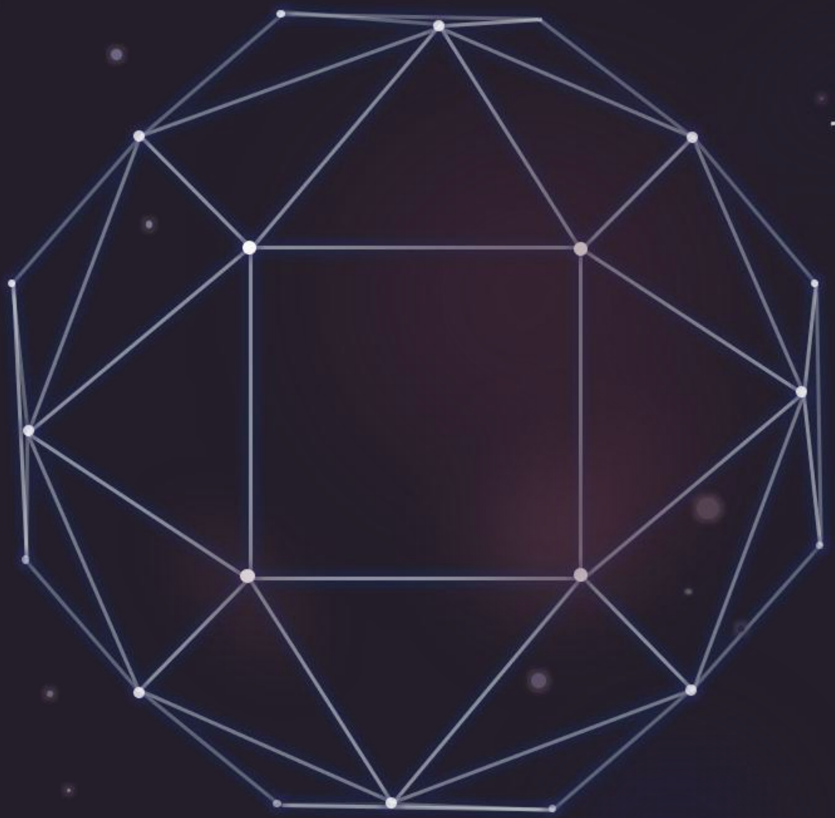
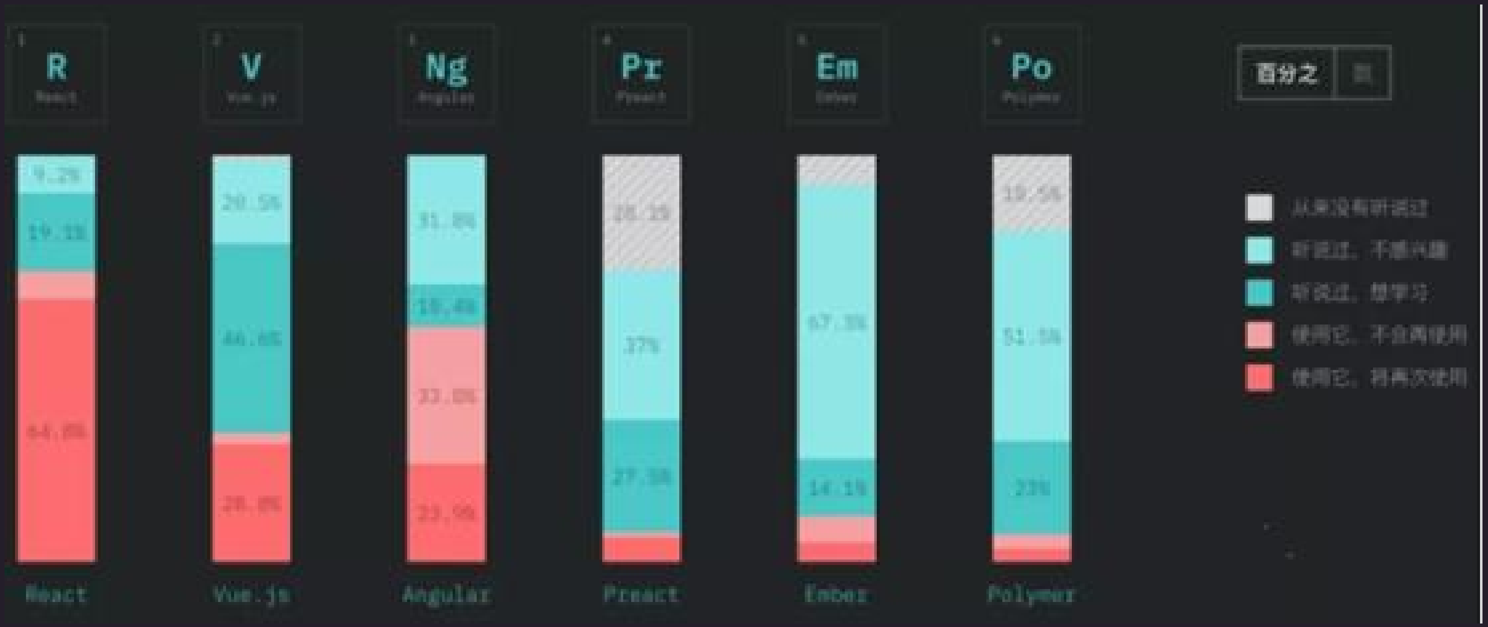


2019

React-Native

从零开发一款App



除Facebook，Instagram，Netflix，微软等众多国际知名互联网公司都是React.js的拥趸者外，国内很多主流互联网公司如腾讯、蚂蚁金服、京东、360、美团、携程等也在用React

目录/CONTENTS

目录/CONTENTS

1-搭架开发环境之react-native

步骤1 请准备装有macOs系统的电脑

步骤2 安装Node和watchman

```
brew install node
brew install watchman
```

注:Watchman - 用于更改的文件和目录监视工具

步骤3 安装yarn

```
npm install -g yarn
```

注:(Yarn是 Facebook 提供的替代 npm 的工具，可以加速 node 模块的下载)
官网:https://yarn.bootcss.com/docs/getting-started/

步骤4 react-native-cli

```
npm install -g react-native-cli
```

注:React Native 的命令行工具用于执行创建、初始化、更新项目、运行打包服务（packager）等任务

步骤5 下载VSCode
下载地址:https://code.visualstudio.com/

步骤6 安装vs插件 React Native Snippet
快速代码补全
(rnce - 创建组件)
(rncsl - 创建组件)
(rncslwc - 创建可以传递子组件的组件)
(rnss - 快速创建样式)

步骤7 安装sdkman工具

```
curl -s "https://get.sdkman.io" | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"
sdk version
```

1-搭架开发环境之iOS

1.在Appstore中下载Xcode

2.申请开发者账号

准备:手机号\邮箱\信用卡(Visa或者master)
参考
https://jingyan.baidu.com/article/a501d80c671653ec630f5e07.html

3.制作下载安装开发者证书(开发环境+产线环境)
https://jingyan.baidu.com/article/afd8f4de8210eb34e286e9ef.html

4.安装iOS包管理工具(cocoapods)
安装此软件需要翻墙,推荐不翻墙的方法
需要先安装ruby环境(mac 系统默认安装好的)
- 输入gem查看
- 移除默认的镜像地址

```
gem  sources --remove https://rubygems.org/
```

- 安装国内的镜像

```
gem sources -a https://gems.ruby-china.com
```

- 验证是否替换成功

```
gem sources -l
```

- 安装cocoapods

```
sudo gem install cocoapods
```

1-搭架开发环境之android

1.下载android studio
下载地址:https://developer.android.google.cn/studio/

2.安装Java 环境
查看本地java版本 java -version
推荐安装方式
sdk install java 13.0.1.j9-adpt

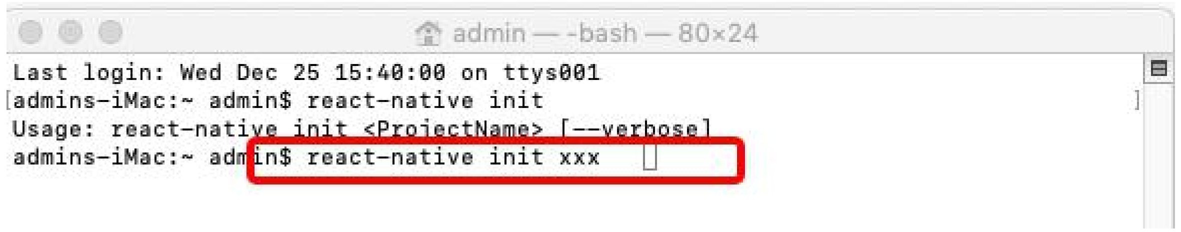
3.下载包管理工具(gradle)
sdk install gradle 5.5

4.安装模拟器
教程地址 https://jingyan.baidu.com/article/4f34706e088aabe387b56d3c.html

2-运行第一个demo

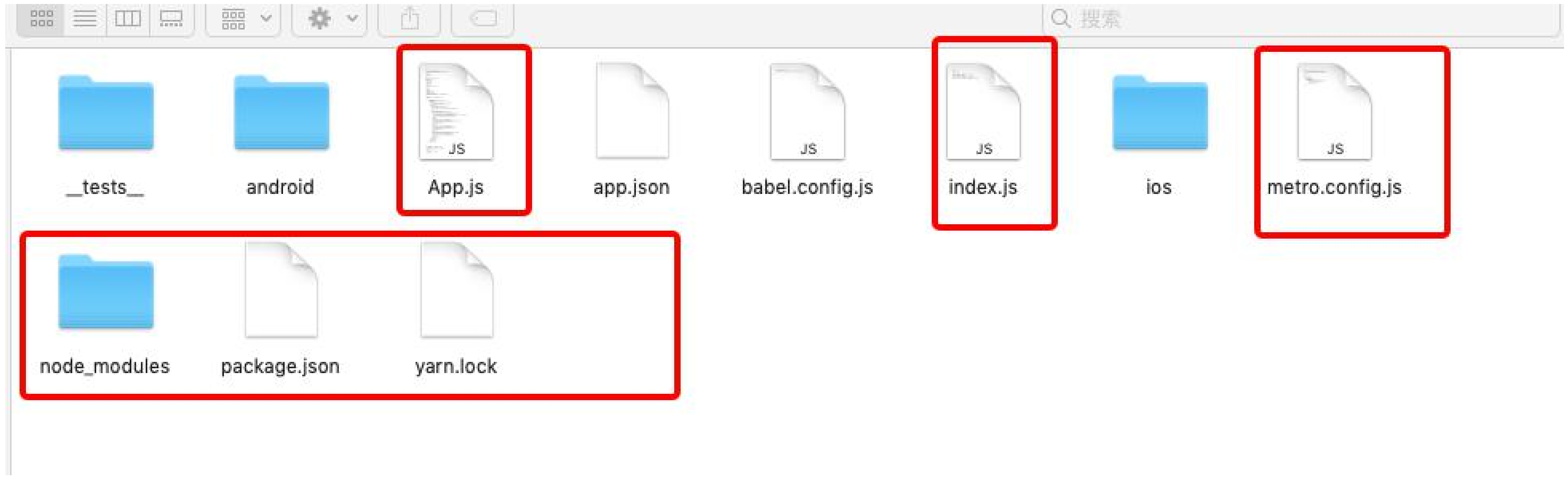
步骤1.使用命令行初始化项目

react-native init 项目名称



```
Last login: Wed Dec 25 15:40:00 on ttys001
admins-iMac:~ admin$ react-native init
Usage: react-native init <ProjectName> [--verbose]
admins-iMac:~ admin$ react-native init xxx
```

初始化完成后的项目结构如下



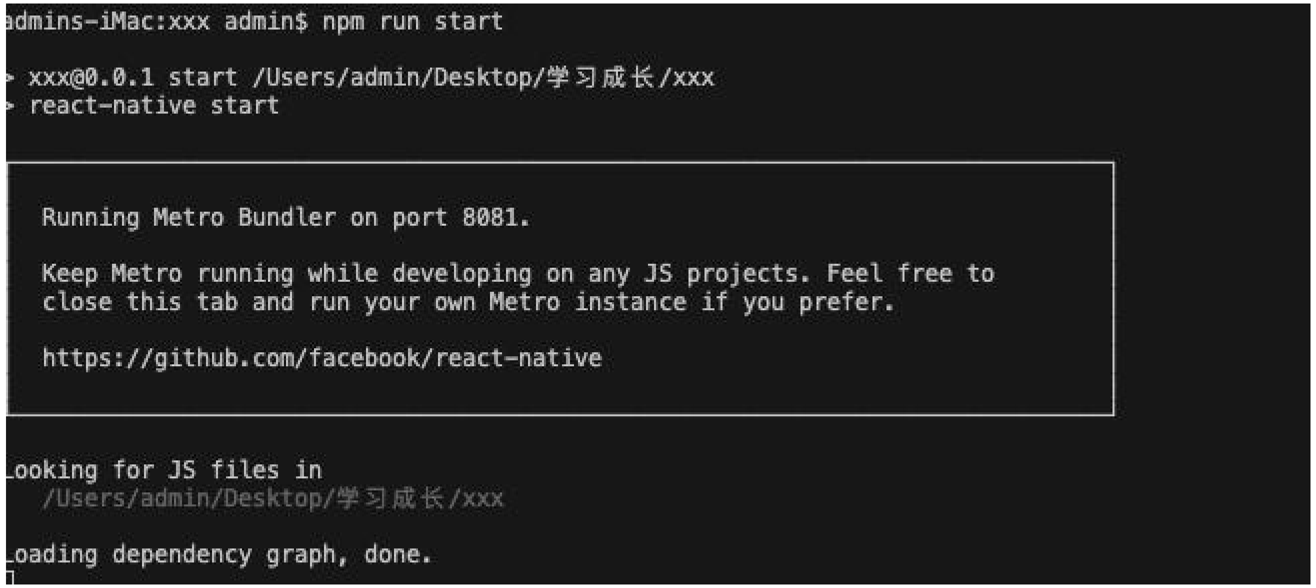
2-运行第一个demo

步骤2.使用命令行进行项目里面安装依赖包

cd xxx & yarn

步骤3.运行项目

npm run start



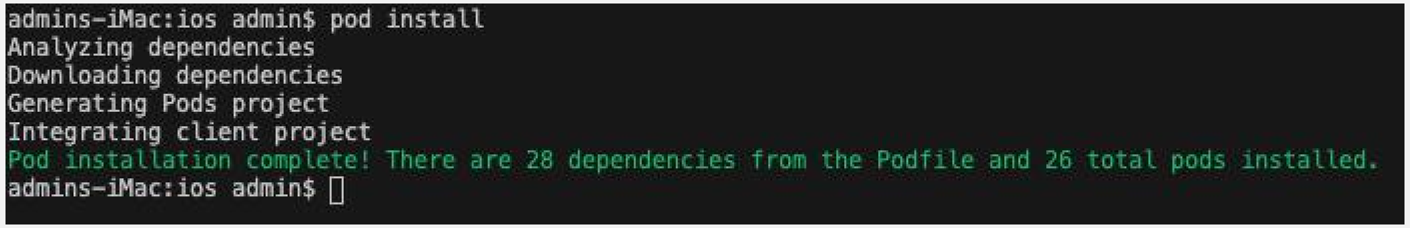
```
admins-iMac:xxx admin$ npm run start

> xxx@0.0.1 start /Users/admin/Desktop/学习成长/xxx
> react-native start

┌──────────────────────────────────────────────────────────────────────────────┐
│                                                                              │
│  Running Metro Bundler on port 8081.                                         │
│                                                                              │
│  Keep Metro running while developing on any JS projects. Feel free to        │
│  close this tab and run your own Metro instance if you prefer.               │
│                                                                              │
│  https://github.com/facebook/react-native                                    │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘

Looking for JS files in
   /Users/admin/Desktop/学习成长/xxx

Loading dependency graph, done.
```

2-运行第一个demo之iOS

1.使用命令行进入iOS项目中安装相关依赖

cd ./ios

pod install

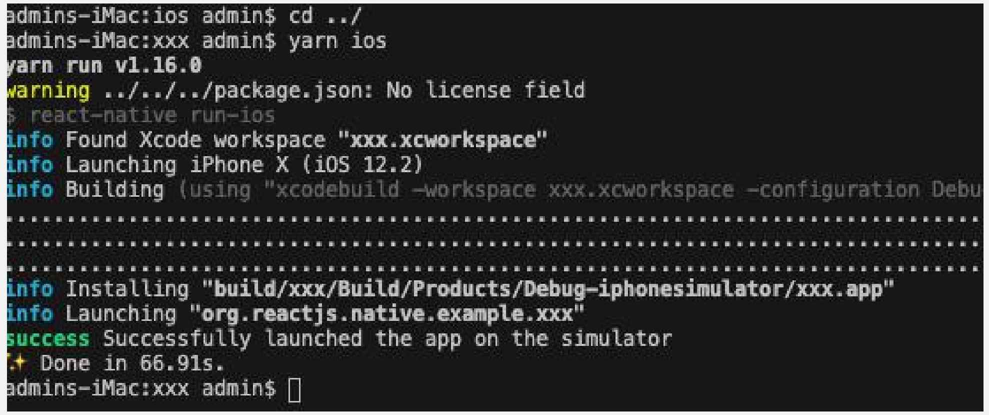


```
admins-iMac:ios admin$ pod install
Analyzing dependencies
Downloading dependencies
Generating Pods project
Integrating client project
Pod installation complete! There are 28 dependencies from the Podfile and 26 total pods installed.
admins-iMac:ios admin$ 
```

2.在模拟器上运行iOS工程

cd ../

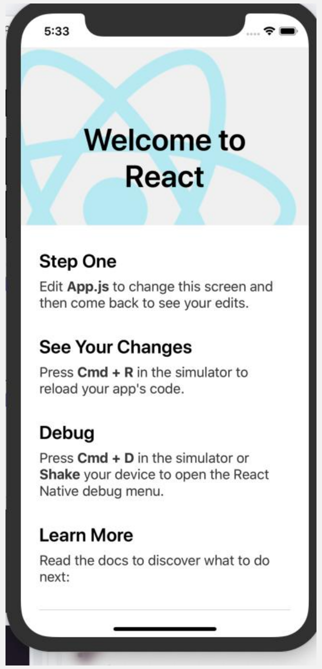
yarn ios(或者npm run ios)



```
admins-iMac:ios admin$ cd ../
admins-iMac:xxx admin$ yarn ios
yarn run v1.16.0
warning ../../../package.json: No license field
$ react-native run-ios
info Found Xcode workspace "xxx.xcworkspace"
info Launching iPhone X (iOS 12.2)
info Building (using "xcodebuild -workspace xxx.xcworkspace -configuration Debu
..............................................................................
..............................................................................
..............................................................................
info Installing "build/xxx/Build/Products/Debug-iphonesimulator/xxx.app"
info Launching "org.reactjs.native.example.xxx"
success Successfully launched the app on the simulator
✨ Done in 66.91s.
admins-iMac:xxx admin$ 
```

2-运行第一个demo

运行起来的效果图



5:33

Welcome to React

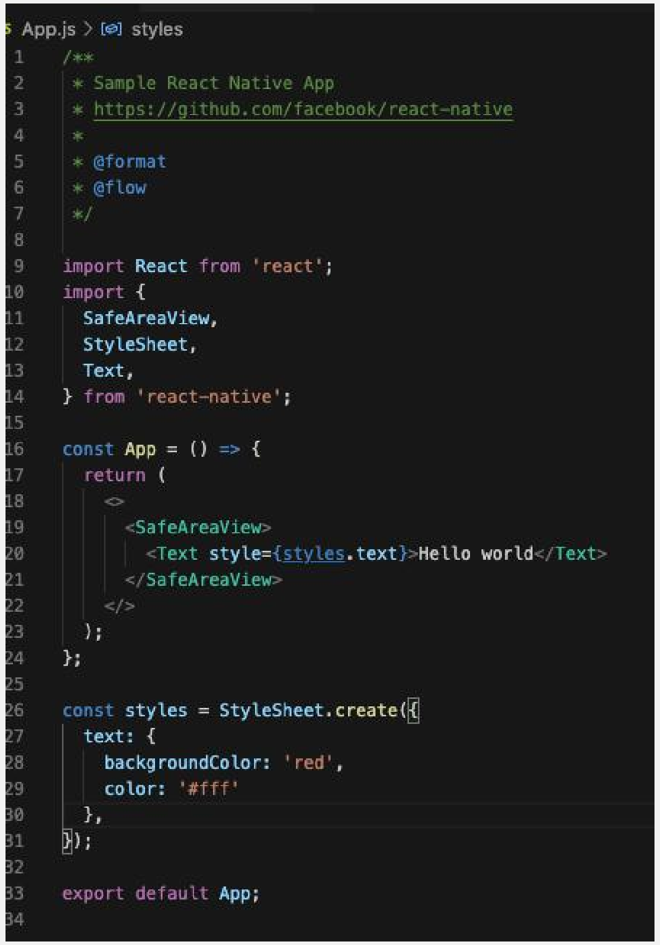
Step One
Edit **App.js** to change this screen and then come back to see your edits.

See Your Changes
Press **Cmd + R** in the simulator to reload your app's code.

Debug
Press **Cmd + D** in the simulator or **Shake** your device to open the React Native debug menu.

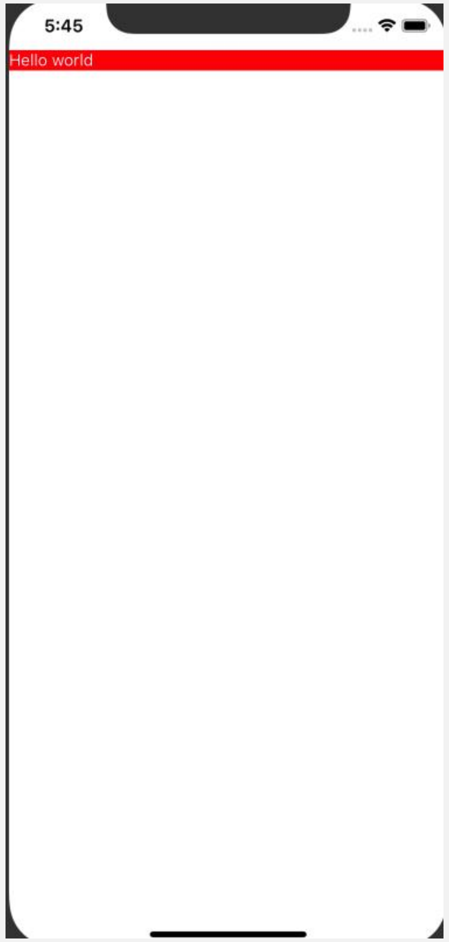
Learn More
Read the docs to discover what to do next:

修改一下App.js文件里面的代码,保存一下,模拟器的页面立即进行响应



App.js > styles

```
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 *
 * @format
 * @flow
 */

import React from 'react';
import {
  SafeAreaView,
  StyleSheet,
  Text,
} from 'react-native';

const App = () => {
  return (
    <>
      <SafeAreaView>
        <Text style={styles.text}>Hello world</Text>
      </SafeAreaView>
    </>
  );
};

const styles = StyleSheet.create({
  text: {
    backgroundColor: 'red',
    color: '#fff'
  },
});

export default App;
```

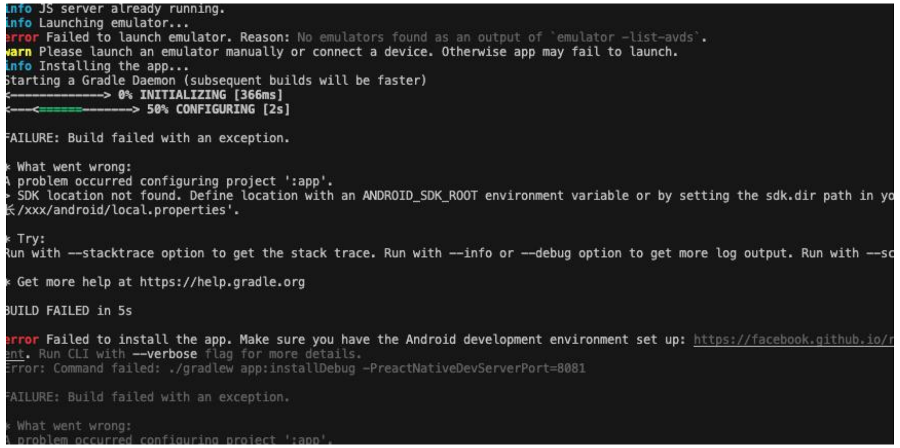


5:45

Hello world

2-运行第一个demo之android

步骤1 运行命令启动android
npm run android

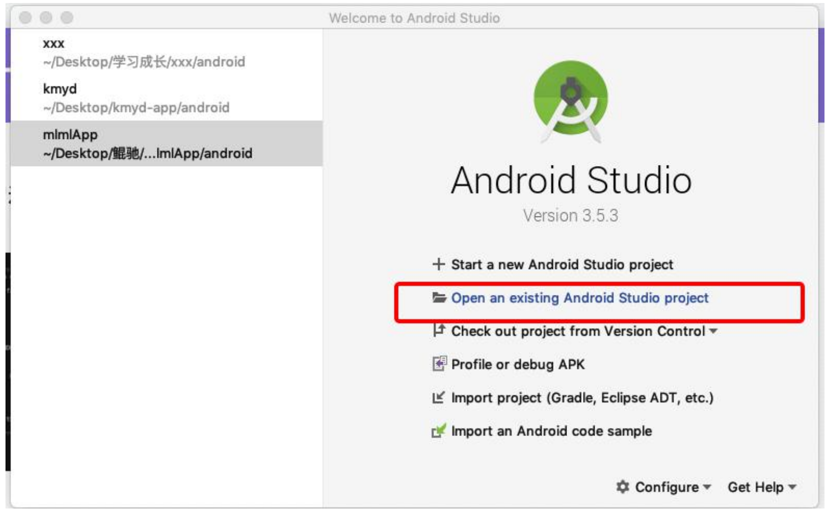


```
info JS server already running.
info Launching emulator...
error Failed to launch emulator. Reason: No emulators found as an output of `emulator -list-avds`.
warn Please launch an emulator manually or connect a device. Otherwise app may fail to launch.
info Installing the app...
Starting a Gradle Daemon (subsequent builds will be faster)
<-------------> 0% INITIALIZING [366ms]
<---<======-------> 50% CONFIGURING [2s]

FAILURE: Build failed with an exception.

* What went wrong:
A problem occurred configuring project ':app'.
> SDK location not found. Define location with an ANDROID_SDK_ROOT environment variable or by setting the sdk.dir path in yo
长/xxx/android/local.properties'.

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output. Run with --sc

* Get more help at https://help.gradle.org

BUILD FAILED in 5s

error Failed to install the app. Make sure you have the Android development environment set up: https://facebook.github.io/r
ent. Run CLI with --verbose flag for more details.
Error: Command failed: ./gradlew app:installDebug -PreactNativeDevServerPort=8081

FAILURE: Build failed with an exception.

* What went wrong:
A problem occurred configuring project ':app'.
```

步骤2 使用android studio 打开android项目

2-运行第一个demo之android

步骤2 使用android studio 打开android项目


Welcome to Android Studio

xxx
~/Desktop/学习成长/xxx/android

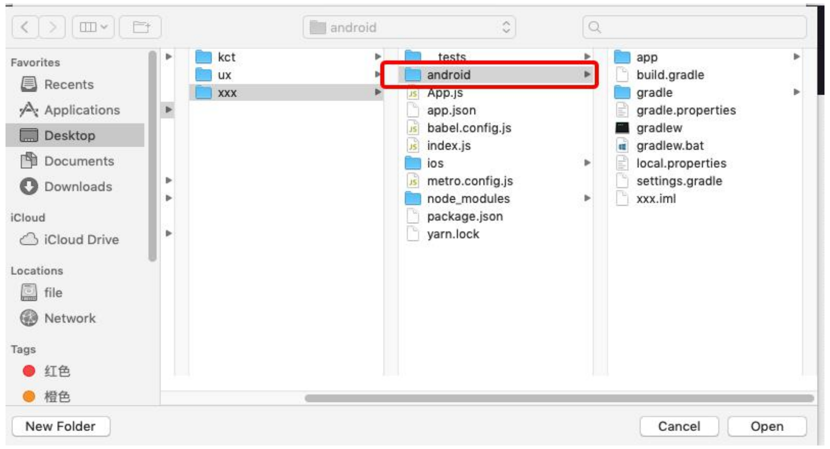
kmyd
~/Desktop/kmyd-app/android

mlmlApp
~/Desktop/鲲驰/...lmlApp/android

Android Studio
Version 3.5.3

+ Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control ▾
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample

Configure ▾ Get Help ▾


android

Favorites: Recents, Applications, Desktop, Documents, Downloads
iCloud: iCloud Drive
Locations: file, Network
Tags: 红色, 橙色

kct
ux
xxx

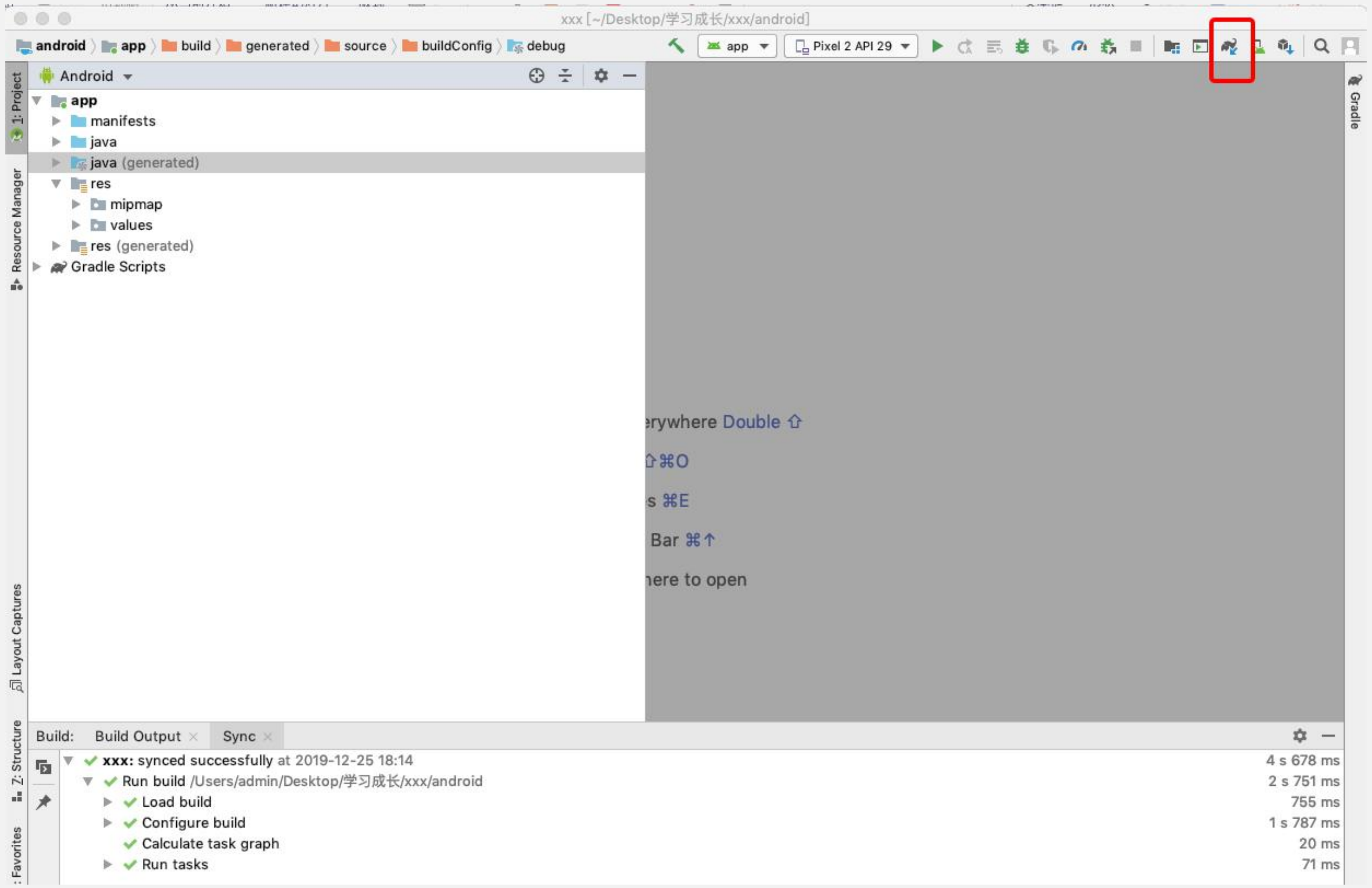
__tests__
android
App.js
app.json
babel.config.js
index.js
ios
metro.config.js
node_modules
package.json
yarn.lock

app
build.gradle
gradle
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle
xxx.iml

New Folder Cancel Open

2-运行第一个demo之android

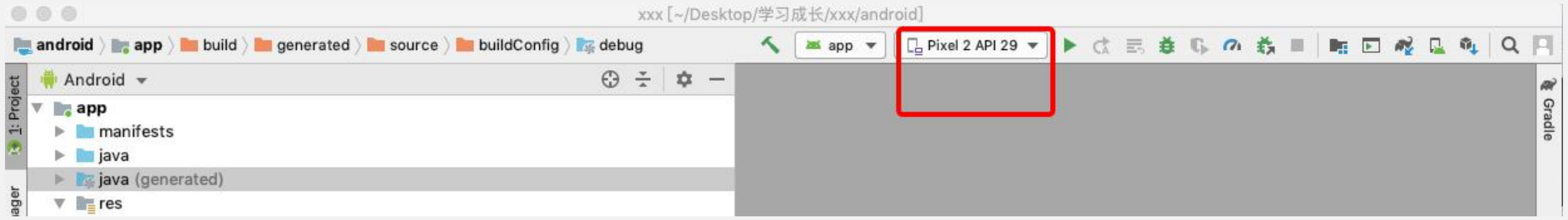
步骤3 安装依赖包



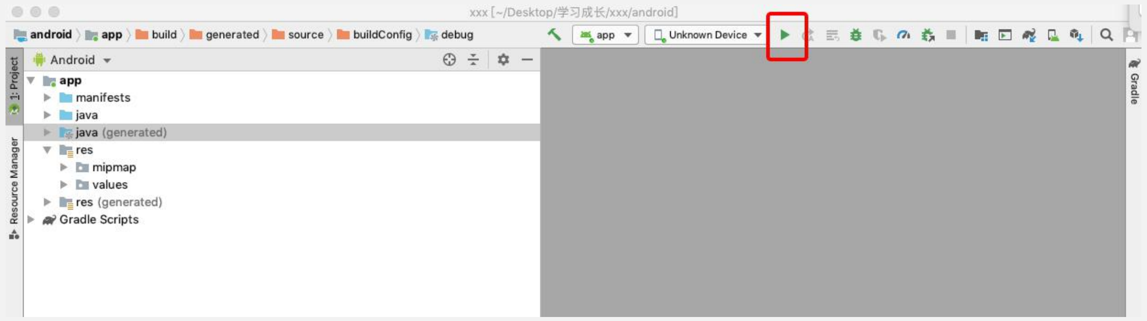
```
xxx: synced successfully at 2019-12-25 18:14          4 s 678 ms
  Run build /Users/admin/Desktop/学习成长/xxx/android    2 s 751 ms
    Load build                                          755 ms
    Configure build                                   1 s 787 ms
    Calculate task graph                                 20 ms
    Run tasks                                            71 ms
```

2-运行第一个demo之android

步骤4 选择模拟器或者真机

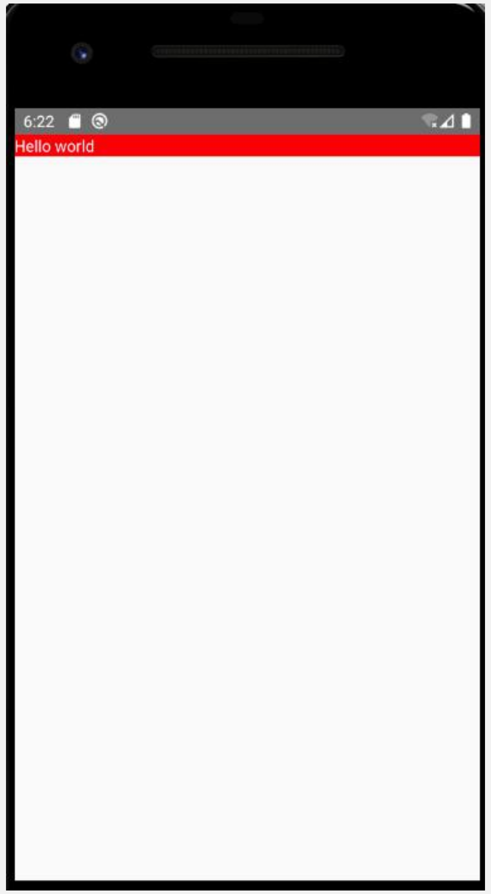


步骤5 点击运行按钮

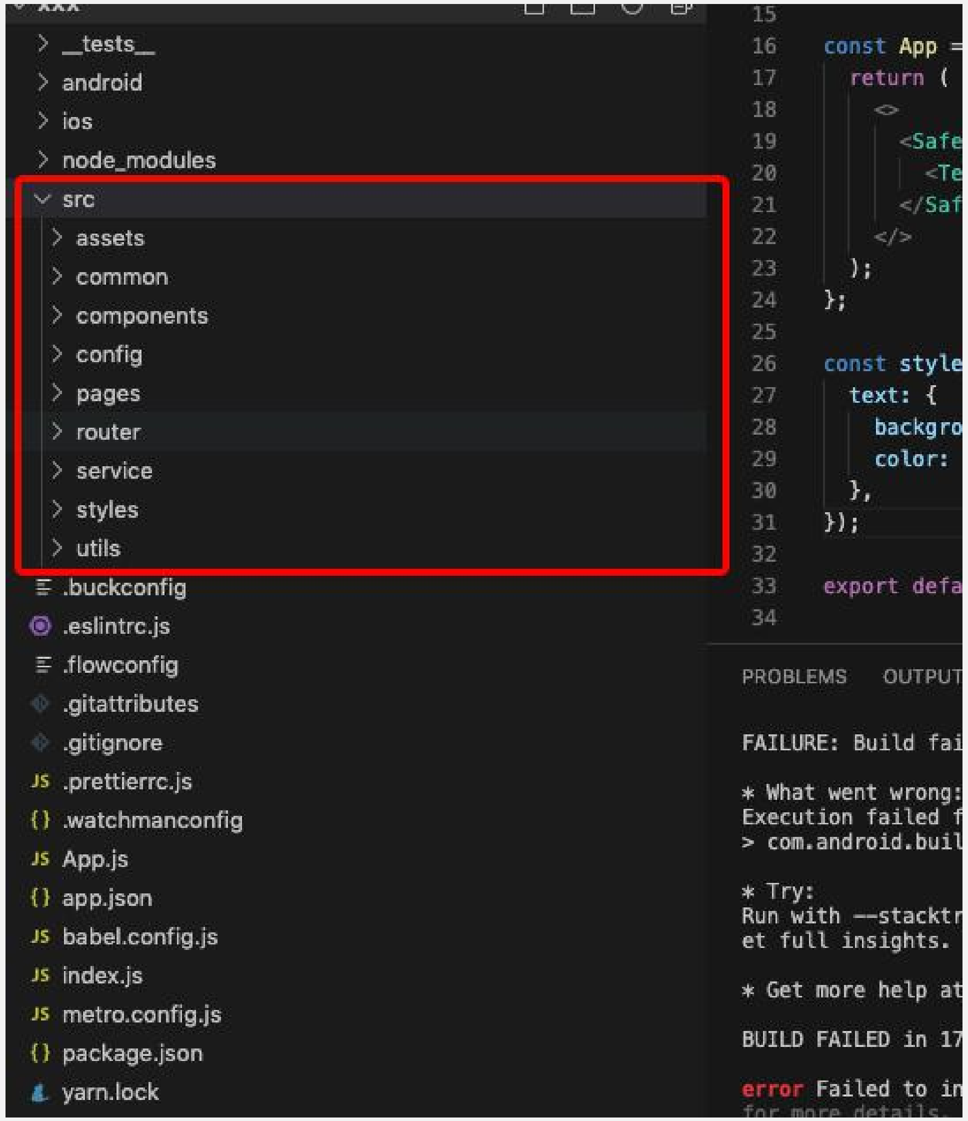


2-运行第一个demo之android

运行成功的效果图如下



项目结构设计



```
xxx
> __tests__
> android
> ios
> node_modules
v src
  > assets
  > common
  > components
  > config
  > pages
  > router
  > service
  > styles
  > utils
.buckconfig
.eslintrc.js
.flowconfig
.gitattributes
.gitignore
.prettierrc.js
.watchmanconfig
App.js
app.json
babel.config.js
index.js
metro.config.js
package.json
yarn.lock
```

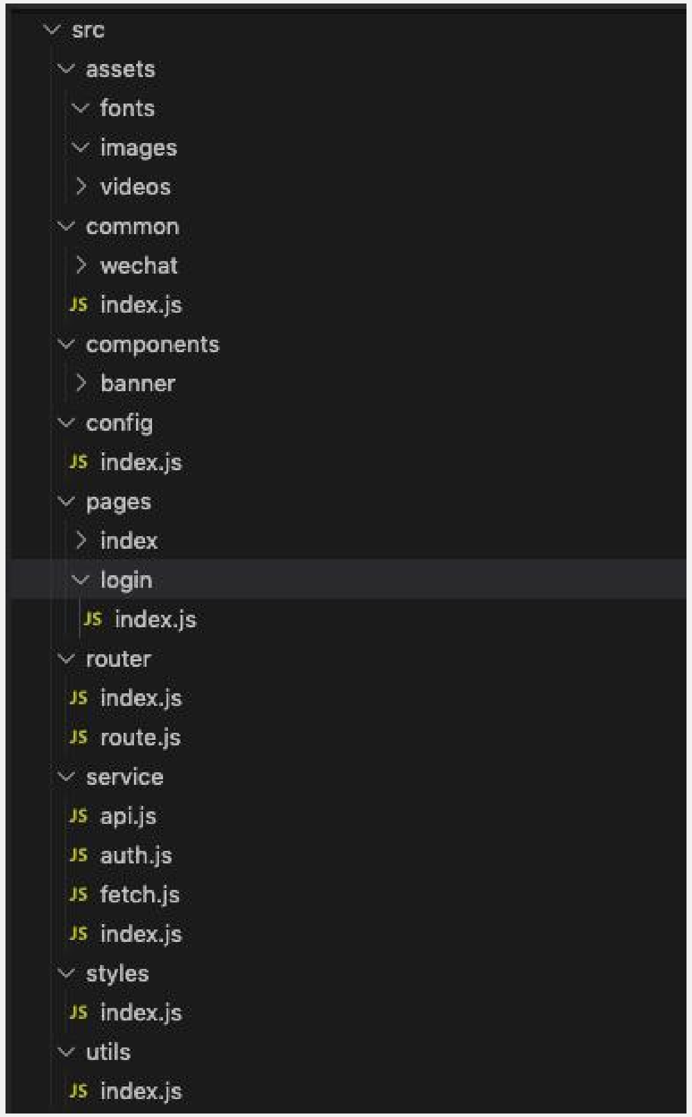
```
const App =
  return (
    <>
      <Safe
        <Te
      </Saf
    </>
  );
};

const style
  text: {
    backgro
    color:
  },
});

export defa
```

PROBLEMS OUTPUT

```
FAILURE: Build fai

* What went wrong:
Execution failed f
> com.android.buil

* Try:
Run with --stacktr
et full insights.

* Get more help at

BUILD FAILED in 17

error Failed to in
for more details.
```

assets-存放资源文件（图片,字体,音视频等）
common - 存放公共文件(wxshare.js)
components - 公共组件
config - 配置(请求主机地址,版本号等)
pages- app页面
router-路由地址
service 请求接口 (封装请求方法 处理请求异常)
styles 公共的样式 如字体颜色 字体大小等
utils 工具文件集合

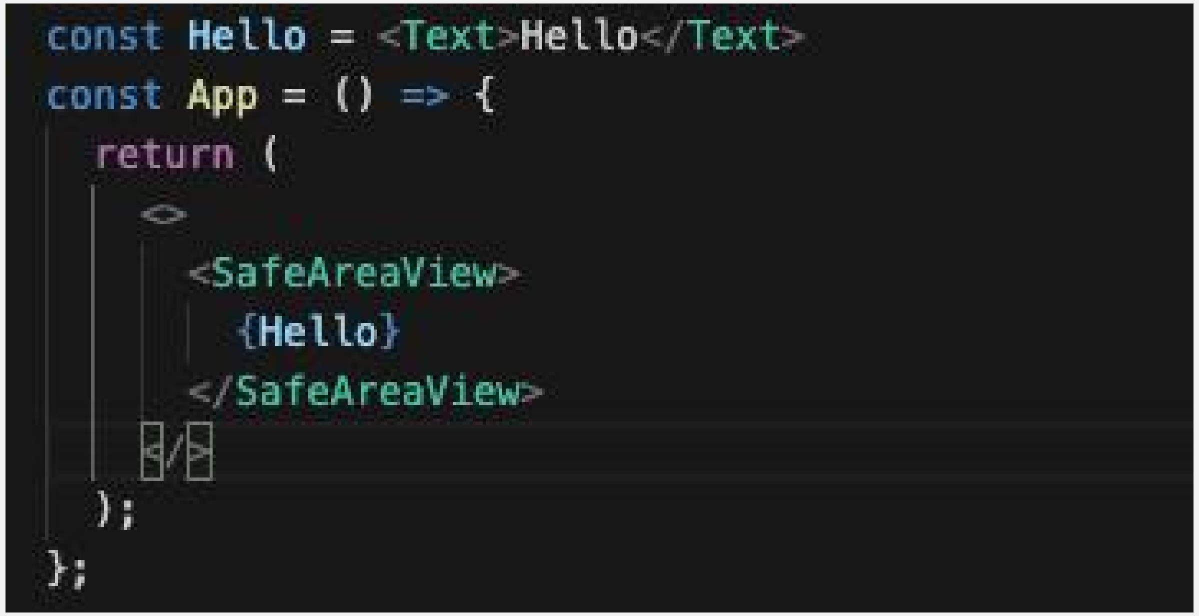
项目结构设计



```
src
  assets
    fonts
    images
    videos
  common
    wechat
    index.js
  components
    banner
  config
    index.js
  pages
    index
    login
      index.js
  router
    index.js
    route.js
  service
    api.js
    auth.js
    fetch.js
    index.js
  styles
    index.js
  utils
    index.js
```

JSX语法

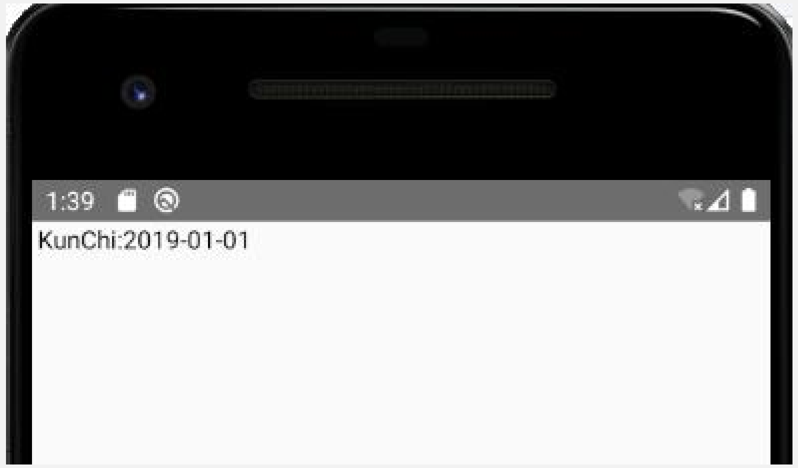
1.State状态机
2.事件处理
3.React 条件渲染
4.React 列表 & Keys
5.React 组件 API
6.React 组件生命周期
7.React Refs
8.React组件件传值
9.React几种常用组件
10.组件设置默认值



```
const Hello = <Text>Hello</Text>
const App = () => {
  return (
    <>
      <SafeAreaView>
        {Hello}
      </SafeAreaView>
    </>
  );
};
```

JSX语法- State

React 把组件看成是一个状态机（State Machines）。通过与用户的交互，实现不同状态，然后渲染UI，让用户界面和数据保持一致。React 里，只需更新组件的 state，然后根据新的 state 重新渲染用户界面（不要操作 DOM）


1:39

KunChi:2019-01-01

```
import React, { Component } from 'react'
import { Text, View } from 'react-native'
export class Index extends Component {
    state = {
        name: 'KunChi',
        date: '2019-01-01'
    }

    render() {
        return (
            <View>
                <Text> {this.state.name}:{this.state.date} </Text>
            </View>
        )
    }
}

export default Index
```

JSX语法- State

```
import React, { Component } from 'react'
import { Text, View } from 'react-native'
export class Index extends Component {
    state = {
        name: 'KunChi',
        date: '2019-01-01'
    }
    componentDidMount() {
        this.state.date = new Date().toLocaleString()
    }
    render() {
        return (
            <View>
                <Text> {this.state.name}:{this.state.date} </Text>
            </View>
        )
    }
}

export default Index
```

UI同步更新数据需要使用setState函数

```
this.setState({
      date: new Date().toLocaleString()
})
```

```
// 强制让组件刷新
 this.forceUpdate()
```

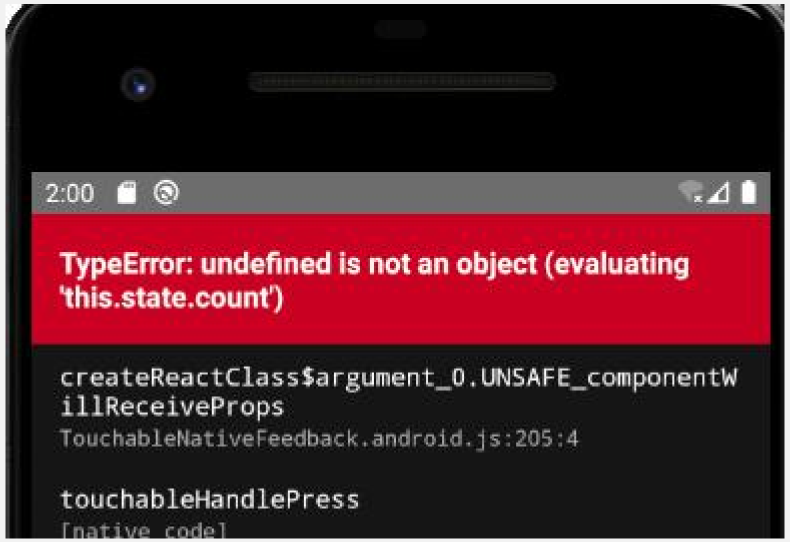
扩展

```
this.setState((prevState, props) => ({
    //do something here
}));
```

prevState 表示上一个状态值
props 表示当前props的值

JSX语法-事件处理

```
import React, { Component } from 'react'
import { Text, View, Button } from 'react-native'
export default  class Index extends Component {
    state = {
        count: 1
    }
    render() {
        return (
            <View>
                <Text> {this.state.count}</Text>
                <Button title="加1" onPress={this.addOne}></Button>
            </View>
        )
    }
    addOne() {
        this.setState({
            count: ++this.state.count
        })
    }
}
```



2:00

TypeError: undefined is not an object (evaluating 'this.state.count')

```
createReactClass$argument_0.UNSAFE_componentW
illReceiveProps
TouchableNativeFeedback.android.js:205:4

touchableHandlePress
[native code]
```

思考:addOne中的this指的是那个对象？

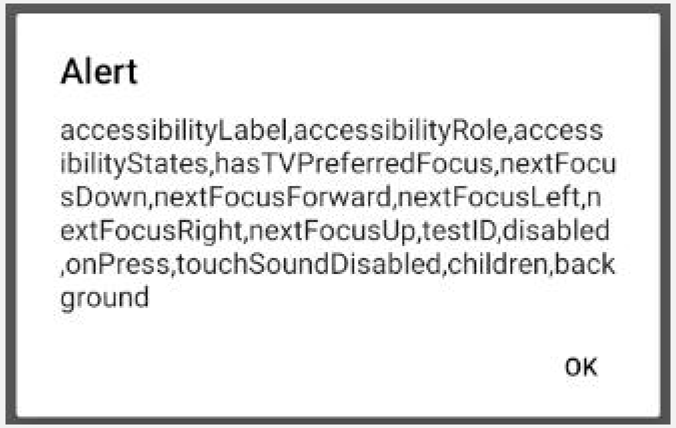
JSX语法-事件处理

下面是<Button /> 组件的定义

```
export interface ButtonProps {
    title: string;
    onPress: (ev: NativeSyntheticEvent<NativeTouchEvent>) => void;
    color?: string;
    accessibilityLabel?: string;
    disabled?: boolean;

    /**
     * Used to locate this button in end-to-end tests.
     */
    testID?: string;
}

export class Button extends React.Component<ButtonProps> {}
```

JSX语法-事件处理

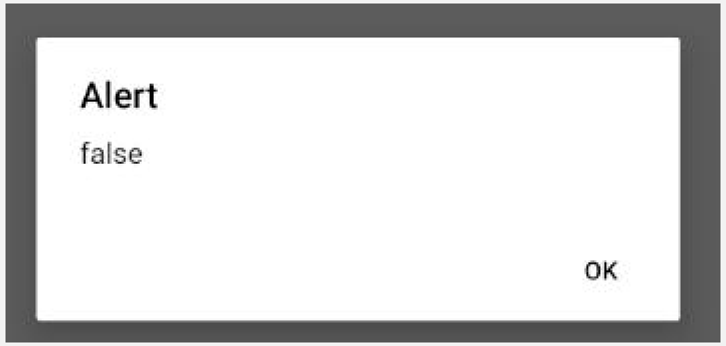
```
import React, { Component } from 'react'
import { Text, View, Button } from 'react-native'
export defaultclass Index extends Component {
    state = {
        count: 1
    }
    render() {
        return (
            <View>
                <Text> {this.state.count}</Text>
                <Button title="加1" onPress={this.addOne}></Button>
            </View>
        )
    }
    addOne() {
        alert(Object.keys(this))
    }
}
```


Alert

accessibilityLabel,accessibilityRole,accessibilityStates,hasTVPreferredFocus,nextFocusDown,nextFocusForward,nextFocusLeft,nextFocusRight,nextFocusUp,testID,disabled,onPress,touchSoundDisabled,children,background

JSX语法-事件处理

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

export default class Index extends Component {
    state = {
        count: 1
    }
    render() {
        return (
            <View>
                <Text> {this.state.count}</Text>
                <Button ref='btn' title="加1"
onPress={this.addOne}></Button>
            </View>
        )
    }
    addOne() {
        alert(this instanceof Index)
    }
}
```


Alert

false

通过instanceof 检测对象类型

JSX语法-事件处理

正确写法1

```
export default class Index extends Component {
    state = {
        count: 1
    }
    render() {
        return (
            <View>
                <Text> {this.state.count}</Text>
                <Button ref='btn' title="加1"
onPress={this.addOne.bind(this)}></Button>
            </View>
        )
    }
    addOne() {
        alert(this instanceof Index)
    }
}
```

JSX语法-事件处理

正确写法 2

Button组件调用方式类似如下
this.addOne.bind(Index组件)()

```
export default class Index extends Component {
    state = {
        count: 1
    }

    constructor() {
        super(...arguments)
        this.addOne = this.addOne.bind(this)
    }

    render() {
        return (
            <View>
                <Text> {this.state.count}</Text>
                <Button ref='btn' title="加1"
onPress={this.addOne}></Button>
            </View>
        )
    }
}
```

JSX语法-事件处理

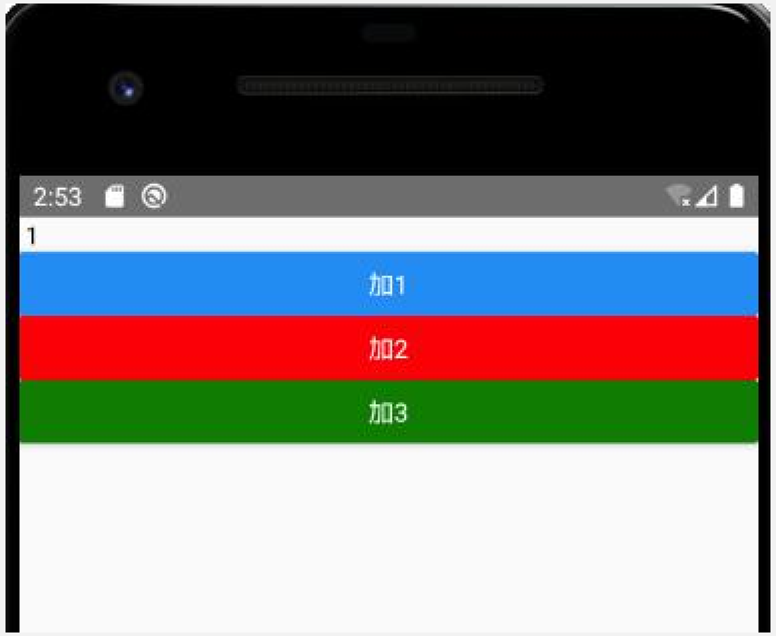
正确写法3

使用ES6语法 避免出现作用域不明确的问题

```
export default class Index extends Component {
    state = {
        count: 1
    }
    constructor() {
        super(...arguments)
    }
    render() {
        return (
            <View>
                <Text> {this.state.count}</Text>
                <Button ref='btn' title="加1"
onPress={this.addOne}></Button>
            </View>
        )
    }
    addOne = () => {
        alert(this instanceof Index)
    }
}
```

JSX语法-事件处理

思考？怎么实现下面的需求



JSX语法-事件处理

```
 <Button ref='btn' title="加1" onPress={this.addOne}></Button>
 <Button ref='btn' title="加2" onPress={this.addTwo} color="red"></Button>

addOne = () => {
    let { count } = this.state
    this.setState({
        count: count + 1
    })
}
addTwo = () => {
    let { count } = this.state
    this.setState({
        count: count + 2
    })
}
```

JSX语法-事件处理

闭包函数实现数据传递

```
<Button ref='btn' title="加1" onPress={() => {
                this.add(1)
}}></Button>
 <Button ref='btn' title="加2" onPress={() => {
                this.add(2)
}} color="red"></Button>
<Button ref='btn' title="加3" onPress={() => {
                this.add(3)
}} color="green"></Button>
add = (num) => {
    let { count } = this.state
    this.setState({
        count: count + num
    })
}
```

JSX语法-React 条件渲染

A,B用户看到不同的内容？

JSX语法-React 条件渲染

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

export default class Index extends Component {
    state = {
        username: "B"
    }
    render() {
        const A = <Text>A用户看到的内容</Text>
        const B = <Text>B用户看到的内容</Text>
        return (
            <View>
                {this.state.username === 'A' ? A : B}
            </View>
        )
    }
}
```

如果是C用户怎么显示？

JSX语法-React 条件渲染

注意:ShowUserInfo首字母必须大写 如果要使用标签写法的话<ShowUserInfo username="A" />

```
import React, { Component } from 'react'
import { Text, View, Button } from 'react-native'
const ShowUserInfo = ()=>{
    const A = <Text>A用户看到的内容</Text>
    const B = <Text>B用户看到的内容</Text>
    if (props.username == 'A') {
        return A
    }
    if (props.username == 'B') {
        return B
    }
    return null
}
```

```
export default class Index extends Component {
    state = {
        username: "B"
    }
    render() {
        return (
            <View>
                {ShowUserInfo({ username: 'A' })}
                <ShowUserInfo username="A" />
            </View>
        )
    }
}
```

JSX语法-React 条件渲染

```
const ShowUserInfo = (props) => {
    const A = <Text>A用户看到的内容</Text>
    const B = <TouchableOpacity
onPress={props.onPress}><Text>B用户看到的内容
</Text></TouchableOpacity>
    if (props.username == 'A') {
        return A
    }
    if (props.username == 'B') {
        return B
    }
    return null
}
```

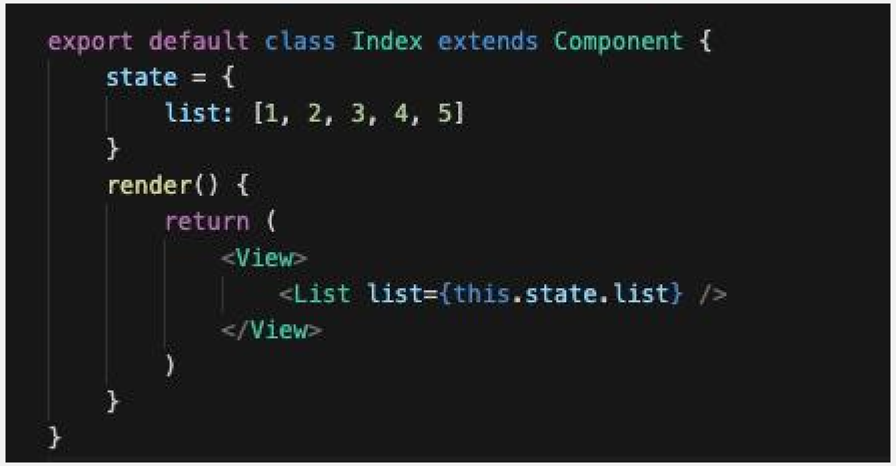
```
<ShowUserInfo username="B"
onPress={this.greet} />
```

如果条件不满足,则返回null即可

JSX语法-React 列表 & Keys

如何将一组数据渲染到页面上?

```
const List = (props) => {
    return props.list.map(item => {
        return <Text>{item}</Text>
    })
}


// 简写
const List = (props) => {
    return props.list.map(item => (<Text>{item}</Text>))
}
```



```
export default class Index extends Component {
    state = {
        list: [1, 2, 3, 4, 5]
    }
    render() {
        return (
            <View>
                <List list={this.state.list} />
            </View>
        )
    }
}
```

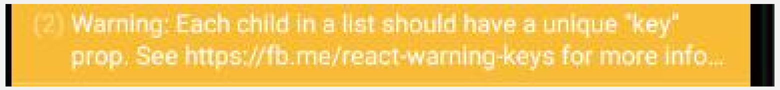
JSX语法-React 列表 & Keys

为什么要为列表中每一个元素指定key值?

- Keys 可以在 DOM 中的某些元素被增加或删除的时候帮助 React 识别哪些元素发生了变化。因此应当给数组中的每一个元素赋予一个确定的标识。

- 一个元素的 key 最好是这个元素在列表中拥有的一个独一无二的字符串。通常，我们使用来自数据的 id 作为元素的 key.

- 当元素没有确定的 id 时，可以使用他的序列号索引 index 作为 key.
如果列表可以重新排序，不建议使用索引来进行排序，因为这会导致渲染变得很慢



```
(2) Warning: Each child in a list should have a unique "key"
prop. See https://fb.me/react-warning-keys for more info...
```

JSX语法-React 列表 & Keys

请注意Key添加的位置

```
const List = (props) => {
    return props.list.map((item, index) => (<Item key={item.name}
title={item.name}></Item>))
}

const Item = (props) => {
    return <Text>{props.name}</Text>
}
```

JSX语法-React 组件 API

设置状态：setState
强制更新：forceUpdate

JSX语法-React 组件 API

设置状态：setState

setState(object nextState[, function callback])

callback 回调函数

setState()并不会立即改变this.state，而是创建一个即将处理的state。setState()并不一定是同步的，为了提升性能React会批量执行state和DOM渲染

如果想要立即更新如何处理?

执行 this.forceUpdate()

JSX语法-React 组件 API

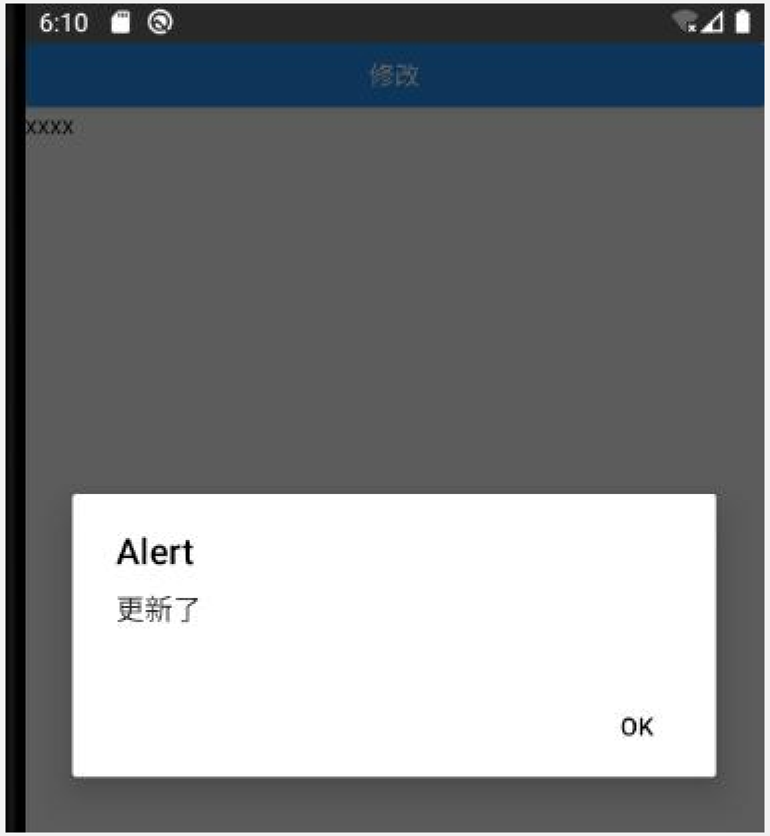
思考下面的代码页面会不会更新数据为aaaa?

```
this.state.username = 'aaaa'
    this.setState({
})
```

JSX语法-React 组件 API

setState()总是会触发一次组件重绘，除非在shouldComponentUpdate()中实现了一些条件渲染逻辑。

```
export default class Index extends Component {
    state = {
        username: 'xxxx'
    }
    componentDidUpdate() {
        alert("更新了")
    }
    render() {
        return (
            <View>
                <Button title="修改" onPress={() => {
                    this.setState({
                    })
                }}></Button>
                <Text>{this.state.username}</Text>
            </View>
        )
    }
}
```



JSX语法-React 组件 API

如何避免数据没有更新，页面刷新？

方法1 如图

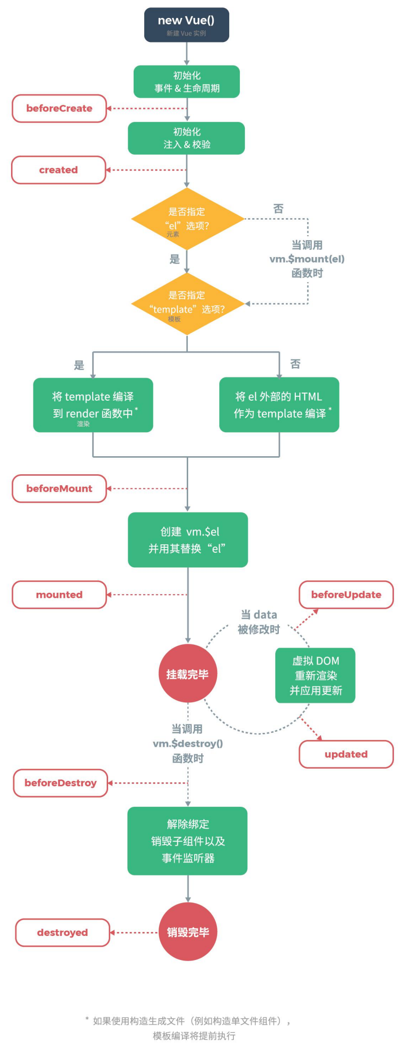
```
export default class Index extends PureComponent {
    state = {
        username: 'xxxx'
    }
    componentDidUpdate() {
        alert("更新了")
    }
    render() {
        return (
            <View>
                <Button title="修改" onPress={() => {
                    this.setState({
                    })
                }}></Button>
                <Text>{this.state.username}</Text>
            </View>
        )
    }
}
```

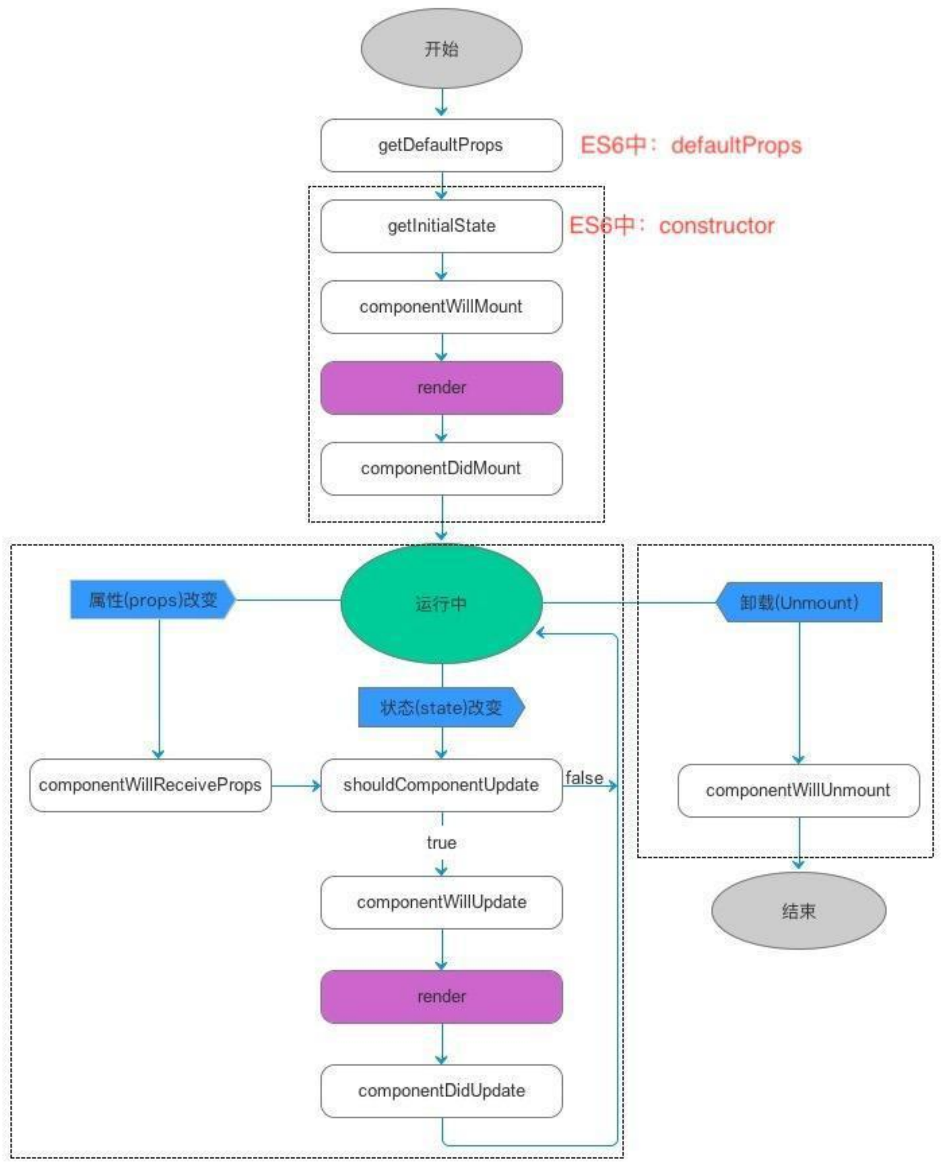
JSX语法-React 组件 API

方法2 重写 shouldComponentUpdate 方法

```
shouldComponentUpdate(nextProps, nextState) {
    if (nextState.username && this.state.username != nextState.username) {
        return true
    }
    return false
}
```

JSX语法-React 组件生命周期





JSX语法-React 组件生命周期

componentWillMount 在渲染前调用,在客户端也在服务端。

componentDidMount : 在第一次渲染后调用，只在客户端。之后组件已经生成了对应的DOM结构，可以通过this.getDOMNode()来进行访问。 如果你想和其他JavaScript框架一起使用，可以在这个方法中调用setTimeout, setInterval或者发送AJAX请求等操作(防止异步操作阻塞UI)。

componentWillReceiveProps 在组件接收到一个新的 prop (更新后)时被调用。这个方法在初始化render时不会被调用。

shouldComponentUpdate 返回一个布尔值。在组件接收到新的props或者state时被调用。在初始化时或者使用forceUpdate时不被调用。
可以在你确认不需要更新组件时使用。

componentWillUpdate在组件接收到新的props或者state但还没有render时被调用。在初始化时不会被调用。

componentDidUpdate 在组件完成更新后立即调用。在初始化时不会被调用。

componentWillUnmount在组件从 DOM 中移除之前立刻被调用。

JSX语法-React 组件生命周期

1.思考下面代码页面输出是什么？

```
import React, { Component, PureComponent } from 'react'
import { Text, View, Button, TouchableOpacity } from 'react-native'
export default class Index extends Component {
    state = {
        username: ''
    }
    constructor() {
        super(...arguments)
        this.state.username = 'A'
    }

    render() {
        return (
            <View>
                <Text>{this.state.username}</Text>
            </View>
        )
    }
}
```

JSX语法-React 组件生命周期

2.思考下面代码页面输出的结果是什么?

```
import React, { Component, PureComponent } from 'react'
import { Text, View, Button, TouchableOpacity } from 'react-native'
export default class Index extends Component {
    state = {
        username: ''
    }
    constructor() {
        super(...arguments)
        this.state.username = 'A'
    }
    componentWillMount() {
        this.state.username = 'B'
    }
    render() {
        return (
            <View>
                <Text>{this.state.username}</Text>
            </View>
        )
    }
}
```

JSX语法-React 组件生命周期

3.思考下面代码页面输出的是什么?

```
import React, { Component, PureComponent } from 'react'
import { Text, View, Button, TouchableOpacity } from 'react-native'
export default class Index extends Component {
    state = {
        username: ''
    }
    constructor() {
        super(...arguments)
        this.state.username = 'A'
    }
    componentDidMount() {
        this.state.username = 'C'
    }
    render() {
        return (
            <View>
                <Text>{this.state.username}</Text>
            </View>
        )
    }
}
```

JSX语法-React Refs

React 支持一种非常特殊的属性 Ref ，你可以用来绑定到 render() 输出的任何组件上

有什么用?

可以获取组件实例对象,及其对应的属性

JSX语法-React Refs

```
<TextInput ref="text"></TextInput>
```

```
let keys = Object.keys(this.refs.text)
console.log(JSON.stringify(keys))
```

```
["measure","measureInWindow","measureLayout","setNativeProps","foc
us","blur","isFocused","clear","_getText","_setNativeRef","_renderIOSLe
gacy","_renderIOS","_renderAndroid","_onFocus","_onPress","_onChan
ge","_onSelectionChange","_onBlur","_onTextInput","_onScroll","props","
context","refs","updater","state","_reactInternalFiber","_reactInternalInsta
nce","_inputRef","__isMounted","_lastNativeText"]
```

JSX语法-React Refs

如何获取元素的位置信息和宽高信息？

```
import React, { Component, PureComponent } from 'react'
import { Text, View, TextInput, UIManager, findNodeHandle } from 'react-native'
export default class Index extends Component {
    componentDidMount() {
        // 直接在componentDidMount 需要在下一次检测循环中获取
        setTimeout(() => {
            // 方法1
            const handle = findNodeHandle(this.refs.text)
            UIManager.measure(handle, (x, y, width, height, pageX, pageY) => {
                console.log(width)
                console.log(height)
            })
        }, 0);
    }
    render() {
        return (
            <View>
                <TextInput ref="text"></TextInput>
            </View>
        )
    }
}
```

JSX语法-React Refs

```
import React, { Component, PureComponent } from 'react'
import { Text, View, TextInput } from 'react-native'


export default class Index extends Component {

    componentDidMount() {
        // 直接在componentDidMount 需要在下一次检测循环中获取
        setTimeout(() => {
            // 方法2
            this.refs.text.measure((x, y, width, height, pageX, pageY) => {
                console.log(width)
                console.log(height)
            })
        }, 0);
    }


    render() {
        return (
            <View>
                <TextInput ref="text"></TextInput>
            </View>
        )
    }
}
```

通过this.refs.text.measure方法直接主动获取布局信息

JSX语法-React Refs

```
  render() {
      return (
          <View>
              <TextInput ref="text"
onLayout={this.getTextInputLayout}></TextInput>
          </View>
      )
   }
   getTextInputLayout = (e) => {
      console.log(e.nativeEvent.layout)
   }
```

布局完成回调函数中获取布局信息

JSX语法- React组件件传值

A.父组件向子组件传值

B.子组件向父组件传值

JSX语法- 组件间传值

问题1.父组件需要向A组件传递什么值？

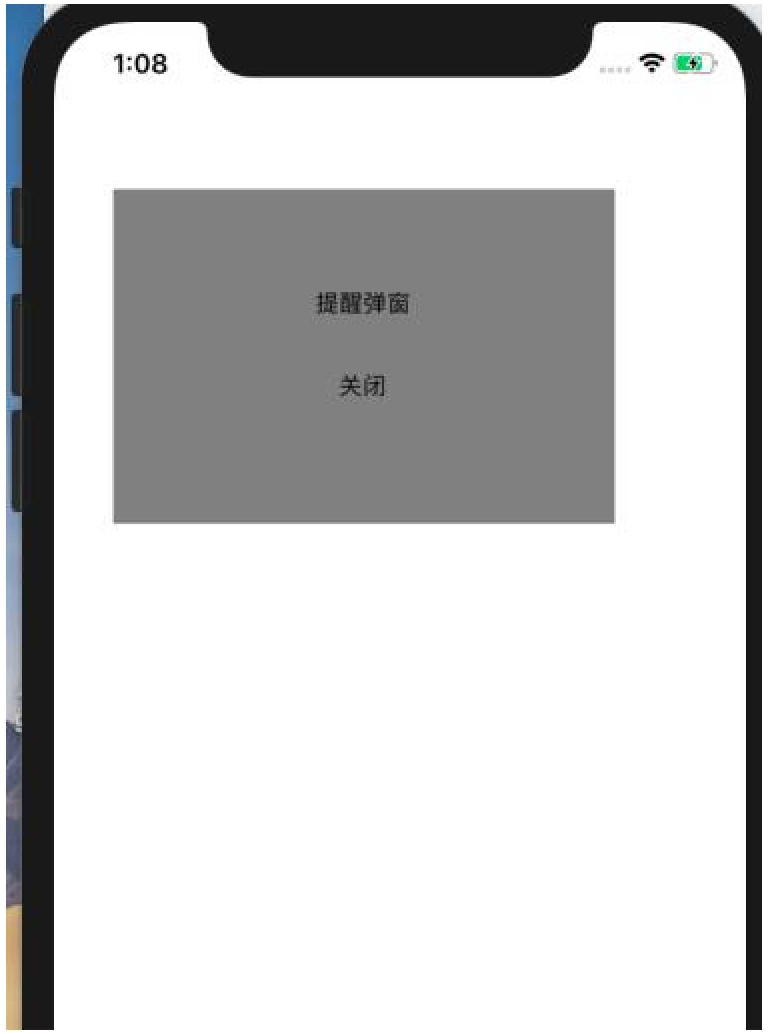
JSX语法- 组件间传值

A.数据 - 需要将父组件的数据传递给子组件(渲染数据,网络请求参数,逻辑计算等)

B.函数 - (组件间通讯,传递执行逻辑)

C.组件 - (让子组件将传递的子组件作为其子组件渲染)

组件间传值

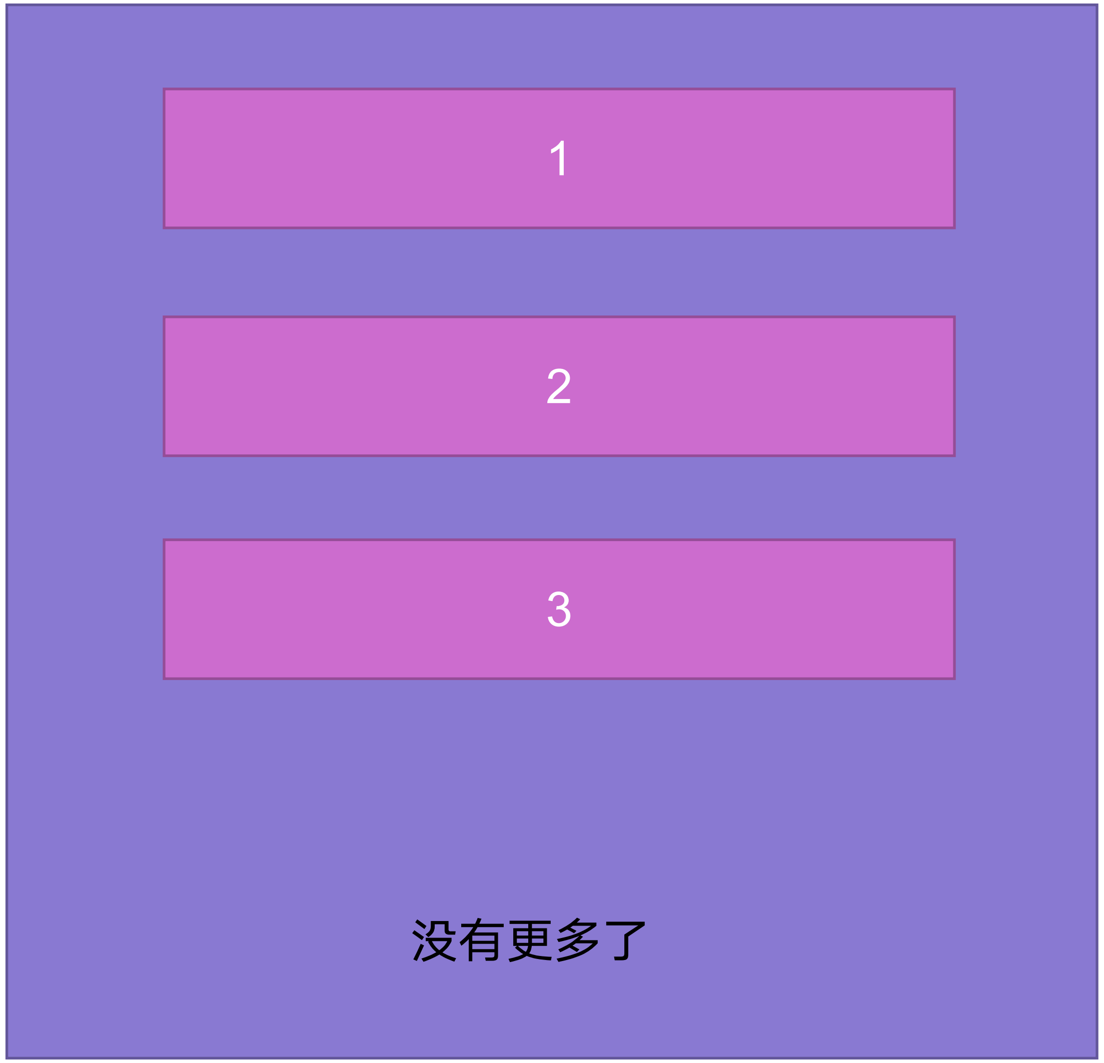


JSX语法- 组件间传值

```
class Alert extends Component{
    render(){
        return <Modal
visible={this.props.visible}>
               <View>
               <Text>{this.props.title}</Text>
              <TouchableHighlight
onPress={this.props.onClose}>
                  <Text>关闭</Text>
              </TouchableHighlight>
               </View>
        </Modal>
    }
}
```

```
class index extends Component {
 state = {
    showAlert:true
 }
  render() {
    return (
      <SafeAreaView>
        <Alert visible={this.state.showAlert}
title="提醒弹窗" onClose={this.onClose}/>
      </SafeAreaView>
    );
  }
  onClose=()=>{
      this.setState({
          showAlert:false
      })
  }
}
```

JSX语法- 组件间传值



JSX语法- 组件间传值

父组件

```
 <SafeAreaView>
       <List footerComponent={<Text>{this.state.noMore?'没有更多了':'加载更多...'}</Text>}/>
     </SafeAreaView>
```

子组件

```
class List extends Component{
   render(){
       return <View style={{alignItems:'center'}}>
           {/* 省略渲染列表代码.... */}
           {this.props.footerComponent}
       </View>
   }
}
```

JSX语法-常用的几种组件

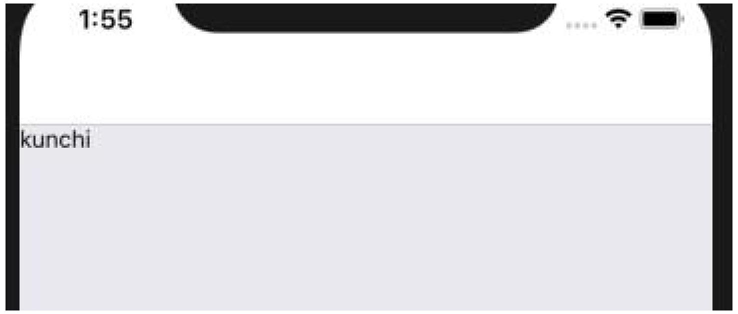
1.继承自Component组件
2.没有继承关系的组件
3.高阶组件
4.子组件是函数的组件

JSX语法-常用的几种组件

1.有继承关系的组件

```
class UserInfo extends Component{
    render(){
        return <View>
           <Text>{this.props.username}</Text>
        </View>
    }
}
```

```
<UserInfo username={'kunchi'} />
```



kunchi

JSX语法-常用的几种组件

2.没有继承关系的组件

```
//1
function UserInfo(props) {
    return(<View>
        <Text>{props.username}</Text>
     </View>)
 }

// 2
 const UserInfo = (props)=>{
    return(<View>
        <Text>{props.username}</Text>
    </View>)
 }
```

```
<UserInfo username={'kunchi'} />
```

注意首字母必须大小

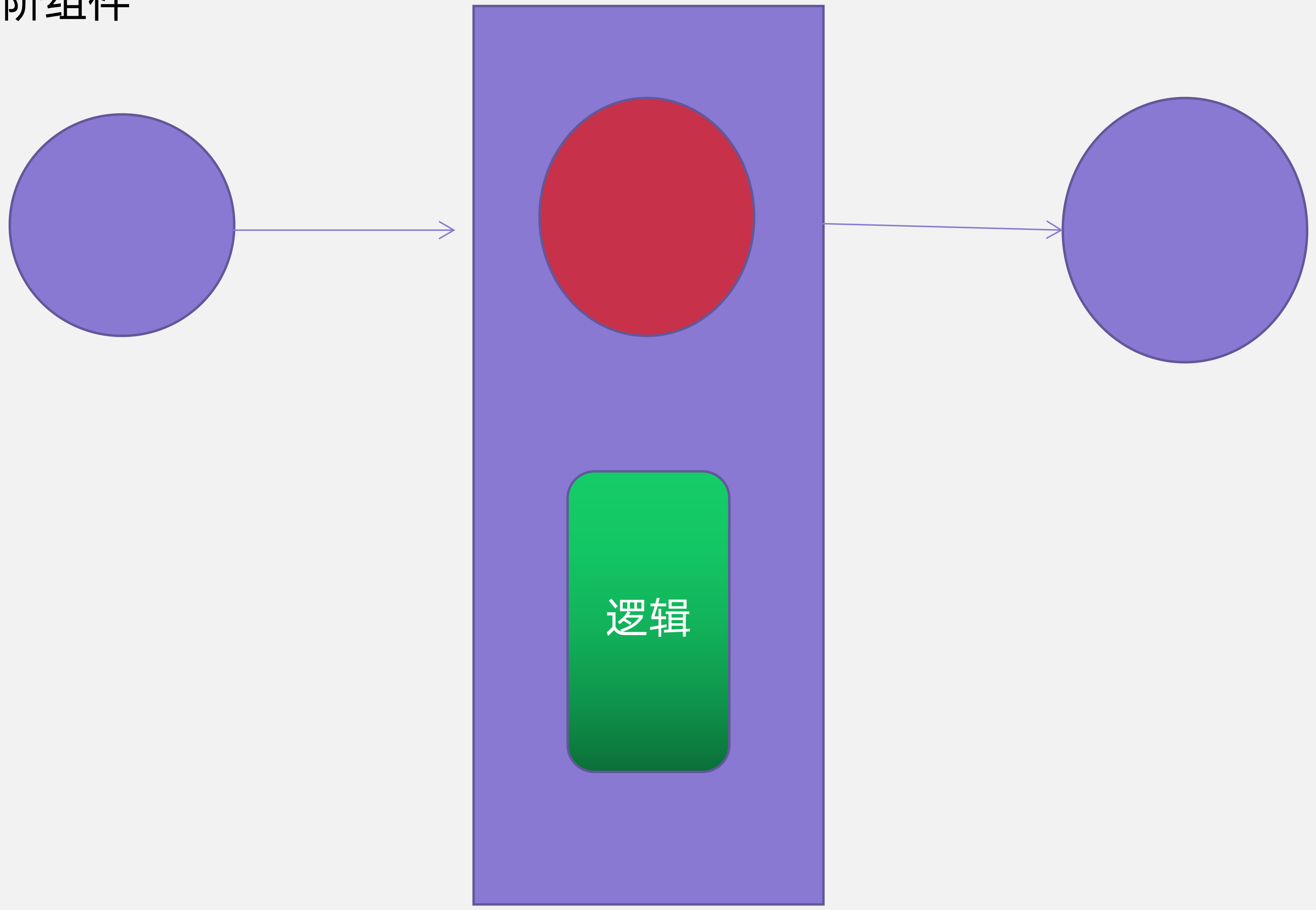
JSX语法-常用的几种组件

高阶组件

抽离出逻辑代码,实现逻辑的重用,和UI布局进行隔离,少写重复代码,早点下班

JSX语法-常用的几种组件

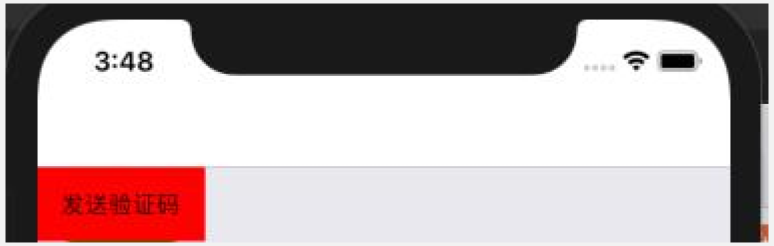
3.高阶组件

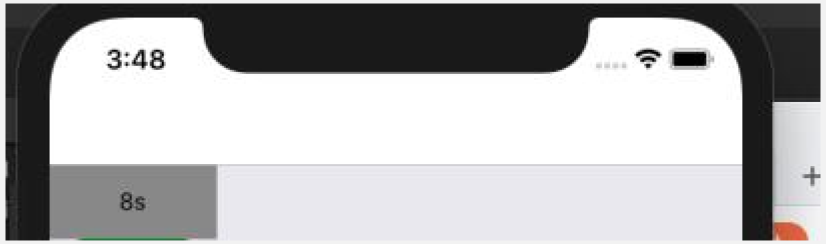


JSX语法-常用的几种组件

写一个高阶组件实现定时器逻辑

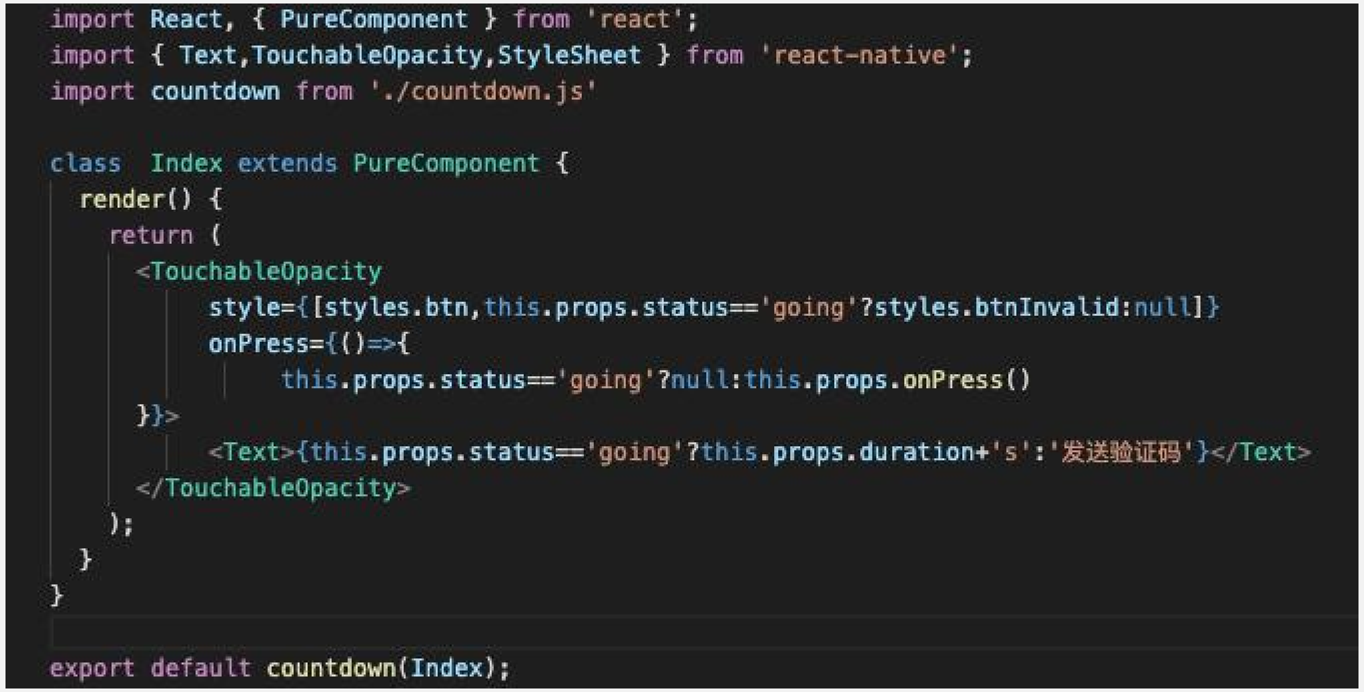
```
import CountDownBtn1 from './CountDownBtn1'

 <CountDownBtn1
        maxDuration={10}
        status={this.state.status1} onPress={()=>{
                 this.setState({
                     status1:'going'
                 })
  }}
         onChange={(status)=>{
                  this.setState({
                      status1:status
           })
    }}/>
```





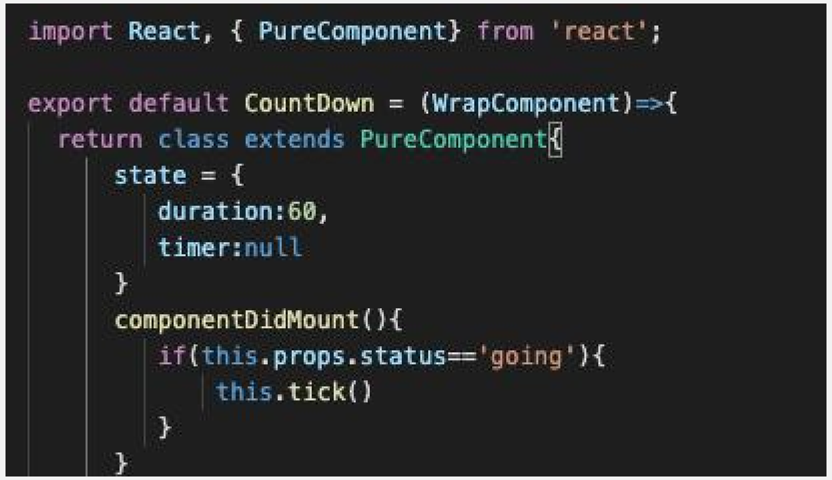
JSX语法-常用的几种组件

CountDownBtn1 代码如下

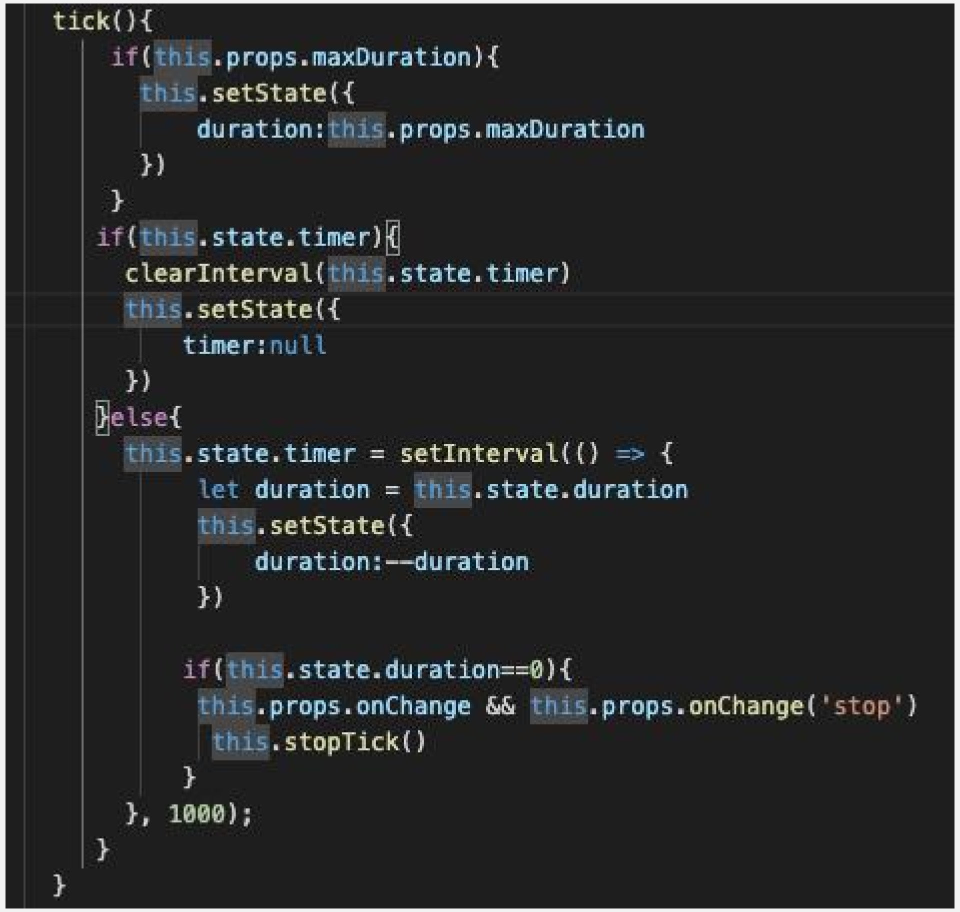


```
import React, { PureComponent } from 'react';
import { Text,TouchableOpacity,StyleSheet } from 'react-native';
import countdown from './countdown.js'

class  Index extends PureComponent {
  render() {
    return (
      <TouchableOpacity
           style={[styles.btn,this.props.status=='going'?styles.btnInvalid:null]}
           onPress={()=>{
                this.props.status=='going'?null:this.props.onPress()
      }}>
           <Text>{this.props.status=='going'?this.props.duration+'s':'发送验证码'}</Text>
      </TouchableOpacity>
    );
  }
}

export default countdown(Index);
```

JSX语法-常用的几种组件

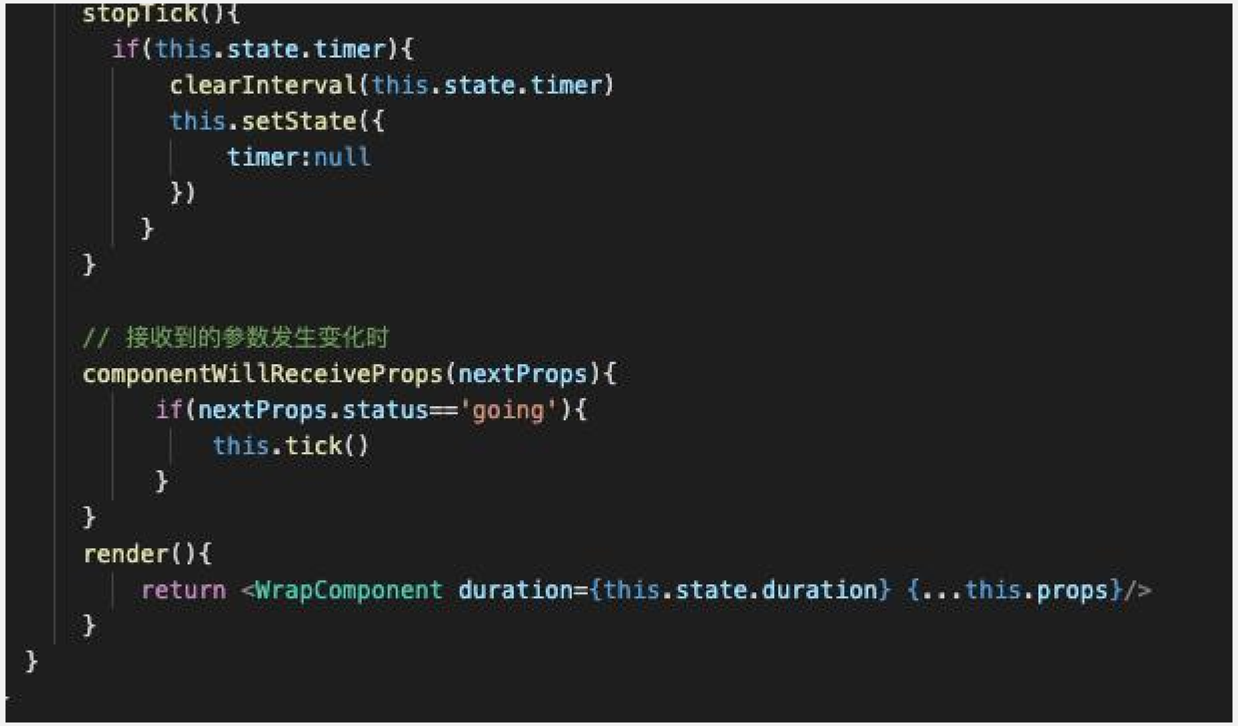


```
import React, { PureComponent} from 'react';

export default CountDown = (WrapComponent)=>{
  return class extends PureComponent{
      state = {
          duration:60,
          timer:null
      }
      componentDidMount(){
          if(this.props.status=='going'){
              this.tick()
          }
      }
```



```
tick(){
    if(this.props.maxDuration){
      this.setState({
          duration:this.props.maxDuration
      })
    }
   if(this.state.timer){
     clearInterval(this.state.timer)
     this.setState({
         timer:null
     })
   }else{
     this.state.timer = setInterval(() => {
         let duration = this.state.duration
         this.setState({
             duration:--duration
         })

         if(this.state.duration==0){
          this.props.onChange && this.props.onChange('stop')
           this.stopTick()
         }
     }, 1000);
   }
}
```

JSX语法-常用的几种组件

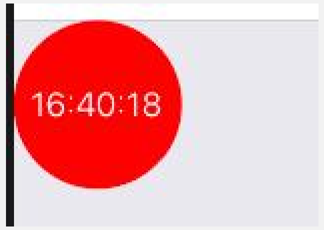


```
    stopTick(){
      if(this.state.timer){
          clearInterval(this.state.timer)
          this.setState({
              timer:null
          })
      }
    }

    // 接收到的参数发生变化时
    componentWillReceiveProps(nextProps){
        if(nextProps.status=='going'){
            this.tick()
        }
    }
    render(){
        return <WrapComponent duration={this.state.duration} {...this.props}/>
    }
}
```

JSX语法-常用的几种组件

4.子组件是函数的组件

写一个的倒计时组件



JSX语法-常用的几种组件

封装父组件实现业务逻辑

渲染部分交给子组件

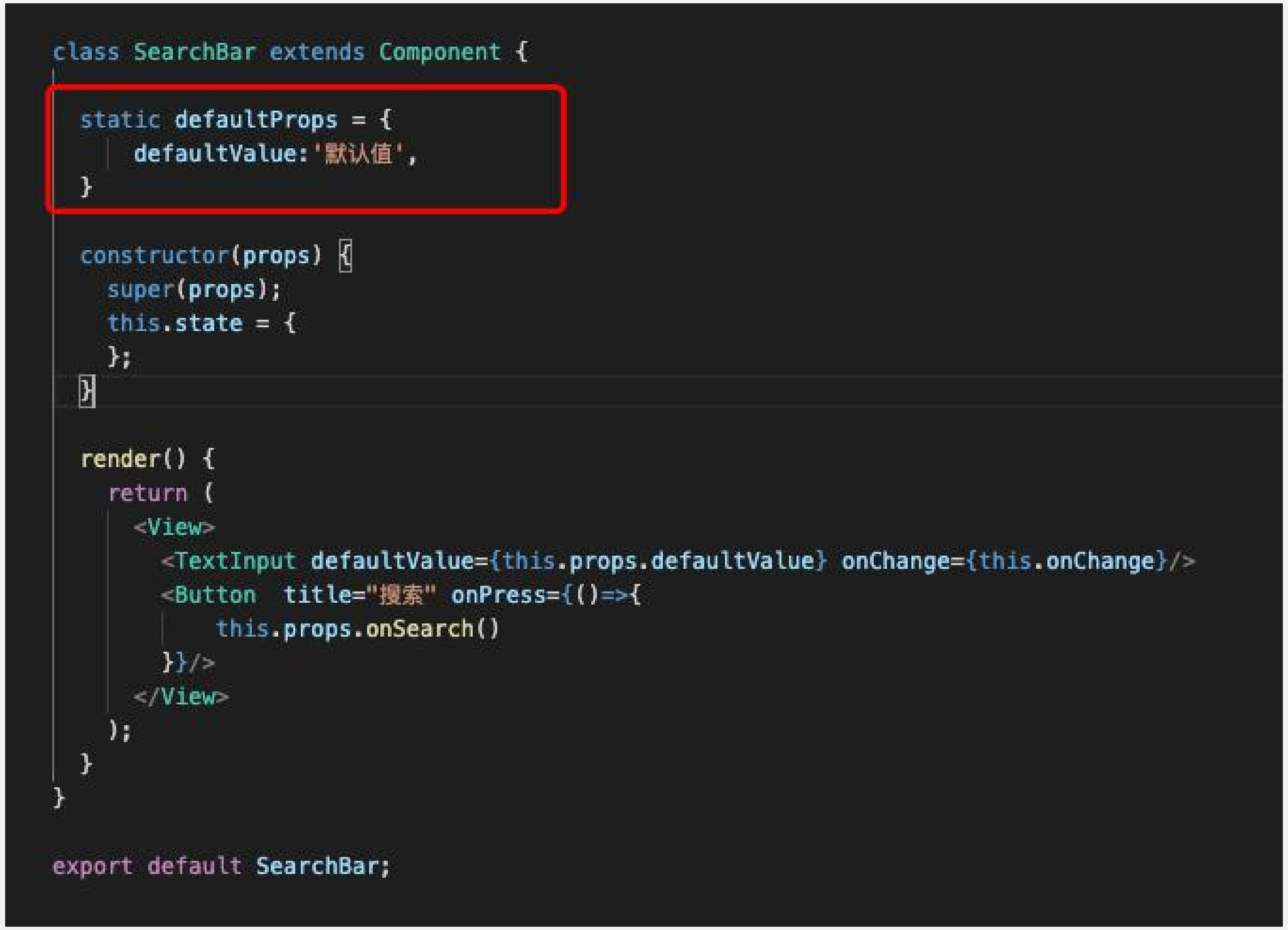
```
import React, { Component } from 'react';
class Index extends Component {
 state = {
    date: new Date(),
    timer:null
};
  componentDidMount(){
  this.tick()
  }
  tick(){
    this.state.timer = setInterval(() => {
        this.setState({
            date:new Date()
        })
    }, 1000);
  }

  render() {
    return (
      <>
      {this.props.children(this.state.date)}
      </>
    );
  }
}

export default Index;
```

JSX语法-常用的几种组件

Clock子组件是一个函数,将数据渲染到组件上

```
class Index extends Component {
    render() {
        return (
                <Clock>
                    {(date) => <View style={styles.clock}>
                        <Text
style={styles.clockText}>{this.formateTime(date)}</Text>
                    </View>}
                </Clock>

        );
    }
    formateTime(date = {}) {
        const toTwoUnit = (num) => {
            return num > 10 ? '' + num : '0' + num
        }
        let components = [date.getHours(), date.getMinutes(),
date.getSeconds()]
        components = components.map(item => toTwoUnit(item))
        return components.join(":")
    }
}
```

JSX语法 - 给组件添加默认值

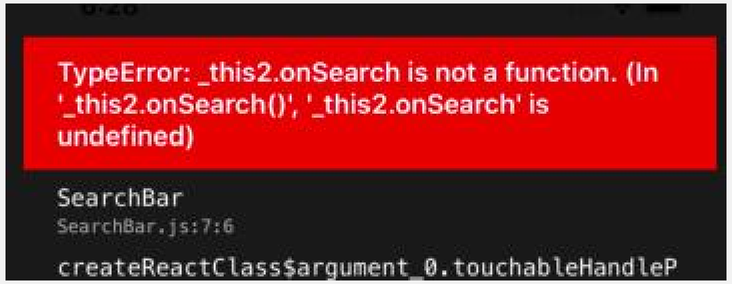


```jsx
class SearchBar extends Component {

  static defaultProps = {
      defaultValue:'默认值',
  }

  constructor(props) {
    super(props);
    this.state = {
    };
  }

  render() {
    return (
      <View>
        <TextInput defaultValue={this.props.defaultValue} onChange={this.onChange}/>
        <Button  title="搜索" onPress={()=>{
            this.props.onSearch()
        }}/>
      </View>
    );
  }
}

export default SearchBar;
```

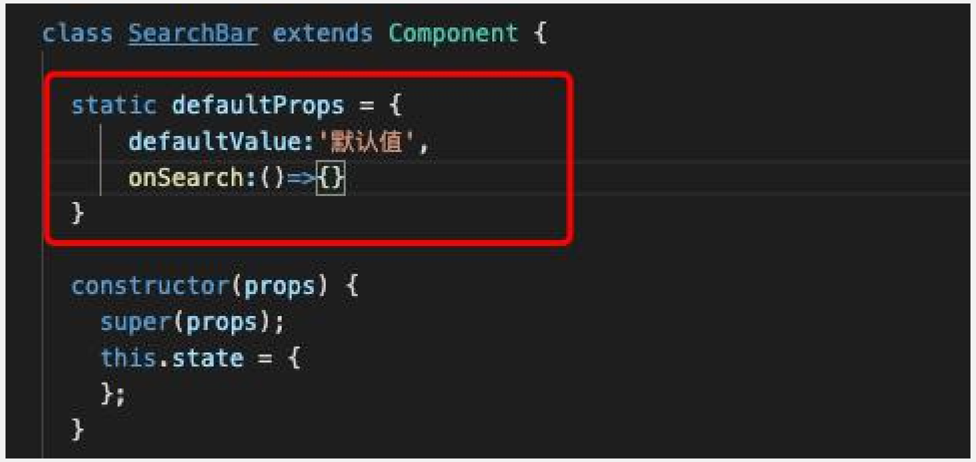
- 给组件添加默认值

```
componentDidMount(){
        alert(SearchBar.defaultProps.defaultValue)
}
```

如果不设置这个默认,点击搜索按钮会如何?



```
TypeError: _this2.onSearch is not a function. (In
'_this2.onSearch()', '_this2.onSearch' is
undefined)

SearchBar
SearchBar.js:7:6
createReactClass$argument_0.touchableHandleP
```



```
class SearchBar extends Component {

  static defaultProps = {
    defaultValue:'默认值',
    onSearch:()=>{}
  }

  constructor(props) {
    super(props);
    this.state = {
    };
  }
```

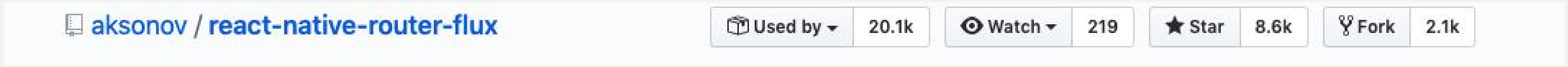
路由管理

1.为什么要进行对路由进行管理？
2.如何定义路由？
3.如何使用路由进行页面跳转和传值？
4.路由的可配置参数有哪些？

路由管理

官方推荐: react-navigation
(https://reactnavigation.org/)

社区推荐: react-native-router-flux
https://github.com/aksonov/react-native-router-flux


aksonov / react-native-router-flux

路由管理

布局形式(同时支持iOS和android)
stack(类似聊天列表进入聊天页面)
bottomTabs(底部导航页面)
sideMenu(从左右两个滑出来导航)

路由管理-安装

```
yarn add react-navigation
yarn add react-navigation-stack
yarn add react-navigation-tabs
yarn add react-native-gesture-
handler
 yarn add react-native-reanimated

npm install --save react-navigation
```

```
yarn add react-navigation-router-flux
```

路由管理-注册-stack

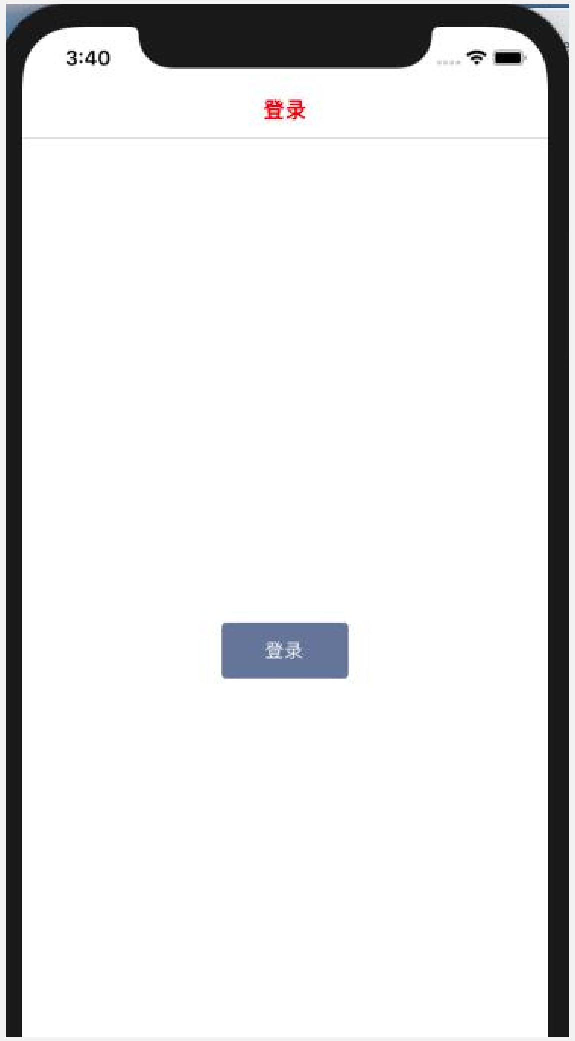
```
import routes,{Login,Me,Index} from './routes'
import {createAppContainer} from 'react-navigation';
import {createStackNavigator} from 'react-navigation-stack'
const StackNavigation = createStackNavigator({
    Login: {
        screen:Login,
        navigationOptions:{
            title:"登录",
            headerTitleStyle:{
                color:'red'
            }
        }
    },
     Index: routes.Index  //  路由配置简写
     },
     {
        initialRouteName:'Login' // 程序首次进入的页面
     }
 )
 const App = createAppContainer(StackNavigation)

export default App
```

```
import React from 'react'; //不能缺省
import routes,{Login,Me,Index} from './routes'
import {Router,Scene,Modal,Stack} from 'react-native-
router-flux'

// 创建一个根组件
const App = ()=>{
    return <Router>
    <Stack key="root">
     <Scene key="Login" component={Login} title="登录"
titleStyle={{color:'red'}}/>
     <Scene key="Index" component={Index} title="首页"/>
   </Stack>
   </Router>
}
export default App
```

路由管理-注册-stack

```
//routes.js

import Login from
'../pages/login/index';
import Index from
'../pages/index/index';
import Me from '../pages/me';

export default {
    Login,
    Index,
    Me
}

export {
    Login,
    Index,
    Me
}
```

```
//index.js

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


AppRegistry.registerComponent(app
Name, () => App);
```

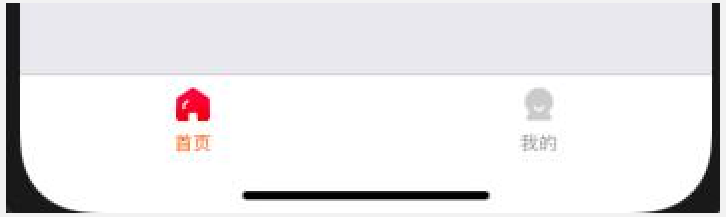


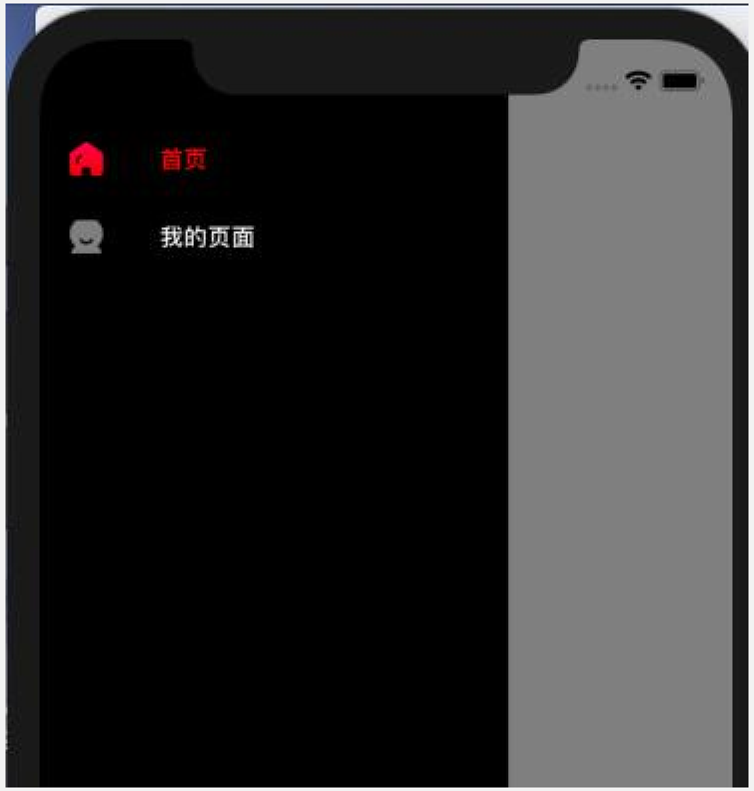
路由管理-注册-stack

```
import routes,{Login,Me,Index} from './routes'
import {createAppContainer} from 'react-navigation';
import {createStackNavigator} from 'react-navigation-stack'
import { createBottomTabNavigator } from 'react-navigation-tabs';
import React from 'react';
import {Image} from 'react-native';
const tabNavigation = createBottomTabNavigator({
  Index:{
    screen:Index,
    navigationOptions:{
        tabBarLabel:'首页',
        tabBarIcon:({focused,tintColor})=>{
         return  <Image  resizeMode="contain" style={{width:20,height:20}}
source={focused?require("../assets/images/index_press.png"):require("../assets/images/index.png")}/>
        }
      }
  },
  Me:{
    screen:Me,
    navigationOptions:{
        tabBarLabel:'我的',
        tabBarIcon:({focused,tintColor})=>{
         return  <Image resizeMode="contain" style={{width:20,height:20}}
source={focused?require("../assets/images/me_press.png"):require("../assets/images/me.png")}/>
        }
    }
  }
},
{
    initialRouteName: "Index",
    lazy: true,
    tabBarOptions: {
      inactiveTintColor: "#8F8F8F",
      activeTintColor: "#ED5601",
      labelStyle: {
         fontSize: 11
      }
    }
});
const App = createAppContainer(tabNavigation)
export default App
```

```
import {Image} from 'react-native';
import React from 'react'; //不能缺省
import routes,{Login,Me,Index} from './routes'
import {Router,Scene,Modal,Stack,Tabs} from 'react-native-router-flux'

// // 创建一个根组件
const App = ()=>{
    return <Router>
    <Tabs key="root" activeTintColor="#ED5601" inactiveTintColor="#8F8F8F"
labelStyle={{fontSize:11}}>
     <Scene key="Index" component={Index} title="首页" icon={tabIcon}/>
     <Scene key="Me" component={Me} title="我的" icon={tabIcon}/>
   </Tabs>
   </Router>
}
export default App

const tabIcon = ({ focused, title }) => {
        let list = {
          '首页': {
            icon: require('../assets/images/index.png'),
            activeIcon: require('../assets/images/index_press.png')
          },
          '我的': {
            icon: require('../assets/images/me.png'),
            activeIcon: require('../assets/images/me_press.png')
          }
        }
        let item = list[title]
        if (!focused) {
          return (
            <Image resizeMode="contain" style={{ width: 20, height: 20 }} source={item.icon} />
          );
        } else {
          return (
            <Image resizeMode="contain" style={{ width: 20, height: 20 }} source={item.activeIcon}
/>
          );
        }
}
```

路由管理-注册-bottomTabs





路由管理-注册-sideMenu

```
import routes,{Login,Me,Index} from './routes'
import {createAppContainer} from 'react-navigation';
import { createDrawerNavigator } from 'react-navigation-drawer';
import React from 'react';
import {Image} from 'react-native';
const drawerNavigator = createDrawerNavigator({
    Index:{
    screen:Index,
    navigationOptions:{
      drawerLabel:'首页',
      drawerIcon:({focused})=>{
          return <Image style={{width:20,height:20}}
source={focused?require("../assets/images/index_press.png"):require("../assets/images/index.pn
g")}/>
      }
    }
  },
  Me:{
    screen:Me,
    navigationOptions:{
       drawerLabel:'我的页面',
            drawerIcon:({focused})=>{
        return <Image style={{width:20,height:20}}
source={focused?require("../assets/images/me_press.png"):require("../assets/images/me.png")}/
>
       }
    }
  }
},
{
    initialRouteName: 'Index',
    drawerBackgroundColor:'black',
    drawerType:'front',
    contentOptions: {
      activeTintColor: 'red',
      inactiveTintColor:'white'
 },
})
const App = createAppContainer(drawerNavigator)

export default App
```

```
import {Image,Text,SafeAreaView,ScrollView} from 'react-native';
import React from 'react'; //不能缺省
import routes,{Login,Me,Index} from './routes'
import {Router,Scene,Modal,Stack,Tabs,Drawer} from 'react-native-router-flux'
import { DrawerItems } from 'react-navigation-drawer';
const App = ()=>{
    return <Router>
    <Drawer  key="root" contentComponent={(props)=>{
       return   <ScrollView>
       <SafeAreaView
       >
         <DrawerItems activeTintColor="red" {...props} renderIcon={(t)=>{
         return tabIcon({focused:t.focused,title:t.route.routes[0].params.title})
         }}/>
       </SafeAreaView>
     </ScrollView>
    }}>
     <Scene key="Index"  drawer={true} component={Index} title="首页" drawerIcon={tabIcon}
drawerLabel="首页" />
     <Scene key="Me"  drawer={true} component={Me} title="我的" drawerIcon={tabIcon}
drawerLabel="我的" />
   </Drawer>
   </Router>
}
const tabIcon = ({ focused, title}) => {
        let list = {
          '首页': {
            icon: require('../assets/images/index.png'),
            activeIcon: require('../assets/images/index_press.png')
          },
          '我的': {
            icon: require('../assets/images/me.png'),
            activeIcon: require('../assets/images/me_press.png')
          }
        }
        let item = list[title]
        if (!focused) {
          return (
            <Image resizeMode="contain" style={{ width: 20, height: 20 }} source={item.icon} />
          );
        } else {
          return (
            <Image resizeMode="contain" style={{ width: 20, height: 20 }} source={item.activeIcon}
/>
          );
        }
}
export default App
```

路由管理-Stack页面跳转和传值

Reset - 用新状态替换当前状态
Replace - 用给定的 key 替换另一条路由
Push - 在堆栈顶部添加一条路由，并向前导航至该路由
Pop - 导航回到之前的路由
PopToTop - 导航到堆栈的顶部路由，销毁所有其他路线

```
import { StackActions } from 'react-navigation';

const pushAction = StackActions.push({
  routeName: 'Profile',
  params: {
    myUserId: 9,
  },
});

this.props.navigation.dispatch(pushAction);
```

```
pop: () => void;
popAndPush
popTo
push
refresh: (props?: any) => void;
replace
reset

// 刷新当前页面
Actions.refresh({name:'xxx'})

// props参数改变时触发
componentWillReceiveProps(nextProps){
alert(JSON.stringify(nextProps.navigation.state.params.name))
}
```

路由管理-注册-bottomTabs

```
import { SwitchActions } from 'react-navigation';

// 切换tab页面
this.props.navigation.dispatch(SwitchActions.jumpTo({routeName:'Me' }));

// 获取参数
this.props.navigation.state.params.name
```

```
import {Actions} from 'react-native-router-flux';

// 跳转页面
Actions.jump('Me',{name:'xxx',age:'xxx'})
```

路由管理-打开关闭-Drawer

```
import { DrawerActions } from 'react-navigation-drawer';

// 打开
this.props.navigation.dispatch(DrawerActions.openDrawer())

// 关闭
this.props.navigation.dispatch(DrawerActions.closeDrawer())
```

```
import {Actions} from 'react-native-router-flux'

// 打开
Actions.drawerOpen()

// 关闭
Actions.drawerClose()
```

路由管理-注册-SwitchNavigator

SwitchNavigator 的用途是一次只显示一个页面。默认情况下，它不处理返回操作，并在你切换时将路由重置为默认状态

```
export default
createAppContainer(createSwitchNavigato
r(
  {
    AuthLoading: AuthLoadingScreen,
    App: AppStack,
    Auth: AuthStack,
  },
  {
    initialRouteName: 'AuthLoading',
  }
));
```

路由切换

```
import { SwitchActions } from 'react-
navigation';

this.props.navigation.dispatch(SwitchAction
s.jumpTo({ routeName }));
```

常用功能介绍

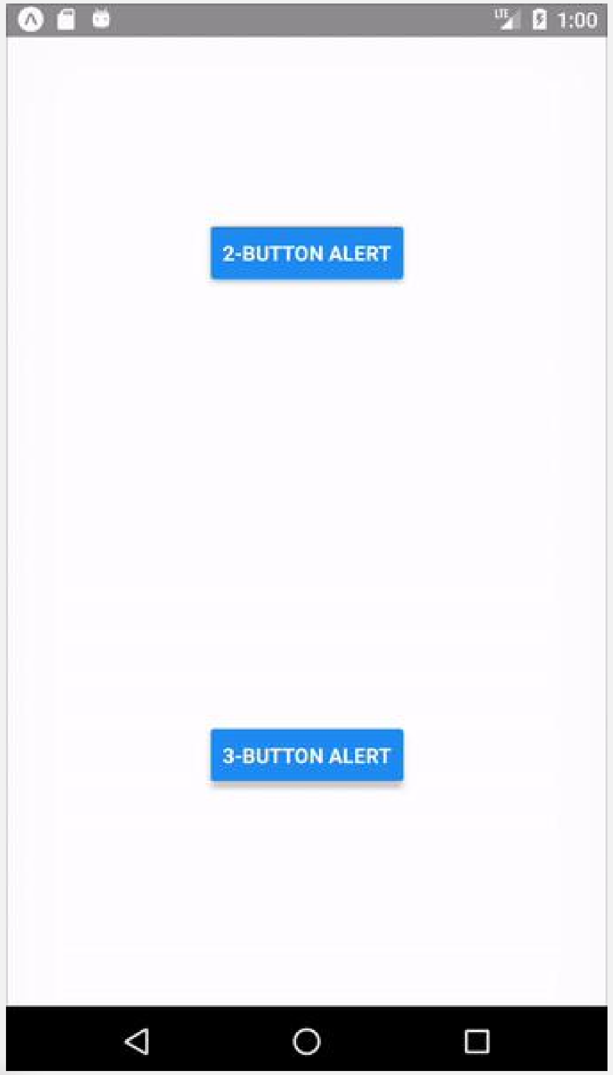
https://facebook.github.io/react-native/docs/getting-started.html

1.网络请求
2.弹窗
3.数据存储
4.复制粘贴功能
5.获取屏幕尺寸信息
6.获取平台信息

常用功能介绍-网络请求

```
fetch('http://chenxiaoping.com/demo', {
    //请求方式，GET或POST
    method: 'POST',

    //请求头定义
    headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
    },

    //body: JSON.stringify({
    //   firstParam: 'value1',
    //   secondParam: 'value1',
    // }),
}).then((response) => response.json()).then(
    //响应体，resonse,json拿到的就已经是转化好的jsonObject了，使用起来就非常简便
    (responseJson) => {
        //输出打印current_user_url字段，输出的内容可以直接在androidStudio日志输出里面看到
        console.log("请求回调： " + responseJson.current_user_url);
    }
)
```

常用功能介绍-弹窗



```
// Works on both Android and iOS
Alert.alert(
  'Alert Title',
  'My Alert Msg',
  [
    {
      text: 'Ask me later',
      onPress: () => console.log('Ask me later pressed')},
    {
      text: 'Cancel',
      onPress: () => console.log('Cancel Pressed'),
      style: 'cancel',
    },
    {text: 'OK', onPress: () => console.log('OK Pressed')},
  ],
  {cancelable: false},
);
```

常用功能介绍-数据存储

业务场景 缓存用户token

仓库地址
https://github.com/react-native-community/async-storage

安装
yarn add @react-native-community/async-storage

react-native link

cd ios && pod install

常用功能介绍-数据存储

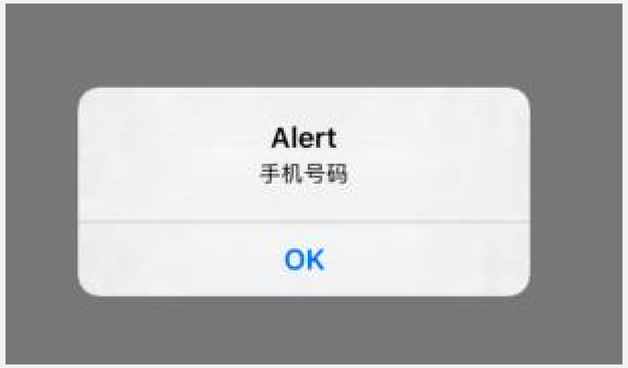
使用

```
import AsyncStorage from '@react-native-community/async-storage';

// 存储数据
storeData = async () => {
  try {
    await AsyncStorage.setItem('@storage_Key', 'stored value')
  } catch (e) {
    // saving error
  }
}
// 读取数据
getData = async () => {
  try {
    const value = await AsyncStorage.getItem('@storage_Key')
    if(value !== null) {
      // value previously stored
    }
  } catch(e) {
    // error reading value
  }
}
```

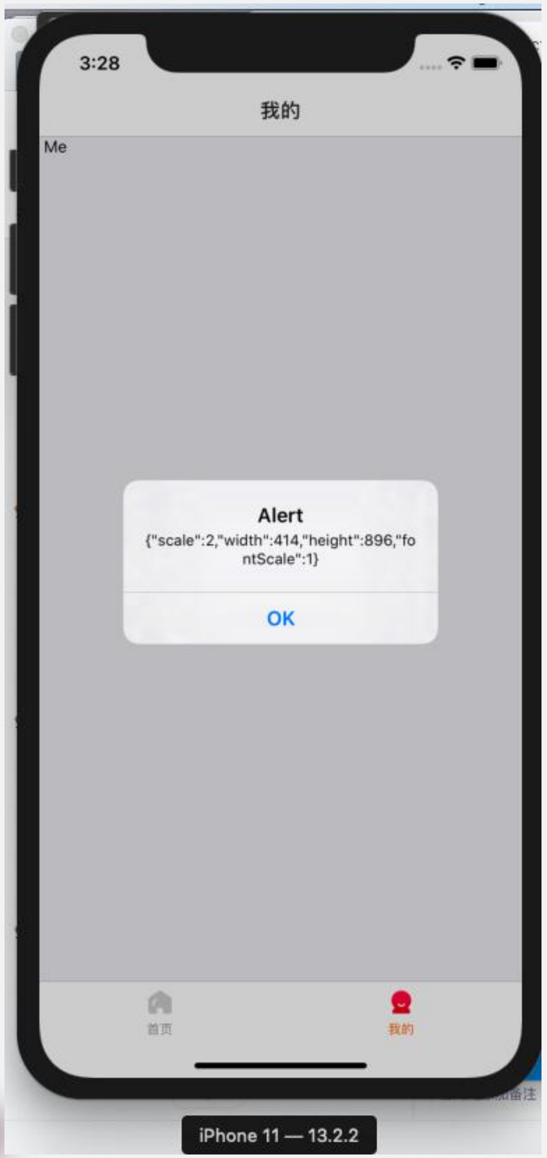
常用功能介绍 – 复制粘贴功能

```
import { Clipboard } from 'react-native';


// 写入数据
Clipboard.setString("手机号码")


// 读取数据
Clipboard.getString().then(res=>{
    alert(res)
})
```

```
readData= async ()=>{
    try{
        const data = await Clipboard.getString()
        alert(data)
    }catch(e){
    }
  }


  this.readData()
```



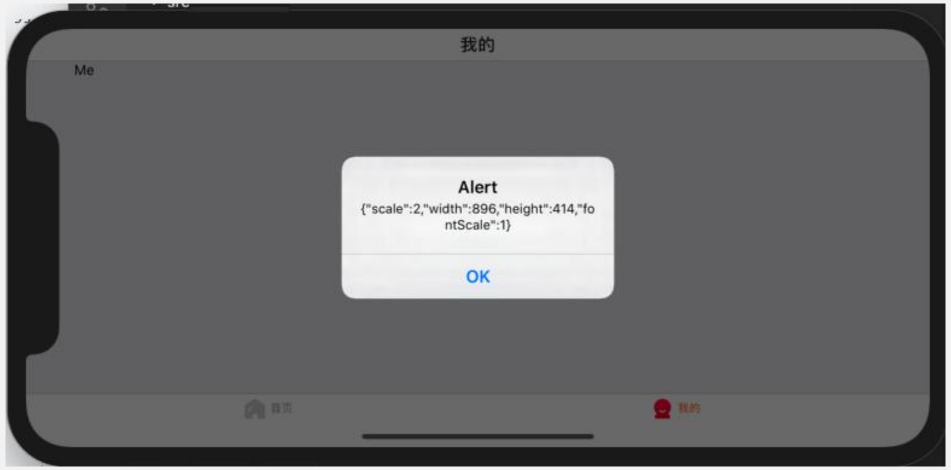
常用功能介绍 - 获取屏幕尺寸信息



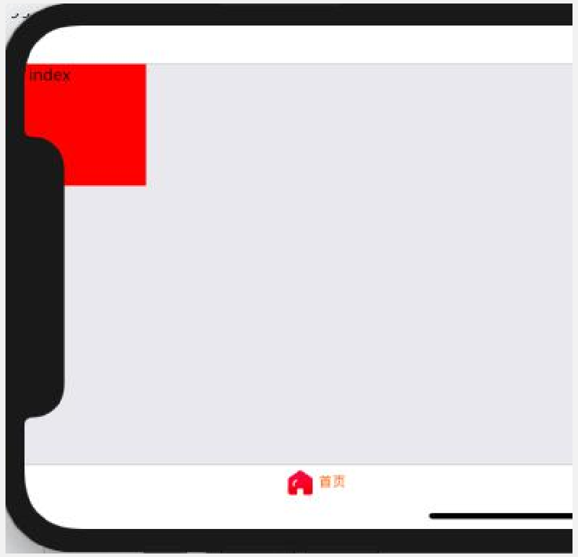
```
{"scale":2,"width":414,"height":896,"fontScale":1}
```

```
import {Dimensions } from 'react-native';
alert(JSON.stringify(Dimensions.get('window')))
const {width,height,scale,fontScale} = Dimensions.get('screen')
```

常用功能介绍-处理屏幕旋转问题



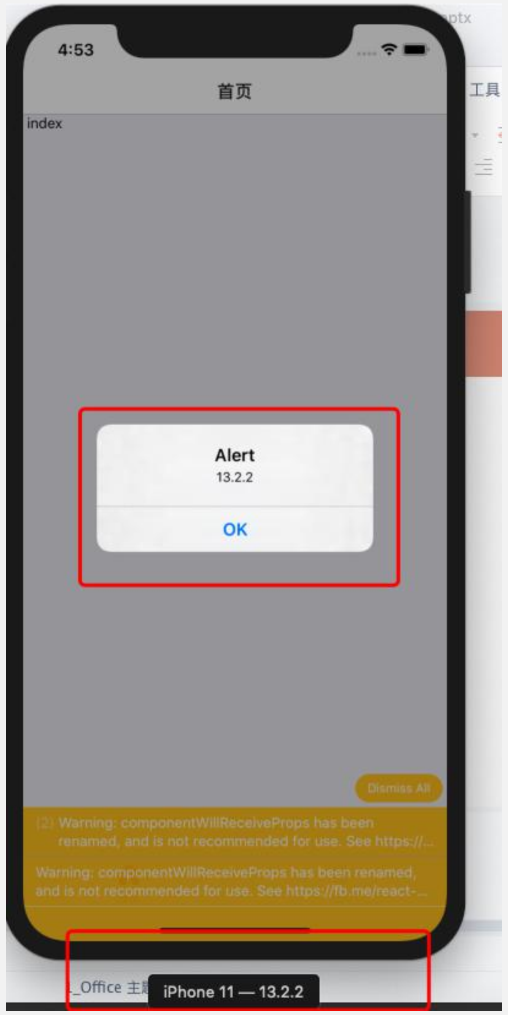
Alert
{"scale":2,"width":896,"height":414,"fontScale":1}
OK



```
Dimensions.addEventListener('change',({window,screen})=
>{
        alert(window)
  })

Dimensions.removeEventListener('change')
```

常用功能介绍-获取平台信息

```
import { Platform} from 'react-native';
Platform.OS         // ios android
Platform.Version  // 13.2.2
```

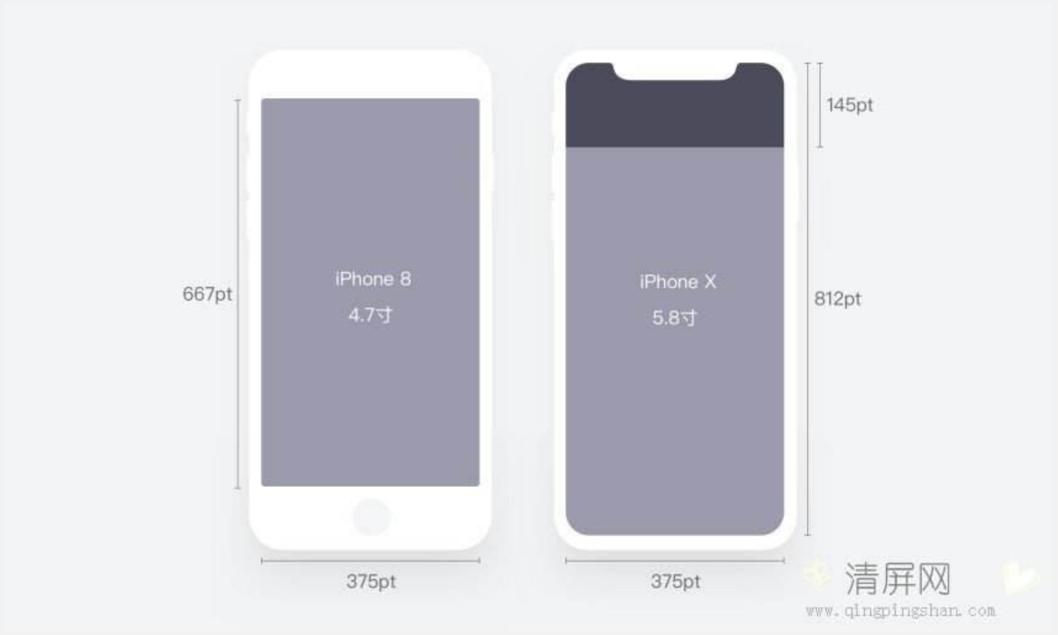


Warning: componentWillReceiveProps has been renamed, and is not recommended for use. See https://...

Warning: componentWillReceiveProps has been renamed, and is not recommended for use. See https://fb.me/react-...

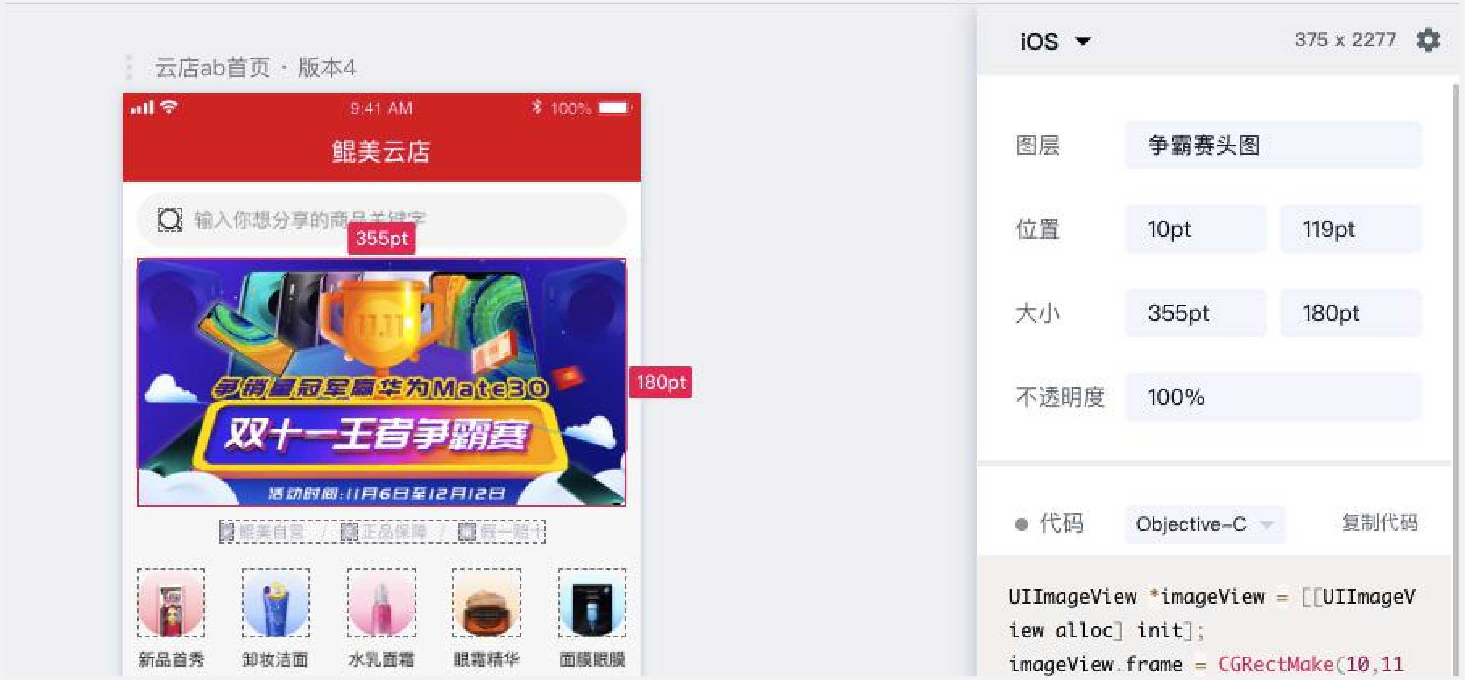
屏幕适配

设备名称	屏幕尺寸	PPI	Asset	竖屏点（point）	竖屏分辨率（px）
iPhone XS MAX	6.5 in	458	@3x	414 x 896	1242 x 2688
iPhone XS	5.8 in	458	@3x	375 x 812	1125 x 2436
iPhone XR	6.1 in	326	@2x	414 x 896	828 x 1792
iPhone X	5.8 in	458	@3x	375 x 812	1125 x 2436
iPhone 8+ , 7+ , 6s+ , 6+	5.5 in	401	@3x	414 x 736	1242 x 2208
iPhone 8, 7, 6s, 6	4.7 in	326	@2x	375 x 667	750 x 1334
iPhone SE, 5, 5S, 5C	4.0 in	326	@2x	320 x 568	640 x 1136
iPhone 4, 4S	3.5 in	326	@2x	320 x 480	640 x 960
iPhone 1, 3G, 3GS	3.5 in	163	@1x	320 x 480	320 x 480
iPad Pro 12.9	12.9 in	264	@2x	1024 x 1366	2048 x 2732
iPad Pro 10.5	10.5 in	264	@2x	834 x 1112	1668 x 2224
iPad Pro, iPad Air 2, Retina iPad	9.7 in	264	@2x	768 x 1024	1536 x 2048
iPad Mini 4, iPad Mini 2	7.9 in	326	@2x	768 x 1024	1536 x 2048
iPad 1, 2	9.7 in	132	@1x	768 x 1024	768 x 1024



屏幕适配

pt：iOS开发单位，即point，绝对长度，约等于0.16毫米



图层	争霸赛头图	
位置	10pt	119pt
大小	355pt	180pt
不透明度	100%	

代码 Objective-C

```
UIImageView *imageView = [[UIImageV
iew alloc] init];
imageView.frame = CGRectMake(10,11
```

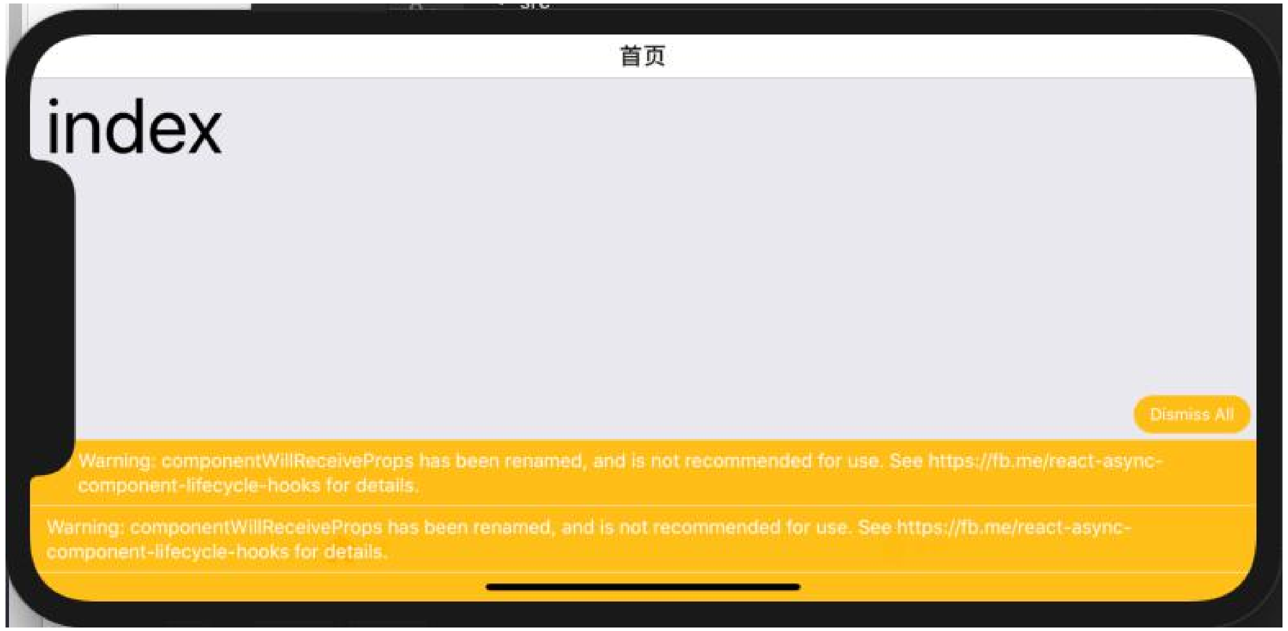
屏幕适配

原理 根据设计稿尺寸 动态计算其它屏幕的对应的pt

设计稿尺寸 x (实际屏幕宽度尺寸/设计稿宽度)

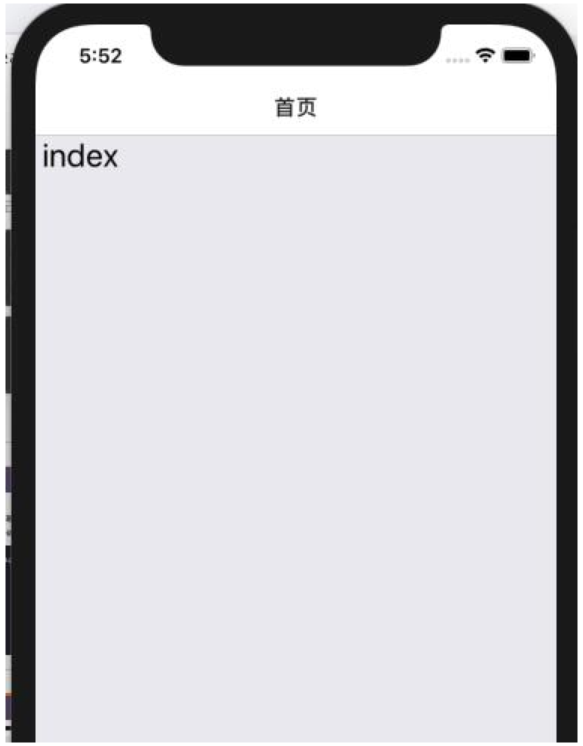
```
const sw = (width)=>{
     return parseInt(width * Dimensions.get('window').width/375.0)
 }

const style1 =StyleSheet.create({
    text:{
     marginLeft:sw(40),
     width:sw(355),
     height:sw(100),
     backgroundColor:'red'
   }
})
```

屏幕适配



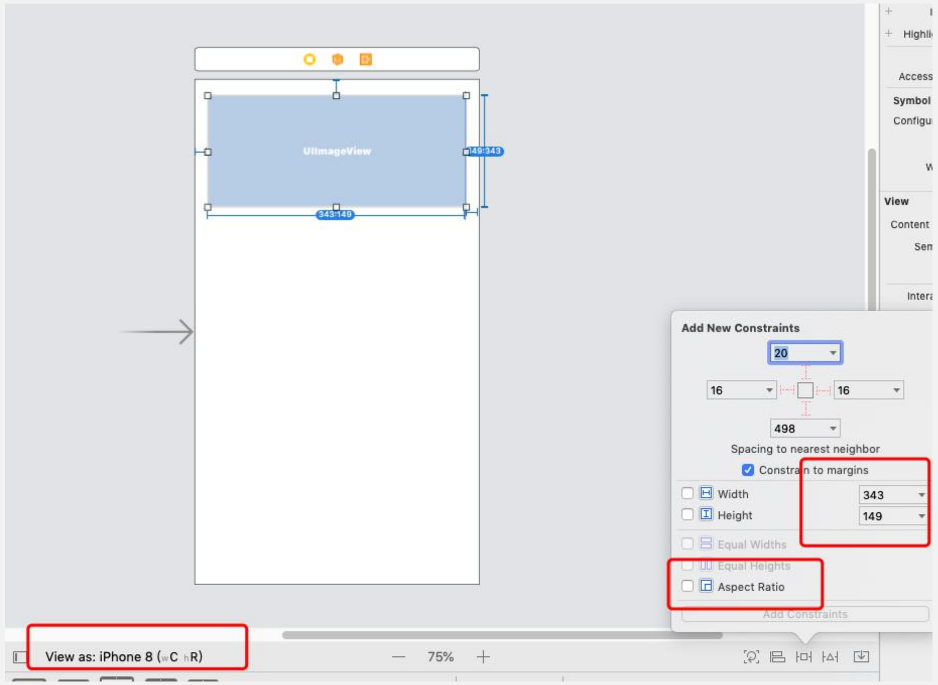
Warning: componentWillReceiveProps has been renamed, and is not recommended for use. See https://fb.me/react-async-component-lifecycle-hooks for details.

Warning: componentWillReceiveProps has been renamed, and is not recommended for use. See https://fb.me/react-async-component-lifecycle-hooks for details.



大屏的目的是显示更多的内容,而不仅仅是把内容放大

屏幕适配


Add New Constraints

20

16 16

498

Spacing to nearest neighbor

☑ Constrain to margins

Width	343
Height	149

Equal Widths

Equal Heights

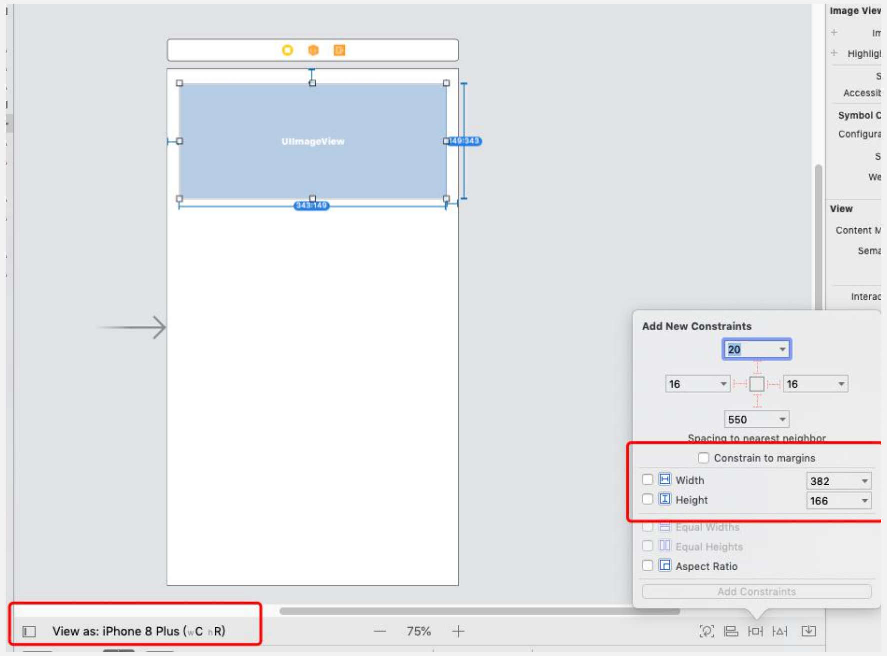
Aspect Ratio

Add Constraints

View as: iPhone 8 (wC hR)

75%

ipone8显示尺寸


Add New Constraints

20

16 16

550

Spacing to nearest neighbor

☐ Constrain to margins

Width	382
Height	166

Equal Widths

Equal Heights

Aspect Ratio

Add Constraints

View as: iPhone 8 Plus (wC hR)

75%

iphone 8P 显示尺寸

屏幕适配

1.字体使用设计稿上的pt单位

2.如果是按照比例显示使用 aspectRatio(宽/高的值)

```
text:{
   marginLeft:sw(40),
   width:100,
   aspectRatio:1.5,
   backgroundColor:'red'
}
```

3.特殊布局使用sw 进行适配

4.如果是非绝对布局(position),请全部使用flex布局

打包发布

ios 打包发布详情流程
https://www.jianshu.com/p/d3dc262cffa4

android 打包发布详细流程

https://www.jianshu.com/p/3acba4233bc6

结束

欢迎上船,稳住 我们能赢!