

# H5、React Native、Native应用对比分析

王利华

@vczero

# H5、React Native、Native指什么？

**H5 / Hybrid**：使用JavaScript与HTML、CSS技术体系开发App内场景的核心工程技术

**Native**：iOS/Android ——原生应用（都懂得，不解释）。

**React Native**：React & Native，应运而生！



# React Native的产生

- “鉴往知来”——从过去的教训中总结经验，从用户的角度开拓未来
- “HTML5差强人意，与原生应用相比还是有些差距”——更高的追求！用户体验！
- “人才宝贵，快速迭代”——Web开发者相对较多，寻找平衡点
- “跨平台！跨平台！跨平台！”——单一技术栈
- “xx是世界上最好的语言”——工程学的范畴，没有最好，只有最适合

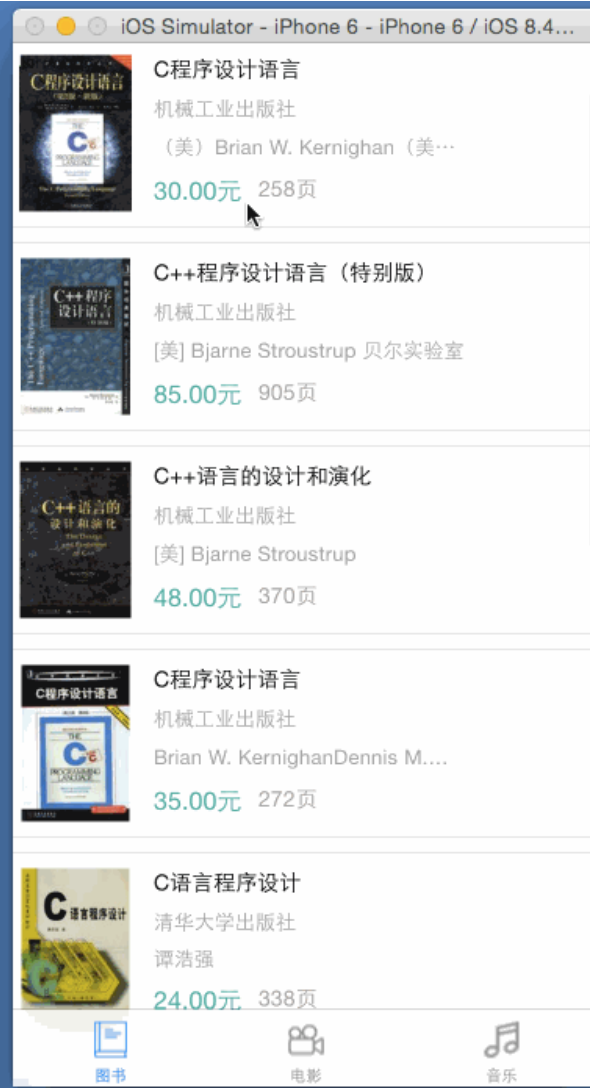
## HTML5 vs React Native ? HTML5 : React Native

结论（React Native）：

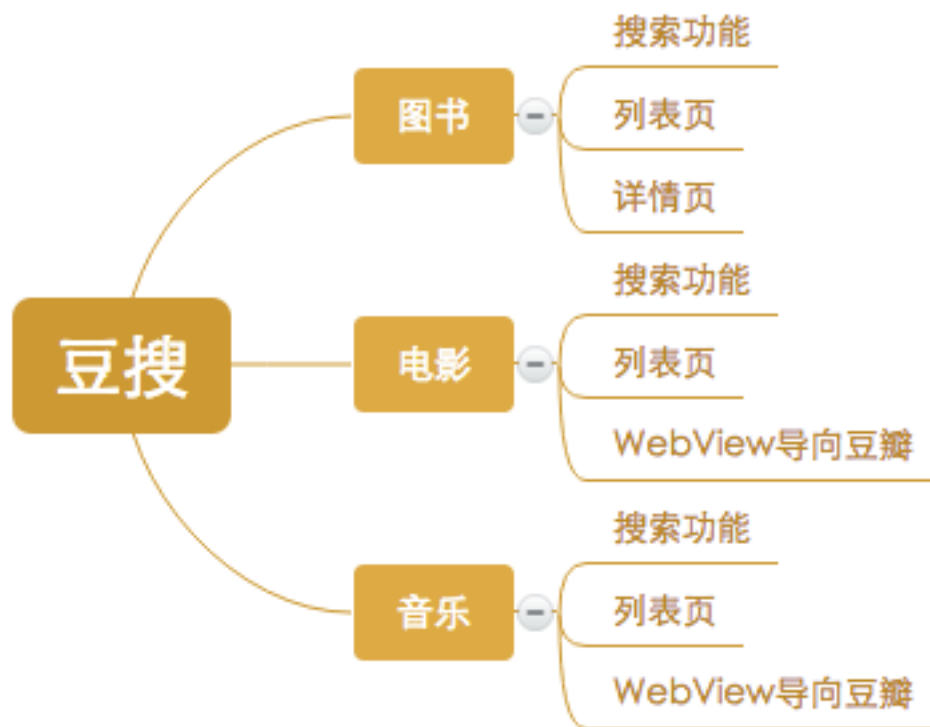
1. 原生应用的用户体验
2. 跨平台特性 & 热更新
3. 开发人员单一技术栈
4. 上手快，入门容易
5. 社区繁荣

# 三款 ( H5/RN/N ) 豆搜App效果

是否分得清哪个是React Native开发？哪个是Native应用？你的第一感觉是什么？



# 实验工程方案



注：功能一致，组件使用保持一致

# 工程方案 - H5&RN&N

1. 在H5/Hybrid应用中，使用AngularJS开发单页webApp
2. 将该WebApp内嵌入到iOS WebView中
3. 在iOS代码中，使用Navigation稍微控制下跳转

WebApp地址：<http://vczero.github.io/search/html/index.html>

WebApp项目地址：<https://github.com/vczero/search>

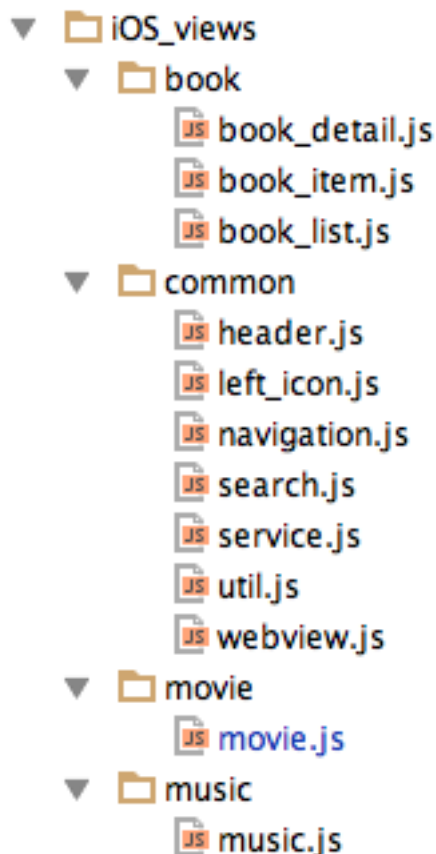
H5项目地址：[https://github.com/vczero/search\\_Hybrid](https://github.com/vczero/search_Hybrid)

在React Native中，封装必要的功能组件。

项目地址：<https://github.com/vczero/React-Dou>。

使用React Native大致相同的组件开发App，  
不使用任何第三方库，代码布局。

项目地址：<https://github.com/vczero/iOS-Dou>



# 对比分析

## 开发方式

- (1) 代码结构
- (2) UI布局
- (3) UI截面图
- (4) 路由／Navigation
- (5) 第三方生态链

## 性能 & 体验

- (1) 内存
- (2) CPU
- (3) 动画
- (4) 安装包体积
- (5) Big ListView
- (6) 真机体验

## 更新 & 维护

- (1) 更新能力
- (2) 维护成本

# NO1: 开发方式



# 代码结构

▼ **search**  
2 targets, iOS SDK 8.4

▼ **html**

- ▶ **build**
- ▶ **css**
- ▶ **doc**
- ▶ **Gruntfile.js** A
- ▶ **html**
- ▶ **icon.png** A
- ▶ **imgs**
- ▶ **js**
- ▶ **lib**
- ▶ **node\_modules**
- ▶ **package.json** A
- ▶ **README.md** A

▼ **search**

- ▶ **AppDelegate.h**
- ▶ **AppDelegate.m**
- ▶ **ViewController.h**
- ▶ **ViewController.m** M
- ▶ **Main.storyboard**
- ▶ **Images.xcassets**
- ▶ **LaunchScreen.xib**
- ▶ **Supporting Files** M
- ▶ **searchTests**

▼ **React-Dou (~ /work/github/React-Dou)**

▼ **douApp**

- ▶ **.idea**
- ▶ **android**
- ▶ **ios**
- ▼ **iOS\_views**
  - ▼ **book**
    - ▶ **book\_detail.js**
    - ▶ **book\_item.js**
    - ▶ **book\_list.js**
  - ▼ **common**
    - ▶ **header.js**
    - ▶ **left\_icon.js**
    - ▶ **navigation.js**
    - ▶ **search.js**
    - ▶ **service.js**
    - ▶ **util.js**
    - ▶ **webview.js**
  - ▼ **movie**
    - ▶ **movie.js**
  - ▼ **music**
    - ▶ **music.js**
- ▶ **node\_modules**
  - ▶ **.flowconfig**
  - ▶ **.gitignore**
  - ▶ **.watchmanconfig**
  - ▶ **index.android.js**
  - ▶ **index.ios.js**
  - ▶ **package.json**
- ▶ **LICENSE**
- ▶ **ReactDou.gif**

▼ **iOS-Dou**  
2 targets, iOS SDK 8.4

▼ **iOS-Dou**

- ▶ **Images**
- ▶ **Util**
- ▶ **AppDelegate.h**
- ▶ **AppDelegate.m**
- ▶ **BookViewController.h**
- ▶ **BookViewController.m**
- ▶ **BookDetailViewController.h**
- ▶ **BookDetailViewController.m**
- ▶ **MovieViewController.h**
- ▶ **MovieViewController.m**
- ▶ **MusicViewController.h**
- ▶ **MusicViewController.m** M
- ▶ **DWebView.h**
- ▶ **DWebView.m**
- ▶ **Main.storyboard**
- ▶ **Images.xcassets**
- ▶ **LaunchScreen.xib**
- ▶ **Supporting Files** M
- ▶ **iOS-DouTests**
- ▼ **Products**
  - ▶ **iOS-Dou.app**
  - ▶ **iOS-DouTests.xctest**

结论：从前端角度而言，**React Native**跨平台特性，不需要开发者深入的了解各平台就能开发一款高效**App**。同时，语言层面而言，**JavaScript**运用很广泛，入门门槛相对较低。**React Native**虽然抛弃了**MVC**分离实践，但是从业务角度而言，更为合理。

总而言之：对前端，对移动领域是**利好**的消息。

# UI布局

Web开发布局目前大多是 DIV + CSS。

React Native: FlexBox + JSX

```
<ScrollView style={styles.flex_1}>
  <View style={[styles.search, styles.row]}>
    <View style={styles.flex_1}>
      <Search placeholder="请输入图书的名称" onChangeText={this._changeText}/>
    </View>
    <TouchableOpacity style={styles.btn} onPress={this._search}>
      <Text style={styles.fontFFF}>搜索</Text>
    </TouchableOpacity>
  </View>
  {
    this.state.show ?
    <ListView
      dataSource={this.state.dataSource}
      renderRow={this._renderRow}
    />
    : Util.loading
  }
</ScrollView>
```

Native (iOS) 布局

```
UILabel *publisher = [[UILabel alloc] init];
publisher.frame = CGRectMake(bookImgWidth + 10, 50, 200, 30);
publisher.textColor = [UIColor colorWithRed:0.400 green:0.400 blue:0.435 alpha:1];
publisher.font = [UIFont fontWithName:@"Heiti TC" size:13];
publisher.text = data[@"publisher"];
[bookInfo addSubview:publisher];
```

React Native既综合了Web布局的优势，采用了FlexBox和JSX，

又使用了Native原生组件。比如我们使用一个文本组件。

```
<Text style={{width:100;height:30;background-color:'red'}}>测试</Text>
```

# UI截面图——H5/Hybrid



第一层列表页是完整的布局，  
实际上这就是Web页面  
第二层灰色的是Native的WebView组件。

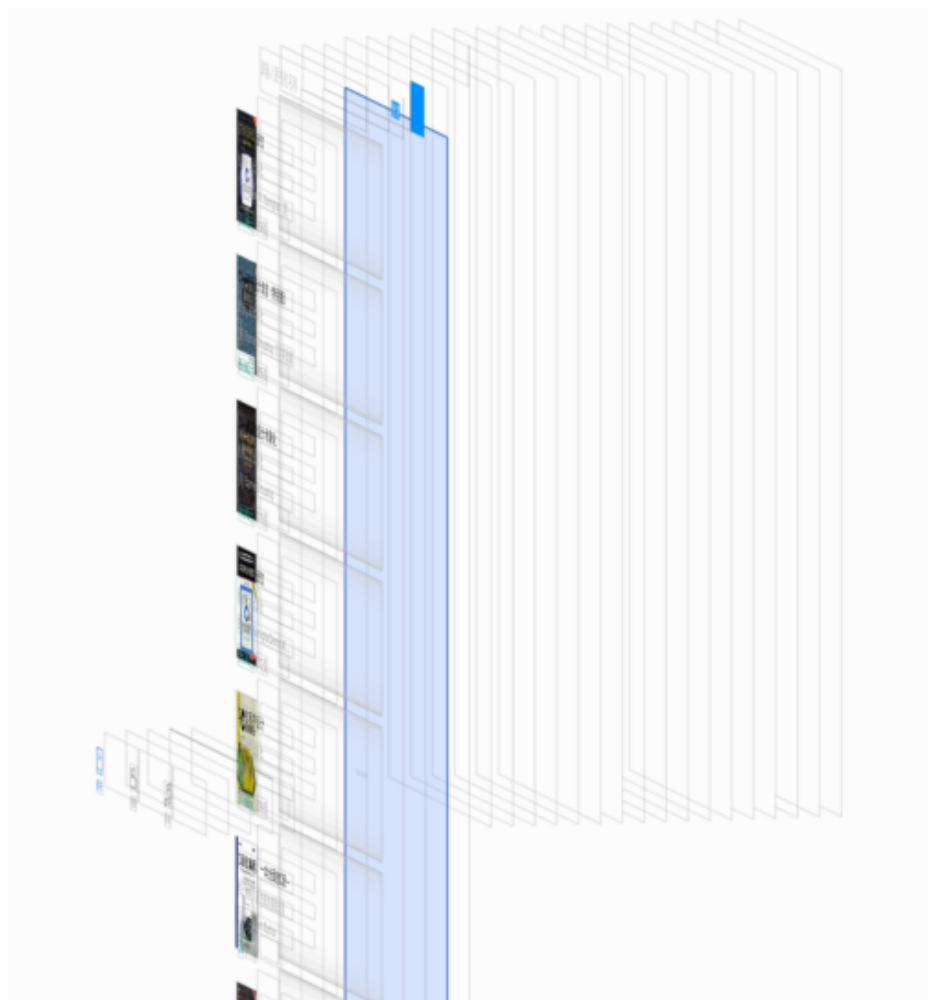
# UI截图图——Native



Native的组件特别多，即使是列表页，其中某一项都是一个组件（控件）。

调用原生组件的性能是否更好？

# UI截图图——React Native



React Native调用的全部是Native组件。并且层次更深，因为React Native做了组件的封装。蓝色边框的是RCTScrollView组件。



# 路由 / Navigation

```
BookViewController *bookViewController = [[BookViewController alloc] init];
bookViewController.view.backgroundColor = [UIColor whiteColor];
bookViewController.tabBarItem.title = @"图书";
bookViewController.tabBarItem.image = [UIImage imageNamed:@"book.png"];
bookViewController.tabBarItem.badgeValue = "1";
UINavigationController *bookNav = [[UINavigationController alloc] initWithRootViewController:bookViewController];
```

```
MovieViewController *movieViewController = [[MovieViewController alloc] init];
movieViewController.view.backgroundColor = [UIColor whiteColor];
movieViewController.tabBarItem.title = @"电影";
movieViewController.tabBarItem.image = [UIImage imageNamed:@"movie.png"];
UINavigationController *movieNav = [[UINavigationController alloc] initWithRootViewController:movieViewController];
```

```
MusicViewController *musicViewController = [[MusicViewController alloc] init];
musicViewController.view.backgroundColor = [UIColor whiteColor];
musicViewController.tabBarItem.title = @"音乐";
musicViewController.tabBarItem.image = [UIImage imageNamed:@"music.png"];
UINavigationController *musicNav = [[UINavigationController alloc] initWithRootViewController:musicViewController];
```

```
int width = [[UIScreen mainScreen] applicationFrame].size.width;
UIView *blackView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, width, 44)];
blackView.backgroundColor = [UIColor blackColor];
```

```
[self.tabBarController.tabBar insertSubview:blackView atIndex:0];
self.tabBarController.tabBar.opaque = YES;
```

```
self.tabBarController.viewControllers=@[bookNav, movieNav, musicNav];
self.window.rootViewController = self.tabBarController;
[self.window makeKeyAndVisible];
```

```
var douApp = React.createClass({
  getInitialState: function() {
    return {
      selectedTab: '图书'
    };
  },
  render: function() {
    return (
      <TabBarIOS>
        <TabBarIOS.Item
          title="图书"
          selected={this.state.selectedTab === '图书'}
          icon={require('image!book')}
          onPress={() => {
            this.setState({
              selectedTab: '图书'
            });
          }}
        />
        <Navigation component={Book} />
      </TabBarIOS.Item>

      <TabBarIOS.Item
          title="电影"
          selected={this.state.selectedTab === '电影'}
          icon={require('image!movie')}
          onPress={() => {
            this.setState({
              selectedTab: '电影'
            });
          }}
        />
        <Navigation component={Movie} />
      </TabBarIOS.Item>

      <TabBarIOS.Item
          title="音乐"
          selected={this.state.selectedTab === '音乐'}
          icon={require('image!music')}
          onPress={() => {
            this.setState({
              selectedTab: '音乐'
            });
          }}
        />
        <Navigation component={Music} />
      </TabBarIOS.Item>
    </TabBarIOS>
  );
});
```



## 第三方生态链

“我的是我的，你的也是我的。”——我不是“疯狂女友”，我是React Native！  
“城市列表”组件，使用JSX封装一个；觉得性能太低，OK，封装一个原生组件。  
这个iOS图表库不错，拿来用呗！ => 完美！

这一切都是基于React Native提供的模块扩展方案。

所以说：iOS第三方库 + 部分JavaScript库 = React Native 生态库

# NO2: 性能 & 体验

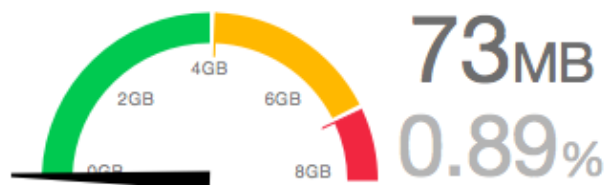
以下测试，都是基于相同的case。  
测试平台：模拟器，iphone6，iOS8.4

# 内存——Native

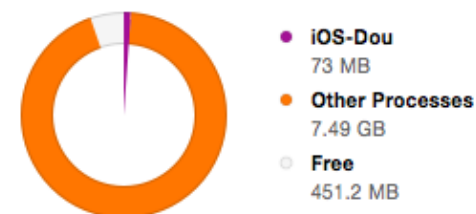
Memory

[Profile in Instruments](#)

Memory Use



Usage Comparison



Memory

Duration: 2 min  
High: 87.9 MB  
Low: 20.3 MB



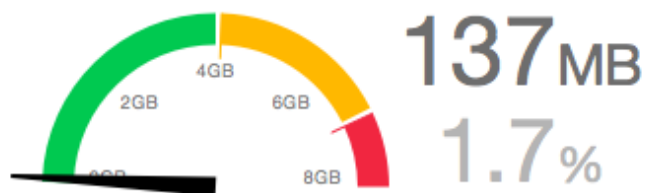
原生应用启动后，占用内存是20~25M；峰值是87.9M，均值是72M；内存释放比较及时。

# 内存——H5/Hybrid

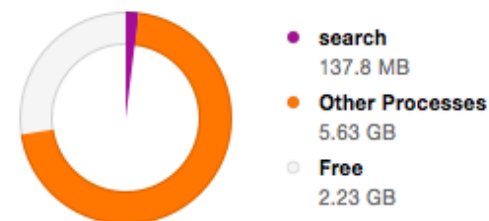
Memory

[Profile in Instruments](#)

Memory Use



Usage Comparison



Memory

137.9 MB

Duration: 3 min 1 sec  
High: 137.9 MB  
Low: 29.4 MB



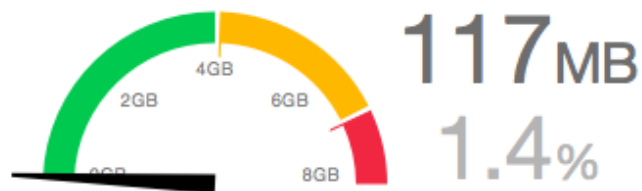
Hybird App启动，占用内存35~55M;峰值在137.9M，因为整个应用在WebView中，内存释放不明显，存在缓存。

# 内存——React Native

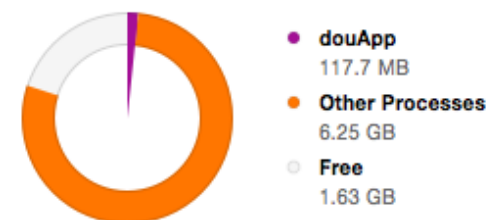
Memory

[Profile in Instruments](#)

Memory Use



Usage Comparison



Memory

Duration: 2 min 6 sec  
High: 142 MB  
Low: 35.6 MB



React Native App启动占用内存35~60M;峰值在142M, 内存相对释放明显。

总结：React Native和Web View在简单App上相差不大。  
二者主要：内存消耗主要是在网页数据上。如果应用较大则会跑赢WebView，  
因为会有更多的原生组件来完成这些功能。

# CPU-Native

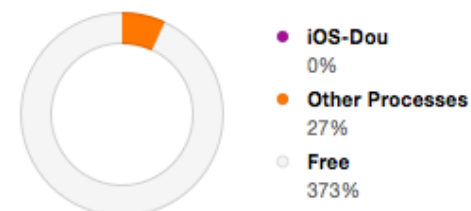
CPU

[Profile in Instruments](#)

Percentage Used

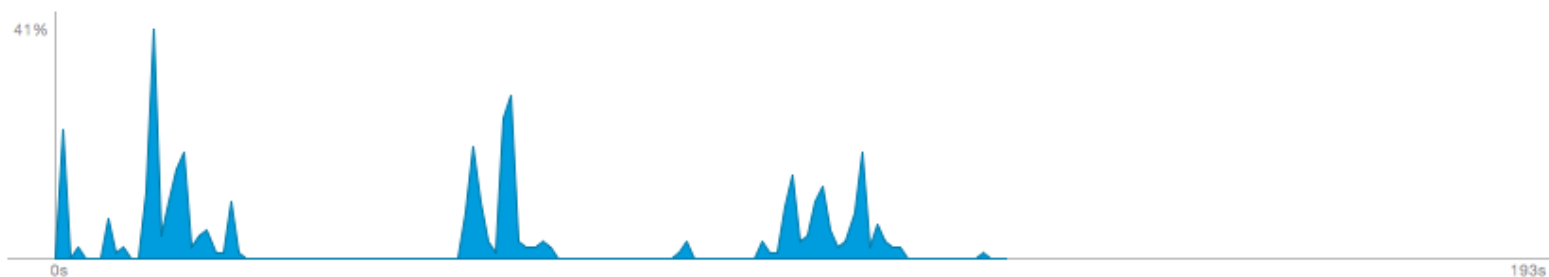


Usage Comparison



Usage over Time

Duration: 2 min 7 sec  
High: 41%  
Low: 0%



Native应用程序CPU的情况，最高值在41%。

# CPU-H5/Hybrid

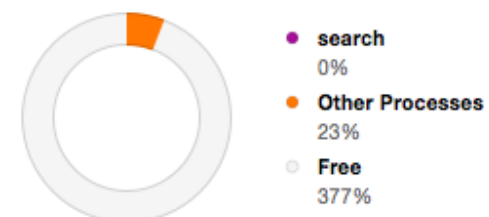
CPU

[Profile in Instruments](#)

Percentage Used

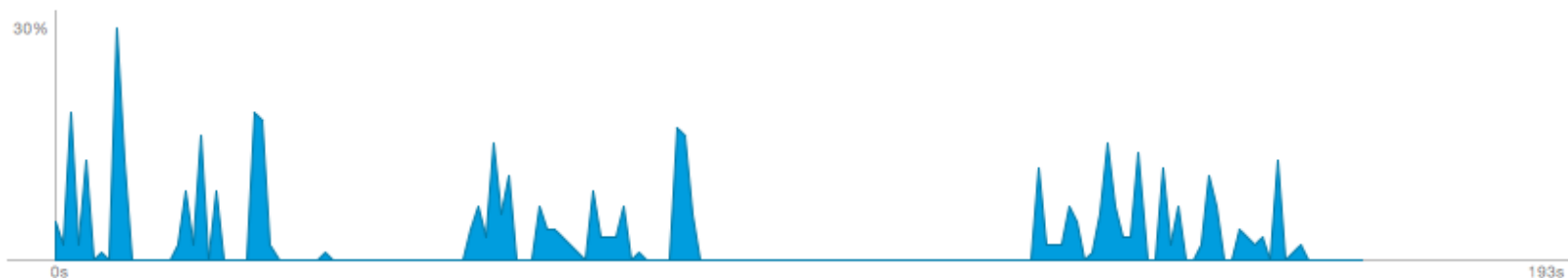


Usage Comparison



Usage over Time

Duration: 4 min 5 sec  
High: 30%  
Low: 0%



Hybird App的最高值在30%。



# CPU-React Native

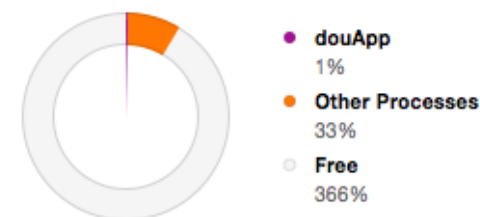
CPU

[Profile in Instruments](#)

Percentage Used

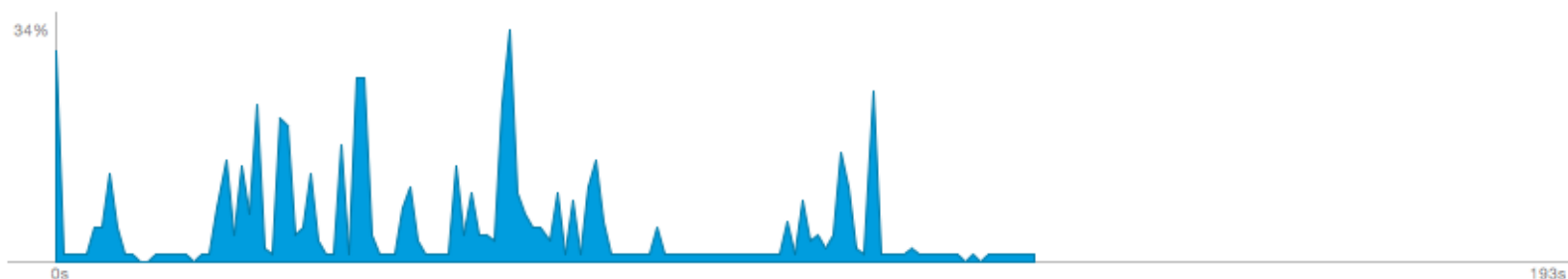


Usage Comparison



Usage over Time

Duration: 2 min 17 sec  
High: 34%  
Low: 0%



React Native的最高值在34%。

总结：CPU使用率大体相近，React Native的占用率低于Native。

# 动画

1. React Native提供了Animated API实现动画。基本OK。React Native不适合做游戏，尤其布局能力较弱。
2. Native Animation提供动画
3. H5/Hybrid：采用js动画能力

React Native Animated API / 封装Native动画库 可以满足基本需求

# 安装包体积

## Hybird App

$34(\text{App壳}) + 5(\text{HTML}) + 125(\text{Angular}) + 29(\text{An-route}) + 6(\text{min.js}) + 4(\text{min.css}) = 203 \text{ KB}$

## Native App

83KB

## React Native App

不含bundle: 843KB

含bundle: 995KB

React Native框架打包后:

$843(\text{不含bundle}) - 32(\text{Hybird\_app空壳, 初识项目}) = 811\text{KB}$

相比快速迭代和热更新，比Native多了811KB一点都不重要，可以将图片素材、静态资源线上更新缓存起来即可减少很多体积。

总结：牺牲一点体积，换更大的灵活性！

世界上哪有那么美的事，除非丑，就会想得美，：)

# Big ListView & Scroll 性能

循环列表项500次:

1. H5页面惨不忍睹
2. React Native还可以接受
3. Native 采用UITabView更高效，因为不渲染视图外部分。

# 真机体验

机型：iphone4s，iOS7；用最差的机型跑最美的东西

Native > React Native > Hybird

如果非要给个数字的话，个人主观感受是：

Native： 95%+ 流畅度

React Native: 85~90% 流畅度

H5/Hybird： 70% 流畅度

总结：Native／React Native的体验相对而言更流畅。

# NO3: 更新 & 维护

# 更新能力

1. H5/Hybrid: 随时更新, 适合做营销页面, 目前携程一些BU全部都是H5页面; 但是重要的部分还是Native。
2. React Native: React Native部分可以热更新, bug及时修复。
3. Native: 随版本更新, 尤其iOS审核严格, 需要测试过关, 否则影响用户。

# 维护成本

1. H5/Hybrid: Web代码 + iOS/Android平台支持
2. React Native: 可以一个开发团队 + iOS/Android工程师; 业务组件颗粒度小, 不用把握全局即可修改业务代码。
3. Native: iOS/Android开发周期长, 两个开发团队。

总结: React Native 统一了开发人员技术栈, 代码维护相对容易。



# 总结

## 1. 开发方式

- (1) 代码结构：React Native更为合理，组件化程度高
- (2) UI布局：Web布局灵活度 > React Native > Native
- (3) UI截面图：React Native使用的是原生组件，
- (4) 路由／Navigation：React Native & Native更胜一筹
- (5) 第三方生态链：Native modules + js modules = React Native modules

## 2. 性能 & 体验

- (1) 内存：Native最少；因为React Native含有框架，所以相对较高，但是后期平稳后会优于Native。
- (2) CPU：React Native居中。
- (3) 动画：React Native动画需求基本满足。
- (4) 安装包体积：React Native框架打包后，811KB。相比热更新，可以忽略和考虑资源规划。
- (5) Big ListView
- (6) 真机体验：Native >= React Native > H5/Hybrid

## 3. 更新 & 维护

- (1) 更新能力: H5/Hybrid > React Native > Native
- (2) 维护成本: H5/Hybrid <= React Native < Native

TURING 图灵原创

王利华 魏晓军 冯诚祺 编著


OSC 源创会

年终盛典 2015 · 北京



# React Native

## 入门与实战

 中国工信出版集团

 人民邮电出版社  
POSTS & TELECOM PRESS

