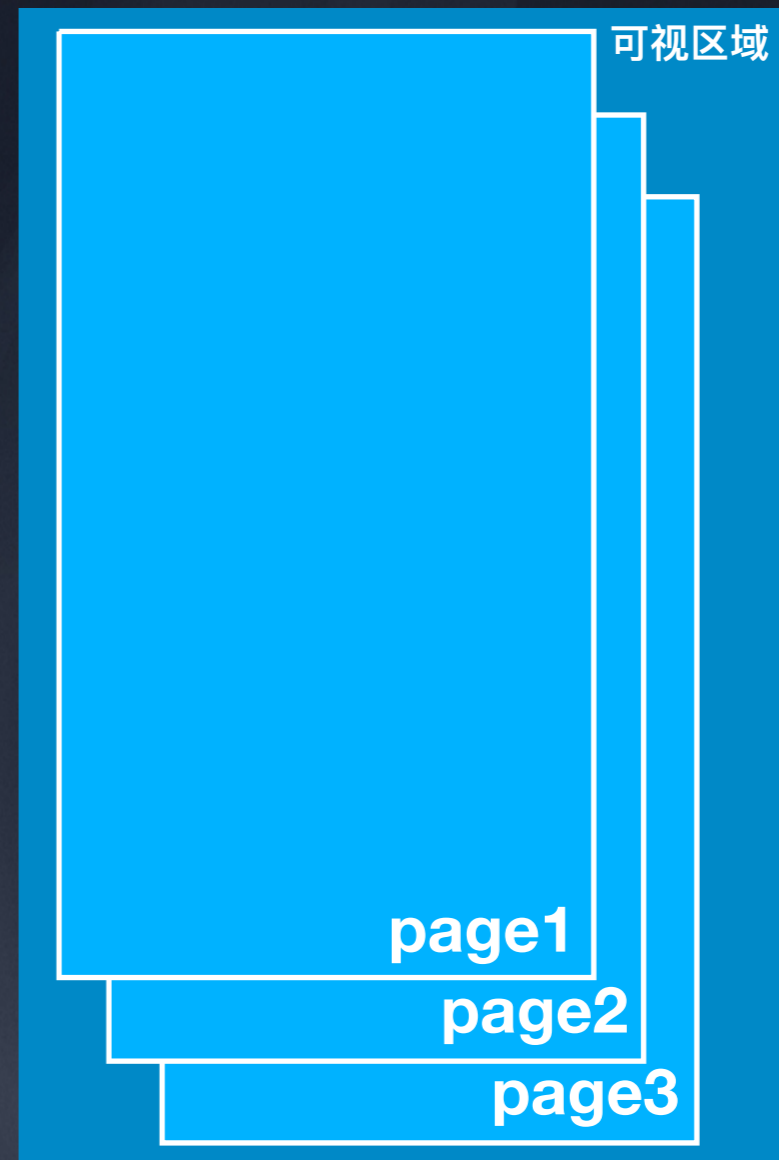
单页面路由解法

2.0 路由探索	想法	JS 实现问题多，采用 Native 定制	优点	声明式定义，任意切换路由
	方案	基于 Weex 扩展机制定制路由组件		完美适配透明皮肤
	缺点	迭代更新需重新发版	复用Native手势系统及过场动画	

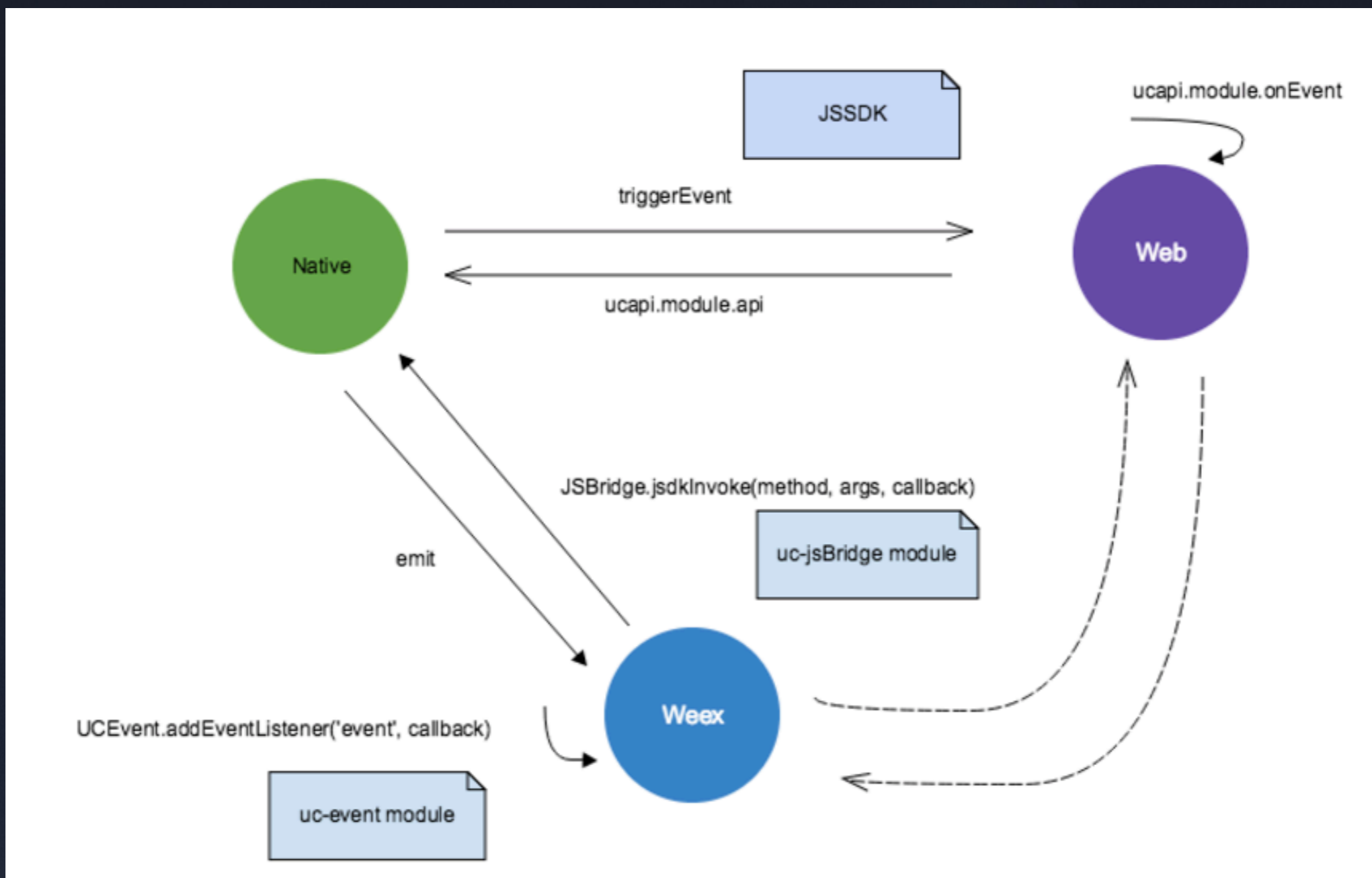
```

1 <template>
2   <uc-nav>
3     <uc-scene
4       sceneName="page1"
5       :isDefault="true"
6     >
7       <div class="content"></div>
8     </uc-scene>
9     <uc-scene sceneName="page2">
10      <div class="content"></div>
11    </uc-scene>
12    <uc-scene sceneName="page3">
13      <div class="content"></div>
14    </uc-scene>
15  </uc-nav>
16 </template>

```



三端通信能力完善



业务服务能力表现

分类	结果
基础组件	SVG、富文本组件、Web组件等支持
动画能力	Canvas、Lottie等支持
复杂交互	上推下拉组件（支持多Tab栏）
页面跳转	支持单页面内路由能力
外部通信	支持三端通信能力
H5降级	整套降级解决方案

UC-Weex典型业务展示



继续探索

- 新的通信架构设计
- 业务模块化解决方案
- 新的业务场景探索

Q&A