React Native 研究分享

利炳根

2016.1.15

内容目录

1 页面导航

1.1 navigator

- initialRoute:路由初始化配置信息
- initialRouteStack：路由栈初始化
- cofigureScene:场景转换动画配置
- renderScene:渲染场景
- push:导航器跳转到一个新的路由
- pop:回到上一页
- replace:替换当前页的路由，并立即加载新路由的视图。
- navigationBar:导航栏
- navigator:上级导航
- sceneStyle:场景样工

1.2 Route

- component:显示的第一个组件
- title:导航栏标题
- passProps:传递参数
- backButtonTitle:后退按钮文字
- backButtonIcon:后退按钮图标
- leftButtonTitle:后退按钮文字
- leftButtonIcon:后退按钮图标
- onLeftButtonPress:点击导航栏右侧按钮执行的函数
- rightButtonTitle:导航栏右侧按钮文字
- onRightButtonPress:点击导航栏右侧按钮执行的函数
- wrapperStyle:外观样式

1.3 tabbar

- barTintColor:标签栏的背景颜色
- style:标签栏样式
- tintColor:当前被选中的标签图标的颜色
- translucent:是否需要半透明化

1.4 TabBarIOS.Item

- badge:图标右上角红色气泡
- icon:标签图标
- onPress:标签被选中时调用的函数
- selected:是否选中标签
- selectedIcon:被选中标签图标
- style:标签样式
- systemIcon:系统图标
- title:标题

2 手势响应系统

2.1 Touchable系列组件与TouchableHighlight

- Touchable: 利用响应系统，以声明的方式来配置触控处理
- TouchableHighlight: 按钮或者网页链接。

2.2 响应者-注册

- 一个View只要实现了正确的协议方法，就可以成为触摸事件的响应者。
- 是否成为响应者，是否响应触摸。
- View.props.onStartShouldSetResponder: (evt) => true
- 在用户开始触摸的时候（手指刚刚接触屏幕的瞬间），是否愿意成为响应者？
- View.props.onMoveShouldSetResponder: (evt) => true
- 如果View不是响应者，那么在每一个触摸点开始移动（没有停下也没有离开屏幕）时再询问一次：是否愿意响应触摸交互呢？

2.3 响应者-授权与弃权

- View.props.onResponderGrant: (evt) => {}
- View开始响应触摸事件，需要做高亮的时候，使用户知道到底点到哪里。
- View.props.onResponderReject: (evt) => {}
- 响应者现在“另有其人”而且暂时不会“放权”，请另作安排。

2.4 响应者-响应触摸事件

- View.props.onResponderMove: (evt) => {}
- 用户正在屏幕上移动手指时（没有停下也没有离开屏幕）。
- View.props.onResponderRelease: (evt) => {}
- 触摸操作结束时触发，比如"touchUp"（手指抬起离开屏幕）。
- View.props.onResponderTerminationRequest: (evt) => true
- 有其他组件请求接替响应者，当前的View是否“放权”？返回true的话则释放响应者权力。
- View.props.onResponderTerminate: (evt) => {}
- 响应者权力已经交出。这可能是由于其他View通过onResponderTerminationRequest请求的，也可能是由操作系统强制夺权（比如iOS上的控制中心或是通知中心）。

3 动画

3.1 动画库

- Animated:更精细的交互控制的动画
- LayoutAnimation:全局的布局动画

3.2 Animated

- 非常容易地实现各种各样的动画和交互方式。
- 具备极高的性能。
- 仅关注动画的输入与输出声明，建立可配置的变化函数。
- 使用简单的start/stop方法来控制动画按顺序执行。
- 整个配置都是声明式的，可以实现更进一步的优化，只要序列化好配置，可以在高优先级的线程执行动画。

3.3 Animated-核心API

- 两个值类型，Value用于单个的值，而ValueXY用于向量值；
- 三种动画类型，spring，decay，还有timing
- 三种组件类型，View，Text和Image。
- 使用Animated.createAnimatedComponent方法来对其它类型的组件创建动画。

3.4 Animated-spring

- 基础的单次弹跳物理模型，符合Origami设计标准。
- friction: 控制“弹跳系数”、夸张系数，默认为7。
- tension: 控制速度，默认40。

3.5 Animated-decay

. 以一个初始速度开始并且逐渐减慢停止。

. velocity: 起始速度，必填参数。

. deceleration: 速度衰减比例，默认为0.997。

3.6 Animated-timing

- 从时间范围映射到渐变的值。
- duration: 动画持续的时间（单位是毫秒），默认为500。
- easing: 一个用于定义曲线的渐变函数。阅读Easing模块可以找到许多预定义的函数。iOS默认为Easing.inOut(Easing.ease)。
- delay: 在一段时间之后开始动画（单位是毫秒），默认为0。

3.7 LayoutAnimation

- 在全局范围内创建和更新动画，这些动画会在下一次渲染或布局周期运行。
- 常用来更新flexbox布局，因为它可以无需测量或者计算特定属性就能直接产生动画。
- 当布局变化可能影响到父节点时，使用LayoutAnimation，就不需要显式声明组件的坐标，所有受影响的组件能够同步运行动画。
- 对动画本身的控制没有Animated或者其它动画库那样方便

3.8 setNativeProps

- setNativeProps 方法可以使我们直接修改基于原生视图的组件的属性，而不需要使用setState来重新渲染整个组件树。

- 如果动画丢帧（低于60帧每秒），可以使用setNativeProps或者shouldComponentUpdate来优化它们。

4 直接操作组件

4.1 setNativeProps

- 在React Native中，setNativeProps就是等价于直接操作DOM节点的方法。
- 在不得不频繁刷新而又遇到了性能瓶颈的时候。
- 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的大量开销。
- setNativeProps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。

4.2 避免和render方法的冲突

- 更新一个由render方法来维护的属性，可能会碰到一些bug。

- 每一次组件重新渲染都可能引起属性变化，这样一来，之前通过setNativeProps所设定的值就被完全忽略和覆盖掉。

4.3 shouldComponentUpdate /setState

- 通过运用 shouldComponentUpdate方法，可以避免重新渲染那些实际没有变化的子组件所带来的额外开销。

- 使setState的性能已经可以与setNativeProps相媲美了。

5 性能

5.1 缓慢的导航器切换

- Navigator的动画由JavaScript线程所控制。切换过程中，如果JavaScript线程卡住，动画就被卡住。
- 在动画的进行过程中，利用InteractionManager来选择性的渲染新场景所需的最小限度的内容。
- InteractionManager.runAfterInteractions的参数中的回调，会在navigator切换动画结束的时候被触发。
- 在场景切换完成之前，显示一个灰色的占位页面或者是一个转动的动画。
- 可以绘制部分的页面内容。

5.2 ListView初始化渲染太慢及列表过长

- initialListSize：定义首次渲染中绘制的行数。快速地显示出页面，可以设置initialListSize为1。
- pageSize：决定每一帧所渲染的行数。默认值为1。 如果页面很小，渲染的开销不大，值可以更大。
- scrollRenderAheadDistance：指定一个超过视野范围之外所需要渲染的行数。
- removeClippedSubviews：超出屏幕的子视图（同时overflow值为hidden）会从它们原生的父视图中移除。14.0版后默认为true。
- Android上，overflow值总是hidden。在iOS上，确保在行容器上设置overflow: hidden。

5.3 重绘变化非常少的页面

- ListView rowHasChanged函数，通过快速的算出某一行是否需要重绘，来减少不必要工作。

- 使用不可变的数据结构，只需检查引用是否相等。不可变的数据结构在提速方面非常有用 。可以使重绘整个组件更加快速，而且代码量更少。

- shouldComponentUpdate函数，指明在什么样的确切条件下，希望这个组件得到重绘。

- 编写返回值由props和state所决定的组件，可以用PureRenderMixin。

5.4 移动视图（滚动，切换，旋转）

- 透明背景的文本，位于一张图片上时，或者在每帧重绘视图时需要用到透明合成的任何其他情况下，会严重掉帧。
- 栅格化属性: shouldRasterize。
- 不要过度使用该特性，否则内存使用量将会飞涨。在使用时，要评估性能和内存使用情况。没有需要移动这个视图的需求，关闭这属性。

5.5 动画改变图片的尺寸

- 调整Image组件的宽度或者高度，尤其是大的图片，操作开销会比大。
- 使用transform:{{scale}}的样式属性来改变尺寸。

6 Redux架构

6.1 Redux开发环境搭建

- 安装Redux:npm install --save redux
- 安装React 绑定库:npm install --save react-redux
- 安装开发者工具:npm install --save-dev redux-devtools

6.2 完整的ReactNativeAPP文件结构

- 入口文件
- Actions文件
- Reducers文件
- 容器组件文件
- 展示组件文件

6.3 Action

- Action 是JavaScript普通对象(键值对)。
- 约定type字段的值表示动作。type的值定义为字符串常量。
- 用单独的模块或文件存放Action。
- 纯函数创建Action，不触发。
- 接收Action，实例化Dipatch。
- store.dispatch()/connect()/bindActionCreators()

6.4 Reducer

- 职责:定义State如何更新
- Reducer: (previousState, action) => newState
- 纯函数，接收Old State和Action，返回New State
- 严禁:修改传入参数，副作用操作(API请求、路由跳转)。
- 只要传入参数一样，返回必须一样。单纯执行计算。

6.5 Store 职责

- Redux应用只有一个store，使用reducer组合拆分数据
- 维持应用的 state。
- 提供 getState() 方法获取 state。
- 提供 dispatch(action) 方法更新 state。
- 通过 subscribe(listener) 注册监听器。

6.6 数据流

- Redux是严格单向数据流。
- 调用 store.dispatch(action)分发行为。
- Redux store 把当前的state树和收到的Action传入调用的 reducer 函数，返回新的state树。
- 根 reducer 把多个子 reducer 输出合并成一个单一的 state 树。
- Redux store 保存根 reducer 返回的完整 state 树。
- 组件通过setState(newState)方法更新state，触发UI更新。

Thank you!!!