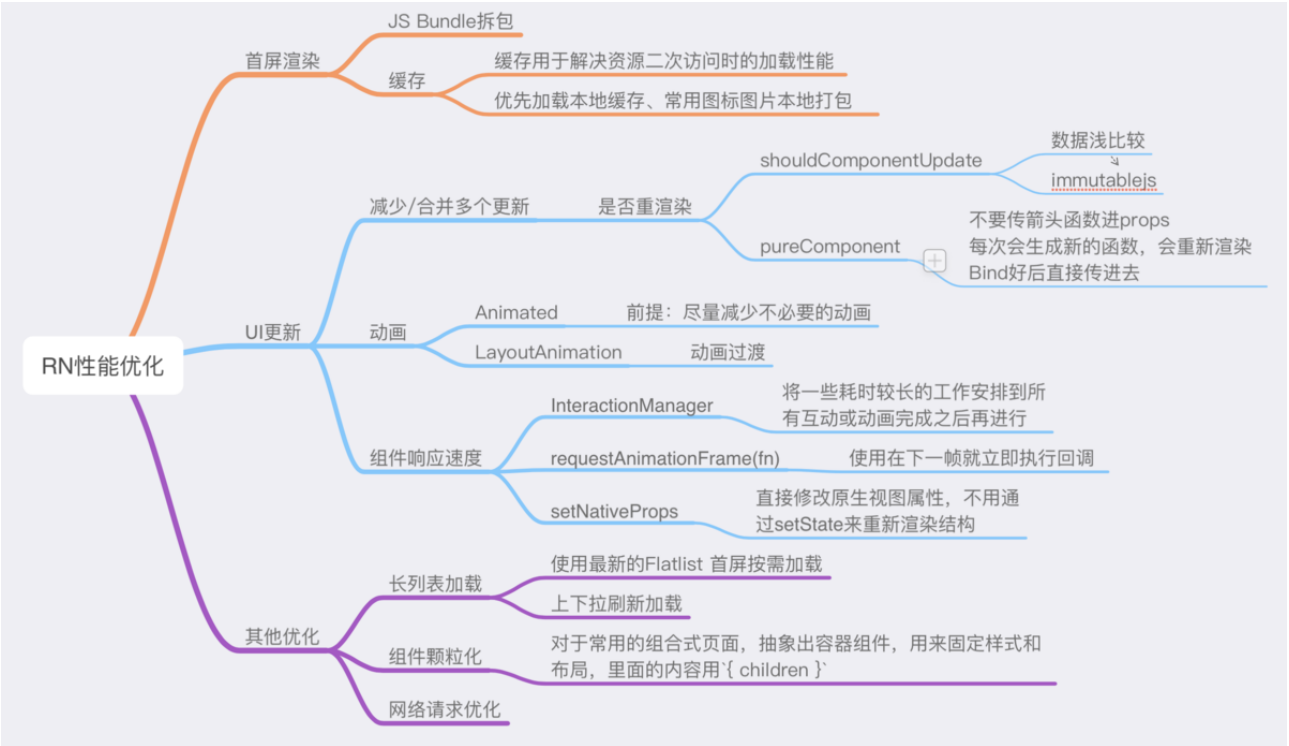
ReactNative性能优化



优化方法：

一、去掉console.log和console.warn

方法：手动去除，或者安装接入babel-plugin-transform-remove-console

npm install babel-plugin-transform-remove-console —save-dev

配置.babelrc：

```
{
  ...
  "plugins": [
    ...
  ],
  "env": {"production": {
      "plugins": [["transform-remove-console", {"exclude": ["error"]}]]
  }
  }
}
```

在我们的项目中以及去除了。

二、采用setNativeProps

在RN中，如果需要频繁刷新view，建议使用setNativeProps，避免使用setState导致的频繁render。
官方原文建议：
在（不得不）频繁刷新而又遇到了性能瓶颈的时候。 直接操作组件并不是应该经常使用的工具。一般来说只是用来创建连续的动画，同时避免渲染组件结构和同步太多视图变化所带来的大量开销。setNativeProps 是一个“简单粗暴”的方法，它直接在底层（DOM、UIView等）而不是React组件中记录state，这样会使代码逻辑难以理清。所以在使用这个方法之前，请尽量先尝试用setState 和shouldComponentUpdate方法来解决问题。

在我们的项目中动画比较少，就没有使用这个方法优化。

三、使用PureComponent或shouldComponentUpdate更新优化UI

更新优化UI：
1、减少更新或者合并多个更新

PureComponent

当组件更新时，如果组件的 props 和 state 都没发生改变， render 方法就不会触发，省去 Virtual DOM 的生成和比对过程，达到提升性能的目的。
具体原理是 在shouldComponentUpdate回调中，对oldState和newState 及 oldProps和newProps进行浅比较，如不同，才return true，进而回调render。

重写shouldComponentUpdate

```
shouldComponentUpdate() {
    return this.state.update;
```

```
}
```
控制在需要的时候才去刷新，根据需求或者业务去控制。不能随便使用。

在我们的项目中，初始化渲染次数过多的组件，使用PureComponent，减少了组件的渲染次数，它会自动检查组件是否需要重新渲染。这时，只有PureComponent检测到state或者props发生变化时，PureComponent才会调用render方法，因此，你不用手动写额外的检查，就可以在很多组件中改变state。比如：AmapScrollView、Accordion、AmapSwitch、AmapText、button，checkBox，checker，carousel，calendar、categoryList，circleProgress、datePicker、Dialog、Drawer、DraggableList、ExpModal、Icon、Image、ImageBackGround、Images、InputItem、KeyboardScrollView、Label、LinerGradient、List、marqueeVertical、marqueeView、NavBar、NoticeBar、PassWordInput、Grid、menulist，Template。都使用了这个方法来优化。
浅比较可以使用PureComponent，意味着嵌套对象和数组是不会被比较的。这个时候可以使用shouldComponentUpdate，来进行深比较，比如：ListView、SectionList根据状态判断来刷新。
2、提高组件响应速度：

- setNativeProps直接在底层更新Native组件属性（其实没有解决JS端与Native端的数据同步问题）
- 立即执行更新回调

3、动画优化

- 通过使用Annimated类库，一次性把更新发送到Native端，由Native端自己负责更新
- 把一些耗时操作放到动画与UI更新之后执行

四、长列表加载

使用用复用性更强的FlatList或SectionList
FlatList或SectionList优化：

- Item采用PureComponent或重写shouldComponentUpdate
- getItemLayout 如果行高是固定的，可以使用getItemLayout，避免动态测量内容尺寸的开销。

在我们的项目中包括我们的组件ListView和SectionList还有swipe-list，可以有待优化。

五、首屏进入时间优化：

1、减少bundle包大小

- 图片压缩。
- 把不重要图片上传到图床
- 分包加载

2、文件压缩、

3、缓存

在我们的项目中做了图片缓存，但是没有做图片和文件的压缩。分包拆包已经完成

六、懒加载

JS在执行import时有时很需要时间，首屏不需要展示的组件可以懒加载

```
import React, { Component } from 'react';
import { TouchableOpacity, View, Text } from 'react-native';

// 先把这个组件赋值为null
let VeryExpensive = null;

export default class Optimized extends Component {
  state = { needsExpensive: false };

  didPress = () => {
    if (VeryExpensive == null) {
        // 真正需要这个组件的时候才加载
      VeryExpensive = require('./VeryExpensive').default;
    }

    this.setState(() => ({
      needsExpensive: true,
    }));
  };

  render() {
    return (
      <View style={{ marginTop: 20 }}>
        <TouchableOpacity onPress={this.didPress}>
          <Text>Load</Text>
        </TouchableOpacity>
           // 根据需要判断是否渲染该组件
        {this.state.needsExpensive ? <VeryExpensive /> : null}
      </View>
    );
  }
}
```

在我们的项目中没有使用这个方法优化。

七、使用BindingX

理论上任何 "频繁通信+UI更新" 的场景都可以使用BindingX来优化卡顿问题。
安装： npm install react-native-bindingx --save
适用场景：

- 监听 pan 手势，更新UI。
- 监听滚动容器(如List)的onscroll事件，更新UI。
- 监听设备传感器方向变化，更新UI。
- 动画。(即监听设备的每一帧的屏幕刷新回调事件，更新UI)。

在我们的项目中也没有使用这个方法。

React Native 性能优化用到的工具:

1.React Native 官方调试工具

这个官网说的很清楚了，具体内容可见 直达链接。

2.react-devtools

React Native 是跑在原生 APP 上的，布局查看不能用浏览器插件，所以要用这个基于 Electron 的 react-devtools。写本文时 React Native 最新版本还是 0.61，不支持最新 V4 版本的 react-devtools，还得安装旧版本。具体安装方法可见这个

3.XCode

iOS 开发 IDE，查看分析性能问题时可以用 **instruments** 和 **Profiler** 进行调试。

4.Android Studio

Android 开发 IDE，查看性能的话可以使用 **Android Profiler**， 官方网站写的非常详细。

5.iOS Simulator

iOS 模拟器，它的 Debug 可以看一些分析内容。

6.Android 真机 -> 开发者选项

Android 开发者选项有不少东西可看，比如说 GPU 渲染分析和动画调试。真机调试时可以开启配合使用。

在我们的项目中主要使用的是真机测试，通过在iOS真机上在render里面打印，来观察render的次数，和show Perf Monitor 来开启性能监控，观察fps和内存变化

其他优化记录：

1. 把一些没有生命周期的组件，写成纯函数的组件，比如：WingBlank，WhiteSpace，AmapSwiper，KeyboardScrollView