

# React Native学习文档

## 一、开发环境

### 安装的依赖：

- 安装 Node 12 以上的版本(目前我使用**12.16.3**)
- Java Development Kit [JDK] 1.8 （暂不支持 1.9 及更高版本）
- 安装Android SDK （目前我使用**24.4.1**）

### Node的安装与配置

- 先到 [官网](#) 去下载node版本
- 到~/.bashrc文件中配置对应node环境

```
#export NODE_HOME=/home/zys/software/node-v12.16.3-linux-x64
export PATH=${JAVA_HOME}/bin:$NODE_HOME/bin
```

- 使用 `source ~/.bashrc` 使环境生效
- 使用 `node -v` 查看版本是否生效

### Yarn的安装

- 安装yarn








```
npm install yarn -g // 使用npm全局安装yarn
```

- 检查是否安装成功

```
yarn -v
```

### JDK的安装与配置

- 到[该网站](#)下载JDK

Product / File Description	File Size	Download
Linux ARM 64 RPM Package	59.1 MB	 <a href="#">jdk-8u291-linux-aarch64.rpm</a>
Linux ARM 64 Compressed Archive	70.79 MB	 <a href="#">jdk-8u291-linux-aarch64.tar.gz</a>
Linux ARM 32 Hard Float ABI	73.5 MB	 <a href="#">jdk-8u291-linux-arm32-vfp-hflt.tar.gz</a>
Linux x86 RPM Package	109.05 MB	 <a href="#">jdk-8u291-linux-i586.rpm</a>
Linux x86 Compressed Archive	137.92 MB	 <a href="#">jdk-8u291-linux-i586.tar.gz</a>
Linux x64 RPM Package	108.78 MB	 <a href="#">jdk-8u291-linux-x64.rpm</a>
Linux x64 Compressed Archive	138.22 MB	 <a href="#">jdk-8u291-linux-x64.tar.gz</a>

- 配置环境变量

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:${JRE_HOME}/bin
```

- 使用 `source ~/.bashrc` 使环境生效
- 使用 `java -version` 查看版本是否生效

## Android SDK的安装与配置

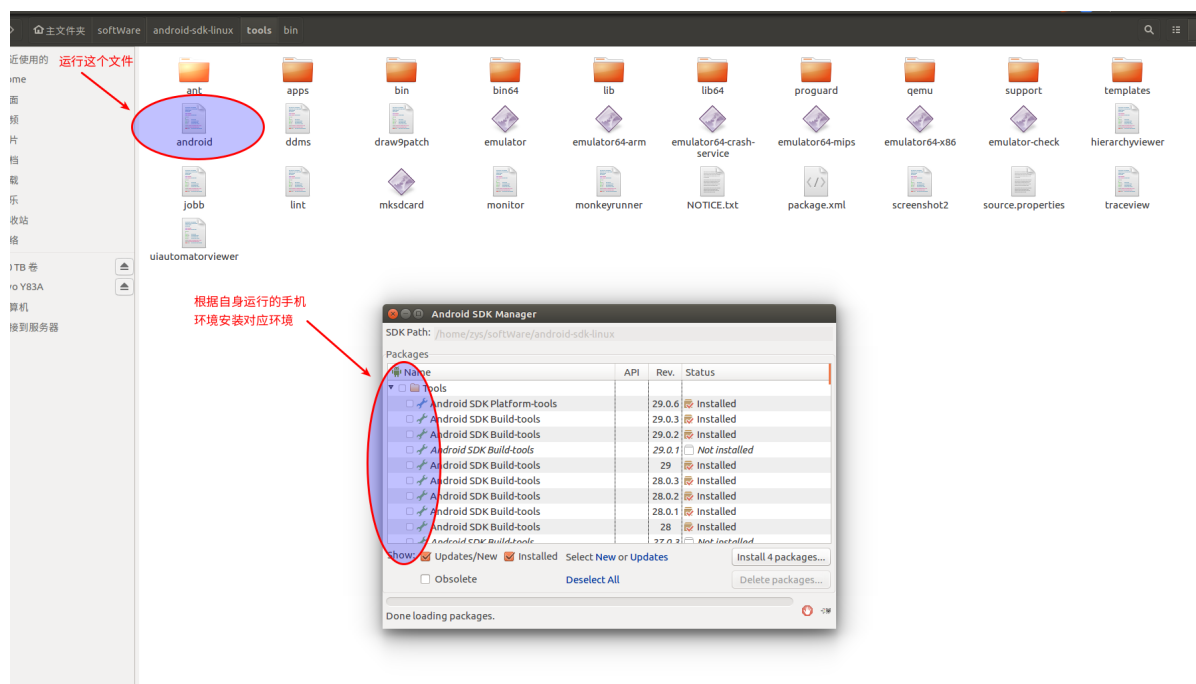
- 首先打开 [网站](#)，然后一直向下拉，找到 SDK Tools 进行下载

版本	平台	下载	大小	SHA-1校验码	官方SHA-1校验码截图
3859397	Windows	<a href="#">sdk-tools-windows-3859397.zip</a>	132 MB	7f6037d3a7d6789b4fdc06ee7af041e071e9860c51f667a4eb5913df9871fd2	<a href="#">查看</a>
	Mac OS X	<a href="#">sdk-tools-darwin-3859397.zip</a>	82 MB	4a81754a760fce88cba74d69c364b05b31c53d57b26f9f82355c61d5fe4b9df9	
	Linux	<a href="#">sdk-tools-linux-3859397.zip</a>	130 MB	444e22ce8ca0f67353bda4b85175ed3731cae3ffa695ca18119cbacef1c1bea0	
24.4.1	Windows	<a href="#">installer_r24.4.1-windows.exe</a>	144 MB	f9b59d72413649d31e633207e31f456443e7ea0b	<a href="#">查看</a>
		<a href="#">android-sdk_r24.4.1-windows.zip</a>	190 MB	66b6a6433053c152b22bf8cab19c0f3fef4eba49	
	Mac OS X	<a href="#">android-sdk_r24.4.1-macosx.zip</a>	98 MB	85a9cccb0b1f9e61f616335c5f07107553840cd	
	Linux	<a href="#">androiddk_r24.4.1-linux.tgz</a>	311 MB	725bb360f0f7d04eacff5a2d57abdd49061326d	

- 配置环境变量

```
export ANDROID_HOME=/home/zys/software/android-sdk-linux
export PATH=${ANDROID_HOME}/tools:${ANDROID_HOME}/platform-tools:$PATH
```

- 使用 `source ~/.bashrc` 使环境生效
- 运行Android SDK Manager及安装对应手机环境



## 二、开发工具

- [VS Code](#)是目前非常受JS 开发者欢迎的 IDE 工具
- [Ignite](#)是一套整合了 Redux 以及一些常见 UI 组件的脚手架。它带有一个命令行可以生成 app、组件或是容器。如果你喜欢它的选择搭配，那么不妨一试。
- [App Center](#)是由微软提供的热更新服务。热更新可以使你绕过 AppStore 的审核机制，直接修改已经上架的应用。对于国内用户，我们也推荐由本网站提供的[Pushy](#)热更新服务
- 调试工具使用rn推荐的工具 [react-native-debugger](#)来调试
  1. 可以查看标签结构
  2. 不能查看网络请求
  3. 想要查看网络请求

找到项目的入口文件 `index.js` 加入以下代码即可

```
GLOBAL.XMLHttpRequest = GLOBAL.originalXMLHttpRequest ||
GLOBAL.XMLHttpRequest
```

## 三、React Native Demo

### 搭建react native demo

1. 使用下面的命令创建react native项目

```
npx react-native init 项目名
```

2. 文件目录结构

	App.js	---	项目的根组件
	index.js	---	项目的入口文件
	package.json	---	项目的描述文件
	.eslintrc.js	---	eslint的配置文件
	.prettierrc.js	---	格式化配置文件
	android	---	编译安卓相关
	ios	---	编译ios相关

3. Index.js代码内容

```
import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

4. App.js代码

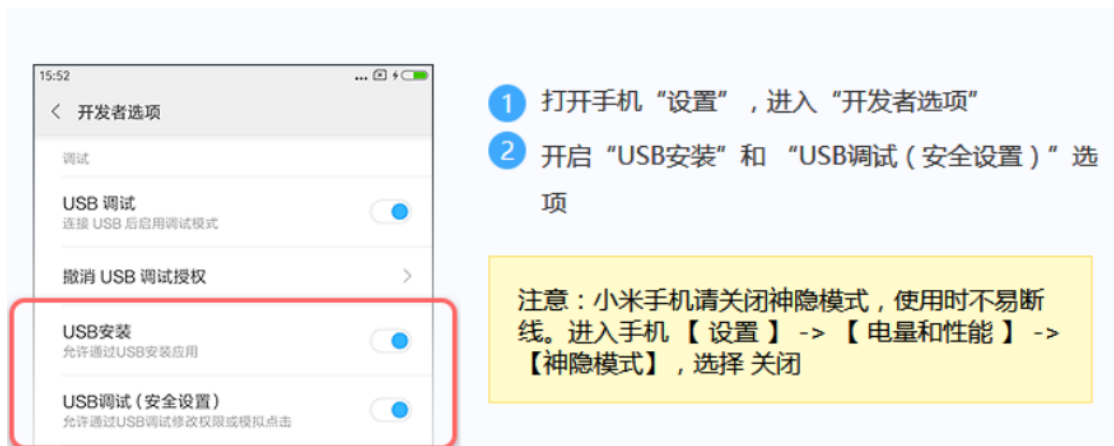
```
import React from 'react';
import type {Node} from 'react';
import {
  Text,
  View,
} from 'react-native';

const App: () => Node = () => {
```

```
return (  
  <View style={{flex:1}}>  
    <Text>Hello World</Text>  
  </View>  
)  
};  
  
export default App;
```

## 安装并运行apk

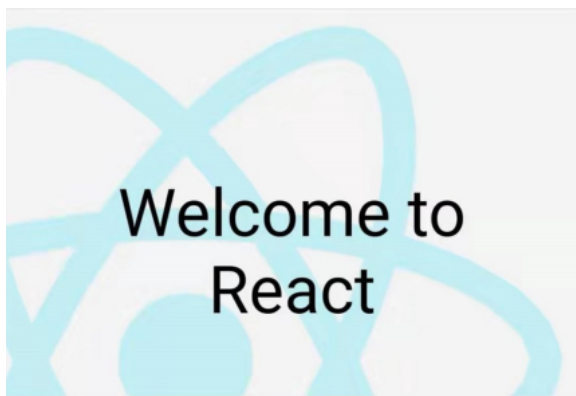
1. 准备一台 Android 手机, 通过数据线 连接 到电脑, 设置启用 USB调试
2. 如果没有安卓手机, 可以使用安卓模拟器也可以, 推荐使用 Genymotion , 自行百度下载安装
3. 一般的手机在 设置 中可以直接找到 开发者选项 进行开启, 如果 找不到 , 就自行百度查一下



4. 手机连接电脑成功后运行检测命令 `adb devices` , 如果有输出设备列表与 ID 相关的字符串就证明手机和电脑是连接成功了, 如果没有显示设备号, 则说明连接有问题, 一定要保证手机和电脑是正常连接状态

```
$ adb devices  
List of devices attached  
feeb8f39                device
```

5. 运行 `yarn android` 安装apk到手机上
6. 运行 `yarn start` 查看手机上安装的内容
7. 运行成功手机上的界面效果如下:



## Step One

Edit **App.js** to change this screen and then come back to see your edits.

## See Your Changes

Double tap **R** on your keyboard to reload your app's code.

## Debug

Press **Cmd or Ctrl + M** or **Shake** your device to open the React Native debug menu.

## 四、基础知识

---

主要讲解和web开发的不同之处

- flex布局
- 样式继承
- 单位
- 屏幕宽度和高度
- 变换

### flex布局

- 所有容器默认都是 `flexbox`
- 默认是纵向排列 也就是 `flex-direction:column`

### 样式继承

背景颜色、字体颜色、字体大小等没有继承

### 单位

- 不能加 `px` 单位
- 不能加 `vw` `vh` 等单位
- 可以加百分比单位
- 表示的是与设备像素密度无关的逻辑像素点

## 屏幕宽度和高度

```
import {Dimensions} from "react-native";
const screenWidth = Math.round(Dimensions.get('window').width);
const screenHeight = Math.round(Dimensions.get('window').height);
```

## 变换

```
<Text style={{transform:[{translateY:300},{scale:2}]}>变换</Text>
```

## 插值表达式

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

const Index = () => <View>
  <Text>{"开心"}</Text>
  <Text>{123}</Text>
</View>

export default Index;
```

## 组件

- 函数组件
  - 没有state (通过hooks可以有)
  - 没有生命周期(通过hooks可以有)
  - 适合简单的场景
- 类组件
  - 适合复杂的场景
  - 有state
  - 有生命周期

### 函数组件

```
class Index extends Component {
  render() {
    return (
      <View>
        <Btn></Btn>
      </View>
    );
  }
}
// 函数组件
const Btn = () => <Button title="点我" />
```

## 类组件

```
import React, { Component } from 'react';
import { View, Text } from 'react-native';
class Index extends Component {
  render() {
    return (
      <View>
        <Text>类组件</Text>
      </View>
    );
  }
}
export default Index;
```

## 状态 state

```
import React, { Component } from 'react';
import { View, Text } from 'react-native';
class Index extends Component {
  // 1 声明state
  state = {
    num: 100
  }
  render() {
    return (
      <View>
        { /* 2 使用state */ }
        <Text onPress={this.handlePress} >{this.state.num}</Text>
      </View>
    );
  }
  // 3 修改state
  handlePress = () => {
    this.setState({ num: 1010 });
  }
}
export default Index;
```

## 属性 props

父子传递数据的关键

```
import React, { Component } from 'react';
import { View, Text } from 'react-native';
class Index extends Component {
  render() {
    return (
      <View>
        <BigText fontColor="red" >大博妞</BigText>
      </View>
    );
  }
}
```

```

class BigText extends Component {
  render() {
    // 通过props来接收父组件传递的数据
    return <Text style={{ color: this.props.fontColor }} >
      { /* children 其实就是插槽 类似vue中的slot */ }
      {this.props.children}
    </Text>
  }
}

export default Index;

```

## 事件

绑定时间需要特别注意 this的指向问题,可以总结为如下的方式

- 使用箭头函数
- 通过bind重新绑定this
- 匿名函数

```

import React, { Component } from 'react';
import { View, Text } from 'react-native';
class Index extends Component {
  state = { num: 100 }
  // 丢失 state
  handlePress1() {
    console.log(this.state);
  }
  // 正常
  handlePress2 = () => {
    console.log(this.state);
  }
  // 正常
  handlePress3() {
    console.log(this.state);
  }
  // 正常
  handlePress4() {
    console.log(this.state);
  }
  // 正常
  render() {
    return (
      <View>
        { /* 导致事件函数中获取不到state */ }
        <Text onPress={this.handlePress1} >事件1</Text>
        { /* 正常 */ }
        <Text onPress={this.handlePress2} >事件1</Text>
        { /* 正常 */ }
        <Text onPress={this.handlePress3.bind(this)} >事件3</Text>
        { /* 正常 */ }
        <Text onPress={() => this.handlePress4()} >事件4</Text>
      </View>
    );
  }
}

```

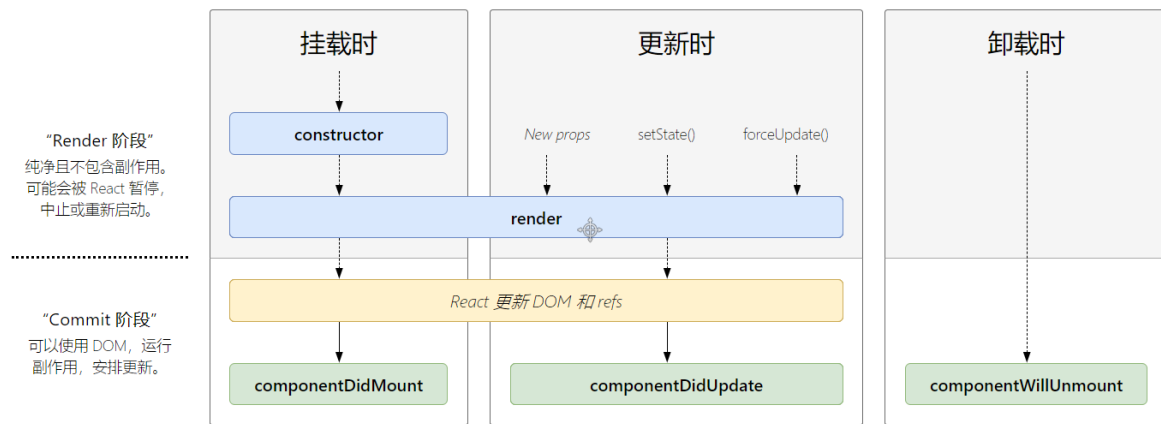


```
export default Index;
```

## 生命周期

生命周期指的是 `react` 组件 的从创建到销毁的整个过程中会自动触发的函数

[在线图示](#)



## 主要的生命周期

- `constructor`
  - 组件被实例化的时候出发 一般用做对组件做初始工作,如设置 `state` 等
- `render`
  - 组件开始渲染时触发
  - 组件被更新时触发 - `state`和`props`发生改变时触发
- `componentDidMount`
  - 组件挂载完毕,可以发送异步请求获取数据
- `componentWillUnmount`
  - 组件被卸载时触发
  - 一般用在清除定时器或者取消订阅等

## 五、组件

1. View
2. Text
3. TouchableOpacity
4. Image
5. ImageBackground
6. TextInput
7. 其他 (自己查阅[官网](#))
  1. button
  2. FlatList
  3. ScrollView
  4. StatusBar
  5. TextInput

## View

- 相当于以前 web 中的 div
- 不支持设置字体大小,字体颜色等
- 不能直接放文本内容
- 不支持直接绑定点击事件 (一般使用 `TouchableOpacity` 来代替)

## Text

### 文本标签

- 文本标签 可以设置字体颜色、大小等
- 支持绑定点击事件
- `<Text>` 元素在布局上不同于其它组件：在 Text 内部的元素不再使用 flexbox 布局，而是采用文本布局。这意味着 `<Text>` 内部的元素不再是一个个矩形，而可能会在行末进行折叠。
- React Native 实际上还是有一部分样式继承的实现，不过仅限于文本标签的子树。

```
<Text style={{ fontWeight: 'bold' }}>
  I am bold
  <Text style={{ color: 'red' }}>and red</Text>
</Text>
```

- 支持嵌套文本

```
<Text style={styles.baseText}>
  I am bold
  <Text style={styles.innerText}> and red</Text>
</Text>
```

- 嵌套视图 (仅限 iOS)

```
<Text>
  There is a blue square
  <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}}
/>

  in between my text.
</Text>
```

## TouchableOpacity

### 可以绑定点击事件的块级标签

- 相当于块级的容器
- 支持绑定点击事件 `onPress`
- 可以设置点击时的透明度

```
<TouchableOpacity activeOpacity={0.5} onPress={this.handleOnPress} >
</TouchableOpacity>
```

# Image

## 图片标签

- 渲染本地图片时

```
<Image source={require("../girl.png")} />
```

- 渲染网络图片时,必须加入宽度和高度

```
<Image source={{uri:"https://timgsa.baidu.com/timg?image&quality=80&size=b9999_10000&sec=1590514654506&di=38fa919d4c78fb776536b922bb94eec3&imgtype=0&src=http%3A%2F%2Fimages.ali213.net%2Fpicfile%2Fpic%2F2013%2F03%2F28%2F927_xgzwl%2520%25281%2529.jpg"}} style={{width:200,height:300}}} />
```

- 在 Android 上支持 GIF 和 WebP 格式图片

默认情况下 Android 是不支持 GIF 和 WebP 格式的。你需要在 `android/app/build.gradle` 文件中根据需要手动添加以下模块:

```
dependencies {  
    // 如果你需要支持Android4.0(API level 14)之前的版本  
    implementation 'com.facebook.fresco:animated-base-support:1.3.0'  
  
    // 如果你需要支持GIF动图  
    implementation 'com.facebook.fresco:animated-gif:2.0.0'  
  
    // 如果你需要支持WebP格式, 包括WebP动图  
    implementation 'com.facebook.fresco:animated-webp:2.1.0'  
    implementation 'com.facebook.fresco:webpsupport:2.0.0'  
  
    // 如果只需要支持WebP格式而不需要动图  
    implementation 'com.facebook.fresco:webpsupport:2.0.0'  
}
```

# ImageBackground

一个可以使用图片当作背景的容器,相当于以前的 `div+背景图片`, 必须指定宽高样式

```
<ImageBackground source={...} style={{width: '100%', height: '100%'}}>  
    <Text>Inside</Text>  
</ImageBackground>
```

# TextInput

## 输入框组件

- 可以通过 `onChangeText` 事件来获取输入框的值
- 本组件的属性提供了多种特性的配置, 譬如自动完成、自动大小写、占位文字, 以及多种不同的键盘类型 (如纯数字键盘) 等等
- 当 `multiline=false` 时, 为元素的某一个边添加边框样式 (例如: `borderBottomColor`, `borderLeftWidth` 等) 将不会生效。为了能够实现效果你可以使用一个 `View` 来包裹 `TextInput`

```

<View
  style={{
    backgroundColor: value,
    borderBottomColor: '#000000',
    borderBottomWidth: 1,
  }}>
  <TextInput
    multiline
    onChangeText={text => onChangeText(text)}
    value={value}
  />
</View>

```

## WebView

`WebView` 创建一个原生的 `WebView`，可以用于访问一个网页。

```

<WebView
  source={{uri: 'https://github.com/facebook/react-native'}}
  style={{marginTop: 20}}
/>

```

## FlatList、ScrollView、SectionList三者的区别

- `FlatList`组件用于显示一个垂直的滚动列表，其中的元素之间结构近似而仅数据不同，适于长列表数据，且元素个数可以增删。和`ScrollView`不同的是，`FlatList`并不立即渲染所有元素，而是优先渲染屏幕上可见的元素。`SectionList`主要是渲染一组需要分组的数据。

## TouchableHighlight、TouchableNativeFeedback、TouchableOpacity、TouchableWithoutFeedback区别

- [TouchableHighlight](#)来制作按钮或者链接。注意此组件的背景会在用户手指按下时变暗。
- [TouchableNativeFeedback](#)，它会在用户手指按下时形成类似墨水涟漪的视觉效果。
- [TouchableOpacity](#)会在用户手指按下时降低按钮的透明度，而不会改变背景的颜色。
- [TouchableWithoutFeedback](#)处理点击事件的同时不显示任何视觉反馈