沪江

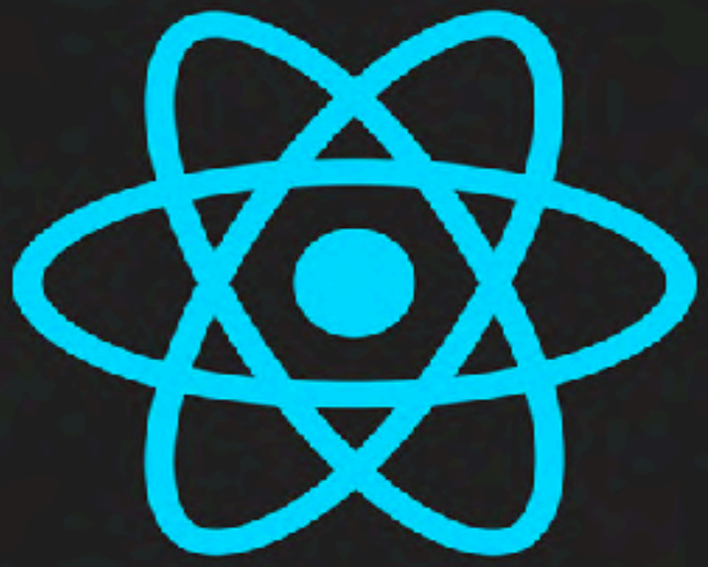
基于React Native的三端融合实践

About me

- 陈达孚
- 沪江Web前端架构部
- Web、Native端融合

摘要

- 为什么选择React Native
- 三端融合
- React Native的上线保证



React Native for Cross-platform
Apps on iOS & Android

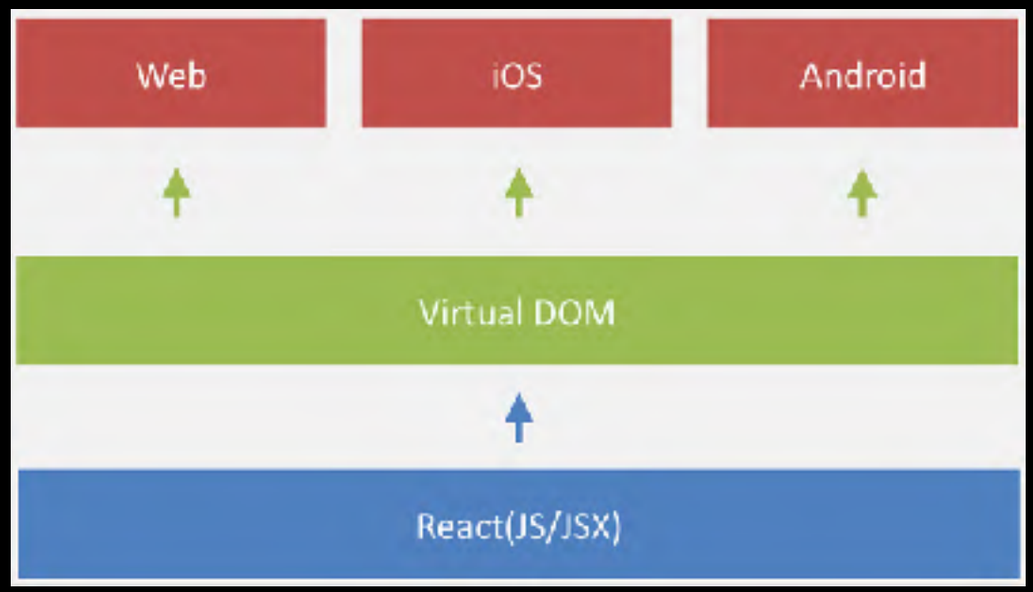
为什么选择React Native

React Native优势

- 两端共享一套业务代码
- 接近原生的性能
- 社区活跃

React Native劣勢

- Learn once write anywhere
- 部分组件性能
- 早期开发成本和融合开发成本



沪江应用现状

- 三端独立完成，复用率低
- 通过Web容器接入在线页面

问题-原生

- 体验好
- 频繁发版
- 复用率低

活动大促前的上线准备

问题-在线页面

- 业务方需要写大量兼容代码
- 体验较差
- 遇到网络问题展示不出

道

Technology

三端融合?

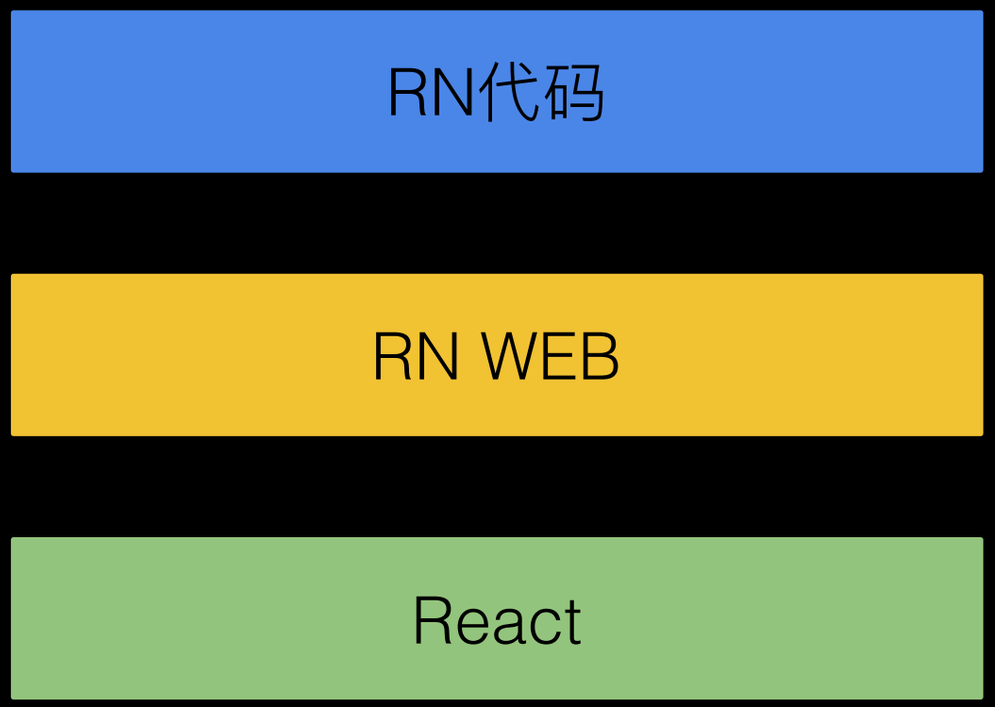
- 对立和统一
- 一套代码三端复用
- 限定来控制复杂度

思路与方案

- React Web组件 + React
- Babel编译
- 完整Web框架

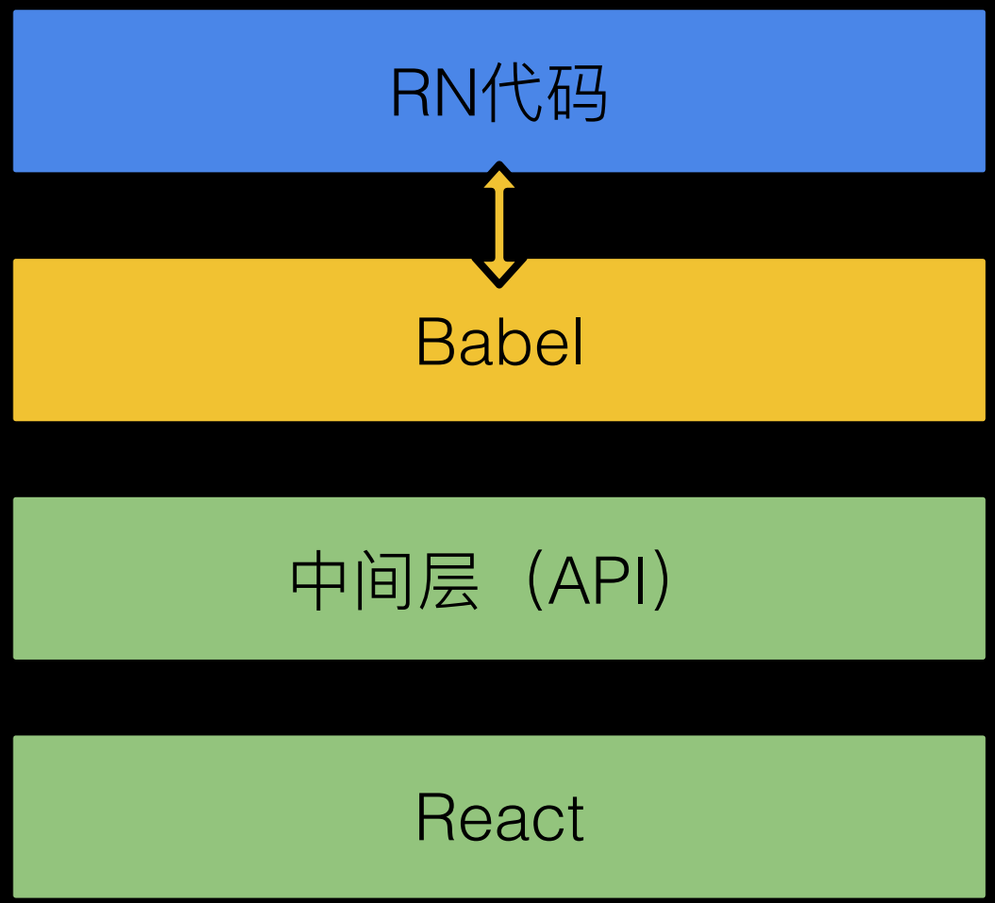
React Web组件 + React

- 初次开发成本低
- 组件代码冗余
- 不确定API
- 底层优化复杂



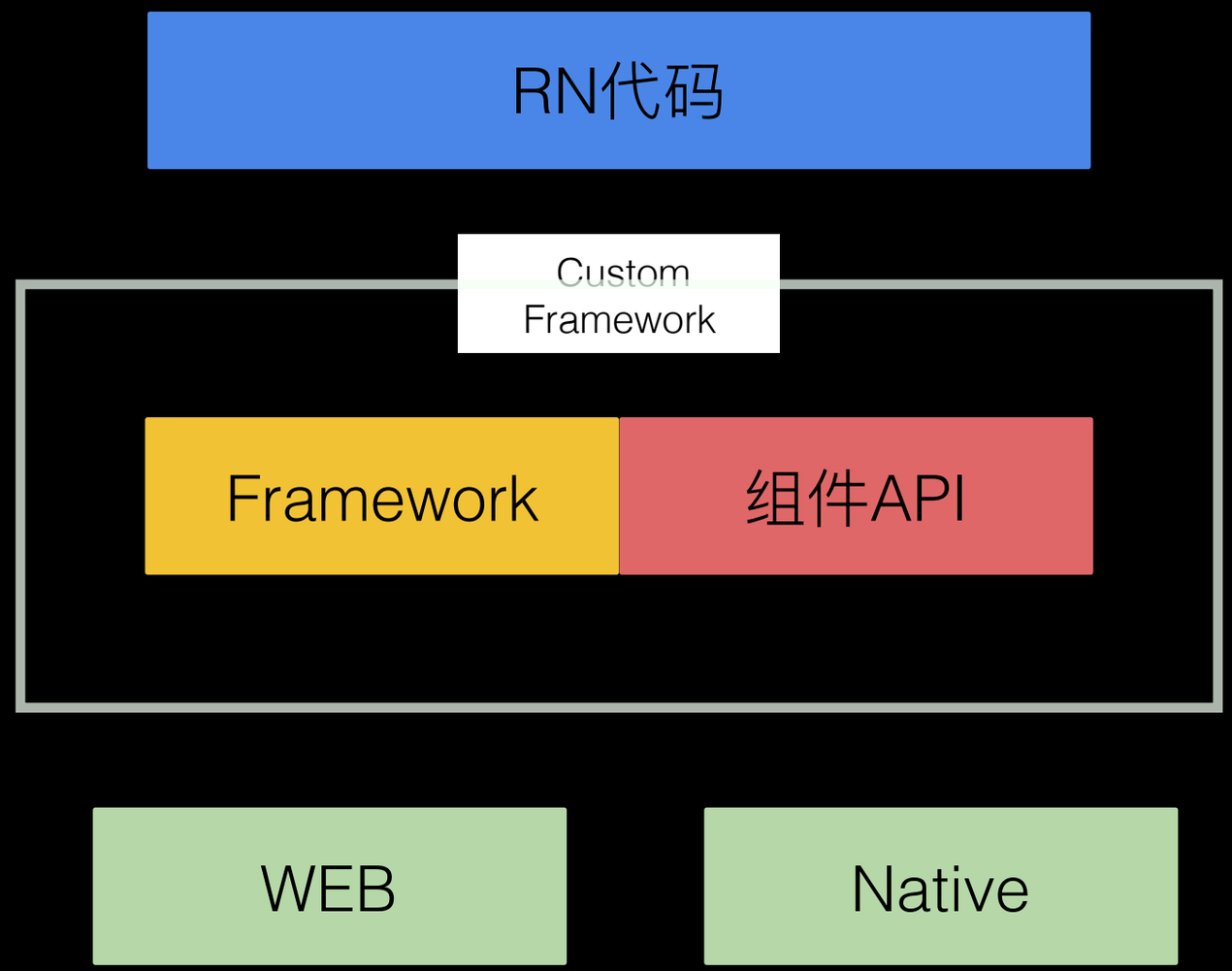
Babel编译+React

- 复杂度过高
- 业务方可以后期调整



Web组件 + 定制化框架

- 将React Native代码视作一套DSL语言
- 部分组件三端兼容性
- 部分社区三端组件



Web组件 + 定制化框架

- 摆脱组件和框架依赖关系中的不确定性（React到React-DOM，deprecated API）
- 减少组件中的冗余代码
- providesModule

Web组件 + 定制化框架

- 兼容性
- API支持能力
- 实际开发中的情况

常用组件及API

- View，ScrollView，ListVIew，Text, Image, TouchableHighlight, StyleSheet
- Web思路下的开发大量组件和API并不会使用
- 空方法并提示

入口文件修改

```
import { AppRegistry } from 'react-native';
import Discover from './src/index';

AppRegistry.registerComponent('DictDiscover', () => Discover);
```

```
import React, { render } from 'react';
import '@hujiang/cassia-global';
import Discover from './src/index';

render(<Discover />, document.getElementById('root'));
```

打包工具设置别名

```
alias: {
  'react': 'cassia'
},
```

组件框架

- 什么需要在组件中做
- 什么需要在框架中做

Web组件

- 处理React Native组件中的props，转换到相应的Web实现，或者初步处理交给框架二次处理
- 处理React Native的API，调用Web API，降级方案中调用Hybrid API

Web组件体积优化

```
import {
  StyleSheet,
  View
} from 'react-native';
```

Babel

```
import StyleSheet from 'node_modules/@hujiang/cassia-web/src/StyleSheet';
import View from 'node_modules/@hujiang/cassia-web/src/View';
```

框架

- 组件生命周期
- setState与更新队列
- 事件系统
- Vitrual DOM

框架精简

- 不稳定的API
- React提供，React Native不提供的API
- PropType
- 合成事件
- 旧语法和旧浏览器的支持
- 简化了部分事务系统和调度系统

框架添加功能

- Props(Style)处理
- 部分公共API
- React Native框架API:
  - findNodeHandle
  - setNativeProps
  - NativeMethods….

整合的优势

- 减少API的频繁引入
- 优化组件的编写成本
- 更小的包体积
- 减少重复的props检查
- Webpack2

四端融合

- 在Platform中判断WebView
- Web模式渲染
- WebView提供更强大的API
- callback => promise
- Adapter

React Native的上线保证

开发上线RN

- 横向团队在RN开发中的职责
- 完整的技术架构方案
- 技术保证

开发环境

- 测试容器
- Web开发通过发测试包迅速查看在APP中的运行情况
- 热更新应用代码

保障

- 前端监控
- 崩溃监控
- 数据上报
- 业务方通过配置中心选择

更新

- 拆包
- 打包工具
- 自动生成配置信息
- MMP2.0
- 针对版本的管理更新机制

功能性组件开发

- Login
- BI
- Share
- APP
- RouterManger

性能优化

- 最好的方法是扬长避短
- 预加载
- 针对性的优化（ListView）
- RN自身的优化

Before:

```
Animated.timing(this.state.animatedValue, {
  toValue: 1,
  duration: 500,
}).start();
```

After:

```
Animated.timing(this.state.animatedValue, {
  toValue: 1,
  duration: 500,
  useNativeDriver: true, // <-- Add this
}).start();
```

整体架构

业务代码

打包系统

Native自定义组件

Web组件

预加载

Web框架

监控系统

配置下发

离线包下

未来

- 更多的优化组件
- fiber
- 业务结合中的调整

大前端时代

- Web端和Native端的差异与统一
- Native，React Native，Hybrid，PWA

谢谢!