

react native教程

编者 xiedrsz



中科软信息系统有限公司

sinosoft

目录

[简介 2](#_Toc476041853)

[概述 2](#_Toc476041854)

[搭建开发环境 2](#_Toc476041855)

[Hello World 2](#_Toc476041856)

# 简介

## 概述

React Native看起来很像React，只不过其基础组件是原生组件而非web组件。要理解React Native应用的基本结构，首先需要了解一些基本的React的概念，比如JSX语法、组件、state状态以及props属性。如果你已经了解了React，那么还需要掌握一些React Native特有的知识，比如原生组件的使用。

# 搭建开发环境

windows Android react native 开发环境已搭建完成，详见《react native开发环境搭建》

# Hello World

程序代码：

import React, { Component } from 'react';

import { AppRegistry, Text } from 'react-native';

class HelloWorldApp extends Component {

render() {

return (

<Text>Hello world!</Text>

);

}

}

// 注意，这里用引号括起来的'HelloWorldApp'必须和你init创建的项目名一致

AppRegistry.registerComponent('HelloWorldApp', () => HelloWorldApp);

你可以新建一个项目，然后用上面的代码覆盖你的index.ios.js或是index.android.js 文件，然后运行看看。

React Native内置了对**ES2015**标准的支持，你可以放心使用而无需担心兼容性问题。上面的示例代码中的import、from、class、extends、以及() =>箭头函数等新语法都是ES2015中的特性。

上面的代码定义了一个名为HelloWorldApp的新的组件（**Component**），并且使用了名为**AppRegistry**的内置模块进行了“注册”操作。你在编写React Native应用时，肯定会写出很多新的组件。而一个App的最终界面，其实也就是各式各样的组件的组合。组件本身结构可以非常简单——唯一必须的就是在render方法中返回一些用于渲染结构的JSX语句。

AppRegistry模块则是用来告知React Native哪一个组件被注册为整个应用的根容器。你无需在此深究，因为**一般在整个应用里AppRegistry.registerComponent这个方法只会调用一次**。上面的代码里已经包含了具体的用法，你只需整个复制到index.ios.js或是index.android.js文件中即可运行。

# Props属性

大多数组件在创建时就可以使用各种参数来进行定制。用于定制的这些参数就称为props（属性）。

以常见的基础组件Image为例，在创建一个图片时，可以传入一个名为source的prop来指定要显示的图片的地址，以及使用名为style的prop来控制其尺寸。

代码如下：

import React, { Component } from 'react';

import { AppRegistry, Image } from 'react-native';

class Bananas extends Component {

render() {

let pic = {

uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'

};

return (

<Image source={pic} style={{width: 193, height: 110}} />

);

}

}

AppRegistry.registerComponent('Bananas', () => Bananas);

注：在iOS上使用http链接的图片地址可能不会显示，参见：

<https://segmentfault.com/a/1190000002933776>

请注意{pic}外围有一层括号，我们需要用括号来把pic这个变量嵌入到JSX语句中。括号的意思是括号内部为一个js变量或表达式，需要执行后取值。因此我们可以把任意合法的JavaScript表达式通过括号嵌入到JSX语句中。

自定义的组件也可以使用props。通过在不同的场景使用不同的属性定制，可以尽量提高自定义组件的复用范畴。只需在render函数中引用this.props，然后按需处理即可。下面是一个例子：

import React, { Component } from 'react';

import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {

render() {

return (

<Text>Hello {this.props.name}!</Text>

);

}

}

class LotsOfGreetings extends Component {

render() {

return (

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

<Greeting name='Rexxar' />

<Greeting name='Jaina' />

<Greeting name='Valeera' />

</View>

);

}

}

AppRegistry.registerComponent('LotsOfGreetings', () => LotsOfGreetings);

上面的例子出现了一样新的名为View的组件。View 常用作其他组件的容器，来帮助控制布局和样式。

# State状态

我们使用两种数据来控制一个组件：**props**和**state**。props是在父组件中指定，而且一经指定，在被指定的组件的生命周期中则不再改变。 对于需要改变的数据，我们需要使用state。

一般来说，你需要在**constructor**中初始化state（译注：这是ES6的写法，早期的很多ES5的例子使用的是getInitialState方法来初始化state，这一做法会逐渐被淘汰），然后在需要修改时调用**setState**方法。

假如我们需要制作一段不停闪烁的文字。文字内容本身在组件创建时就已经指定好了，所以文字内容应该是一个prop。而文字的显示或隐藏的状态（快速的显隐切换就产生了闪烁的效果）则是随着时间变化的，因此这一状态应该写到state中：

import React, { Component } from 'react';

import { AppRegistry, Text, View } from 'react-native';

class Blink extends Component {

constructor(props) {

super(props);

this.state = { showText: true };

// 每1000毫秒对showText状态做一次取反操作

setInterval(() => {

this.setState({ showText: !this.state.showText });

}, 1000);

}

render() {

// 根据当前showText的值决定是否显示text内容

let display = this.state.showText ? this.props.text : ' ';

return (

<Text>{display}</Text>

);

}

}

class BlinkApp extends Component {

render() {

return (

<View>

<Blink text='I love to blink' />

<Blink text='Yes blinking is so great' />

<Blink text='Why did they ever take this out of HTML' />

<Blink text='Look at me look at me look at me' />

</View>

);

}

}

AppRegistry.registerComponent('BlinkApp', () => BlinkApp);

实际开发中，我们一般不会在定时器函数（setInterval、setTimeout等）中来操作state。典型的场景是在接收到服务器返回的新数据，或者在用户输入数据之后。你也可以使用一些“状态容器”比如Redux来统一管理数据流（译注：但我们不建议新手过早去学习redux <http://redux.js.org/index.html>）。

State的工作原理和React.js完全一致。