# React Native

## 一、关于React Native

## 二、React Native环境搭建（windows）

## 三、React Native专项

1、React Native样式定义

2、React Native图像

3、React组件生命周期

4、React Native ES6

5、React Native动画

6、React Native调试

7、Android APK签名

## 四、React Native 组件（Component）

（一）View组件

（二）Text组件

（三）TouchableHighlight/TouchableOpacity组件

（四）TouchableWithoutFeedback/TouchableNativeFeedback组件

（五）Image组件

（六）TextInput组件

（七）Switch组件

（八）Slider

（九）Picker

（十）Modal

（十一）ListView

（十二）ScrollView

（十三）RefreshControl

（十四）WebView

（十五）MapView

（十六）ProgressBarAndroid

（十七）StatusBar

（十八）ToolbarAndroid

（十九）Navigator

（二十）DrawerLayoutAndroid

## 一、关于React Native



### 1、React Native 是什么？

*React Native enables you to build world-class application experiences on native platforms using a consistent developer experience based on JavaScript and React. The focus of React Native is on developer efficiency across all the platforms you care about — learn once, write anywhere. Facebook uses React Native in multiple production apps and will continue investing in React Native.*

官方网站：<http://facebook.github.io/react-native/>

React Native是一个可以用Javascript语言来编写APP的框架。React Native使用React作为视图渲染层，使用并封装设备原生API，不同于一些基于Webview的跨平台框架（如Ionic等），React Native的组件渲染和设备调用都是以原生形式生成，不需要引入其他的第三方插件，这让React Native应用的响应和体验更接近于原生应用。

React Native可以开发跨平台APP（IOS/Android），但并不是不同平台共用一套代码，而是需要为不同的平台编写各自一套代码。但是，这并不意味着使用React Native开发APP就一定低效难维护，相反地， React Natvie提供了统一规范的语法，使得在不同平台下的代码编写都变得一致；在设备API调用方面，React Native同样进行了封装，使不同平台的调用都趋于一致。这就使得开发人员不需要为不同平台的应用转换使用不同的技术或语言，同时，在不同的平台中，可以针对不同平台渲染不同的UI，甚至执行使用有差异的功能，最大程度保持了应用开发的灵活性。

正如React Native官网所说的，React Native的对跨平台应用开发的定义并不是“write once, run anywhere”，而是“learn once,write anywhere”。

### 2、React Native / Ionic

新的技术，好的技术总会引起关注和讨论，于是，在原生应用（Native App）与混合应用（Hybrid App）之间，无数人进行着对比和争论，而在混合应用（Hybrid App）的世界里，又同样有无数人进行对比和争论。

关于原生应用（Native App）与混合应用（Hybrid App），我们不打算探讨，因为网上大把的资料，我们把重点放在目前混合应用开发最最热门的两项技术，同时也代表两个不同的方向，就是React Native和Ionic之间，当然，关于这两项技术，网上也有很多的资料，但是为了更好的理解React Native，在学习和开发React Native时，真正能“Think in React Native”，在这里还是简单的作下介绍，更详细对比资料可以在网上搜索。

在混合应用（Hybrid App）开发方面，有很多不同的框架，目前最火热的当数React Native和Ionic。

1、在实现上，React Native的组件渲染是基于原生的，而Ionic是基于WebView，因此，在体验方面，React Native体验更接近（或者说就是原生应用）的体验；

2、UI渲染上，React Native使用React组件化形式，封装实现原生应用UI，Ionic使用HTML 5 布局渲染；

3、在逻辑上，React Native基于React，而Ionic基于Angular JS；

4、在设备调用上，React Native调用原生设备API，而Ionic要依靠Cordova来调用设备API；

5、在开发实现上，React Native需要为不同的平台开发各自平台的代码，而Ionic则是一套代码，可以在不同平台上使用，这也牵出另一个对比——开发效率；

6、在开发效率上，React Native要比Ionic慢很多。

虽然React Native封装了一些常用的组件方便开发者使用，但所有的组件都是基础功能，在实际开发使用中，基本上都需要开发者重新封装，而且，React Native使用React来渲染组件和UI，在React Native，只提供了基本的Flexbox布局和常用的一些style样式属性，没有像HTML5中的CSS样式那么丰富，个性化定制略差；而在开发速度上，React Native使用JSX语法，并不像HTML语言那么简单易懂。

Ionic使用HTML5来实现UI，因此，书写语法是HTML5，样式支持是CSS，所有HTML5（标签、属性等）和CSS样式都可以完全支持并使用，书写更简单，开发效率更高。

在开发效率上，React Native要比Ionic差，但是React Native组件化开发的概念正是现在应用开发所推崇的，也是我们应该提倡的，当应用开发趋于成熟，组件化开发可以让组件使用模块化，大大提高组件重用性。

7、在性能和用户体验上，无疑React Native会更胜一筹。

8、在开发成本上，React Native要高于Ionic，这个简单明了，不解释。

9、在学习成本和学习曲线上。React Native要比Ionic难，学习成本更高，门槛也更高。

10、在成熟度和发展空间上，React Native刚推出不久，尽管目前各种坑也在不断地暴露，而Facebook将React Native开源的做法，也让React Native的未来发展空间更具有无限的可能性和暇想空间；Ionic所依靠的HTML5、Cordova、Angular JS等技术已都是非常成熟了。

**总结：React Native是混合应用（Hybrid App）开发的未来，甚至是App开发的未来。**

*关于这部分的话题，可以参考以下链接：*

Ionic Framework vs React Native

https://medium.com/react-id/ionic-framework-hybrid-app-vs-react-native-4facdd93f690#.ktrq0ivzx

Native vs Ionic vs Nativescript vs React Native

https://www.trustedhousesitters.com/engineering/news/hackathon-jan-2016-native-vs-ionic-vs-nativescript-vs-react-native/

Ionic vs React Native — Text Blast Case Study

https://medium.com/@fisch/ionic-vs-react-native-9fa307a9f0e1#.ptjme5ind

### 3、资源

**（1）文档**

React Native 官方文档

https://facebook.github.io/react-native/

React Native中文文档（极客学院）

http://wiki.jikexueyuan.com/project/react-native/

React Native 中文网

http://reactnative.cn/

**（2）基本**

React Native For Beginners – The Next Big Thing?

https://devdactic.com/react-native-for-beginners/

React-Native layout examples

http://browniefed.com/blog/react-native-layout-examples/

Cold Dive into React Native: A Beginner's Tutorial

https://www.toptal.com/ios/cold-dive-into-react-native-a-beginners-tutorial

Why You Should Consider React Native For Your Mobile App

https://www.smashingmagazine.com/2016/04/consider-react-native-mobile-app/

Beginning Mobile App Development with React Native

http://beginning-mobile-app-development-with-react-native.com/book-preview.html

React Native Tutorial: Building Apps with JavaScript

https://www.raywenderlich.com/126063/react-native-tutorial

Execution sequence of a React component’s lifecycle methods

http://javascript.tutorialhorizon.com/2014/09/13/execution-sequence-of-a-react-components-lifecycle-methods/

The Component Lifecycle

https://www.kirupa.com/react/component\_lifecycle.htm

React Native 中组件的生命周期

http://www.race604.com/react-native-component-lifecycle/

深入浅出 React Native：使用 JavaScript 构建原生应用

https://zhuanlan.zhihu.com/p/19996445

react-native 之布局篇

https://segmentfault.com/a/1190000002658374

React Native 基础练习指北

https://segmentfault.com/a/1190000002645929

https://segmentfault.com/a/1190000002647733

React Native通信机制详解

http://blog.cnbang.net/tech/2698/

ReactNative 开发实践

http://www.jianshu.com/p/6d4cce9d914f

**（3）学习专题**

USE REACT NATIVE

http://www.reactnative.com/

React Native 入门指南

https://github.com/vczero/react-native-lesson

江清清博客React Native专题

http://www.lcode.org/react-native/

**（4）专项文章**

React Native ListView with Section Headers

http://moduscreate.com/react-native-listview-with-section-headers/

Dive into React Native performance

https://code.facebook.com/posts/895897210527114/dive-into-react-native-performance/

Making React Native apps accessible

https://code.facebook.com/posts/435862739941212/making-react-native-apps-accessible/

BUILDING A NATIVE SOUNDCLOUD ANDROID APP WITH REACT NATIVE AND REDUX

https://wiredcraft.com/blog/native-soundcloud-android-app/

React Native Animations

http://blog.huynh.io/2015/08/06/react-native-animations/

React-native Animated API Basic Example

http://browniefed.com/blog/react-native-animated-api-basic-example/

The Shapes of React Native

http://browniefed.com/blog/the-shapes-of-react-native/

React-native press and hold button actions

http://browniefed.com/blog/react-native-press-and-hold-button-actions/

新手理解Navigator的教程

http://bbs.reactnative.cn/topic/20/%E6%96%B0%E6%89%8B%E7%90%86%E8%A7%A3navigator%E7%9A%84%E6%95%99%E7%A8%8B

**（5）组件**

https://js.coach/

**（6）其他**

https://rnplay.org/

## 五、React Native 组件（Component）

#### （一）View组件

<View>组件是React Native中最基本的组件，它是一个布局窗口，几乎所有的其他组件在使用时都要配合<View>组件一起使用。在React Native中，<View>可以嵌套使用，并且在渲染组件的时候，只能有一个<View>根组件，其他所有组件都必须包含在这个根组件当中。如：

'use strict';

import React, {

Component,

StyleSheet,

View,

Text,

TouchableHighlight,

} from 'react-native';

class ViewDemo extends Component {

render() {

return (

<View style={styles.container}>

<View style={styles.box}>

<Text>这是一个包含于VIew组件中的Text文本组件</Text>

<TouchableHighlight>

<Text>可触控高亮组件</Text>

</TouchableHighlight>

<View style={[styles.box,styles.innerBox]}>

<Text>这是一个嵌套View组件</Text>

</View>

</View>

<View style={[styles.box,styles.background,styles.borderRadiusT]}>

<Text>设置背景颜色，边框和边框圆角</Text>

</View>

<View style={[styles.box,styles.background,styles.opacityT]}>

<Text>设置背景颜色，透明度</Text>

</View>

<View style={styles.subContainer}>

<View style={styles.item}>

<Text>项目一</Text>

</View>

<View style={styles.item}>

<Text>项目二</Text>

</View>

<View style={styles.item}>

<Text>项目三</Text>

</View>

<View style={styles.item}>

<Text>项目四</Text>

</View>

</View>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

box:{margin:10,padding:10,borderWidth:1,borderColor:'#0a8acd',borderStyle:'solid'},

innerBox:{backgroundColor:'#ccc',margin:10},

background:{backgroundColor:'#fff333'},

borderRadiusT :{borderRadius:15},

opacityT:{opacity:0.5},

subContainer:{flexDirection: 'row',margin:10,justifyContent:'center',alignItems:'center'},

item:{flex:1,height:100,backgroundColor:'#ddd',padding:10,borderWidth:1,justifyContent:'center',alignItems:'center',alignSelf:'center'}

})

module.exports = ViewDemo;



#### （二）Text组件

Text组件是另一个非常常用的组件，使用非常简单。要点有：

1、Text组件同样可以嵌套使用。

2、在任何需要使用文字的地方，都需要使用Text组件包含，不能直接在View组件下直接包含文字，如以下代码将报错：

<View>

文字内容

</View>

正确的写法是：

<View>

<Text>

文字内容

</Text>

</View>

在View组件中已经讲过，在渲染组件的时候，只能有一个<View>根组件，其他所有组件都必须包含在这个根组件当中，因此，上面正确的代码中，如果Text组件已经包含在某个View组件或其他组件中时，上面代码中的View组件包含不是必须的。

3、Text组件的样式可以继承，但仅限于子Text组件中

4、numberOfLines属性，用来控制显示文本的行数

<Text numberOfLines={2}>这是一段长文本，这真是一段长文本，这真是一段很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本。</Text>

5、onPress属性，传入一个函数，当文本被点击按下时触发函数

<Text onPress={this.onPressMe.bind(this)}>点击我</Text>

Text组件示例代码：

'use strict';

import React, {

Component,

StyleSheet,

View,

Text,

TouchableHighlight,

} from 'react-native';

class TextDemo extends Component {

constructor(props){

super(props);

this.state = {

text:'点我点我'

}

}

onPressMe(){

this.setState({

text: '叫你点你就点，你傻呀！',

});

}

render() {

return (

<View style={styles.container}>

<Text style={styles.title}>Text组件</Text>

<Text>

父Text组件

<Text>子Text组件</Text>

</Text>

<View style={styles.divider}></View>

<View style={styles.box}>

<Text>普通文本</Text>

<Text style={styles.colorText}>文字颜色</Text>

<Text style={styles.fontFamilyText}>文字字体</Text>

<Text style={styles.fontSizeText}>文字大小</Text>

<Text style={styles.fontStyleText}>文字样式-斜体</Text>

<Text style={styles.fontWeightText}>粗体文字</Text>

<Text style={styles.lineHeightText}>文字行高</Text>

<Text style={styles.textShadow}>文字阴影</Text>

</View>

<View style={styles.divider}></View>

<Text style={{fontStyle:'italic'}}>

父Text组件的样式是斜体

<Text style={{color:'#0a8acd'}}>子Text组件继承了父Text组件的“斜体”样式</Text>

</Text>

<View style={styles.divider}></View>

<Text>这是一段长文本，这真是一段长文本，这真是一段很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本。</Text>

<Text numberOfLines={2} style={{color:'red'}}>这里只显示两行：但这是一段长文本，这真是一段长文本，这真是一段很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本，这真的是一段很长很长的文本。</Text>

<View style={styles.divider}></View>

<Text onPress={this.onPressMe.bind(this)}>{this.state.text}</Text>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

box:{paddingLeft:10,paddingRight:10},

colorText:{color:'#fff100'},

fontFamilyText:{fontFamily:'微软雅黑'},

fontSizeText:{fontSize:40},

fontStyleText:{fontStyle:'italic'},

fontWeightText:{fontWeight:'700'},

lineHeightText:{flex:1,lineHeight:50,backgroundColor:'#ddd',textAlignVertical:'center'},

textShadow:{textShadowColor:'#444',textShadowOffset:{width:2,height:2},textShadowRadius:4}

})

module.exports = TextDemo;



#### （三）TouchableHighlight/TouchableOpacity组件

TouchableHightlight组件是一个触摸（点击）可响应的容器，当用户触摸或点击带有TouchableHightlight组件包含的区域或实体时，会响应TouchableHighlight组件中onPress属性指定的函数，并且当被点击时，组件包含的区域会高亮显示，高亮颜色由组件的underlayColor属性指定。相当于HTML语言中的<A>标签。

注意的是TouchableHighlight组件中只能包含一个基本子组件，如果有多个，可以将多个子组件包含在<View>组件中，再把<View>组件包含在TouchableHighlight组件中。

TouchableHightlight组件的属性：

1、activeOpacity：指定数字类型属性，用于指定触摸（点击）被激活时的透明程度。

2、underlayColor：指定一个颜色字符属性，定义触摸（点击）被激活时的背景颜色。

3、onShowUnderlay:指定一个函数，当背景颜色显示时触发。

4、onHideUnderlay:指定一个函数，当背景颜色隐藏时触发。

TouchableOpacity组件与TouchableHightlight组件非常类似，TouchableOpacity组件用于显示触摸（点击）有透明效果的交互。TouchableOpacity组件的activeOpacity属性用于指定触摸（点击）被激活时的透明程度。

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

TouchableHighlight,

TouchableOpacity

} from 'react-native';

class TouchableHighlightOpacity extends Component {

constructor(props){

super(props);

this.state = {

basic:'一个基本的按钮',

colorBtn:'触摸变色按钮',

color:'#0a8acd'

}

}

onPressBasic(){

this.setState({

basic: '你点我干嘛呀'

});

}

onPressColorBtn(){

this.setState({

colorBtn:'我是变色龙'

});

}

onShowUnderlayFunc(){

this.setState({

color:'#fff'

});

}

onHideUnderlayFun(){

this.setState({

color:'#000'

});

}

render() {

return (

<View style={styles.container}>

<Text style={styles.title}>TouchableHighlight组件</Text>

<TouchableHighlight onPress={this.onPressBasic.bind(this)} style={styles.basic}>

<Text style={{textAlign:'center'}}>{this.state.basic}</Text>

</TouchableHighlight>

<TouchableHighlight underlayColor="red" onPress={this.onPressColorBtn.bind(this)} style={styles.basic}>

<Text style={{textAlign:'center'}}>{this.state.colorBtn}</Text>

</TouchableHighlight>

<TouchableHighlight underlayColor="red" onShowUnderlay={this.onShowUnderlayFunc.bind(this)} onHideUnderlay={this.onHideUnderlayFun.bind(this)} onPress={this.onPressColorBtn.bind(this)} style={styles.basic}>

<Text style={[{textAlign:'center'},{color:this.state.color}]}>{this.state.colorBtn}</Text>

</TouchableHighlight>

<TouchableHighlight underlayColor="red" activeOpacity={0.5} onShowUnderlay={this.onShowUnderlayFunc.bind(this)} onHideUnderlay={this.onHideUnderlayFun.bind(this)} onPress={this.onPressColorBtn.bind(this)} style={styles.basic}>

<Text style={[{textAlign:'center'},{color:this.state.color}]}>{this.state.colorBtn}</Text>

</TouchableHighlight>

<View style={styles.divider}></View>

<Text style={styles.title}>TouchableOpacity组件</Text>

<TouchableOpacity onPress={this.onPressBasic.bind(this)} activeOpacity={0.5} style={styles.basic}>

<Text style={{textAlign:'center'}}>{this.state.basic}</Text>

</TouchableOpacity>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

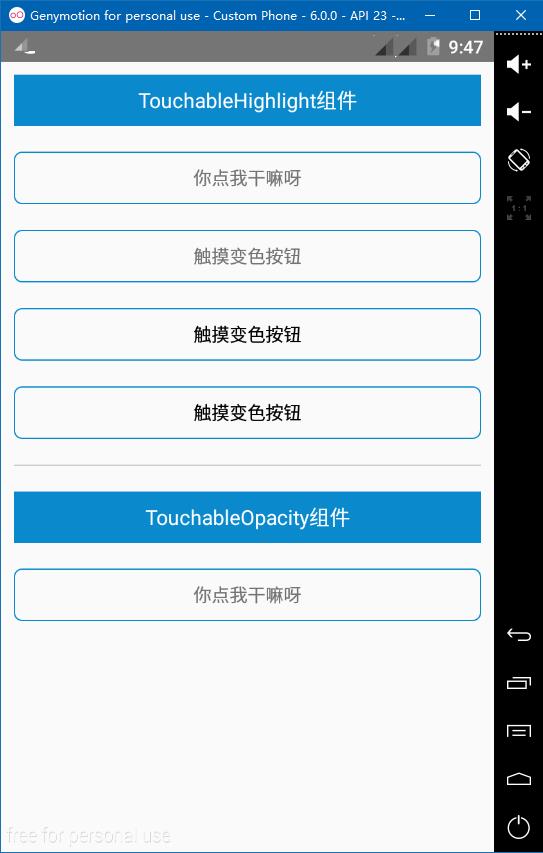
divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

basic:{margin:10,padding:10,borderWidth:1,borderColor:'#0a8acd',borderStyle:'solid',borderRadius:6},

})

module.exports = TouchableHighlightOpacity;



#### （四）TouchableWithoutFeedback/TouchableNativeFeedback组件

TouchableWithoutFeedback组件也是一个触摸（点击）可响应的容器，当用户触摸或点击带有TouchableHightlight组件包含的区域或实体时，会响应TouchableHighlight组件中onPress属性指定的函数，TouchableWithoutFeedback组件不能指定style样式，只能在其包含的子组件中使用。

与TouchableHighlight组件一样，TouchableWithoutFeedback组件只能包含一个基本子组件，如果有多个，可以将多个子组件包含在<View>组件中，再把<View>组件包含在TouchableHighlight组件中。

常用属性：

1. delayLongPress:指定数字类型属性，指定长按事件的延迟（毫秒数）
2. delayPressIn:指定数字类型属性，指定触摸（点击）进入事件的延迟（毫秒数）.
3. delayPressOut:指定数字类型属性，指定触摸（点击）退出事件的延迟（毫秒数）
4. disabled:指定布尔类型属性，如果为true，则禁用所有的交互事件
5. hitSlop：指定一个对象属性 {top: number, left: number, bottom: number, right: number}，定义响应外延范围。
6. onLongPress:指定一个函数，当长按时触发。
7. onPress:指定一个函数，当被触摸（点击）时触发。
8. onPressIn:指定一个函数，当触摸（点击）进入时触发。
9. onPressIOut:指定一个函数，当触摸（点击）退出时触发。

TouchableNativeFeedback组件利用设置的原生形态来渲染触摸的反馈效果，这个组件同样只能包含一个子节点。TouchableNativeFeedback组件通过background属性来指定原生触摸操作反馈的效果背景，background属性类型有三种：

1. TouchableNativeFeedback.SelectableBackground()：安卓默认的对象被选中时的背景
2. TouchableNativeFeedback.SelectableBackgroundBorderless()：当被按下时产生一个无边框的动画背景
3. TouchableNativeFeedback.Ripple(color, borderless) ：指定颜色（color）和边框（borderless），当被按下时产生一个动画背景，如果参数borderless是true，那么动画会被渲染到按钮视图范围外。

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

TouchableWithoutFeedback,

TouchableNativeFeedback

} from 'react-native';

class TouchableWithoutFeedbackDemo extends Component {

constructor(props){

super(props);

this.state = {

basic:'TouchableWithoutFeedback按钮',

colorBtn:'触摸变色按钮',

color:'#0a8acd',

nativeColor:'red'

}

}

onPressBasic(){

this.setState({

basic: '你点我干嘛呀'

});

}

onPressFunc(){

this.setState({

colorBtn: '点击进入退出'

});

}

onPressInFunc(){

this.setState({

color:'red'

});

}

onPressOutFunc(){

this.setState({

color:'blue'

});

}

onPressNative(){

this.setState({

nativeColor:'blue'

});

}

render() {

return (

<View style={styles.container}>

<Text style={styles.title}>TouchableWithoutFeedback组件</Text>

<TouchableWithoutFeedback onPress={this.onPressBasic.bind(this)}>

<View style={styles.basic}>

<Text style={{textAlign:'center'}}>{this.state.basic}</Text>

</View>

</TouchableWithoutFeedback>

<TouchableWithoutFeedback onPress={this.onPressFunc.bind(this)} onPressIn={this.onPressInFunc.bind(this)} onPressOut={this.onPressOutFunc.bind(this)}>

<View style={styles.basic}>

<Text style={[{textAlign:'center'},{color:this.state.color}]}>{this.state.colorBtn}</Text>

</View>

</TouchableWithoutFeedback>

<TouchableWithoutFeedback delayPressIn={1000} delayPressOut={1000} onPress={this.onPressFunc.bind(this)} onPressIn={this.onPressInFunc.bind(this)} onPressOut={this.onPressOutFunc.bind(this)}>

<View style={styles.basic}>

<Text style={[{textAlign:'center'},{color:this.state.color}]}>{this.state.colorBtn}</Text>

</View>

</TouchableWithoutFeedback>

<TouchableWithoutFeedback onPress={this.onPressBasic.bind(this)} disabled={true}>

<View style={styles.basic}>

<Text style={{textAlign:'center'}}>这个按钮点击不了</Text>

</View>

</TouchableWithoutFeedback>

<View style={styles.divider}></View>

<TouchableWithoutFeedback hitSlop={{top:30,left:30,bottom:30,right:30}} onPress={this.onPressFunc.bind(this)} onPressIn={this.onPressInFunc.bind(this)} onPressOut={this.onPressOutFunc.bind(this)}>

<View style={{backgroundColor:'#ccc'}}>

<Text style={[{textAlign:'center'},{color:this.state.color}]}>{this.state.colorBtn}</Text>

</View>

</TouchableWithoutFeedback>

<Text style={styles.title}>TouchableNativeFeedback组件</Text>

<TouchableNativeFeedback onPress={this.onPressNative.bind(this)} background={TouchableNativeFeedback.SelectableBackground()}>

<View style={styles.basic}>

<Text style={[{textAlign:'center'},{color:this.state.nativeColor}]}>{this.state.colorBtn}</Text>

</View>

</TouchableNativeFeedback>

<TouchableNativeFeedback onPress={this.onPressNative.bind(this)} background={TouchableNativeFeedback.SelectableBackgroundBorderless()}>

<View style={styles.basic}>

<Text style={[{textAlign:'center'},{color:this.state.nativeColor}]}>{this.state.colorBtn}</Text>

</View>

</TouchableNativeFeedback>

<TouchableNativeFeedback onPress={this.onPressNative.bind(this)} background={TouchableNativeFeedback.Ripple('red', true)}>

<View style={styles.basic}>

<Text style={[{textAlign:'center'},{color:this.state.nativeColor}]}>{this.state.colorBtn}</Text>

</View>

</TouchableNativeFeedback>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

basic:{margin:10,padding:10,borderWidth:1,borderColor:'#0a8acd',borderStyle:'solid',borderRadius:6},

})

module.exports = TouchableWithoutFeedbackDemo;



#### （五）Image组件

React Native提供了Image组件用来显示图片，图片来源可以本地，也可以是网络上的图片。

常用属性：

1. source:指定图片的来源，可以是图片网址或本地图片路径。如果是本地图片，source指定的路径格式为require(‘./img/image.png’)
2. resizeMode:指定图片的显示方式，属性值为枚举类型，属性值分别为”cover”（默认）、”contain”和”stretch”。这三个属性值指定的图片显示方式与图片在web中的显示方式类似，cover为适应缩放，contain为全图包含显示，stretch为真实图片截取显示。

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

Image

} from 'react-native';

class ImageDemo extends Component {

constructor(props){

super(props);

this.state = {

status:'',

result:''

};

}

startFunc(){

this.setState({

status : '开始加载网络图片...'

});

}

endFunc(){

this.setState({

status : '网络图片加载完成.'

});

}

onLoadFunc(){

this.setState({

result : '加载结果'

});

}

render() {

return (

<View style={styles.container}>

<Text style={styles.title}>Image组件</Text>

<Text>使用网络图片</Text>

<Image style={{width:100,height:100,borderWidth:1,borderColor:'#ccc'}} source={{uri: 'http://facebook.github.io/react/img/logo\_og.png'}} />

<Text>使用本地图片</Text>

<Image style={{width:100,height:100,borderWidth:1,borderColor:'#ccc'}} source={require('./img/beautiful.jpg')} />

<View style={{flexDirection:'row',justifyContent:'center',alignItems:'center'}}>

<View style={{flex:1}}>

<Text>resizeMode-cover</Text>

<Image style={{width:120,height:100,borderWidth:1,borderColor:'#ccc'}} resizeMode="cover" source={require('./img/beautiful.jpg')} />

</View>

<View style={{flex:1}}>

<Text>resizeMode-contain</Text>

<Image style={{width:120,height:100,borderWidth:1,borderColor:'#ccc'}} resizeMode="contain" source={require('./img/beautiful.jpg')} />

</View>

<View style={{flex:1}}>

<Text>resizeMode-stretch</Text>

<Image style={{width:120,height:100,borderWidth:1,borderColor:'#ccc'}} resizeMode="stretch" source={require('./img/beautiful.jpg')} />

</View>

</View>

<View>

<Image style={{width:150,height:150,borderWidth:1,borderColor:'#ccc'}} source={{uri: 'http://hdimagesnew.com/wp-content/uploads/2015/11/biRjno7iy.jpg'}} onLoadStart={this.startFunc.bind(this)} onLoadEnd={this.endFunc.bind(this)} onLoad={this.onLoadFunc.bind(this)} />

<Text>{this.state.status}</Text>

<Text>{this.state.result}</Text>

</View>

</View>

);

}

}

const styles = StyleSheet.create({

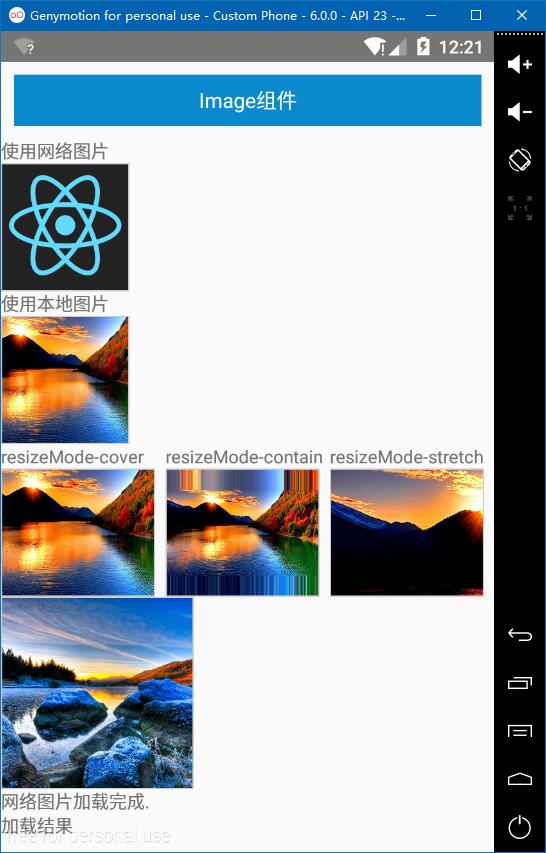
container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

})

module.exports = ImageDemo;



#### （六）TextInput组件

TextInput是用一个文本输入框组件。

TextInput常用属性：

1、autoCapitalize：设置输入框自动大写提示，可选值有'none'、'sentences'、'words'、'characters'；

2、autoCorrect：如果设置为false，则禁用自动更正，默认为true；

3、autoFocus：设置是否自动获得焦点，默认为false；

4、blurOnSubmit：设置提交时是否失去焦点。当文本框为单行时默认值为true，当文本框为多行时默认值为false。

5、defaultValue：设置文本框初始的Value值，该值只有当用户开始输入时才会改变；

6、editable：设置文本框是否可编辑，默认值为true；

7、keyboardType：设置点击文本框（文本框获得焦点）时出现键盘的类型，可选值有'default', 'email-address', 'numeric', 'phone-pad', 'ascii-capable', 'numbers-and-punctuation', 'url', 'number-pad', 'name-phone-pad', 'decimal-pad', 'twitter', 'web-search'；

8、maxLength：设置文本框接受输入的最大字符数量

9、multiline：设置文本框是否可以多行输入；

10、onBlur：文本框失去焦点时触发事件函数

11、onChange：文本框内容变化时触发事件函数

12、onChangeText：文本框内容变化时触发事件函数，与onChange不同的是，onChange把文本框内容作为参数传递给触发事件函数。

13、onFocus：文本框获得焦点时触发事件函数

14、onSelectionChange：当文件框内容选择变化时触发事件函数

15、onSubmitEditing：当提交按钮按下时触发事件函数，如果为多行文本框则无效；

16、placeholder：文本框提示信息

17、placeholderTextColor：文本框提示信息颜色

18、secureTextEntry：设置是否以安全保护形式输入，类似HTML中的密码输入框

19、selectTextOnFocus：设置是否在文本框获得焦点时自动选择全部文本框内容

20、selectionColor：设置文本框内容的高亮颜色

21、value：设置文本框的value值

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

TextInput

} from 'react-native';

class TextInputDemo extends Component {

constructor(){

super();

this.state = {

bgColor:'red'

}

}

onFocusFunc(){

this.setState({

bgColor:'#0a8acd'

});

}

onBlurFunc(){

this.setState({

bgColor:'red'

});

}

render() {

return (

<View>

<Text style={styles.title}>TextInput组件</Text>

<TextInput style={styles.input} onChangeText={(text) => this.setState({text})} value={this.state.text} />

<Text>{this.state.text}</Text>

<TextInput style={styles.inputBorder} autoCapitalize='words' />

<TextInput style={styles.inputBorder} autoCorrect={false} autoFocus={true} />

<TextInput style={styles.inputBorder} editable={false} placeholder='这个文本框不能编辑' />

<TextInput style={styles.inputBorder} maxLength={6} placeholder='最大长度6位' placeholderTextColor="blue" />

<TextInput style={styles.inputBorder} multiline={true} placeholder='多行文本框' />

<TextInput style={styles.inputBorder} secureTextEntry={true} />

<TextInput style={styles.inputBorder,{backgroundColor:this.state.bgColor}} onFocus={this.onFocusFunc.bind(this)} onBlur={this.onBlurFunc.bind(this)}/>

<TextInput style={styles.inputBorder} selectTextOnFocus={true} placeholder="当获得焦点时自动选择" selectionColor='#fff' />

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

input:{height:40},

inputBorder:{height:40,borderBottomWidth:1,borderBottomColor:'#0a8acd'}

})

module.exports = TextInputDemo;



#### （七）Switch组件

Switch组件是一个单选切换按钮（true||false），Switch组件在切换状态时会响应onValueChange属性指定的回调函数。

Switch组件的常用属性有：

1. disabled：设置Switch组件是否可用；
2. onValueChange：当Switch组件切换时触发事件函数；
3. value：Switch组件的值，为true时Switch组件呈开启状态，为false时呈关闭状态；

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

Switch

} from 'react-native';

class SwitchDemo extends Component {

constructor(){

super();

this.state = {

show:false

};

}

renderView(){

if(this.state.show){

return (

<View style={styles.box}>

<Text>你看到我了</Text>

</View>

);

}else{

return false;

}

}

render() {

return (

<View style={styles.container}>

<Text style={styles.title}>Switch组件</Text>

<Switch

onValueChange={(value) => this.setState({show: value})}

value={this.state.show} />

<Switch

style={{alignSelf:'center'}}

onValueChange={(value) => this.setState({show: value})}

value={this.state.show} />

<Switch

style={{alignSelf:'flex-start'}}

onValueChange={(value) => this.setState({show: value})}

value={this.state.show} />

{this.renderView()}

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

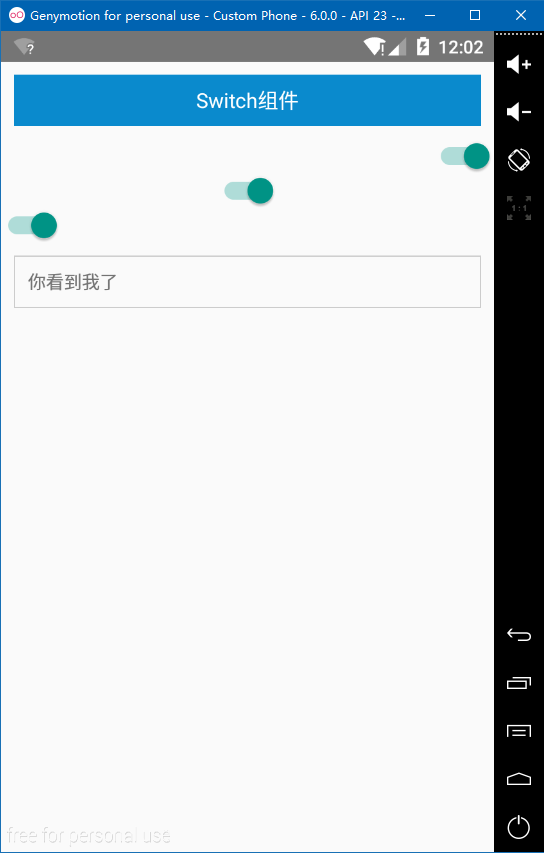
divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

box:{borderWidth:1,borderColor:'#ccc',padding:10,margin:10}

})

module.exports = SwitchDemo;



#### （八）Slider组件

Slider组件是一个新的组件，只有React Native 0.24以上版本才有。Slider组件用于让用户更方便的在一定的数值范围内选择。

Slider组件的主要属性有：

1、disabled：是否禁用Slider组件，如果为true则Slider组件无法使用；

2、maximumValue：设置Slider组件的最大值，默认为1；

3、minimumValue：设置Slider组件的最小值，默认为0；需注意的是，如果minimumValue如果设置为大于0的其他数值，则必须指定step属性；

4、Value：设置Slider组件的初始值，在使用过程中获得Slider的值。在使用Value设置初始值时，初始值Value必须在maximumValue和minimumValue之间；如果没有设置maximumValue和minimumValue，则Value初始值为0；

5、step：设置每次滑动的步长值；

6、onValueChange：当Slider组件Value值变化时实时调用此函数；

7、onSlidingComplete：当Slider组件每次拖动完成时调用此函数。

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

Slider

} from 'react-native';

class SliderDemo extends Component {

constructor(){

super();

this.state = {

s:0.3,

c:2,

d:5,

e:0

};

}

render() {

return (

<View style={styles.container}>

<Text style={styles.title}>Slider组件</Text>

<Slider disabled={true}/>

<View style={styles.divider}></View>

<Slider value={this.state.s} onValueChange={(value) => this.setState({s: value})} />

<Text>{this.state.s}</Text>

<View style={styles.divider}></View>

<Slider value={2} maximumValue={10} minimumValue={0} />

<View style={styles.divider}></View>

<Slider value={this.state.c} maximumValue={10} minimumValue={1} step={1} onValueChange={(value) => this.setState({c: value})} />

<Text>{this.state.c}</Text>

<View style={styles.divider}></View>

<Slider value={this.state.d} maximumValue={10} minimumValue={1} step={1} onSlidingComplete={(value) => this.setState({e: value})} />

<Text>{this.state.e}</Text>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

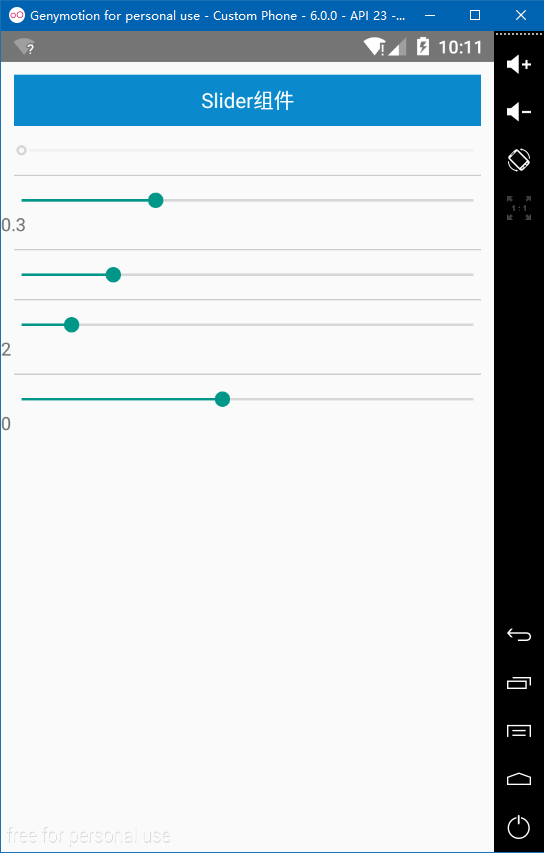
divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

box:{borderWidth:1,borderColor:'#ccc',padding:10,margin:10}

})

module.exports = SliderDemo;



#### （九）Picker

Picker渲染一个单选组件。

常用属性有：

1、onValueChange：当选择单选组件时调用函数，调用函数时会传入两个参数，分别为itemValue代表单选项的value值，以及itemPosition代表单选项的位置索引（从0开始）

2、selectedValue：默认选中的单选项

3、style：单选组件样式

4、enabled：单选组件是否生效，仅Android下可用

5、mode：单选组件的呈现方式，仅Android下可用。可选有“dialog”对话框模式（默认）和“dropdown”下拉列表模式。

6、prompt：设置单选组件以“dialog”对话框模式显示时的标题

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

Picker,

TouchableHighlight

} from 'react-native';

class PickerDemo extends Component {

constructor(props){

super(props);

this.state = {

gender:'man',

index:null

};

}

render() {

return (

<View>

<Text style={styles.title}>Picker组件</Text>

<Picker

selectedValue={this.state.gender}

onValueChange={(itemValue,itemPosition) => this.setState({gender: itemValue,index:itemPosition})}>

<Picker.Item label="男" value="man" />

<Picker.Item label="女" value="woman" />

</Picker>

<Text>选择了：{this.state.index}-{this.state.gender}</Text>

<View style={styles.divider}></View>

<Picker

enabled={false}

selectedValue={this.state.gender}

onValueChange={(gender) => this.setState({gender: gender})}>

<Picker.Item label="男" value="man" />

<Picker.Item label="女" value="woman" />

</Picker>

<View style={styles.divider}></View>

<Picker

mode='dropdown'

selectedValue={this.state.gender}

onValueChange={(gender) => this.setState({gender: gender})}>

<Picker.Item label="男" value="man" />

<Picker.Item label="女" value="woman" />

</Picker>

<View style={styles.divider}></View>

<Picker

style={{paddingHorizontal:10,backgroundColor:'#0a8acd',color:'#fff'}}

prompt='请选择性别'

selectedValue={this.state.gender}

onValueChange={(gender) => this.setState({gender: gender})}>

<Picker.Item label="男" value="man" />

<Picker.Item label="女" value="woman" />

</Picker>

</View>

);

}

}

const styles = StyleSheet.create({

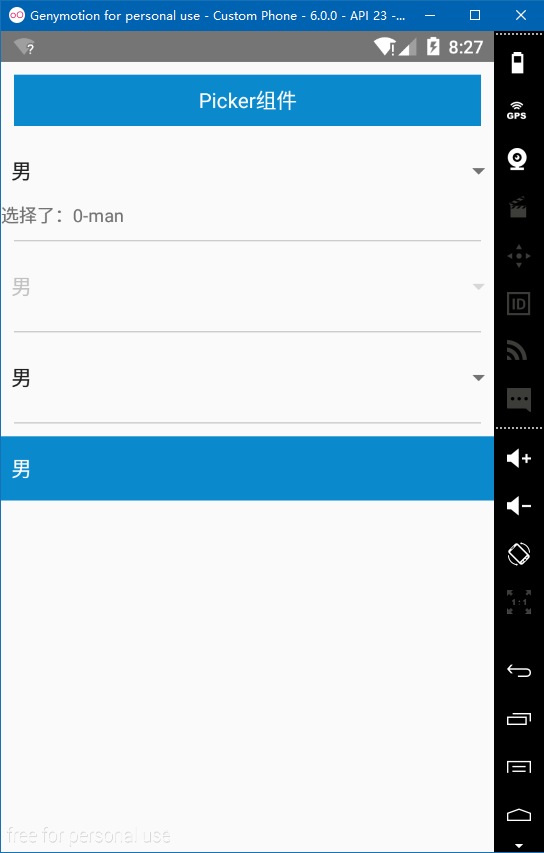
container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

})

module.exports = PickerDemo;



#### （十）Modal组件

Modal模态组件是APP的常用组件之一。

Modal组件的常用属性：

1、animated：定义显示、关闭Modal组件时是否使用动画效果；

2、onRequestClose：定义关闭Modal组件调用函数；

3、onShow：定义当Modal显示时调用的函数；

4、transparent：定义是否使用透明效果；

5、visible：定义Modal是否显示

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

Modal,

TouchableHighlight

} from 'react-native';

class ModalDemo extends Component {

constructor(props){

super(props);

this.state = {

animated: true,

modalVisible: false,

transparent: false,

showText:'',

modalStatus:'Modal目前处于关闭状态'

};

}

\_setModalVisible(visible) {

this.setState({

modalVisible: visible

});

}

\_showModal(){

this.setState({

showText:'Modal显示了',

modalStatus:'Modal打开过了'

});

}

render() {

return (

<View>

<Modal

animated={this.state.animated}

transparent={this.state.transparent}

visible={this.state.modalVisible}

onRequestClose={() => {this.\_setModalVisible(false)}}

onShow={this.\_showModal.bind(this)}

>

<View style={{width:300,height:200,marginTop:200,alignSelf:'center',justifyContent: 'center',padding: 20,backgroundColor:'rgba(229,27,81,0.7)',borderRadius:10}}>

<View style={{alignItems: 'center'}}>

<Text style={{color:'#fff'}}>Modal弹出窗口的内容</Text>

<Text style={{color:'#fff'}}>{this.state.showText}</Text>

<View style={{borderWidth:1,borderColor:'#f8f8f8',paddingVertical:8,paddingHorizontal:12,marginTop:16,borderRadius:6}}>

<Text style={{color:'#fff'}} onPress={this.\_setModalVisible.bind(this, false)}>

关闭

</Text>

</View>

</View>

</View>

</Modal>

<Text style={styles.title}>Modal组件</Text>

<TouchableHighlight style={{flex:1,justifyContent:'center',margin:20,padding:20,alignItems:'center',backgroundColor:'#eee'}} underlayColor='#ddd' onPress={()=>this.\_setModalVisible(true)}>

<Text style={{fontSize:18}}>Open Modal</Text>

</TouchableHighlight>

<Text style={{textAlign:'center'}}>{this.state.modalStatus}</Text>

</View>

);

}

}

const styles = StyleSheet.create({

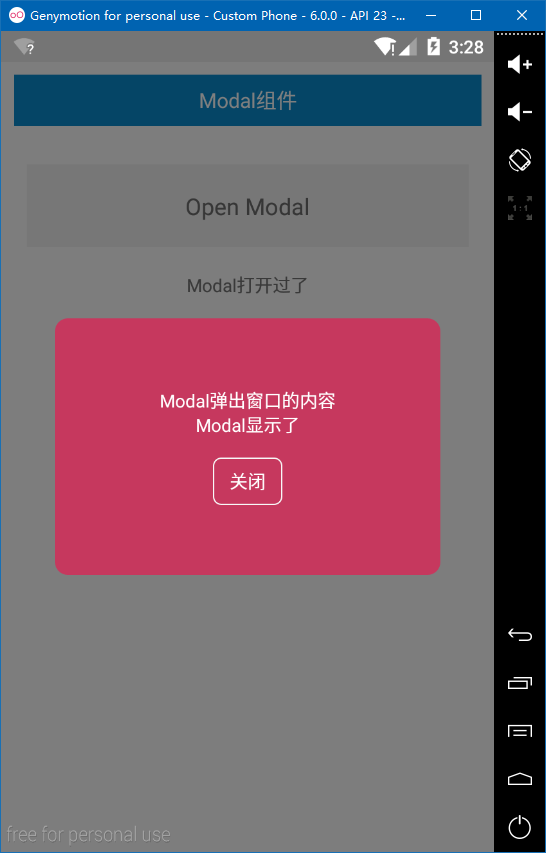
container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

})

module.exports = ModalDemo;



#### （十一）ListView组件

ListView组件是一个非常重要的组件，主要用于以列表形式显示动态数据，如显示一个新闻列表、产品列表等。

**1、使用ListView的步骤如下：**

（1）引入ListView依赖

import React, {

Component,

StyleSheet,

Text,

View,

**ListView**

} from 'react-native';

（2）在构造方法创建（实例化）一个ListView数据源对象

this.ds = new ListView.DataSource({rowHasChanged: (r1,r2) => r1 !== r2});

（3）定义或获取数据源（数组）

this.data = [

{"id":1,"title":"title1","excerpt":"excerpt11111","content":"content111111111111111111111"},

{"id":2,"title":"title2","excerpt":"excerpt222222","content":"content222222222222222222222"},

{"id":3,"title":"title3","excerpt":"excerpt333333","content":"content333333333333333333333"},

{"id":4,"title":"title4","excerpt":"excerpt444444","content":"content444444444444444444444"},

{"id":5,"title":"title5","excerpt":"excerpt555555","content":"content555555555555555555555"},

{"id":6,"title":"title6","excerpt":"excerpt666666","content":"content666666666666666666666"},

{"id":7,"title":"title7","excerpt":"excerpt777777","content":"content777777777777777777777"},

{"id":8,"title":"title8","excerpt":"excerpt888888","content":"content888888888888888888888"},

{"id":9,"title":"title9","excerpt":"excerpt999999","content":"content999999999999999999999"},

{"id":10,"title":"title10","excerpt":"excerpt10101010","content":"content101010101010101010"},

{"id":11,"title":"title11","excerpt":"excerpt1111111111","content":"content111111111111111111111"},

{"id":12,"title":"title12","excerpt":"excerpt1212121212","content":"content121212121212121212121"},

];

（4）将数据源（数组）指定给已实例化的ListView对象

this.state = {

dataSource:this.ds.cloneWithRows(this.data)

};

（5）定义ListView列表条目的渲染函数（样式）

\_renderRow(rowData){

return (

<View style={{padding:10,borderBottomWidth:1,borderBottomColor:'#ccc',marginBottom:5}}>

<Text style={{fontSize:18,fontWeight:'bold',color:'#444'}}>{rowData.id} {rowData.title}</Text>

<Text>{rowData.excerpt}</Text>

</View>

);

}

（6）将ListView加入库渲染视图，指定数据源和渲染格式

<ListView dataSource={this.state.dataSource} renderRow={ this.\_renderRow.bind(this) } />

**2、一个基本的ListView实例**

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

ListView

} from 'react-native';

class ListViewDemo extends Component {

constructor(props){

super(props);

this.data = [

{"id":1,"title":"title1","excerpt":"excerpt11111","content":"content111111111111111111111"},

{"id":2,"title":"title2","excerpt":"excerpt222222","content":"content222222222222222222222"},

{"id":3,"title":"title3","excerpt":"excerpt333333","content":"content333333333333333333333"},

{"id":4,"title":"title4","excerpt":"excerpt444444","content":"content444444444444444444444"},

{"id":5,"title":"title5","excerpt":"excerpt555555","content":"content555555555555555555555"},

{"id":6,"title":"title6","excerpt":"excerpt666666","content":"content666666666666666666666"},

{"id":7,"title":"title7","excerpt":"excerpt777777","content":"content777777777777777777777"},

{"id":8,"title":"title8","excerpt":"excerpt888888","content":"content888888888888888888888"},

{"id":9,"title":"title9","excerpt":"excerpt999999","content":"content999999999999999999999"},

{"id":10,"title":"title10","excerpt":"excerpt10101010","content":"content101010101010101010"},

{"id":11,"title":"title11","excerpt":"excerpt1111111111","content":"content111111111111111111111"},

{"id":12,"title":"title12","excerpt":"excerpt1212121212","content":"content121212121212121212121"},

];

this.ds = new ListView.DataSource({rowHasChanged: (r1,r2) => r1 !== r2});

this.state = {

dataSource:this.ds.cloneWithRows(this.data)

};

}

\_renderRow(rowData){

return (

<View style={{padding:10,borderBottomWidth:1,borderBottomColor:'#ccc',marginBottom:5}}>

<Text style={{fontSize:18,fontWeight:'bold',color:'#444'}}>{rowData.id} {rowData.title}</Text>

<Text>{rowData.excerpt}</Text>

</View>

);

}

render() {

return (

<View style={styles.container}>

<Text style={styles.title}>ListView组件</Text>

<ListView dataSource={this.state.dataSource} renderRow={ this.\_renderRow.bind(this) } />

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

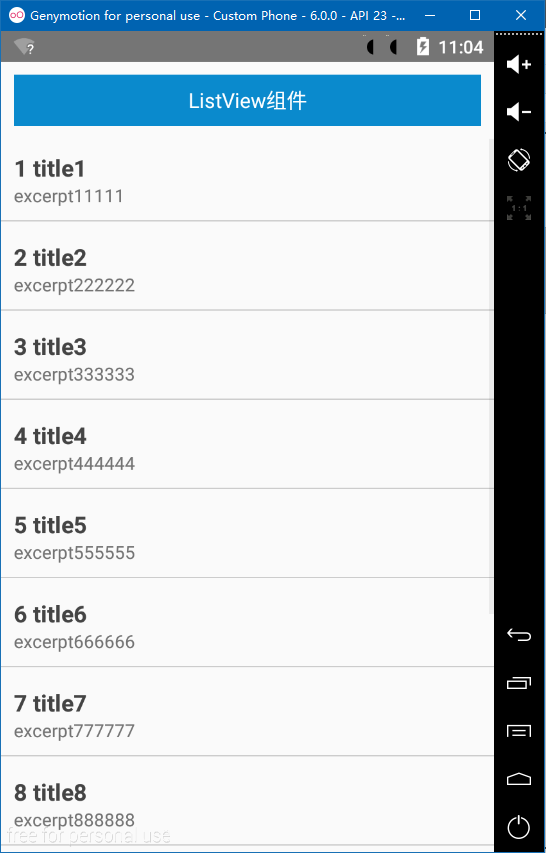
divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

box:{borderWidth:1,borderColor:'#ccc',padding:10,margin:10}

})

module.exports = ListViewDemo;



**3、ListView常用属性：**

（1）dataSource：指定ListView组件的数据源；

（2）renderRow：指定数据行的渲染形式，通过指定一个回调返回函数来设定。

renderRow指定函数可传入参数为：

(rowData, sectionID,rowID,highlightRow)

rowData为数据源单条数据对象，sectionID为分段区块ID，rowID为数据条目ID,hightlightRow是当某行数据通过highlightRow函数调用高亮时，ListView会得到相应的通知。

（3）renderSectionHeader：定义分段区块的标题

（4）renderHeader：定义ListView总标题

（5）renderFooter：定义ListView底部

（6）renderSeparator：定义列表项目的分隔格式

（7）initialListSize：定义初始化渲染的项目行数

（8）onEndReached：当完成数据渲染后滑动到底端不足onEndReachedThreshold像素距离时调用

（9）onEndReachedThreshold：定义onEndReached函数调用的临界值，单位为像素

（10）pageSize：定义每次事件循环的行数

（11）onChangeVisibleRows：当可见行的集合变化时调用

（12）enableEmptySections：定义是否显示空分段区块的标题

（13）renderScrollComponent：指定返回一个可以滚动的组件的函数，默认为ScrollView

（14）scrollRenderAheadDistance：指定当一行接受屏幕范围多少像素之内开始渲染

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

ListView,

ScrollView,

RecyclerViewBackedScrollView,

TouchableHighlight

} from 'react-native';

class ListViewDemo extends Component {

constructor(props){

super(props);

this.data = [

{"id":1,"category":"category1","title":"title1","excerpt":"excerpt11111","content":"content111111111111111111111"},

{"id":2,"category":"category3","title":"title2","excerpt":"excerpt222222","content":"content222222222222222222222"},

{"id":3,"category":"category2","title":"title3","excerpt":"excerpt333333","content":"content333333333333333333333"},

{"id":4,"category":"category1","title":"title4","excerpt":"excerpt444444","content":"content444444444444444444444"},

{"id":5,"category":"category3","title":"title5","excerpt":"excerpt555555","content":"content555555555555555555555"},

{"id":6,"category":"category4","title":"title6","excerpt":"excerpt666666","content":"content666666666666666666666"},

{"id":7,"category":"category4","title":"title7","excerpt":"excerpt777777","content":"content777777777777777777777"},

{"id":8,"category":"category2","title":"title8","excerpt":"excerpt888888","content":"content888888888888888888888"},

{"id":9,"category":"category2","title":"title9","excerpt":"excerpt999999","content":"content999999999999999999999"},

{"id":10,"category":"category3","title":"title10","excerpt":"excerpt10101010","content":"content101010101010101010"},

{"id":11,"category":"category3","title":"title11","excerpt":"excerpt1111111111","content":"content111111111111111111111"},

{"id":12,"category":"category4","title":"title12","excerpt":"excerpt1212121212","content":"content121212121212121212121"},

{"id":13,"category":"category1","title":"title13","excerpt":"excerpt1212121212","content":"content131313131313131313131"},

{"id":14,"category":"category1","title":"title14","excerpt":"excerpt1212121212","content":"content141414141414141414141"},

{"id":15,"category":"category1","title":"title15","excerpt":"excerpt1212121212","content":"content151515151515151515151"},

{"id":16,"category":"category1","title":"title16","excerpt":"excerpt1212121212","content":"content161616161616161616161"},

];

this.ds = new ListView.DataSource({

rowHasChanged: (r1,r2) => r1 !== r2,

sectionHeaderHasChanged: (s1,s2) => s1 !== s2

});

this.state = {

highlightColor:'red',

dataSource:this.ds.cloneWithRowsAndSections(this.convertArrayToMap())

};

}

pressFunc(rowDataid){

console.log(rowDataid);

}

convertArrayToMap(){

var categoryMap = {}; // Create the blank map

this.data.forEach(function(item) {

if (!categoryMap[item.category]) {

// Create an entry in the map for the category if it hasn't yet been created

categoryMap[item.category] = [];

}

categoryMap[item.category].push(item);

});

return categoryMap;

}

\_renderRow(rowData,sectionID,rowID){

return (

<TouchableHighlight onPress={() => this.pressFunc(rowData.id)}>

<View>

<Text style={{fontSize:18,fontWeight:'bold'}}>{rowData.id} {rowData.title} - {rowID}</Text>

<Text style={{width:80,textAlign:'center',backgroundColor:'#0a8acd',borderRadius:6,color:'#fff'}}>{sectionID}</Text>

</View>

</TouchableHighlight>

);

}

\_renderSectionHeader(sectionData, category){

return (

<Text style={{fontSize:18,color:'#fff',backgroundColor:'#0a8acd'}}>{category}</Text>

);

}

\_renderHeader(){

return (

<View style={{justifyContent:'center',alignItems:'center',backgroundColor:'#FF9632'}}>

<Text style={{textAlign:'center',marginVertical:10,color:'#fff'}}>资讯中心</Text>

</View>

);

}

\_separator(sectionID,rowID){

return (

<View key={`${sectionID}-${rowID}`} style={{height:1,backgroundColor:'red',marginBottom:10,marginTop:6}}>

</View>

);

}

render() {

return (

<View style={styles.container}>

<Text style={styles.title}>ListView组件</Text>

<ListView

dataSource={this.state.dataSource}

renderRow={ this.\_renderRow.bind(this) }

renderSectionHeader={this.\_renderSectionHeader.bind(this)}

renderHeader={this.\_renderHeader.bind(this)}

renderScrollComponent={props => <RecyclerViewBackedScrollView {...props} />}

renderSeparator={this.\_separator.bind(this)}

/>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

box:{borderWidth:1,borderColor:'#ccc',padding:10,margin:10}

})

module.exports = ListViewDemo;



#### （十二）ScrollView

ScrollView组件包含滚动视图的容器，并集成了响应功能。使用ScrollView必须指定一个确定的高度。

ScrollView的常用属性有：

1、contentContainerStyle：设置内层子容器的统一样式

2、horizontal：设置是否以水平滚动方式显示（默认是垂直方式显示）

3、keyboardDismissMode：用户拖拽滚动视图时，软键盘的设置，共有三种设置可选，分别为none（默认值，拖拽不隐藏），on-drag（拖拽滚动隐藏软键盘）和interactive（软键盘伴随拖拽操作同步消失，Android平台不支持此设置项，如果设置为此项，Android平台上将与none设置项相同）

4、keyboardShouldPersistTaps：如果设置为false，则软键盘在激活时，点击焦点文本输入框外的地方，键盘会隐藏，默认值。如果为true，滚动视图不会响应点击操作，并且键盘不会自动消失。

5、onContentSizeChange：当滚动内容视图变化时触发指定函数。

6、onScroll：当滚动时触发指定函数

7、refreshControl：用于指定RefreshControl组件，实现ScrollView下拉刷新功能

8、scrollEnabled：设置ScrollView是否可以滚动

9、showsHorizontalScrollIndicator：设置是否显示水平滚动条，默认显示

10、showsVerticalScrollIndicator：设置是否显示垂直滚动条，默认显示

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

TouchableHighlight,

ScrollView

} from 'react-native';

class ScrollViewDemo extends Component {

constructor(props){

super(props);

this.state = {

status:'提示',

data:[

'title1',

'title2',

'title3',

'title4',

'title5',

'title6',

'title7',

'title8',

'title9',

'title10',

'title11',

'title12'

]

}

}

render() {

var createRow = (title, i) => {

return (

<View key={i} style={{padding:10,borderBottomWidth:1,borderBottomColor:'#aaa'}}>

<Text>{title}</Text>

</View>

);

};

return (

<View style={styles.container}>

<Text style={styles.title}>ScrollView组件</Text>

<View>

<ScrollView style={styles.boxContainer} showsVerticalScrollIndicator={false}>

<View style={styles.box}><Text style={styles.centerText}>1</Text></View>

<View style={styles.box}><Text style={styles.centerText}>2</Text></View>

<View style={styles.box}><Text style={styles.centerText}>3</Text></View>

<View style={styles.box}><Text style={styles.centerText}>4</Text></View>

<View style={styles.box}><Text style={styles.centerText}>5</Text></View>

</ScrollView>

<View style={styles.divider}></View>

<ScrollView style={styles.boxHorizontalContainer} contentContainerStyle={{borderWidth:1,borderColor:'#aaa',paddingVertical:10}} horizontal={true} showsHorizontalScrollIndicator={false}>

<View style={styles.boxHorizontal}><Text style={styles.centerText}>1</Text></View>

<View style={styles.boxHorizontal}><Text style={styles.centerText}>2</Text></View>

<View style={styles.boxHorizontal}><Text style={styles.centerText}>3</Text></View>

<View style={styles.boxHorizontal}><Text style={styles.centerText}>4</Text></View>

<View style={styles.boxHorizontal}><Text style={styles.centerText}>5</Text></View>

</ScrollView>

<View style={styles.divider}></View>

<ScrollView

onScroll={() => this.setState({status:'你滚动了ScrollView'})}

style={{height:200,backgroundColor:'#ccc'}}

contentContainerStyle={{paddingLeft:10}}>

{this.state.data.map(createRow)}

</ScrollView>

<Text>{this.state.status}</Text>

</View>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

scrollContainer:{padding:10,height:120},

scroll:{padding:10,backgroundColor:'#eee',paddingBottom:50},

boxContainer:{padding:10,height:120},

box:{flex:1,height:50,backgroundColor:'#0a8acd',marginBottom:10,justifyContent:'center'},

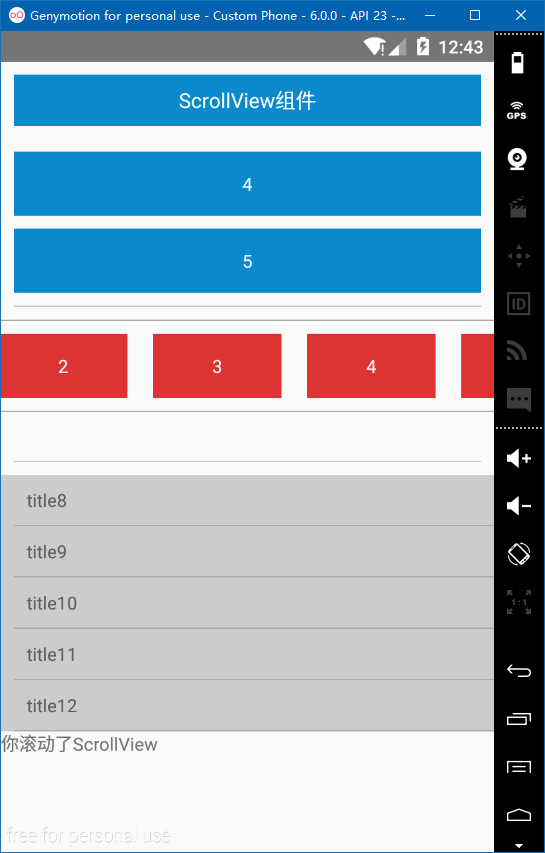
centerText:{color:'#fff',textAlign:'center'},

boxHorizontalContainer:{flex:1,height:100},

boxHorizontal:{width:100,height:50,backgroundColor:'#dd3333',marginHorizontal:10,justifyContent:'center'}

})

module.exports = ScrollViewDemo;



#### （十三）RefreshControl

RefreshControl组件作为ScrollView或ListView组件内部的refreshControl属性被引入ScrollView或ListView组件，用于触发当用户下拉列表时触发的事件。

常用属性有：

1、refreshing：设置是否显示刷新指示器

2、onRefresh：设置下拉刷新触发响应事件

3、colors：设置刷新指示器的颜色变化组合，至少一种颜色，仅Android平台可用

4、enabled：设置下拉刷新是否可用，仅Android平台可用

5、progressBackgroundColor：设置刷新指示器的背景颜色，仅Android平台可用

6、size：设置刷新指示器的大小，仅Android平台可用

7、tintColor：设置刷新指示器的颜色

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

ListView,

ScrollView,

RefreshControl,

TouchableHighlight

} from 'react-native';

class RefreshControlDemo extends Component {

constructor(props){

super(props);

this.state = {

apiUrl:'http://www.sameroad.cn/wp-json/wp/v2/posts',

pageSize:5,

page:1,

loaded: false,

data:[],

isRefreshing:false,

dataSource:new ListView.DataSource({

rowHasChanged:(row1,row2) => row1 !== row2,

})

};

}

componentDidMount(){

this.fetchData();

};

fetchData(){

var REQUEST\_URL = this.state.apiUrl + '?filter[posts\_per\_page]='+this.state.pageSize + '&page='+ this.state.page;

fetch(REQUEST\_URL)

.then((response) => response.json())

.then((responseData) => {

var dataArray = responseData.concat(this.state.data);

this.setState({

data:dataArray,

dataSource: this.state.dataSource.cloneWithRows(dataArray),

loaded:true,

page:this.state.page + 1,

isRefreshing:false

});

})

.done();

};

pressFunc(rowDataid){

console.log(rowDataid);

}

\_formatDate(date){

var dt = new Date(date);

return dt.getFullYear() + '-' + (dt.getMonth() + 1) + '-' + dt.getDate();

}

\_renderRow(rowData){

return (

<TouchableHighlight onPress={() => this.pressFunc(rowData.id)} underlayColor="#eee">

<View style={{padding:10,borderBottomWidth:1,borderColor:'#ccc'}}>

<Text style={{fontSize:18,fontWeight:'bold'}}>{rowData.title.rendered}</Text>

<Text>{this.\_formatDate(rowData.date)}</Text>

</View>

</TouchableHighlight>

);

}

\_onRefresh(){

this.setState({isRefreshing:true})

this.fetchData();

}

renderLoadingView(){

return (

<View style={{flex:1,justifyContent:'center',alignItems:'center',alignSelf:'center'}}>

<Text>正在拼命加载中...</Text>

</View>

);

}

render() {

if(!this.state.loaded){

return this.renderLoadingView();

}

return (

<View style={styles.container}>

<Text style={styles.title}>RefreshControl组件</Text>

<ListView

refreshControl={

<RefreshControl

colors={['#0a8acd','#BDECFF']}

progressBackgroundColor='#eee'

refreshing={this.state.isRefreshing}

onRefresh={this.\_onRefresh.bind(this)}

/>

}

dataSource={this.state.dataSource}

renderRow={ this.\_renderRow.bind(this) }

/>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

box:{borderWidth:1,borderColor:'#ccc',padding:10,margin:10}

})

module.exports = RefreshControlDemo;



#### （十四）WebView

WebView组件渲染一个原生Web视图。

常用属性有：

1、automaticallyAdjustContentInsets：自动调整内容

2、contentInset：{top: number, left: number, bottom: number, right: number} 设置内容尺寸

3、injectedJavaScript：设置注入的JS内容

4、mediaPlaybackRequiresUserAction：设置多媒体的播放是否需要用户点击后才播放

5、onError：当Web视图加载失败时触发指定函数

6、onLoad：当页面加载完成时触发指定函数

7、onLoadEnd：当页面加载结束时触发指定函数（不管是否成功加载）

8、onLoadStart：当页面开始加载时触发指定函数

9、onNavigationStateChange：当导航状态变化时触发指定函数

10、renderError：当发生错误时显示指定渲染的错误界面，需要将startInLoadingState设置为true才有效

11、renderLoading：指定一个加载指示器，需要将startInLoadingState设置为true才有效

12、scalesPageToFit：设置加载网页自动适应WebView视图大小

13、source：设置加载网页内容源或URL地址源

{uri: string, method: string, headers: object, body: string}, {html: string, baseUrl: string}

14、startInLoadingState：设置是否在加载状态开始介入

15、domStorageEnabled：设置是否允许使用DOM缓存，仅可用于Android平台

16、javaScriptEnabled：设置是否启用Javascript，仅可用于Android平台，IOS平台默认启用支持

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

WebView,

TouchableHighlight

} from 'react-native';

class WebViewDemo extends Component {

constructor(props){

super(props);

this.state = {

url: 'http://m.baidu.com',

status:'',

newUrl:'http://www.163.com/',

newStatus:''

};

}

\_renderLoading(){

return (

<View style={{flex:1,justifyContent:'center',alignItems:'center',alignSelf:'center'}}>

<Text>正在拼命加载中...</Text>

</View>

);

}

render() {

const HTML = `

<!DOCTYPE html>\n

<html>

<head>

<title>Hello Static World</title>

<meta http-equiv="content-type" content="text/html; charset=utf-8">

<meta name="viewport" content="width=320, user-scalable=no">

<style type="text/css">

body {margin: 0;padding: 0;background: #dd3333;}

h1 {margin:0,padding: 6px;font-Size:20px;text-align: center;color: #0a8acd;}

p {text-align:center;}

</style>

</head>

<body>

<h1>使用WebView显示静态页面内容</h1>

<p>人通常一旦失去什么，就会害怕，“未来还会再失去吗？”无法褪去的记忆与惊恐，使人的灵魂永远藏着黑暗。一道又一道的刻痕，在时间岁月的累积中，我们失去信赖的能力，失去善良的能力，失去快乐的能力。每一道过往的刻痕，折叠着随时的怨、恨、愤以及攻击。</p>

</body>

</html>

`;

return (

<View style={{flex:1}}>

<Text style={styles.title}>WebView组件</Text>

<View style={{flexDirection:'row'}}>

<TouchableHighlight style={{flex:1,borderWidth:1,borderColor:'#ddd'}} onPress={()=>this.setState({url:'http://m.baidu.com'})}>

<Text style={{textAlign:'center'}}>百度</Text>

</TouchableHighlight>

<TouchableHighlight style={{flex:1,borderWidth:1,borderColor:'#ddd'}} onPress={()=>this.setState({url:'http://xw.qq.com/index.htm'})}>

<Text style={{textAlign:'center'}}>腾讯</Text>

</TouchableHighlight>

<TouchableHighlight style={{flex:1,borderWidth:1,borderColor:'#ddd'}} onPress={()=>this.setState({url:'http://sina.cn/?from=wap'})}>

<Text style={{textAlign:'center'}}>新浪</Text>

</TouchableHighlight>

</View>

<WebView

style={{flex:1,height:200}}

source={{uri:this.state.url}}

onLoadStart={() => this.setState({status:'开始打开网页'})}

onLoadEnd={() => this.setState({status:'网页加载结束'})}

onLoad={() => this.setState({status:'网页加载完成'})}

renderLoading={this.\_renderLoading.bind(this)}

/>

<Text style={{textAlign:'center'}}>{this.state.status}</Text>

<View style={styles.divider}></View>

<WebView

style={{flex:1,height:100}}

source={{html:HTML}}

renderLoading={this.\_renderLoading.bind(this)}

startInLoadingState={true}

/>

<View style={styles.divider}></View>

<View style={{flexDirection:'row'}}>

<TouchableHighlight style={{flex:1,borderWidth:1,borderColor:'#ddd'}} onPress={()=>this.setState({newUrl:'http://www.163.com/'})}>

<Text style={{textAlign:'center'}}>网易</Text>

</TouchableHighlight>

<TouchableHighlight style={{flex:1,borderWidth:1,borderColor:'#ddd'}} onPress={()=>this.setState({newUrl:'http://www.jd.com'})}>

<Text style={{textAlign:'center'}}>京东</Text>

</TouchableHighlight>

<TouchableHighlight style={{flex:1,borderWidth:1,borderColor:'#ddd'}} onPress={()=>this.setState({newUrl:'https://m.taobao.com/#index'})}>

<Text style={{textAlign:'center'}}>淘宝</Text>

</TouchableHighlight>

</View>

<WebView

style={{flex:1,height:200}}

source={{uri:this.state.newUrl}}

onLoadStart={() => this.setState({newStatus:'开始打开网页'})}

onLoadEnd={() => this.setState({newStatus:'网页加载结束'})}

onLoad={() => this.setState({newStatus:'网页加载完成'})}

renderLoading={this.\_renderLoading.bind(this)}

automaticallyAdjustContentInsets={true}

startInLoadingState={true}

/>

<Text style={{textAlign:'center'}}>{this.state.newStatus}</Text>

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

})

module.exports = WebViewDemo;



#### （十五）ViewPagerAndroid

ViewPagerAndroid是一个仅可用于Android平台的翻页容器，ViewPagerAndroid内部使用<View>定义各个子容器，每个子容器作为一个单独的视图，尺寸大小与父窗口ViewPagerAndroid相同。

常用的属性有：

1、initialPage：定义初始显示（选中状态）的子容器索引，从0开始

2、keyboardDismissMode：定义是否在滑动的时候显示软键盘，可选值有“none”显示软键盘（默认值）和“on-drag”隐藏软键盘

3、onPageScroll:当在页面切换时执行指定函数。回调参数中的event.nativeEvent对象包含下面数据：

position：从左开始数起第一个当前可见的页面索引

offset：一个大于等于0，小于1的数字，代表当前切换状态。值x表示当前显示页面有（1-x）的部分可见，而下一页有x的部分可见

4、onPageScrollStateChanged：当页面滑动状态变化时触发指定函数，页面滑动状态有三种状态：

（1）idle：空闲状态，即没有交互

（2）dragging：拖动中状态，表示当前页面正在被拖动

（3）settling：处理中状态，表示当前页面发生过交互，且正在结束开头或收尾的动画

5、onPageSelected：当页面切换完成后触发指定函数。回调参数中的event.nativeEvent对象中的position表示当前被选中页面的索引

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

Image,

ViewPagerAndroid

} from 'react-native';

class ViewPagerAndroidDemo extends Component {

constructor(props){

super(props);

this.state = {

selected: 0

};

}

\_pageSelected(e){

console.log(e);

this.setState({

selected: e.nativeEvent.position

});

}

render() {

var intros = ['复仇者联盟3','复仇者联盟2','漫威动画电影','钢铁侠3','蚁人'];

var images = [

{

intro:'复仇者联盟3',

url:'http://i.annihil.us/u/prod/marvel/i/mg/a/10/54ebd9917f0e7.jpg'

},

{

intro:'复仇者联盟2',

url:'http://ifanboy.com/wp-content/uploads/2012/02/newavengersposter.jpg'

},

{

intro:'漫威动画电影',

url:'http://x.annihil.us/u/prod/marvel/i/mg/6/b0/5375c397eef00.jpg'

},

{

intro:'钢铁侠3',

url:'http://x.annihil.us/u/prod/marvel/i/mg/9/40/508563b9dc2ac.jpg'

},

{

intro:'蚁人',

url:'http://x.annihil.us/u/prod/marvel/i/mg/3/60/55495e0e6a260.jpg'

}];

return (

<View>

<Text style={styles.title}>ViewPagerAndroid组件</Text>

<Text style={{textAlign:'center'}}>{this.state.selected}</Text>

<ViewPagerAndroid style={{flex:1,margin:10,height:500}} initialPage={0} onPageSelected={this.\_pageSelected.bind(this)}>

{images.map((elem,index) => {

return (

<View key={index} style={{flex:1}}>

<Image style={{flex:1}} resizeMode="cover" source={{uri: elem.url}} />

<Text style={{fontSize:20,textAlign:'center'}}>{elem.intro}</Text>

</View>

);

})}

</ViewPagerAndroid>

</View>

);

}

}

const styles = StyleSheet.create({

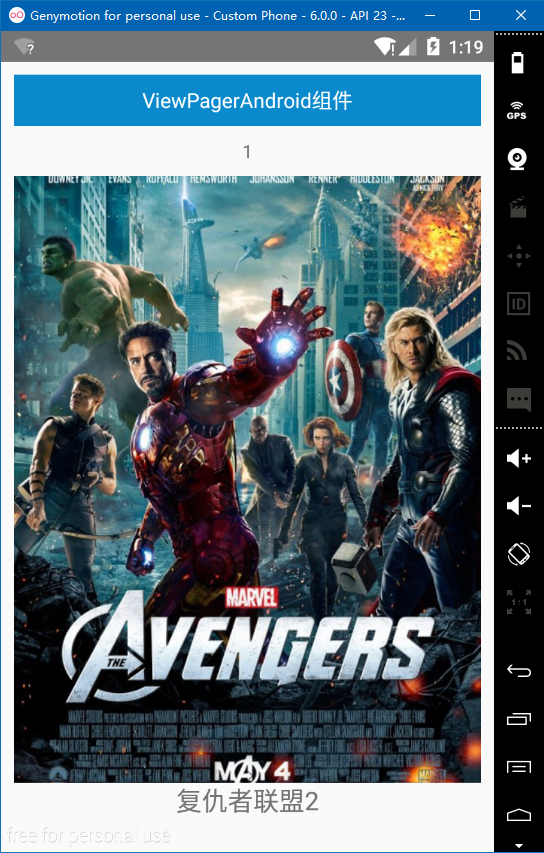
container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

})

module.exports = ViewPagerAndroidDemo;



#### （十六）ProgressBarAndroid

ProgressBarAndroid是一个仅可用于Android平台的进度条组件。

常用属性有：

1、color：指定进度条颜色

2、indeterminate：设置是否显示一个不确定的进度。这个属性在styleAttr设置为Horizontal时必须是 false

3、progress：设置当前进度值，从0到1之间

4、styleAttr：设置进度条的样式，可选值有：

（1）Horizontal：水平样式

（2）Normal：普通样式，默认

（3）Small

（4）Large

（5）Inverse

（6）SmallInverse

（7）LargeInverse

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

ProgressBarAndroid,

} from 'react-native';

class ProgressBarAndroidDemo extends Component {

render() {

return (

<View>

<Text style={styles.title}>ProgressBarAndroid组件</Text>

<ProgressBarAndroid />

<ProgressBarAndroid color='#dd3333' />

<ProgressBarAndroid indeterminate={true} />

<ProgressBarAndroid styleAttr='Small' />

<ProgressBarAndroid styleAttr='Large' />

<ProgressBarAndroid styleAttr='Horizontal' />

<ProgressBarAndroid styleAttr='Horizontal' progress={0.4} />

<ProgressBarAndroid styleAttr='Inverse' />

</View>

);

}

}

const styles = StyleSheet.create({

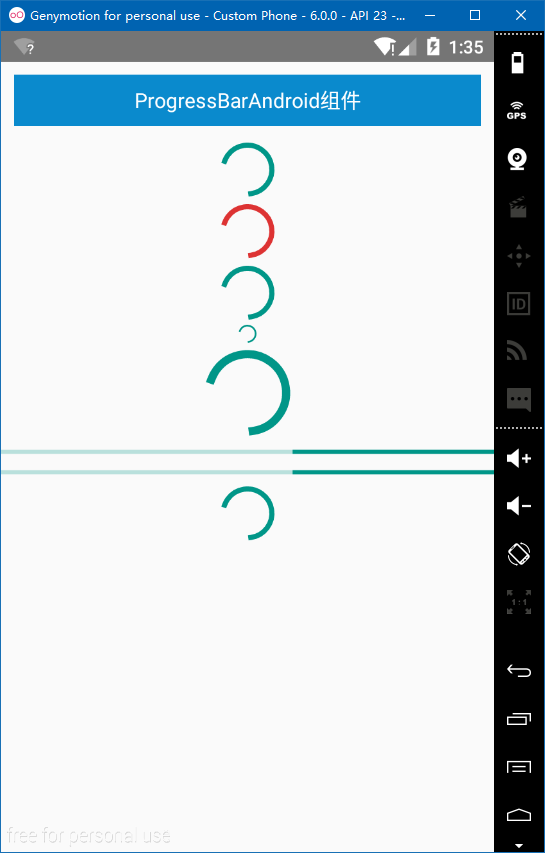
container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

})

module.exports = ProgressBarAndroidDemo;



#### （十七）StatusBar

StatusBar是一个状态栏组件。

常用属性有：

1、animated：设置状态栏变化是否以动画形式呈现

2、hidden：是否隐藏状态栏

3、backgroundColor：设置状态栏背景颜色，仅可用于Android平台

4、translucent：设置状态栏是否与视图重叠

|  |  |
| --- | --- |
|  | <StatusBar /> |
|  | <StatusBar  hidden={true}  /> |
|  | <StatusBar  backgroundColor="rgba(5,165,209,0.8)"  /> |
|  | <StatusBar  backgroundColor="rgba(5,165,209,0.8)"  translucent={true}  /> |
|  | <StatusBar  backgroundColor="rgba(5,165,209,0)"  translucent={true}  /> |

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

Image,

StatusBar,

} from 'react-native';

class Main extends Component {

render() {

return (

<View>

<StatusBar

backgroundColor="rgba(5,165,209,0)"

translucent={true}

/>

<Image style={{height:200}} resizeMode="stretch" source={require('./img/beautiful.jpg')} />

<Text style={styles.title}>StatusBar组件</Text>

</View>

);

}

}

const styles = StyleSheet.create({

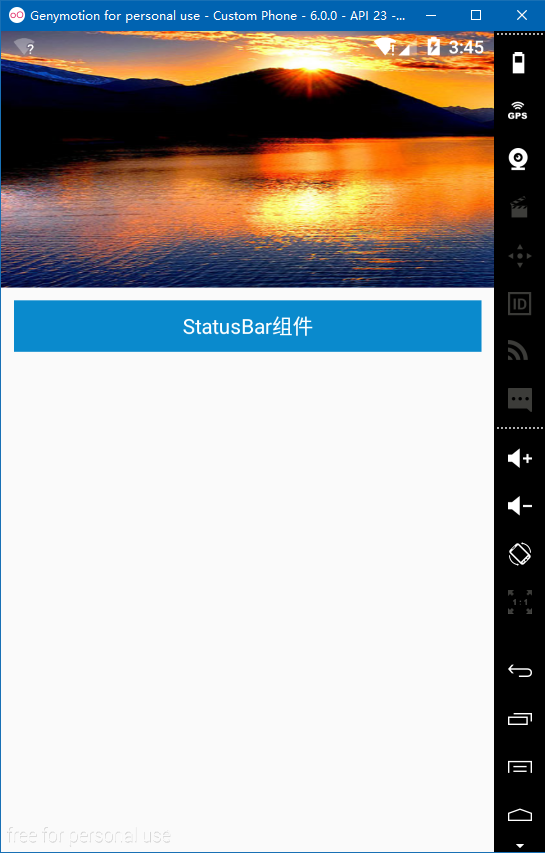
container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

})

module.exports = Main;



#### （十八）ToolbarAndroid

ToolbarAndroid是一个仅可用于Android平台的工具栏组件。ToolbarAndroid可以显示一个导航图标、一个LOGO图标（或者应用标题/子标题，注：LOGO图标和标题不能同时有）和一个功能列表。

常用属性有：

1、actions: [{title: string, icon: optionalImageSource, show: enum('always', 'ifRoom', 'never'), showWithText: bool}]

定义功能列表，在工具栏右边显示。这些功能项目会以图标或文字形式显示，如果放不不，则会放进一个弹出菜单里。

Actions定义一个对象列表，每个对象有以下字段：

（1）title：必须，功能标题

（2）icon：功能图标

（3）show：显示方式。可选项有“always”（总是显示）、“ifRoom”（如果有空间则显示）和“never”（不显示）

（4）showWithText：是否在图标旁边显示文字

2、logo：设置工具栏LOGO

3、navIcon：设置工具栏导航图标

4、overflowIcon：设置功能列表弹出菜单的图标

5、title：设置工具栏标题

6、titleColor：设置工具栏标题颜色

7、subtitle：设置工具栏副标题

8、subtitleColor：设置工具栏副标题颜色

9、onIconClicked：设置当导航图标被点击时的触发函数

10、onActionSelected:设置当功能项目被点击时的触发函数，回调参数为被点击功能项目在功能列表中的位置索引值，从0开始

11、contentInsetStart：设置工具栏左边缘和屏幕左边缘距离

12、contentInsetEnd：设置工具栏右边缘和屏幕右边缘距离

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

ToolbarAndroid

} from 'react-native';

class ToolbarAndroidDemo extends Component {

constructor(props){

super(props);

this.state = {

menu: undefined,

};

}

\_onActionSelected(index){

this.setState({

menu: index

});

}

render() {

var myToolbarActions = [

{title:'设置', icon: require('./img/setting.png'),show:'ifRoom'},

{title:'登陆', icon: require('./img/login.png'),show:'ifRoom'},

{title:'编辑', icon: require('./img/edit.png'),show:'ifRoom'},

{title:'新建', icon: require('./img/edit.png'),show:'ifRoom'},

{title:'分享', icon: require('./img/edit.png'),show:'ifRoom'}

];

var myToolbarActions2 = [

{title:'设置', icon: require('./img/setting.png'),show:'always'},

{title:'登陆', icon: require('./img/login.png'),show:'always'},

{title:'编辑', icon: require('./img/edit.png'),show:'always'},

{title:'新建', icon: require('./img/edit.png'),show:'always'},

{title:'分享', icon: require('./img/edit.png'),show:'always'}

];

var myToolbarActions3 = [

{title:'设置', icon: require('./img/setting.png'),show:'never'},

{title:'登陆', icon: require('./img/login.png'),show:'never'},

{title:'编辑', icon: require('./img/edit.png'),show:'never'},

{title:'新建', icon: require('./img/edit.png'),show:'never'},

{title:'分享', icon: require('./img/edit.png'),show:'never'}

];

var testActions = [

{title:'设置'},

{title:'登陆'},

{title:'编辑'}

];

return (

<View>

<Text style={styles.title}>ToolbarAndroid组件</Text>

<ToolbarAndroid

style={{height:56,backgroundColor:'#ddd'}}

navIcon={require('./img/nav.png')}

title="我的应用"

titleColor="#dd3333"

subtitle="一个React Native应用"

subtitleColor="#0a8acd"

contentInsetStart={10}

contentInsetEnd={120}

onIconClicked={this.props.openDrawer}

actions={myToolbarActions} />

<View style={styles.divider}></View>

<ToolbarAndroid

style={{height:56,backgroundColor:'#ddd'}}

navIcon={require('./img/nav.png')}

title="我的应用"

titleColor="#dd3333"

subtitle="一个React Native应用"

subtitleColor="#0a8acd"

onActionSelected={this.\_onActionSelected.bind(this)}

onIconClicked={this.props.openDrawer}

actions={myToolbarActions2} />

<Text style={{textAlign:'center'}}>{this.state.menu}</Text>

<View style={styles.divider}></View>

<ToolbarAndroid

style={{height:56,backgroundColor:'#ddd'}}

navIcon={require('./img/nav.png')}

logo={require('./img/react.png')}

overflowIcon={require('./img/overflowIcon.png')}

onIconClicked={this.props.openDrawer}

actions={myToolbarActions3} />

<Text style={{textAlign:'center'}}>{this.state.status}</Text>

<View style={styles.divider}></View>

<ToolbarAndroid

style={{height:56,backgroundColor:'#ddd'}}

navIcon={require('./img/nav.png')}

logo={require('./img/react.png')}

onIconClicked={this.props.openDrawer}

actions={testActions} />

</View>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

})

module.exports = ToolbarAndroidDemo;



#### （十九）Navigator

#### （二十）DrawerLayoutAndroid

DrawerLayoutAndroid是一个仅可用于Android平台的抽屉式组件。DrawerLayoutAndroid的直接子视图会成为主视图，用于显示主要页面内容，而抽屉部分的内容通过renderNavigationView属性调用函数来渲染。

常用属性有：

1、renderNavigationView：调用一个函数，用于渲染抽屉视图内容

2、drawerPosition：设置抽屉拖动方向。可选值有DrawerLayoutAndroid.positions.Leftt DrawerLayoutAndroid.positions.Right。

3、drawerWidth：设置抽屉的宽度

4、keyboardDismissMode：设置拖动过程中是否隐藏软键盘，可选值有none(不隐藏，默认值)和on-drag（拖动时隐藏）

5、onDrawerOpen：指定当抽屉视图打开时触发的函数

6、onDrawerClose：指定当抽屉视图关闭时触发的函数

7、onDrawerSlide：指定当抽屉内部发生交互时触发函数

8、onDrawerStateChanged：当抽屉状态变化地触发指定函数。抽屉有三种状态，分别是：

（1）idle：空闲状态，即没有发生任何交互

（2）dragging：正在拖动状态，用户正在进行交互

（3）settling：依靠中状态，用户刚刚结束交互

9、drawerLockMode：设置抽屉的锁定模式，有三种状态：

（1）unlocked：不锁定，可以响应打开和关闭操作，默认值；

（2）locked-losed：抽屉保持关闭，不能用手势打开

（3）locked-open：抽屉保持打开，不能用手势关闭

10、statusBarBackgroundColor：使抽屉占满整个屏幕，并设置状态栏颜色

DrawerLayoutAndroid提供了openDrawer方法用于打开抽屉，closeDrawer用于关闭抽屉。

'use strict';

import React, {

Component,

StyleSheet,

Text,

View,

DrawerLayoutAndroid,

TouchableHighlight

} from 'react-native';

class DrawerLayoutAndroidDemo extends Component {

\_openDrawer(){

this.refs['myDrawer'].openDrawer();

};

\_closeDrawer(){

this.refs['myDrawer'].closeDrawer();

}

render() {

var navView = (

<View style={{flex:1,padding:20}}>

<Text style={{textAlign:'center',lineHeight:50,fontSize:20}}>用户中心</Text>

<Text style={styles.item}>登录</Text>

<Text style={styles.item}>关于</Text>

<Text style={styles.item}>分享</Text>

<Text style={styles.item}>更新</Text>

<Text style={styles.item}>设置</Text>

<TouchableHighlight style={{padding:10,borderWidth:1,width:80}} onPress={this.\_closeDrawer.bind(this)}>

<Text>关闭抽屉</Text>

</TouchableHighlight>

</View>

);

return (

<DrawerLayoutAndroid

ref={'myDrawer'}

drawerWidth={300}

drawerPosition={DrawerLayoutAndroid.positions.Right}

statusBarBackgroundColor='#dd3333'

renderNavigationView={() => navView}>

<View style={styles.container}>

<Text style={styles.title}>DrawerLayoutAndroid组件</Text>

<Text>内容区域</Text>

<TouchableHighlight style={{padding:10,borderWidth:1,width:80}} onPress={this.\_openDrawer.bind(this)}>

<Text>打开抽屉</Text>

</TouchableHighlight>

</View>

</DrawerLayoutAndroid>

);

}

}

const styles = StyleSheet.create({

container:{flex:1},

divider:{height:1,backgroundColor:'#ccc',margin:10},

title:{fontSize:16,textAlign:'center',backgroundColor:'#0a8acd',color:'#fff',margin:10,lineHeight :40,textAlignVertical:'center'},

item:{lineHeight:30}

})

module.exports = DrawerLayoutAndroidDemo;



