一、React简介

React 是一个用于构建用户界面的 JavaScript 库，它是 Facebook 的内部项目，用来架设 Instagram 的网站，并于 2013 年 5 月开源。React 主要用于构建UI，很多人认为 React 是 MVC 中的 V（视图）。由于拥有较高的性能，且代码逻辑非常简单，越来越多的人已开始关注和使用它。

中文官网：https://react.docschina.org

二、React特点

1、声明式设计

react是面向数据编程，不需要直接去控制dom，你只要把数据操作好，react自己会去帮你操作dom，可以节省很多操作dom的代码。这就是声明式开发。

2、使用JSX语法

JSX 是 JavaScript 语法的扩展。React 开发大部分使用 JSX 语法（在JSX中可以将HTML于JS混写）。

3、灵活

react所控制的dom就是id为root的dom，页面上的其他dom元素你页可以使用jq等其他框架 。可以和其他库并存。

4、使用虚拟DOM、高效

虚拟DOM其实质是一个JavaScript对象，当数据和状态发生了变化，都会被自动高效的同步到虚拟DOM中，最后再将仅变化的部分同步到DOM中（不需要整个DOM树重新渲染）。

5、组件化开发

通过 React 构建组件，使得代码更加容易得到复用和维护，能够很好的应用在大项目的开发中。

6、单向数据流

react是单向数据流，父组件传递给子组件的数据，子组件能够使用，但是不能直接通过this.props修改。 这样让数据清晰代码容易维护。

三、React脚手架的安装

使用 `npm/cnpm` 安装React官方脚手架(create-react-app)到全局

```
npm install create-react-app -g    //4.0.3
```

四、创建我们的第一个React项目

4.1、构建一个名为 `demo` 项目

```
create-react-app demo
```

4.2、进入这个文件夹，并且跑起来

```
cd demo && npm start
```

项目运行，自动打开谷歌浏览器看见项目页面

（第一次项目运行会很慢，可能需要10到15分钟，请耐心等待）

五、hello_world程序编写

1、删除在src目录下所有文件

2、src目录下新建index.js文件作为入口文件

```
//1、引入React两个核心模块
import React from 'react';
import ReactDOM from 'react-dom';


//2、通过JSX语法将组件/标签渲染到指定标签上
ReactDOM.render(
    <div>
        hello world!
    </div>
    , document.getElementById('root'));
```

ReactDOM.render(参数1，参数2)

参数1 是JSX语法的标签/组件

参数2 是要把参数1这个标签渲染到的位置

六、VSCode中JSX的编写优化

在vscode中的 `文件>首选项>设置` 中，直接搜索 `include Language` ，进入 `settings.json` ：

找到 `"emmet.includeLanguages"` 字段，添加：

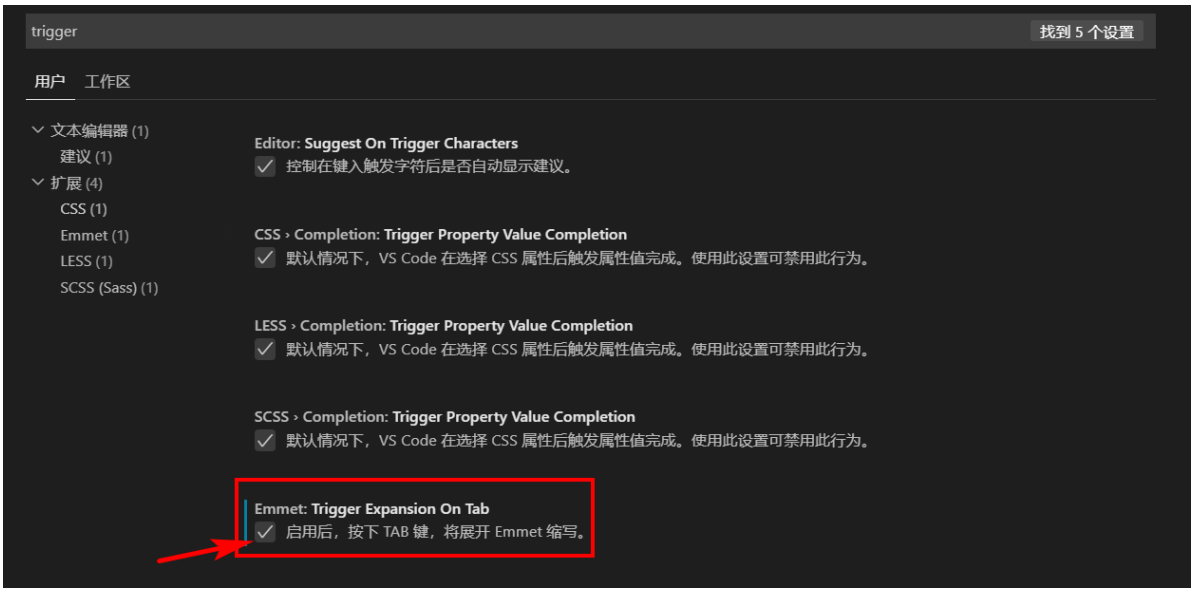
```
"emmet.includeLanguages": {
        "javascript": "javascriptreact"
},
```

"emmet.includeLanguages": {

"javascript": "javascriptreact"

}

搜索 `trigger` ，找到如图位置，打钩：



设置后就可以通过 标签名+Tab 键快速码出标签

七、组件化开发

在React的项目中，都是使用组件化开发的模式，所以，我们可以把刚才的hello world的div定义为一个组件：

定义组件分为3步：

1、导入React核心模块

2、定义组件类

3、导出组件

在src目录下新建App.js文件：

```
//1、导入React核心模块
import React from 'react'

//2、定义组件类
class Hello extends React.Component{   //类
    render(){     //函数
        return (   //返回值
            <div>
                hello world !!! 我是组件222
            </div>
        )
    }
}


//3、导出组件
export default Hello
```

在需要引入该组件的index.js中，导入，就可以直接使用这个组件了：

```
//import 组件名 from '文件路径'
import Hello from './App'   //1、导入Hello组件  (首字母必须大写)

ReactDOM.render(
    <Hello />         // 2、使用Hello组件  (首字母必须大写)
    , document.getElementById('root'));

//注意：Hello是组件名，在使用的时候就应该写JSX标签写法，而不能直接写Hello
```

！！注意：

1、定义组件的时候，return 后面只能有一个根标签，不能有多个，但这个标签内部可以有其他多个标签

2、使用组件的时候，首字母必须大写

八、JSX语法糖

React 使用 JSX 来替代常规的 JavaScript。JSX 是一个看起来很像 XML 的 JavaScript 语法扩展，在React中会被babel编译为JavaScript。

8.1、JSX的特点

- JSX 执行更快，因为它在编译为 JavaScript 代码后进行了优化。
- 它是类型安全的，在编译过程中就能发现错误。
- 使用 JSX 编写模板更加简单快速。

8.2、JSX几个注意的格式：

1、React的JSX是使用大写和小写字母来区分本地组件和HTML组件

（如：html就用 div p h1 ，组件就用 MyButton App Home List等 ）

2、JSX和html的标签属性的区别

HTML标签属性	JSX	原因
for	htmlFor	for在JS中为for循环关键字
class	className	class在JS中为声明类关键字
style	需使用JS对象(使用双花括号--**{{}}**)	

组件中：

```
            <div>
                <p  style={{backgroundColor: "#ccc"}}>
                    hello world !!! 我是组件2223
                </p>

                <img src={MyImg} alt="" className="img1" /> <br/>
                <label htmlFor="username">用户名:
                    <input type="text" id="username"/>
                </label>

            </div>
```

8.3、React的JSX创建出来的是虚拟DOM，而不是HTML

在index.js中：

```
const DomEl = document.createElement("div")
const ReactEl = React.createElement("div",null,"hello");   //虚拟DOM
<div>hello</div>

let Dnum=0
let Rnum=0
for (const i in ReactEl){
    Rnum++
}
for (const i in DomEl){
    Dnum++
}
console.log("ReactDom对象的属性总个数：", Rnum)    // 7
console.log("传统Dom对象的属性总个数：", Dnum)    // 240+
//小结：
//JSX不是HTML
//JSX创建的是ReactDOM(虚拟DOM)
//JSX中的<div>hello</div>  等同于 React.createElement("div",null,"hello");

//！！！以上代码只做了解，只记住结论就行
```

8.4、JSX变量引用、三目运算符、for循环

在JSX中，想要调用变量，需要在return中直接使用单花括号--**{}**调用

index.js中：

```
//第四节：JSX中变量引用、三目运算符、for循环的使用
import App3 from './App3'

//2、通过JSX语法将标签/组件渲染到指定标签上
ReactDOM.render(
    <App3 />
    , document.getElementById('root'));
```

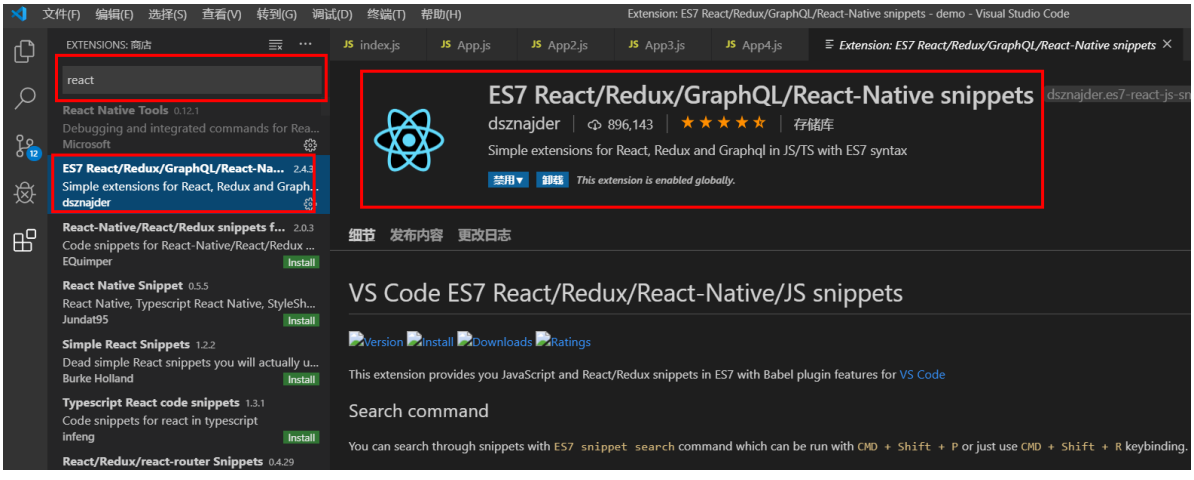
App3.js中

```
import React, { Component } from 'react'

let name = "小明", num1=20, num2=30, arr=[1, 2, 3, 4, 5]
```

```
export default class App3 extends Component {
    render() {
        return (
            <div>
                {/* 这是注释的格式 */}
                {/* JSX中引用变量需要加单花括号 */}
                <p>{name}</p>

                {/* 三目运算符的使用 */}
                <p>num1和num2中比较大的是：{num1>num2? num1: num2}</p>


                {/* for循环的使用 */}
                <ul>
                    {/* 数组名.map(函数) */}
                    {
                        //格式1：
                        arr.map((v,k)=>(
                                <li key={k}>{v}</li>
                            )
                        )
                    }
                    {
                        //格式2：可以把上面的大括号和return换成小括号
                        arr.map((v,k)=>{
                                return <li key={k}>{v}</li>
                            }
                        )
                    }
                </ul>

            </div>
        )
    }
}
```

九、VSCode中，代码片段扩展的安装

在扩展搜索栏中搜索react，选择下图的扩展进行安装



安装后，通过 rcc+回车 ，得到组件的代码片段， clg+回车 快速得到console.log()代码

十、事件的使用

App4.js中:

```
//事件讲解

import React, { Component } from 'react'

//1、实现点击弹框效果(事件基本格式)
// export default class App4 extends Component {
//     handleClick(e){
//         alert(132456)
//     }

//     render() {
//         return (
//             <div>
//                 <button onClick={this.handleClick}>按钮</button>
//             </div>
//         )
//     }
// }


//2、实现累加的功能 (状态的使用1)
// export default class App4 extends Component {
//     constructor(props){
//         super(props)

//         this.state = {
//             num: 20
//         }
//     }

//     handleClick(){
//         //1、事件中，this的指向，在经过绑定后，指向这个App4组件
//         //2、通过调用this.setState()方法来修改this.state里面的数据
//         this.setState({
//             num: this.state.num+1
//         })
//     }

//     render() {
//         return (
//             <div>
//                 <p>{this.state.num}</p>
//                 <button onClick={this.handleClick.bind(this)}>点击增加
1</button>

//                 {/* 下面使用箭头函数可以不用绑定this */}
//                 {/* <button onClick={() => this.handleClick()}>点击增加
1</button> */}
//             </div>
//         )
//     }
// }
```

```
//3、实现双向数据绑定 (状态的使用2)
export default class App4 extends Component {
    constructor(props){
        super(props)

        this.state = {
            str1: "123"
        }
    }
    handleChange(e){
        console.log(e)
        console.log(e.target)

        this.setState({
            str1: e.target.value
        })
    }

    render() {
        return (
            <div>
                <input type="text" onChange={this.handleChange.bind(this)}/>
                <p>{this.state.str1}</p>
            </div>
        )
    }
}
/*  防抖节流
npm i --save lodash.throttle   安装方式



import React,{Component} from "react"
import throttle from 'lodash/throttle'
export default class App4 extends Component {
    constructor(props){
        super(props)

        this.state = {
            str1: "123"
        }
        this.handleChange = throttle(this.handleChange,2000)
    }
    handleChange(e){
        console.log(e)
        console.log(e.target)
        this.setState({
                str1: e.target.value
        })

    }
    handleSubmit(e){
        e.preventDefault()
        this.handleChange(e)
    }

    render() {
```

```
        return (
            <div>
                <input type="text" onChange={this.handleSubmit.bind(this)}/>
                <p>{this.state.str1}</p>
            </div>
        )
    }
}

*/
```

十一、组件状态this.state的基本使用

组件状态this.state的基本使用总结：

```
/*
在组件内部：

定义状态数据：
    constructor(props){
        super(props)

        this.state = {
            num: 20
        }
    }

使用状态数据：
    return (
            <div>
                <p>{this.state.num}</p>
            </div>
        )


修改状态数据：
        1、通过事件或者定时器触发：
        <button onClick={this.handleClick.bind(this)}>点击增加1</button>
        2、在事件函数中：
            this.setState({
                num: this.state.num+1
            })

*/
```

十二、组件属性this.props

App5.js中：

```
import React from 'react'


//定义一个头部组件(子组件)
```

```
class Header extends React.Component{
    //定义默认值
    static defaultProps = {
        bgc : "blue",
        title : "默认标题",
        children: "默认的子元素"
    }

    render(){
        return (
            <div style={{height:60, backgroundColor:this.props.bgc,
textAlign:"center", color:"#fff"}}>
                {this.props.title}
                <p>{this.props.children}</p>
            </div>

        )
    }
}

// 父组件
export default class hello extends React.Component{
    render(){
        return (
            <div>
                <Header title="首页" bgc="green"/>
                <Header title="列表页" bgc="red"/>
                <Header/>

                {/* 子元素的使用 */}
                <Header bgc="pink">Header子元素</Header>
            </div>
        )
    }
}
```

```
/*
this.props基本使用 总结:

1、在父组件中使用子组件的时候，可以给组件设定属性的值
    <组件名 属性名=值 />  例如: <Header title="首页" bgc="green"/>

2、在定义子组件的时候，需要填入值的位置书写  this.props.属性名  来获取定义的值
    <div style={{height:40, backgroundColor:this.props.bgc, textAlign:"center",
color:"#fff"}}>
        {this.props.title}
    </div>

3、在子组件内部可以定义默认值，格式如下:

    //定义默认值
    static defaultProps = {
        属性名: 默认值,
        bgc : "blue",
        title : "默认标题"
```

```
    }

4、子元素的使用(了解)
    在父组件中，以<组件名 属性名=值>  </组件名>方式使用子组件的时候，可以添加子元素：
    <组件名 属性名=值>子元素</组件名>
    例如：<Header bgc="pink">Header子元素</Header>

    在子组件内部，通过  this.props.children  来获取这个子元素
*/
```

十三、state和props小结（注意点、区别）

1、定义组件的时候，状态初始化在 constructor 方法中完成；

```
constructor(props){
    super(props)

    this.state = {
        //状态属性名：值
        num: 20
    }
}
```

this.state定义的属性，称为state属性

2、修改状态属性，要使用 this.setState()

```
this.setState({
    //状态属性名：值
    num: this.state.num + 1
})
```

3、使用组件的时候定义的属性，称为props属性

```
<Header title="列表页" bgc="red"/>   // title和bgc 就是props属性
```

4、React组件显示的变化，都是通过状态state的修改，重写(setState())渲染到页面中

5、state 和 props 主要的区别在于 props 一般是不可变的，而 state 可以根据与用户交互来改变

6、props可以用来区分的同个组件创建(复用)出来的不同标签，它由外部（父组件）传入。state用来定义和修改组件内部的一些数据

下面举个最最通俗的例子：

你爸爸拿给你100块钱叫你去帮他买包烟，这个钱是通过 props 传给你的（父组件通过props可以往子组件传递数据）。

剩下的钱自己去买零食吃，你心情觉得高兴，高兴就是你这个人的状态数据（state就是组件的内部数据）。

你心情的好坏，理论上来讲，你没说出来，你爸爸是不知道你的心情（即子组件的state数据一般和父组件无关）。

十四、子组件中限制传进来的props属性的数据类型

子组件中可以限制传进来的props属性值的数据类型，需要先安装prop-types模块

```
npm install --save prop-types

或者

yarn add prop-types (如果没有安装yarn的话，先安装npm install -g yarn)
```

实现步骤：

1、先导入import PropTypes from 'prop-types'

2、在子组件中定义静态属性propTypes：

static propTypes = {

//props属性名：PropTypes.类型

title: PropTypes.类型

}

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'

// 子组件
class Header extends Component {

    static propTypes = {
        title: PropTypes.string
    }
    render() {
        return (
            <div style={{height:40, backgroundColor:'blue', textAlign:"center",
color: "#fff"}}>
                {this.props.title}
            </div>
        )
    }
}


// 父组件
export default class App extends Component {
    render() {
        return (
            <div>
                <Header title="首页" />
            </div>
        )
    }
}
```

十五、使用context进行props属性值的多级传递

React 组件之间的通信是基于 props 的数据传递，数据需要一级一级从上到下传递。如果组件级别过多传递就会过于麻烦。React中Context配置可以解决组件跨级值传递。

```
实现步骤：
     1. 在父组件中，声明数据类型和值
          1.1 声明上下文数据类型
              static childContextTypes = {
                  数据名：  数据类型
              }
          1.2 向上下文控件中存值
              getChildContext(){
                  return {
                      数据：值
                  }
              }

    2. 在“无论哪一级别”子组件中，只需声明需要的全局上下文数据，就可自动注入子组件的context属性中
        static contextTypes = {
             数据名：  数据类型
        }

        在子组件中使用：
            {this.context.con}
```

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'

class ChildChild extends Component {

    static contextTypes = {
        con: PropTypes.string
    }
    render() {
        return (
            <div>
                子组件中的子组件----{this.context.con}
            </div>
        )
    }
}

// 子组件
class Child extends Component {
    render() {
        return (
            <div>
                <ChildChild/>
            </div>
        )
    }
}
```

```
// 父组件
export default class App extends Component {

    static childContextTypes = {
        con: PropTypes.string
    }

    getChildContext(){
        return {
            con: "爷爷组件"
        }
    }

    render() {
        return (
            <div>
                <Child/>
            </div>
        )
    }
}
```

十六、React生命周期

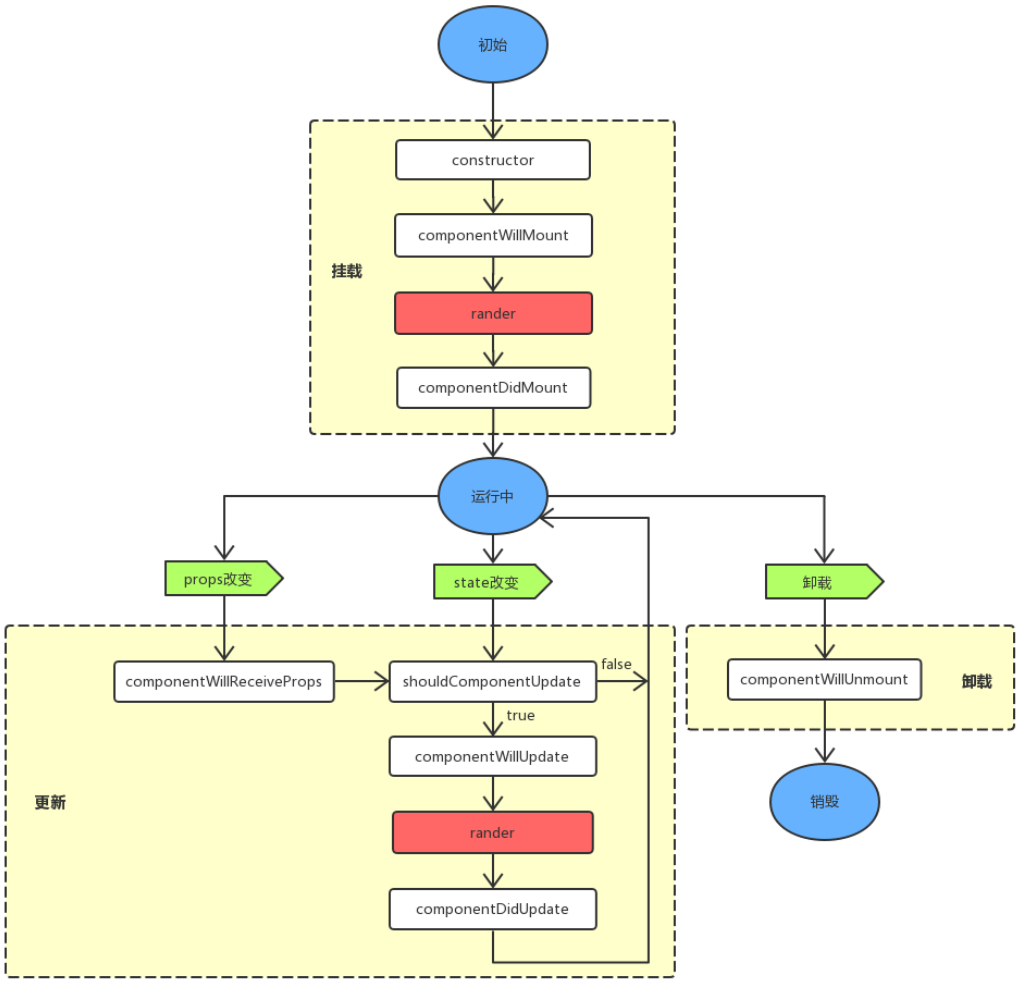
生命周期：就是指一个对象的生老病死。 React的生命周期指从组件被创建到销毁的过程。掌握了组件的生命周期，就可以在适当的时候去做一些事情。

React生命周期可以分成三个阶段：

1、实例化(挂载阶段)：对象创建到完全渲染

2、存在期(更新期)：组件状态的改变

3、销毁/清除期：组件使用完毕后，或者不需要存在与页面中，那么将组件移除，执行销毁。



16.1 实例化/挂载阶段

constructor()

componentWillMount()

render()

componentDidMount()

```
export default class App3 extends Component {
    // 生命周期第一个阶段:  挂载/初始化阶段
    constructor(props){
        console.log("1.1 constructor: 构造初始化")
        // 调用父类构造方法
        super(props)

        // 初始化状态数据
        this.state = {

        }
        // 事件函数this的绑定
        this.handleClick = this.handleClick.bind(this)

    }

    UNSAFE_componentWillMount(){
        console.log("1.2 componentWillMount")
```

```
        //做一些准备性的工作，比如提示正在加载
    }

    componentDidMount() {
        console.log("1.4 componentDidMount")
        //异步加载数据
    }


    handleClick(){
        alert("点击了p标签")
    }

    render() {
        console.log("1.3 render")
        return (
            <div>
                <p onClick={this.handleClick}>这是一个展示生命周期的组件</p>
                <p onClick={this.handleClick}>这是一个展示生命周期的组件</p>
                <p onClick={this.handleClick}>这是一个展示生命周期的组件</p>
            </div>
        )
    }
}
```

16.2 存在期/更新期

存在期:组件已经渲染好并且用户可以与它进行交互。通常是通过一次鼠标点击、手指点按者键盘事件来触发一个事件处理器。随着用户改变了组件或者整个应用的state，便会有新的state流入组件树，并且我们将会获得操控它的机会。

componentWillReceiveProps()

shouldComponentUpdate()

componentWillUpdate()

render()

componentDidUpdate()

在上面的组建中继续书写生命周期函数：

```
    render() {
        console.log("1.3/2.4 render")
        return (
            <div>
                <p onClick={this.handleClick}>这是一个展示生命周期的组件</p>
                <p onClick={this.handleClick}>这是一个展示生命周期的组件</p>
                <p onClick={this.handleClick}>这是一个展示生命周期的组件
{this.state.num}</p>
            </div>
        )
    }
```

```
    handleClick(){
        this.setState({
            num:22
        })
    }

    // 生命周期第二个阶段  存在期/更新期
    componentWillReceiveProps(){
        console.log("2.1 componentWillReceiveProps")
    }
    shouldComponentUpdate(nextProps, nextState) {
        console.log("2.2 shouldComponentUpdate 可以判断修改前后的值是不是一样，不一样才去执行render。减少不必要的render，优化更新性能")
        console.log("旧的值：", this.state.num)
        console.log("新的值：", nextState.num)

        // return true   则执行render
        // return false   则不执行render
        //这里返回值是布尔值，但不应该写死，
        //而是判断新旧两个值是不是不相等，不相等就要执行render(就要返回true)
        return this.state.num !== nextState.num

    }
    componentWillUpdate(nextProps, nextState) {
        console.log("2.3 componentWillUpdate 更新前的生命周期回调")
    }
    componentDidUpdate(prevProps, prevState) {
        console.log("2.5 componentDidUpdate 更新后的生命周期回调")
    }
```

以上执行的是组件内部state数据更新前后的生命周期函数，

其实，**对于组件的props属性值发生改变的时候，同样需要更新视图，执行render**

componentWillReceiveProps() 这个方法是将要接收新的props值的时候执行，而props属性值从父组件而来，所以需要定义父组件：

```
class App3 extends Component {
    ...

    //生命周期第二个阶段  存在期/更新期
    UNSAFE_componentWillReceiveProps(nextProps){
        console.log("2.1 componentWillReceiveProps 这个方法props属性值更新的时候才会执行，更新state数据则不会执行这个方法")

        console.log(nextProps)
    }
    ...
    shouldComponentUpdate(nextProps, nextState) {
        console.log("2.2 shouldComponentUpdate 可以判断修改前后的值是不是一样，不一样才去执行render。减少不必要的render，优化更新性能")

        // return true   则执行render
        // return false   则不执行render
        //这里返回值是布尔值，但不应该写死，
        //而是判断新旧两个值是不是不相等，不相等就要执行render(就要返回true)
```

```
        return (this.state.num !== nextState.num || this.props.fatherNum !== nextProps.fatherNum)  //不仅仅是state数据跟新时候需要执行render，props属性的值更新的时候也要执行render，所以要多加一个判断条件

    }
}

export default class Father extends Component{
    constructor(props){
        // 调用父类构造方法
        super(props)

        // 初始化状态数据
        this.state = {
            fatherNum:0
        }

    }

    componentDidMount() {
        setTimeout(() => {
            this.setState({
                fatherNum:10
            })
        }, 2000);
    }


    render(){

        return(
            <App3 fatherNum={this.state.fatherNum}/>
        )
    }
}
```

16.3 销毁期

componentWillUnmount() 销毁组件前做一些回收的操作

```
    componentDidMount() {
        console.log("1.4 componentDidMount")
        console.log("----------------------------------------")

        //异步加载数据
        document.addEventListener("click", this.closeMenu);
    }
    closeMenu(){
        console.log("click事件 closeMenu")
    }

    // 生命周期第三个阶段  卸载期/销毁期
    componentWillUnmount() {
        console.log("3.1 componentWillUnmount 做一些回收的操作")
        document.removeEventListener("click", this.closeMenu);
```

```
    }
```

index.js中3秒后重新渲染页面：

```
setTimeout(() => {
    ReactDOM.render(
        <div>hello world</div>
        , document.getElementById('root'));
}, 3000);
```

16.4 生命周期小结

React组件的生命周期

3大阶段10个方法

1、初始化期（执行1次）

2、存在期 （执行N次）

3、销毁期 （执行1次）

小结：

componentDidMount ： 发送ajax异步请求

shouldComponentUpdate ： 判断props或者state是否改变，目的：优化更新性能

十七、不受控组件和受控组件

17.1 不受控组件

组件中表单元素没有写value值，与state数据无关，不受组件管理。(不推荐)

```
//不受控组件
import React, { Component } from 'react'

export default class App4 extends Component {


    handleClick(){
        console.log(this.refs.username.value)
        console.log(this.refs.password.value)
    }
    render() {
        return (
            <div>
                {/* 这种组件和组件本身state数据没有关系，所以不受组件管理，称为不受控组件(无约束组件)*/}
                {/* 这种写法的每个表单元素的验证在登录按钮的点击事件完成，用户体验很差 */}
                用户名：<input type="text" ref="username"/> <br/> <br/>
                密 码：<input type="text" ref="password"/><br/> <br/>
                <button onClick={this.handleClick.bind(this)}>登录</button>
            </div>
        )
    }
```

```
}
```

17.2 受控组件

组件中表单元素的value值受组件的state数据控制，并且该表单元素有onChange事件，我们可以在事件中对该表单做实时验证，验证值是否合法然后做相应的操作(推荐)

```
//受控组件
import React, { Component } from 'react'

export default class App4 extends Component {

    constructor(props){
        // 调用父类构造方法
        super(props)

        // 初始化状态数据
        this.state = {
            value:"admin"
        }
        // 事件函数this的绑定
        this.handleChange = this.handleChange.bind(this)

    }

    handleChange(e){
        console.log(e.target.value)
        //可以在这个方法内部做一些实时验证，验证值是否合法做响应的操作
        this.setState({
            value:e.target.value
        })
    }
    render() {
        return (
            <div>

                用户名：<input type="text" value={this.state.value} onChange=
{this.handleChange}/> <br/> <br/>
                密 码：<input type="text" /><br/> <br/>
                <button>登录</button>
            </div>
        )
    }
}
```

17.3 小结

React中的表单组件，分为2类：

1. 不受控组件（和状态无关）
   1.1 在input标签组件中，使用ref属性，来代表组件标识
   1.2 组件渲染后，会自动把有ref属性的dom对象，挂到this.refs上
   this.refs = {
   ref名1 ： dom对象
   ref名2 ： dom对象

}

1.3 在需要的场景（一般指回调），使用this.refs来获取需要的对象，然后再做dom操作

2、 受控组件（和状态紧密相关）

2.1 初始化状态

2.2 在input标签的value属性上，绑定状态（输入框的默认值和状态有关）

2.3 在input标签上维护onChange属性，触发时即时修改状态

2.4 自动：当状态修改，立即触发声明周期的render方法，显示最先的状态值

使用：

如果对值的控制度不高，使用不受控组件

如果要严格把控值的操作，受控组件可以做表单验证等严格的逻辑(推荐)操作

十八、路由原理

SPA： Single Page Application

例如：https://music.163.com/#/

现实生活中的路由：用来管理网络和计算机之间的关系

程序中的路由：用来管理**url地址**和**视图**之间的关系

路由原理：

1、准备视图(html)

2、准备路由的路线(可以是一个对象，键名是路线名称和值是视图地址)

3、通过hash地址的路线，获取“视图地址”

4、在指定标签中，加载需要的视图页面

代码演示：

router.html

```
    <!-- 1、定义视图，(点击)改变哈希地址 -->
```

```
    <ul>
        <li><a href="#/index">首页</a></li>
        <li><a href="#/list">列表页</a></li>
        <!-- 注意：href填的是hash地址，这里的 #/ 不要去掉 ，为的是不改变域名(地址栏中#前面的为域名)-->
    </ul>
    <div id="routerView"></div>

    <script>
        // 2、定义路由
        let louter = {
            //路由名称      视图页面
            "#/index": "./index.html",
            "#/list": "./list.html"
        }
        // 通过 louter[键名] 来获取值
        console.log(louter["#/index"])

        //3、检测哈希地址的改变(事件监听)
        window.addEventListener('hashchange',()=>{
            console.log("location.hash为: ", location.hash)   //location.hash可以获取哈希地址
            // 可以看出location.hash就是我们的键，我们可以通过 louter[键名] 来获取值，即获取视图页面

            let url = louter[location.hash]
            // console.log(url)
            // 加载对应视图
            // 在<div id="routerView"></div>中load加载需要的视图页面
            $("#routerView").load(url)

        })


    </script>
```

记得在head中添加jq

```
<script src="https://cdn.bootcss.com/jquery/1.12.4/jquery.min.js"></script>
```

十九、react路由搭建

react-router中奉行一切皆组件的思想，路由器-Router、链接-Link、路由-Route、独占-Switch、重定向-Redirect都以组件形式存在。
Route渲染优先级：children>component>render

App.js

```
import React, { Component } from 'react'
import RouterPage from './RouterPage'
export default class App extends Component {
    render() {
        return (
            <div className="app">
                <RouterPage />
            </div>
        )
    }
}
ReactDOM.render(<App />, document.getElementById('root'));
```

RouterPage.js

```
import React, { Component } from "react";
import { BrowserRouter, Link,Route,Switch} from "react-router-dom";
import HomePage from "./HomePage";
import UserPage from "./UserPage";
export default class RouterPage extends Component {
    render() {
        return (
            <div>
                <h1>RouterPage</h1>
                <BrowserRouter>
                    <nav>
                        <Link to="/">首页</Link>
                        <br />
                        <Link to="/user">用户中心</Link>
                    </nav>
                    <Switch>
                        <Route path="/" exact component={HomePage} />
                        <Route path="/user"  component={UserPage} />
                        <Route  component={()=><div><h1>404</h1></div>} />
                    </Switch>
                </BrowserRouter>
            </div>
        );
    }
}
```

动态路由

```
import React, { Component } from "react";
import { BrowserRouter, Link,Route,Switch} from "react-router-dom";
import HomePage from "./HomePage";
import UserPage from "./UserPage";
function Search(props){
    console.log(props);
    const {id} = props.match.params
    return (
        <div>
            <h1>{'Search'+id}</h1>
        </div>
    )
}
```

```
export default class RouterPage extends Component {
    render() {
        const id = 123
        return (
            <div>
                <h1>RouterPage</h1>
                <BrowserRouter>
                    <nav>
                        <Link to="/">首页</Link>
                        <br />
                        <Link to="/user">用户中心</Link>
                        <br />
                        <Link to={/search/+id}>搜索</Link>
                    </nav>
                    <Switch>
                        <Route path="/" exact component={HomePage} />
                        <Route path="/user"  component={UserPage} />
                        <Route path="/search/:id"  component={Search} />
                        <Route  component={()=><div><h1>404</h1></div>} />
                    </Switch>
                </BrowserRouter>
            </div>
        );
    }
}
```

路由嵌套

```
import React, { Component } from "react";
import { BrowserRouter, Link,Route,Switch} from "react-router-dom";
import HomePage from "./HomePage";
import UserPage from "./UserPage";
function Search(props){
    console.log(props);
    const {id} = props.match.params
    return (
        <div>
            <h1>{'Search'+id}</h1>
            <Link to="/search/detail">搜索--详情</Link>
        </div>
    )
}
function Detail(props){
    return (
        <div>
            <h1>search----detail</h1>
        </div>
    )
}
export default class RouterPage extends Component {
    render() {
        const id = 123
        return (
            <div>
                <h1>RouterPage</h1>
                <BrowserRouter>
                    <nav>
```

```
                        <Link to="/">首页</Link>
                        <br />
                        <Link to="/user">用户中心</Link>
                        <br />
                        <Link to={/search/+id}>搜索</Link>
                    </nav>
                    <Switch>
                        <Route path="/" exact component={HomePage} />
                        <Route path="/user"  component={UserPage} />
                        <Route path="/search/:id"  component={Search} />
                        <Route path="/search/detail"  component={Detail} />
                        <Route  component={()=><div><h1>404</h1></div>} />
                    </Switch>
                </BrowserRoter>
            </div>
        );
    }
}
```

6.1、安装react路由

yarn add react-router@3.2.0

6.2、在路由配置模块中，引入路由

```
import {
    Router,
    Route,
    hashHistory
} from 'react-router';
```

6.3、配置路由器

```
const router= <Router history={hashHistory}>  {/*路由器*/}
        <Route> {/*路线*/}
            <Route path="" component={}/>
        </Route>

      </Router>
```

其中，Router是路由器，路由器包含了所有路线；Route是路线，路线记录了路线名称和组件的对应关系

6.4、引入相关组件

引入路线中需要用到的组件

```
import Index from './Index'
import List from './List'
```

Index.js首页模块:

```
import React, { Component } from 'react'

export default class Index extends Component {
```

```
  constructor(props){
    super(props)

    console.log('Index初始化 ！！ ')
  }
  render() {
    return (
      <h1>
        欢迎进入首页
      </h1>
    )
  }
}
```

List.js列表页模块：

```
import React, { Component } from 'react'

export default class List extends Component {
  constructor(props){
    super(props)

    console.log('List初始化 ！！ ')
  }
  render() {
    return (
      <h1>
        列表页
      </h1>
    )
  }
}
```

6.5、渲染路由

```
ReactDom.render(
    router,
    document.querySelector('#root')
)
```

6.6、完整代码示例

App.js中

```
import React, {Component} from 'react'
import ReactDOM from 'react-dom'

//1、安装路由

//yarn add react-router@3.2.0

//2、引入路由
import {
```

```
    Router,
    Route,
    hashHistory
} from 'react-router';

//4、引入react路由中需要用到的组件
import Index from './Index'
import List from './List'



export default class App extends Component {
    render() {
        return (
            <ul>
                <li><a href="#/index">去往首页</a></li>
                <li><a href="#/list">去往列表页</a></li>
            </ul>
        )
    }
}

//3、配置react路由
const router1 = <Router history={hashHistory}>
        <Route path='/'  component={App}  />
        <Route path='/index'  component={Index}  />
        <Route path='/list'  component={List}  />
    </Router>


ReactDOM.render(
    //5、渲染路由
    router1
    , document.getElementById('root'));
```

index.js中

```
// 路由原理
// import './day02/App';
```

二十、嵌套路由

7.1、步骤

步骤1、可以在Route中嵌套子路由

```
<Route path='/app'  component={App}>
    <Route path='/app/index'  component={Index}  />
    <Route path='/app/list'  component={List}  />
</Route>
```

步骤2、在父路线的组件中，使用 {this.props.children}来获取子路线组件视图（必须当子组件激活时，才会去创建对应的子组件，并挂到this.props.children）

在视图组件中：

```
            <div>
                 <h1>我是顶级组件</h1>
                 <ul>
                     <li><a href="#/app/index">去往首页</a></li>
                     <li><a href="#/app/list">去往列表页</a></li>
                 </ul>
                 <hr/>
                 {this.props.children}
             </div>
```

7.2、优点

1、可以实现页面的局部刷新

2、按需加载子组件

3、子组件可以使用相对路径，继承父路线来作为前缀

即上面步骤1也可以写成 ：相对路径的写法

```
<Route path='/app'  component={App}>
    <Route path='index'  component={Index}  />     {/* 相对路径的写法 */}
    <Route path='list'  component={List}  />     {/* 相对路径的写法 */}
</Route>
```

二十一、路由传参

8.1、定义路由，在path上声明参数规则

```
<Route path='detail/:newsId'  component={Detail}  >
```

8.2、使用连接地址的时候，传参

```
<a href="#/app/detail/99">
```

8.3、在路线对应的视图组件中，使用this.props获取

```
this.props.routeParams.newsId
```

在Detail.js中：

```
import React, { Component } from 'react'

export default class Detail extends Component {
  constructor(props){
    super(props)

    console.log('Detail初始化 ！！ ')
  }
```

```
  render() {
    return (
      <h1>
        详情页
        {
          this.props.routeParams.newsId
        }
      </h1>
    )
  }
}
```

二十二、其他路由组件

IndexRoute、IndexRedirect和Link组件

1、导入

```
import {
    Router,
    Route,
    hashHistory,
    IndexRoute,
    IndexRedirect,
    Link
} from 'react-router';
```

2、IndexRoute 组件的使用：

```
// <IndexRoute component = {Index}/>组件，当访问#/app 时候，在{this.props.children}
默认展示Index页面
//（没有使用IndexRoute组件的情况下，{this.props.children}不展示任何页面）
const router4 = (
    <Router history={hashHistory}>
        <Route path='/app'  component={App}>
            <IndexRoute component = {Index}/>
            <Route path="index" component={Index}/>
            <Route path='list'  component={List}  />
            <Route path='detail/:newsId'  component={Detail}  />
        </Route>


    </Router>
    )
```

3、IndexRedirect 组件的使用：

```
// <IndexRedirect to="index"/>重定向组件，当访问 #/app 时候，直接重定向到#/app/index
const router5 = (
    <Router history={hashHistory}>
        <Route path='/app'  component={App}>
            <IndexRedirect to="index"/>
            <Route path='index'  component={Index}  />
            <Route path='list'  component={List}  />
            <Route path='detail/:newsId'  component={Detail}  />
        </Route>


    </Router>
    )
```

4、Link 组件的使用：

```
                <ul>
                     {/* to属性的值是连接地址，不需要加# ,本质是a标签*/}
                     <li><Link to="/app/index">去往首页</Link></li>
                     <li><Link to="/app/list">去往列表页</Link></li>
                     <li><Link to="/app/detail/99">去往详情页</Link></li>
                 </ul>
```

二十三、路由传参

在上一天的案例中操作，(/src/router/02_react_router)

1.1、定义路由，在path上声明参数规则

```
<Route path='detail/:newsId'  component={Detail}  >
```

1.2、使用连接地址的时候，传参

```
<a href="#/app/detail/99">
```

1.3、在路线对应的视图组件中，使用this.props获取

```
this.props.routeParams.newsId
```

在Detail.js中：

```
import React, { Component } from 'react'

export default class Detail extends Component {
  constructor(props){
    super(props)

    console.log('Detail初始化 ！！ ')
  }
```

```
  render() {
    return (
      <h1>
        详情页
        {
          this.props.routeParams.newsId
        }
      </h1>
    )
  }
}
```

二十四、其他路由组件

IndexRoute、IndexRedirect和Link组件

1、导入

```
import {
    Router,
    Route,
    hashHistory,
    IndexRoute,
    IndexRedirect,
    Link
} from 'react-router';
```

2、IndexRoute 组件的使用：

```
// <IndexRoute component = {Index}/>组件，当访问#/app 时候，在{this.props.children}
默认展示Index页面
//（没有使用IndexRoute组件的情况下，{this.props.children}不展示任何页面）
const router4 = (
    <Router history={hashHistory}>
        <Route path='/app'  component={App}>
            <IndexRoute component = {Index}/>
            <Route path="index" component={Index}/>
            <Route path='list'  component={List}  />
            <Route path='detail/:newsId'  component={Detail}  />
        </Route>


    </Router>
    )
```

3、IndexRedirect 组件的使用：

```
// <IndexRedirect to="index"/>重定向组件，当访问 #/app 时候，直接重定向到#/app/index
const router5 = (
    <Router history={hashHistory}>
        <Route path='/app'  component={App}>
            <IndexRedirect to="index"/>
            <Route path='index'  component={Index}  />
            <Route path='list'  component={List}  />
            <Route path='detail/:newsId'  component={Detail}  />
        </Route>


    </Router>
    )
```

4、Link 组件的使用：

```
                <ul>
                     {/* to属性的值是连接地址，不需要加# ,本质是a标签*/}
                     <li><Link to="/app/index">去往首页</Link></li>
                     <li><Link to="/app/list">去往列表页</Link></li>
                     <li><Link to="/app/detail/99">去往详情页</Link></li>
                 </ul>
```

二十五、项目技术点介绍及Node环境

《叩丁狼咨询移动端项目》前端技术点介绍：

使用react作为前端框架
使用create-react-app脚手架创建工程
使用react-router实现前端路由部署
使用redux，react-redux进行状态管理
使用Normalize.css进行初始规范化样式
直接使用less进行样式管理
使用阿里的蚂蚁金服移动端UI框架antd-mobile的各种组件
封装自己的react组件
使用第三方组件
进行前后端的联调
项目部署上线

node环境如下：

node -v (10.15.0)

npm -v (6.4.1)

二十六、项目库环境——create-react-app脚手架安装

4.1、安装官方脚手架（用来项目搭建）

cnpm install -g create-react-app

4.2、在项目目录创建工程

create-react-app kdlzx

4.3、进入安装目录

cd kdlzx

4.4、启动项目查看内容

npm start

二十七、项目集成less

1、解包脚手架

yarn eject 或者 npm run eject

解包之后，先尝试启动项目，如果发现类似报错 Cannot find module '@babel/plugin-transform-react-jsx'

1. 删除项目下node_modules文件夹
2. yarn add start （重新下载node_modules）
3. yarn start

2、安装less环境:

1）（**！！已经安装了yarn的就跳过这一步**）安装React团队提供yarn包管理工具

D:\react\kdlzx>**npm i yarn -g**

2）安装less环境

D:\react\kdlzx>**yarn add less less-loader -D**

3、把less加载器配置到webpack配置文件中

在config文件夹中的webpack.config.js文件：

```
48   const cssRegex = /\.css$/;
49   const cssModuleRegex = /\.module\.css$/;
50   const lessModuleRegex = /\.less$/;
51   const sassRegex = /\.(scss|sass)$/;
52   const sassModuleRegex = /\.module\.(scss|sass)$/;
53
```

代码：

```
const lessModuleRegex = /\.less$/;
```

```
// less加载器
{
  test: lessModuleRegex,
  use: getStyleLoaders(
    {
      //暂不配置
    },
    'less-loader'
  ),
},
```

二十八、准备项目目录结构

进入项目文件夹：

1、src目录下，只保留index.js文件，其他全部删除

2、src目录下，新建assets文件夹（存放静态资源），并在其中新建fonts/styles/images等目录

3、src目录下，新建components文件夹（存放组件代码）

4、src目录下，新建pages文件夹（存放页面代码）

5、在pages文件夹新建App.js文件

6、在App.js中 rcc 触发组件代码， 复制index.js中的 ReactDOM.render(, document.getElementById('root')); 到App.js的最后

7、在App.js中补充 import ReactDOM from 'react-dom'

8、在index.js中，引入App：

```
//入口文件，就像一个清单一样

//引入顶级组件
import './pages/App'

//引入全局样式
```

9、启动项目 npm start/yarn start

二十九、测试less

在src/assets/styles/ 目录下新建core.less，别写测试代码：

```
@charset 'utf-8';

@color: #f00;

body{
    background: @color;
}
```

在src/index.js 中引入全局样式：

```
import './assets/styles/core.less'
```

要引入less文件直接作为页面的样式文件，应该将高版本的less-loader降级，本人测试最好的版本是：

`npm install less-loader@6.0.0`

启动项目 npm start

三十、登录页面准备

在src/pages下新建LoginPage.js 文件 (rcc 回车，然后引入外部样式，书写div容器内容):

```
// 1、引入核心模块
import React, { Component } from 'react'

// 2、引入外部样式
import '../assets/styles/loginPage.less'

// 3、声明并且导出组件类
```

```
export default class LoginPage extends Component {
    render() {
        return (
            <div className="login-page">
                login页面
            </div>
        )
    }
}
```

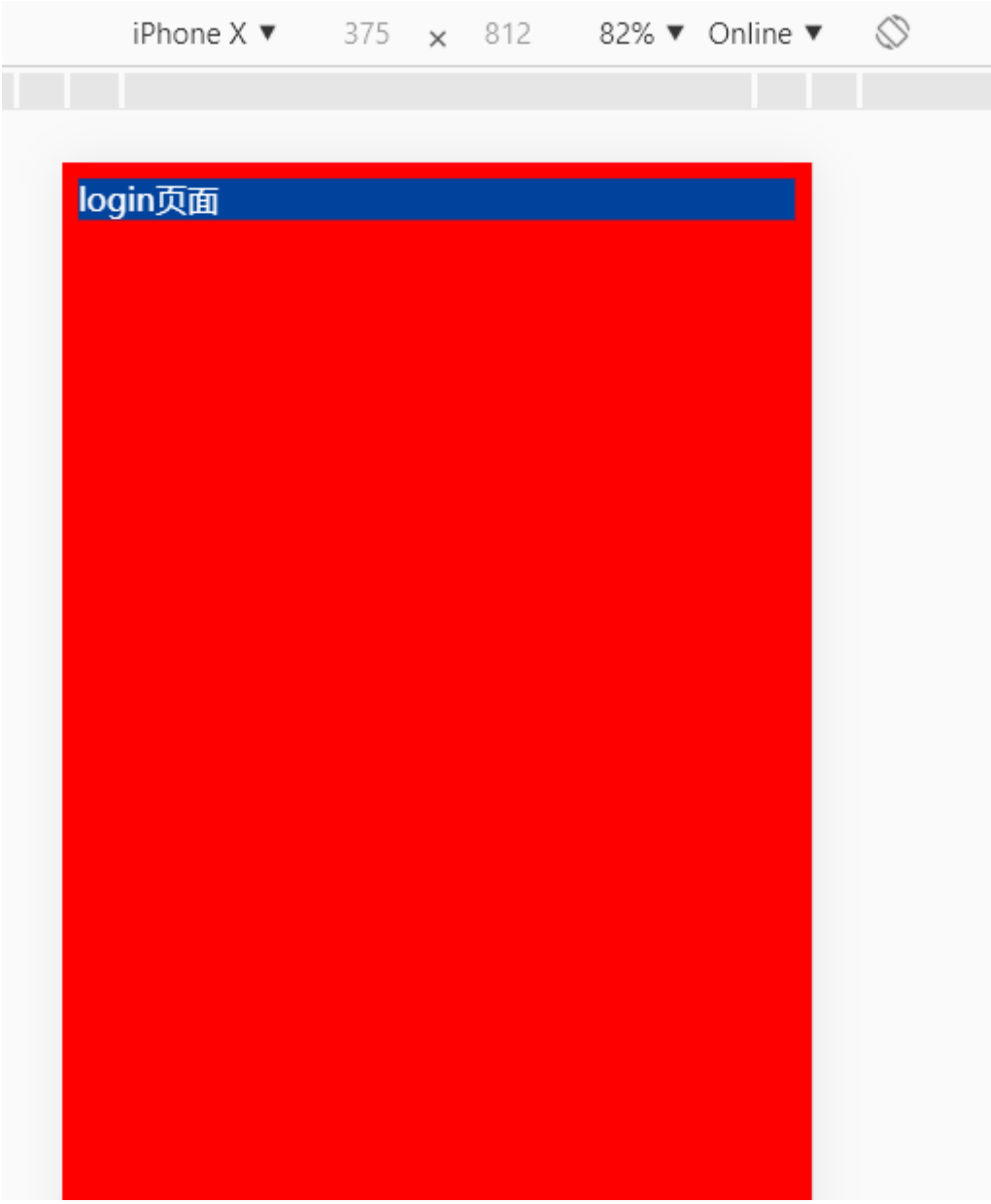
在src/assets/styles/目录下新建loginPage.less ，书写：

```
@charset 'utf-8';

.login-page{

    color:#fff;
    background-color: #00439c;
}
```

在src/pages下的App.js中 (导入登录页，返回Login页面)：

```
import LoginPage from './LoginPage'

        return (
            <LoginPage/>
        )
```

看到效果：



三十一、重置样式、设置页面高度

项目中初始化样式采用 Normalize.css

http://necolas.github.io/normalize.css/

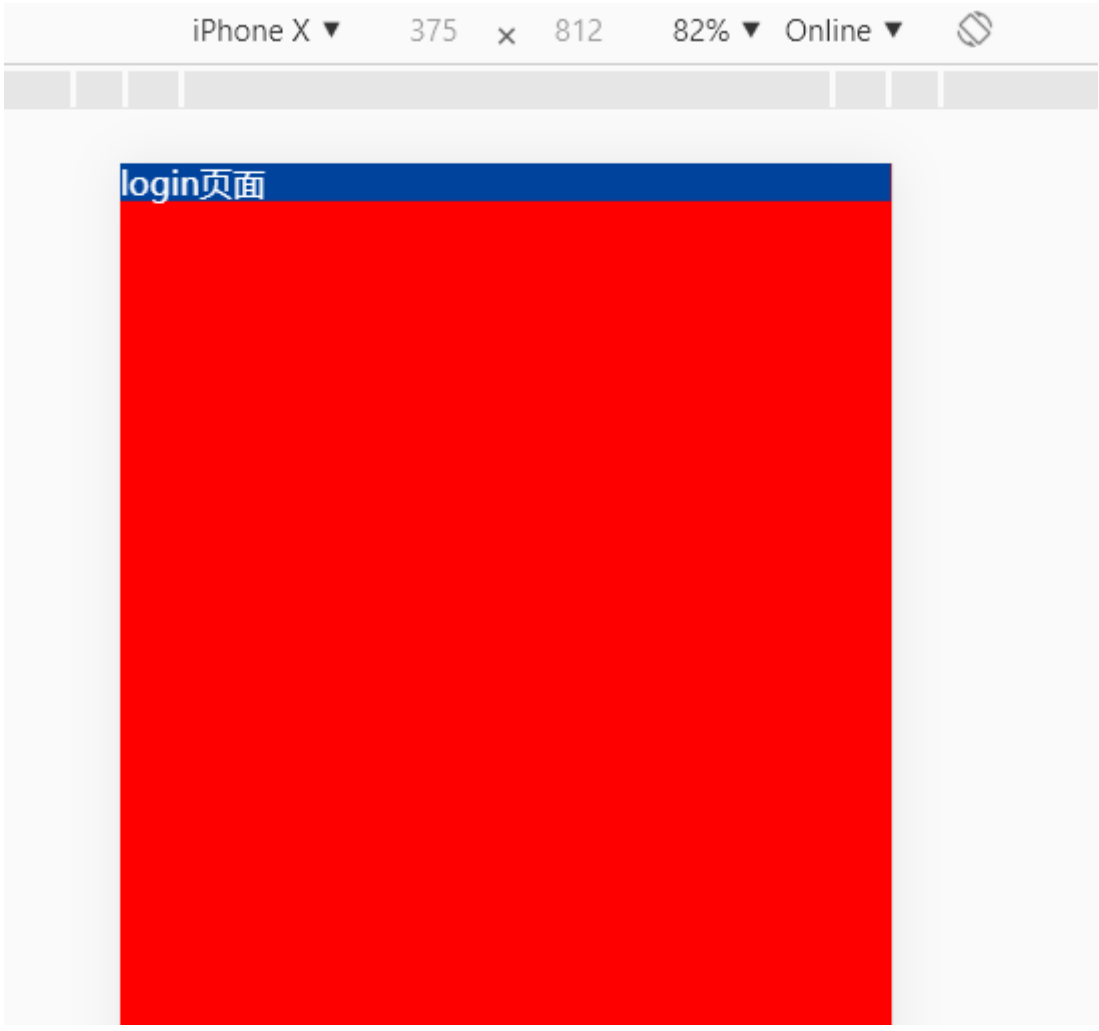
终端中，项目目录下安装 Normalize.css：

D:\react\kdlzx>**yarn add normalize.css**

```
yarn add v1.19.1
warning package-lock.json found. Your project contains lock files generated by tools other than Yarn. It is advised not
to mix package managers in order to avoid resolution inconsistencies caused by unsynchronized lock files. To clear this
warning, remove package-lock.json.
[1/4] Resolving packages...
[2/4] Fetching packages...
info There appears to be trouble with your network connection. Retrying...
info fsevents@2.1.1: The platform "win32" is incompatible with this module.
info "fsevents@2.1.1" is an optional dependency and failed compatibility check. Excluding it from installation.
info fsevents@1.2.9: The platform "win32" is incompatible with this module.
info "fsevents@1.2.9" is an optional dependency and failed compatibility check. Excluding it from installation.
[3/4] Linking dependencies...
warning "@typescript-eslint/eslint-plugin > tsutils@3.17.1" has unmet peer dependency "typescript@>=2.8.0 || >= 3.2.0-de
v || >= 3.3.0-dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev || >= 3.7.0-beta".
[4/4] Building fresh packages...
success Saved lockfile.
success Saved 1 new dependency.
info Direct dependencies
└─normalize.css@8.0.1
info All dependencies
└─normalize.css@8.0.1
Done in 11.29s.
```

安装成功后，在src/pages下的index.js中:

```
//重置样式
import 'normalize.css'
```



接下来设置页面高度：

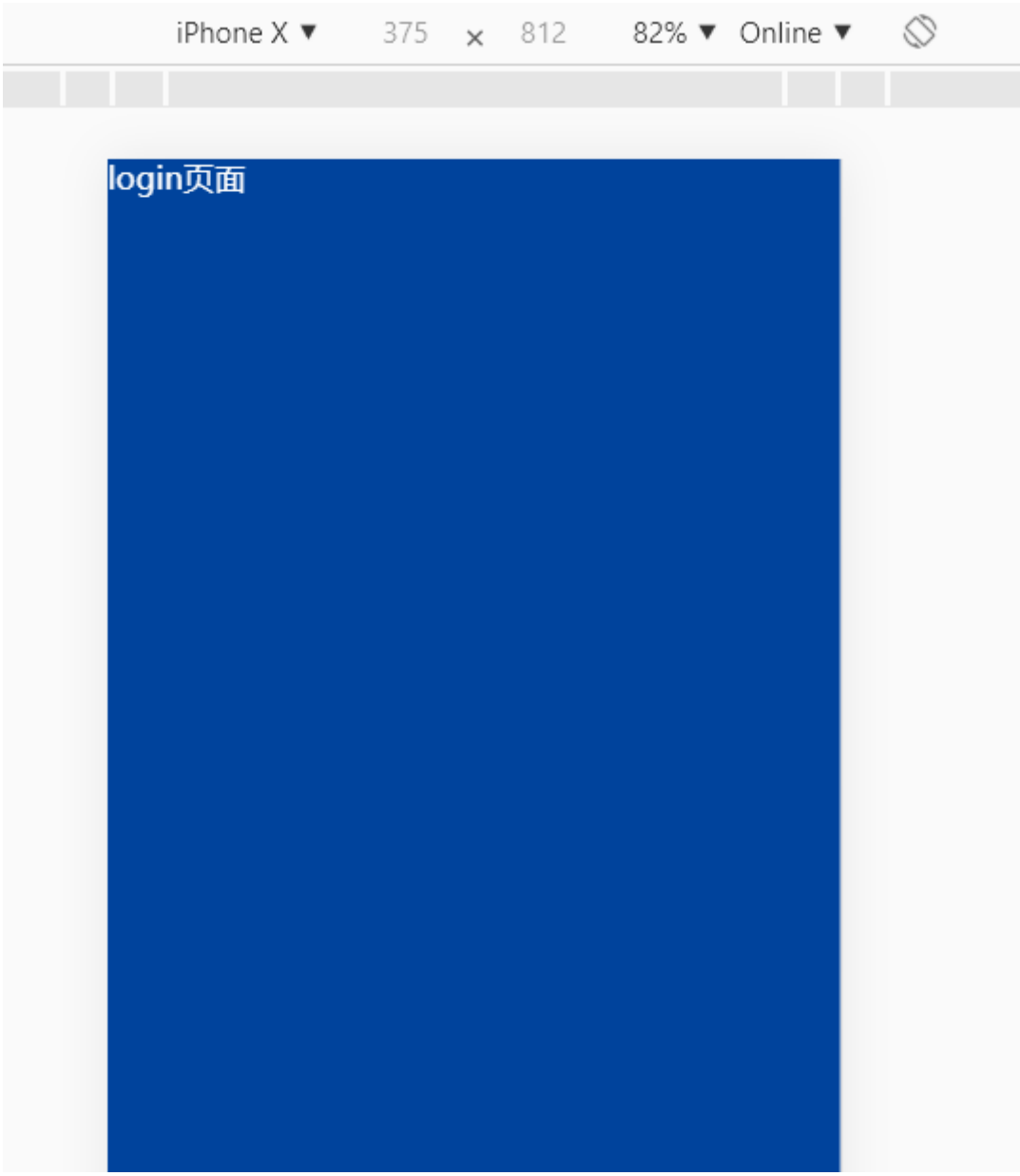
在src/assets/styles/core.less中：

```
@charset 'utf-8';

html,body,#root{ height: 100%;}
```

在src/assets/styles/loginPage中, 添加高度样式：

```
@charset 'utf-8';

.login-page{
    height: 100%;
    color:#fff;
    background-color: #00439c;
}
```

设置后的页面：



三十二、logo图片的完成

LoginPage.js中：

```
import Img from '../components/Img'

...

          <div className="login-page">
                {/* 1、图片 */}
                <Img src="logo.png" />

                {/* 2、表单域 */}
                <form>

                </form>
            </div>
```

Img图片组件代码：

```
import React, { Component } from 'react'

export default class Img extends Component {
    render() {
        return (
            <div className="img-div">
                <img src={require("../assets/images/"+this.props.src)} alt=""/>
            </div>
        )
    }
}
## 需要在webpack.config.js中做如下定义：
{
    test: [/\.bmp$/, /\.gif$/, /\.jpe?g$/, /\.png$/],
    loader: require.resolve('url-loader'),
    options: {
        limit: imageInlineSizeLimit,
        name: 'static/media/[name].[hash:8].[ext]',
        esModule: false
    },
},
```

loginPage.less中：

```
@charset "utf-8";

.login-page{
    /*外观*/
    height:100%;
    color: #fff;
    background-color: #00439c;
    /*布局*/
    overflow: hidden;

    .img-div{
```

```
        /*外观*/
        width: 145px;
        /*布局*/
        margin: 100px auto 0;

    }
}
```

三十三、表单组件的编写

LoginPage.js中:

```
import FormInput from '../components/FormInput'

                {/* 2、表单域 */}
                <form className="login-form">
                    {/* 用户名 */}
                    <FormInput/>
                    {/* 密码 */}
                    <FormInput/>
                    {/* 登录 */}
                    {/* 忘记密码 */}
                    {/* 免费注册、游客登录 */}
                </form>
```

接下来需要编写FormInput组件，在src/components目录下新建FormInput.js：

```
import React, { Component } from 'react'
import '../assets/styles/formInput.less'

export default class FormInput extends Component {
    render() {
        return (
            <div className="input-group">
                {/* 左侧图标 */}
                <i>图</i>
                {/* 右侧输入框 */}
                <input type="text"/>
            </div>
        )
    }
}
```

在src/assets/styles目录下新建formInput.less:

```
@charset 'utf-8';

.input-group{
    width: 100%;
    height: 40px;
    border: 1px solid #fff;
    border-radius: 6px;
```

```
    display: flex;  // 弹性布局
    margin-bottom: 20px;

    i{
        width: 40px;
        height: 40px;
        line-height: 40px;
        text-align: center;
    }

    input{
        color:#fff;
        outline: none;   //去掉聚焦框
        background: transparent;  //背景透明
        border:0 none;
        flex:1;
    }
}
```

三十四、字体图标

把字体文件和iconfont.css文件拷贝到src/assets/fonts/iconfont目录中：

在FormInput.js组件中，引入样式，添加两个类 iconfont icon-shouji：

```
import '../assets/fonts/iconfont/iconfont.css'

                {/* 左侧图标 */}
                <i className="iconfont icon-shouji"></i>
```

在formInput.less中设置字体图标大小：

```
    i{
        width: 40px;
        line-height: 40px;
        font-size:26px;
    }
```

三十五、组件复用

在FormInput.js组件中，设置两个需要替换成props属性，icon类名和input类型：

```
            <div className="input-group">
                 {/* 左侧图标 */}
                 <i className={"iconfont icon-"+this.props.iconClass}></i>
                 {/* 右侧输入框 */}
                 <input type={this.props.type}/>
             </div>
```

在LoginPage.js中，为两个组件设置属性：

```
                <form className="login-form">
                     {/* 用户名 */}
                     <FormInput iconClass="shouji" type="text"/>
                     {/* 密码 */}
                     <FormInput iconClass="mima" type="password"/>
                     {/* 登录 */}
                     {/* 忘记密码 */}
                     {/* 免费注册、游客登录 */}
                 </form>
```

设置placeholder样式：

```
input::-webkit-input-placeholder { /* WebKit browsers 适配谷歌 */
    color: #fff;
    opacity: 0.6;
}
```

三十六、按钮组件——登录按钮

LoginPage.js中：

```
            <form className="login-form">

                ...

                {/* 登录 */}
                <FormBtn />
                {/* 忘记密码 */}
                {/* 免费注册、游客登录 */}
            </form>
```

FormBtn.js中：

```
import React, { Component } from 'react'
import '../assets/styles/formBtn.less'

export default class FormInput extends Component {
    render() {
        return (
            <button className="form-btn">登录</button>
        )
    }
}
```

formBtn.less中：

```
@charset "utf-8";

.form-btn{
    display: block;
    width: 100%;
    height: 40px;
    border: 1px solid #fff;
    border-radius: 6px;
    color:#fff;
    background: transparent;
}
```

三十七、按钮组件——忘记密码

LoginPage.js中：

```
                    {/* 登录 */}
                     <FormBtn>登录</FormBtn>
                     {/* 忘记密码 */}
                     <FormBtn type="ordinary">忘记密码</FormBtn>
```

FormBtn.js中：

```
   render() {
        return (
            this.props.type === "ordinary"?
            <p className="form-btn-p">{this.props.children}</p>:
            <button className="form-btn">{this.props.children}</button>
        )
    }
```

formBtn.less中：

```
.form-btn-p{
    text-align: left;
}
```

三十八、按钮组件——免费注册、游客登录

LoginPage.js中：

```
                    {/* 登录 */}
                     <FormBtn block={true}>登录</FormBtn>
                     {/* 忘记密码 */}
                     <FormBtn type="ordinary">忘记密码</FormBtn>
                     {/* 免费注册、游客登录 */}
                     <FormBtn>免费注册</FormBtn> 
                     <FormBtn>游客登录</FormBtn>
```

FormBtn.js中：

```
    render() {
         let is_block_class = "";
         this.props.block? is_block_class=" block":is_block_class="";
         return (

             this.props.type === "ordinary"?
             <p className="form-btn-p">{this.props.children}</p>:
             <button className={"form-btn" + is_block_class}>
{this.props.children}</button>
         )
     }
```

formBtn.less中重新改写.form-btn类：

```
.form-btn{
    height: 40px;
    border: 1px solid #fff;
    border-radius: 6px;
    color:#fff;
    background: transparent;

    &.block{   //同时具有form-btn和block类名的样式
        width: 100%;

        display: block;

    }

}
```

三十九、Antd-Mobile介绍

常用的reactUI组件例如：antd-design、bootstrap、material、element-ui

Ant Design Mobile 一个基于 Preact / React / React Native 的 UI 组件库

`antd-mobile` 是 Ant Design 的移动规范的 React 实现，服务于蚂蚁及口碑无线业务

ant-design官网：https://ant.design/index-cn

antd-mobile官网：https://mobile.ant.design/index-cn

antd-mobile组件文档：https://mobile.ant.design/docs/react/introduce-cn

四十、项目集成Antd-Mobile

1、安装antd-mobile：

项目目录下： (如果没有安装yarn的话，先安装npm install -g yarn)

```
yarn add antd-mobile    或者   npm install antd-mobile
```

以下2到4步参考antd-mobile组件文档的首页来完成：

2、在pubilc/index.html中引入 FastClick, 在head中插入：

```
<script
src="https://as.alipayobjects.com/g/component/fastclick/1.0.6/fastclick.js">
</script>
<script>
    if ('addEventListener' in document) {
        document.addEventListener('DOMContentLoaded', function() {
            FastClick.attach(document.body);
        }, false);
    }
    if(!window.Promise) {
        document.writeln('<script
src="https://as.alipayobjects.com/g/component/es6-promise/3.2.2/es6-
promise.min.js"'+'>'+'<'+'/'+'script>');
    }
</script>
```

3、src目录下新建test文件夹，test下新建Test1Button.js文件

```
import React from 'react'  //快捷键imr
import ReactDOM from 'react-dom' //快捷键imrd

import { Button } from 'antd-mobile';
ReactDOM.render(<Button>Start</Button>, document.getElementById('root'));
```

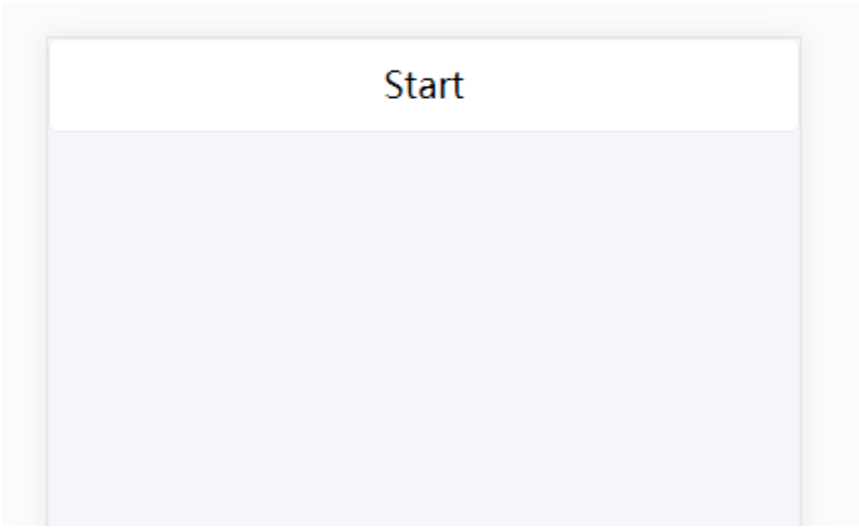
4、src/index.js中，注释以前的顶部组件，换成测试用组件导入，及其样式

```
// 引入顶级组件(后期的 路由组件)
// import './pages/App';

//测试ant mobile的集成
import './test/Text1Button'

//样式的引入
import 'normalize.css'
import './assets/styles/core.less'

//各种引入
//ant mobile样式引入
import 'antd-mobile/dist/antd-mobile.css';
```

5、如果出现警告：[Intervention] Unable to preventDefault inside passive event listener due to target being treated as passive. See https://www.chromestatus.com/features/5093566007214080

则在core.less文件中添加修复样式：

```
body{ touch-action: pan-y; }
```



四十一、头部组件——Logo

在src/pages下新建HomePage.js

```
import React, { Component } from 'react'
import Header from "../components/Header"

// 2、引入外部样式
import '../assets/styles/header.less'


// 3、声明并且导出组件类
export default class LoginPage extends Component {

    render() {
        return (
            <div>
                <Header />
            </div>
        )
    }
}
```

在src/components中新建Header.js （引用antd-mobile的NavBar导航栏组件）

```
import React, { Component } from 'react'
import { NavBar, Icon } from 'antd-mobile';
import Img from "../components/Img"
import "../assets/styles/header.less"

export default class Header extends Component {
    render() {
        return (
```

```
            <div className="header">
                <NavBar

                    leftContent={[
                        <Img key="0" src="logo.png" alt=""/>,
                        // <Icon key="0" type="search" style={{ marginRight:
'16px' }} />,
                        // <Icon key="1" type="ellipsis" />,
                    ]}
                    // mode="light"
                    // icon={<Icon type="left" />}
                    // onLeftClick={() => console.log('onLeftClick')}
                    rightContent={[
                        <Icon key="0" type="search" style={{ marginRight: '16px'
}} />,
                        <Icon key="1" type="ellipsis" />,
                    ]}
                >NavBar</NavBar>

            </div>
        )
    }
}
```

在src/styles下新建global.less

```
@charset "utf-8";

/*设置全局基色*/
@base-color:#00439c;
```

在src/styles下新建header.less

```
@charset "utf-8";
@import './global.less';

.header{
    .am-navbar{
        background-color: @base-color;
    }
    .img-div img{
        height: 30px;
    }
}
```

四十二、头部组件——搜索框

Header.js 中： （引用antd-mobile的SearchBar搜索栏组件）

```
import React, { Component } from 'react'
import { NavBar, SearchBar } from 'antd-mobile';
import Img from "../components/Img"
import "../assets/styles/header.less"

export default class Header extends Component {
```

```
    render() {
        return (
            <div className="header">
                <NavBar
                    leftContent={[
                        <Img key="0" src="logo.png" alt=""/>,
                    ]}
                ><SearchBar placeholder="搜你想搜的" /></NavBar>
            </div>
        )
    }
}
```

header.less的.header类中，继续添加样式：

```
.header{
    ...

    .am-navbar-right,.am-navbar-left{
        flex:0;
    }

    .am-search{
        flex:1;
        background-color: @base-color;

        .am-search-input{  /*form下的div元素*/
            flex:0 1 auto;
        }

        .am-search-cancel{   /* “取消” 的文字样式*/
            color:#fff;
        }
    }
}
```

四十三、banner

HomePage.js中

```
import Banner from "../components/Banner"
            <div>
                {/* 页头 */}
                <Header />
                {/* 广告位Banner */}
                <Banner />
            </div>
```

components中新建Banner.js（引用antd-mobile的Carousel 走马灯组件）

```
import React from 'react'
```

```
import { Carousel, WingBlank } from 'antd-mobile';
// 【1】、引入banner图片文件
import Banner1 from "../assets/images/banner1.png"
import Banner2 from "../assets/images/banner2.png"
import Banner3 from "../assets/images/banner3.png"
import "../assets/styles/banner.less"

export default class Banner extends React.Component {
  state = {
    data: ['1', '2', '3'],
    imgHeight: 176,
  }
  componentDidMount() {
    // simulate img loading
    setTimeout(() => {
      this.setState({
        //  【2】 、使用banner图片文件
        data: [Banner1, Banner2, Banner3],
      });
    }, 100);
  }
  render() {
    return (
      <div className="banner">
          <WingBlank>
        <Carousel
          autoplay={false}
          infinite
          beforeChange={(from, to) => console.log(`slide from ${from} to
${to}`)}
          afterChange={index => console.log('slide to', index)}
        >
          {this.state.data.map(val => (
            <a
              key={val}
              href="###"
              style={{ display: 'inline-block', width: '100%', height:
this.state.imgHeight }}
            >
              <img
                // src={`https://zos.alipayobjects.com/rmsportal/${val}.png`}
                // 【3】 、指定val值
                src={val}
                alt=""
                style={{ width: '100%', verticalAlign: 'top' }}
                onLoad={() => {
                  // fire window resize event to change height
                  window.dispatchEvent(new Event('resize'));
                  this.setState({ imgHeight: 'auto' });
                }}
              />
            </a>
          ))}
        </Carousel>
      </WingBlank>
      </div>
    );
  }
```

```
}
```

assets/style中新建banner.less

```
@charset 'utf-8';

.banner{
    .am-wingblank.am-wingblank-lg{
        margin: 0;
    }
}
```

四十四、学科导航

HomePage.js中

```
import Subject from "../components/Subject"
          <div>
               {/* 页头 */}
               <Header />
               {/* 广告位Banner */}
               <Banner />
               {/* 学科导航 */}
               <Subject />
           </div>
```

components中新建Subject.js中 （引用antd-mobile的Flex布局）

```
import React, { Component } from 'react'
import { Flex } from 'antd-mobile';
import "../assets/styles/subject.less"

export default class Subject extends Component {
    render() {
        return (
            <div class="subject">
                <Flex>
                    <Flex.Item>
                        <a href="###">
                            <i style={{backgroundPositionX:0}}></i>
                            <p>Jave EE</p>
                        </a>
                    </Flex.Item>
                    <Flex.Item>
                        <a href="###">
                            <i style={{backgroundPositionX:-50}}></i>
                            <p>全栈UI设计</p>
                        </a>
                    </Flex.Item>
                    <Flex.Item>
                        <a href="###">
                            <i style={{backgroundPositionX:-100}}></i>
```

```
                            <p>H5前端</p>
                        </a>
                    </Flex.Item>
                    <Flex.Item>
                        <a href="###">
                            <i style={{backgroundPositionX:-150}}></i>
                            <p>Python</p>
                        </a>
                    </Flex.Item>
                </Flex>
                <Flex>
                    <Flex.Item>
                        <a href="###">
                            <i style={{backgroundPositionX:-200}}></i>
                            <p>iOS</p>
                        </a>
                    </Flex.Item>
                    <Flex.Item>
                        <a href="###">
                            <i style={{backgroundPositionX:-250}}></i>
                            <p>大数据</p>
                        </a>
                    </Flex.Item>
                    <Flex.Item>
                        <a href="###">
                            <i style={{backgroundPositionX:-300}}></i>
                            <p>C++</p>
                        </a>
                    </Flex.Item>
                    <Flex.Item></Flex.Item>
                </Flex>
            </div>
        )
    }
}
```

assets/style中新建subject.less

```
@charset "utf-8";

.subject{
    background-color: #fff;
    padding-bottom: 15px;

    a{
        display: block;
        padding-top: 15px;
        text-align-last:center;
        i{
            margin: 0 auto;
            width: 50px;
            height: 56px;
            background-color: #ccc;
            display: block;
            background: url("../images/subjectIcons.png");
            background-size: 350px;
```

```
        }
        p{
            text-align: center;
            margin-bottom: 0;
        }
    }

}
```

四十五、小列表

HomePage.js中，准备数据：

```
const sub_list_data = [
    {
        img: 't0110974b6f051c1e5a.gif',
        title: '用于构建用户界面的 JavaScript 库',
        des: '组件逻辑使用 JavaScript 编写而非模版，你可以轻松地在应用中传递数据，并使得状态与 DOM 分离',
    },
    {
        img: 't0177ec8627c04e438b.gif',
        title: '渐进式的JavaScript 框架',
        des: '不断繁荣的生态系统，可以在一个库和一套完整框架之间自如伸缩',
    },
    {
        img: 't0135a6567622271196.gif',
        title: '一套框架多种平台 移动端&桌面端',
        des: '学会用 Angular 构建应用，把这些代码和能力复用在多种不同平台的应用上',
    },
    {
        img: 't0110974b6f051c1e5a.gif',
        title: '用于构建用户界面的 JavaScript 库',
        des: '组件逻辑使用 JavaScript 编写而非模版，你可以轻松地在应用中传递数据，并使得状态与 DOM 分离',
    },
    {
        img: 't0177ec8627c04e438b.gif',
        title: '渐进式的JavaScript 框架',
        des: '不断繁荣的生态系统，可以在一个库和一套完整框架之间自如伸缩',
    }
];
```

HomePage.js中，jsx结构： (引用antd-mobile的Flex布局)

```
import React, { Component } from 'react'
import { Flex } from 'antd-mobile'
import Img from '../components/Img'
import Header from "../components/Header"
import Banner from "../components/Banner"
import Subject from "../components/Subject"
import '../assets/styles/header.less'
```

```
import '../assets/styles/subList.less'

const sub_list_data = [
    {
        img: 't0110974b6f051c1e5a.gif',
        title: '用于构建用户界面的 JavaScript 库',
        des: '组件逻辑使用 JavaScript 编写而非模版，你可以轻松地在应用中传递数据，并使得状态与 DOM 分离',
    },
    {
        img: 't0177ec8627c04e438b.gif',
        title: '渐进式的JavaScript 框架',
        des: '不断繁荣的生态系统，可以在一个库和一套完整框架之间自如伸缩',
    },
    {
        img: 't0135a6567622271196.gif',
        title: '一套框架多种平台 移动端&桌面端',
        des: '学会用 Angular 构建应用，把这些代码和能力复用在多种不同平台的应用上',
    },
    {
        img: 't0110974b6f051c1e5a.gif',
        title: '用于构建用户界面的 JavaScript 库',
        des: '组件逻辑使用 JavaScript 编写而非模版，你可以轻松地在应用中传递数据，并使得状态与 DOM 分离',
    },
    {
        img: 't0177ec8627c04e438b.gif',
        title: '渐进式的JavaScript 框架',
        des: '不断繁荣的生态系统，可以在一个库和一套完整框架之间自如伸缩',
    }
];

export default class HomePage extends Component {
    constructor(props){
        super(props)
        this.state = {
            subList:sub_list_data
        };
    }
    render() {
        return (
            <div className="home-page">
                <Header />
                <Banner />
                <Subject />
                {/* 小列表 */}
                <div className="home-sub-list">
                    <div className="list-header">相关资讯</div>

                    {
                        this.state.subList.map((obj, k)=>(
                            <div className="list-item" key={k}>
                                <Flex>
                                    <Flex.Item className="list-item-left">
                                        <Img alt="" src={obj.img}/>
                                    </Flex.Item>
                                    <Flex.Item className="list-item-left">
                                        <h3>{obj.title}</h3>
```

```
                                    <p>{obj.des}</p>
                                </Flex.Item>
                            </Flex>
                        </div>
                    ))
                }
            </div>
        </div>
    )
  }
}
```

subList.less中

```
@charset 'utf-8';

.home-page{

    background-color: #fff;

    .home-sub-list{
        h3,p{
            margin: 0;
            padding: 0;
        }
        padding: 0 15px 0;
        border-top: 1px solid #ddd;

        .list-header{
            padding: 15px 0 6px;
            font-size:20px;
            border-bottom: 1px solid #ddd;
        }

        .list-item{
            border-bottom: 1px solid #ddd;
            padding: 12px 0;
        }

        .list-item-left{
            width: 80px;
            flex:0 1 auto;
            margin-right: 5px;

            img{
                width: 80px;
                display: block;
            }
        }

        h3{
            font-size: 16px;
            margin-bottom: 10px;
            overflow: hidden;
```

```
            text-overflow: ellipsis;    /*表示文本超出时候用 “...” 来代替*/
            display: -webkit-box;
            -webkit-box-orient: vertical;
            -webkit-line-clamp: 1;
        }
        p{
            line-height: 1.666em;
            overflow: hidden;
            text-overflow: ellipsis;    /*表示文本超出时候用 “...” 来代替*/
            display: -webkit-box;
            -webkit-box-orient: vertical;
            -webkit-line-clamp: 2;
        }
    }
}
```

四十六、标签页

HomePage.js中 ： （引用antd-mobile的Flex布局 Tabs 标签页 和 Item列表 ）

```
                <div className="home-tab">
                    <Tabs tabs={tabs} initialPage={0} animated={false} useOnPan=
{false}>
                    <div style={{ backgroundColor: '#fff' }}>
                        <Flex className="home-tab-item">
                            <Flex.Item className="home-tab-item-left">
                                <Img src="20191025091907842.gif" />
                            </Flex.Item>
                            <Flex.Item className="home-tab-item-right">
                                <h3>
                                传三星计划为Galaxy S11配置可卷曲显示屏和可旋转摄像头
                                </h3>
                                <div className="txt_info">
                                    <i className="iconfont icon-shijian"></i>
                                    <span>2小时前</span>
                                    <div className="right">
                                        <i className="iconfont icon-liulan"></i>
                                        <span>63</span>
                                    </div>
                                </div>
                            </Flex.Item>
                        </Flex>
                        <Item arrow="horizontal" onClick={() => {}}>小米有品推出小
寻定位书包 拥有8重定位系统</Item>
                        <Item arrow="horizontal" onClick={() => {}}>三星S11系列国行
型号曝光 具备90Hz刷新率</Item>
                        <Item arrow="horizontal" onClick={() => {}}>华为多屏协同技
术机型公布 共15款</Item>
                        <Item arrow="horizontal" onClick={() => {}}>40英寸红米电视
曝光 售价可能在千元以内</Item>
                        <Item arrow="horizontal" onClick={() => {}}>华为手环4发布
售价199元</Item>
                    </div>
                    <div style={{ backgroundColor: '#fff' }}>
                        <Flex className="home-tab-item">
```

```
                            <Flex.Item className="home-tab-item-left">
                                <Img src="20191025104209413.gif" />
                            </Flex.Item>
                            <Flex.Item className="home-tab-item-right">
                                <h3>
                                传丰田计划与铃木合作在印度推出紧凑型电动汽车
                                </h3>
                                <div className="txt_info">
                                    <i className="iconfont icon-shijian"></i>
                                    <span>4小时前</span>
                                    <div className="right">
                                        <i className="iconfont icon-liulan"></i>
                                        <span>85</span>
                                    </div>
                                </div>
                            </Flex.Item>
                        </Flex>
                        <Item arrow="horizontal" onClick={() => {}}>传丰田计划与铃木合作在印度推出紧凑型电动汽车</Item>
                        <Item arrow="horizontal" onClick={() => {}}>外媒：欧盟考虑对自动驾驶政策松绑，特斯拉Autopilot有望上市</Item>
                        <Item arrow="horizontal" onClick={() => {}}>GoFun出行：已经在全国54个城市实现盈利</Item>
                        <Item arrow="horizontal" onClick={() => {}}>丰田e-4me概念车亮相2019东京车展 科技感十足</Item>
                        <Item arrow="horizontal" onClick={() => {}}>外媒：2020年第一季度结束，特斯拉上海工厂单月产能可达1.4万辆</Item>
                    </div>
                    <div style={{ backgroundColor: '#fff' }}>
                        <Flex className="home-tab-item">
                            <Flex.Item className="home-tab-item-left">
                                <Img src="20191025092136621.gif" />
                            </Flex.Item>
                            <Flex.Item className="home-tab-item-right">
                                <h3>
                                恒大新能源汽车全球研究总院落户上海 许家印出席活动
                                </h3>
                                <div className="txt_info">
                                    <i className="iconfont icon-shijian"></i>
                                    <span>3小时前</span>
                                    <div className="right">
                                        <i className="iconfont icon-liulan"></i>
                                        <span>13</span>
                                    </div>
                                </div>
                            </Flex.Item>
                        </Flex>
                        <Item arrow="horizontal" onClick={() => {}}>手机+AIOT“双引擎”持续赋能小米各业务线</Item>
                        <Item arrow="horizontal" onClick={() => {}}>BluePrism进军中国市场 开启RPA行业新赛道</Item>
                        <Item arrow="horizontal" onClick={() => {}}>金融科技迎转型关键期，向前金服迭代升级智能风控平台“听风者”</Item>
                        <Item arrow="horizontal" onClick={() => {}}>周鸿祎：必须以作战视角看待网络安全</Item>
                        <Item arrow="horizontal" onClick={() => {}}>开启人工智能绚烂新世界 微软小冰年度大会发布Avatar Framework</Item>
                    </div>
```

```
                    </Tabs>
                </div>
```

homePage.less中：

```
.home-tab{
    margin-top: 5px;
    .am-list-item .am-list-line{
        padding-right: 0;
        margin-right: 15px;
    }
    .am-tabs-default-bar-tab-active{
        color:@base-color;
    }
    .am-tabs-default-bar-underline{
        border-color:@base-color;
    }
    .am-tabs-default-bar-tab{
        font-size:18px;
    }

    .home-tab-item{
        margin: 10px 15px 5px;
        .home-tab-item-left{
            flex:0 1 auto;
            width: 120px;
            img{
                width: 120px;
                display: block;
            }
        }

        .home-tab-item-right{
            h3{
                font-size:16px;
                line-height:1.4em;
                margin-top: 0;
                overflow: hidden;
                text-overflow: ellipsis;
                display: -webkit-box;
                -webkit-box-orient: vertical;
                -webkit-line-clamp: 2;
            }
            .txt_info{
                color:#ccc;
                .right{
                    float: right;
                }
                i{
                    margin-right: 6px;
                }
                span{
                    color:#aaa;
                }
            }

        }
```

```
    }

}
```

九、页脚(header组件的复用)

HomePage.js中：

```
<Header isFooter={true}/>
```

Header.js组件中：

```
            <div className="header">
                 <NavBar
                 mode="light"
                 leftContent={this.props.isFooter?"":[

                     <img key="0" src={logoImg} alt=".." className="header-
logo"/>
                 ]}

                 >
                     {
                         this.props.isFooter?
                         <div style={{flex:"1",textAlign:"right"}}><img key="0"
src={logoImg} alt=".." className="header-logo" /></div>:
                         <SearchBar placeholder="搜你想搜的"  />
                     }
                 </NavBar>
             </div>
```

最后企业文字：

```
                 {/* 页脚 */}
                 <Header isFooter={true}/>
                 <div className="footer">
                     <p>© 2018-2019 ICP备案：粤ICP备17147191号</p>
                     <p>广州狼码教育科技有限公司</p>
                     <p>办公电话：020-85628002</p>
                     <p>地址：广州市天河区棠下大地商务港D栋603</p>
                 </div>
```

一、列表页ListView组件基本使用

列表页使用组件：

https://mobile.ant.design/components/list-view-cn/#components-list-view-demo-basic-body

在pages中新建ListPage.js：

```
import React, { Component } from 'react'
import Header from "../components/Header"
import MyListView from "../components/MyListView"


// 3、声明并且导出组件类
export default class ListPage extends Component {

    render() {
        return (
            <div>
                <Header />
                <MyListView />
            </div>
        )
    }
}
```

在components中新建MyListView.js:

```
import React, { Component } from 'react'
import { ListView } from 'antd-mobile';

const data = [
    {
        img: 'tO110974b6f051c1e5a.gif',
        title: '用于构建用户界面的 JavaScript 库',
        des: 'React 使创建交互式 UI 变得轻而易举',
    },
    {
        img: 'tO177ec8627c04e438b.gif',
        title: '小米新品发布会定档 MIX新品亮相',
        des: 'Yeelight智能LED灯丝灯发布：复古设计 亮度可调',
    },
    {
        img: 'tO135a6567622271196.gif',
        title: 'iPhone11开始出货 本周五发售',
        des: '首批售卖的iPhone 11、iPhone 11 Pro和iPhone 11 Pro Max已经从工厂送出',
    },
    {
        img: 'tO110974b6f051c1e5a.gif',
        title: '用于构建用户界面的 JavaScript 库',
        des: 'React 使创建交互式 UI 变得轻而易举',
    },
    {
        img: 'tO177ec8627c04e438b.gif',
        title: '小米新品发布会定档 MIX新品亮相',
        des: 'Yeelight智能LED灯丝灯发布：复古设计 亮度可调',
    },
    {
        img: 'tO135a6567622271196.gif',
        title: 'iPhone11开始出货 本周五发售',
        des: '首批售卖的iPhone 11、iPhone 11 Pro和iPhone 11 Pro Max已经从工厂送出',
    },
];

export default class MyListView extends Component {
```

```
    constructor(props) {
        super(props);
        //定义数据源
        const dataSource = new ListView.DataSource({
          rowHasChanged: (row1, row2) => row1 !== row2,  // 设置行数据变化的规则
        });

        //定义初始状态
        this.state = {
          dataSource,
          isLoading: true,
        };
      }

      componentDidMount() {
        // 模拟ajax请求获取数据
        setTimeout(() => {
            // this.rData = genData();
            this.setState({
              dataSource: this.state.dataSource.cloneWithRows(data),  //把数据对象
克隆到listview数据源中
              isLoading: false,   //加载状态
            });
          }, 600);
      }


    renderRow(rowData){
        //每一行的渲染函数

        return <div>{rowData.title}</div>
    }

    render() {


        return (
            <ListView
                dataSource={this.state.dataSource}
                renderRow={this.renderRow}
                // renderHeader={() => <span>header</span>}
                // renderFooter={() => (<div style={{ padding: 30, textAlign:
'center' }}>
                // {this.state.isLoading ? 'Loading...' : 'Loaded'}
                // </div>)}

                // renderSeparator={separator}
                className="am-list"
                // pageSize={4}
                useBodyScroll
                // onScroll={() => { console.log('scroll'); }}
                // scrollRenderAheadDistance={500}
                // 滚动到底部执行onEndReached方法
                // onEndReached={this.onEndReached}
                // onEndReachedThreshold={10}
            />
        )
    }
```

```
}
```

ps:我们自己的数据：

```
const data = [
    {
        img: 't0110974b6f051c1e5a.gif',
        title: '用于构建用户界面的 JavaScript 库',
        des: 'React 使创建交互式 UI 变得轻而易举',
    },
    {
        img: 't0177ec8627c04e438b.gif',
        title: '小米新品发布会定档 MIX新品亮相',
        des: 'Yeelight智能LED灯丝灯发布：复古设计 亮度可调',
    },
    {
        img: 't0135a6567622271196.gif',
        title: 'iPhone11开始出货 本周五发售',
        des: '首批售卖的iPhone 11、iPhone 11 Pro和iPhone 11 Pro Max已经从工厂送出',
    },
];
```

二、列表每一项布局的书写

从上面我们已经知道randerRow函数，就是每一条数据的渲染。而这部分样式布局刚好和我们首页相关资讯一致。所以：

2.1、先抽取首页循环return的部分作为组件

2.1.1、在components文件夹中新建 ListItem.js 组件:

```
import React, { Component } from 'react'
import Img from "./Img"
import { Flex } from 'antd-mobile';
export default class ListItem extends Component {
    render() {
        return (

            <div className="sub_list_item">
                <Flex>
                    <Flex.Item className="sub_list_left">
                        <Img src={this.props.imgSrc} alt=""/>
                    </Flex.Item>
                    <Flex.Item className="sub_list_right">
                        <h3>{this.props.title}</h3>
                        <p>{this.props.des}</p>
                    </Flex.Item>
                </Flex>
            </div>

        )
    }
}
```

HomePage.js中修改为：

```
                {/* 小列表(相关资讯) */}
                 <div className="sub_list">
                     <div className="sub_list_header">相关资讯</div>

                     {
                         this.state.sub_list_data.map((obj, key)=>{

                             return <div key = {key}>
                                 <ListItem
                                         imgSrc={obj.img}
                                         title={obj.title}
                                         des={obj.des}
                                     />
                             </div>
                         })
                     }
                 </div>
```

2.1.2、抽取homePage.less中该部分样式到global.less并封装成 “函数” 可以 “调用”

global.less中增加代码：

```
.list-item(){
    .sub_list{

        border-top: 1px solid #ddd;
        padding: 0 15px;
        h3,p{margin: 0;}
        .sub_list_header{
            padding: 15px 0 6px;
            font-size: 20px;
            border-bottom: 1px solid #ddd;

        }
        .sub_list_item{
            padding: 12px 0;
            border-bottom: 1px solid #ddd;

        }
        .img-div img{
            width: 80px;
            display: block;
        }

        .sub_list_left{
            flex: 0 1 auto;
            margin-right: 5px;

        }

        .sub_list_right{
```

```
            h3{
                margin-bottom: 10px;
                font-size: 16px;
                .max-line(1);
            }
            p{
                line-height: 1.666em;
                .max-line(2);
            }
        }
    }

}
```

homePage.less中：

```
.home-page{

    background-color: #fff;

    .list-item();

    .home-tab{
        ...
    }
}
```

以上必须确保抽取后不能影响首页原有的样式内容

2.2、列表页MyListView组件的randerRow函数中的书写

ListPage.js中：

```
            <div className="list-page">
                <Header />
                <div className="sub_list">
                    <div className="sub_list_header">相关资讯</div>

                    <MyListView />
                </div>

             </div>
```

randerRow函数中：

```
renderRow(rowData){
    //每一行的渲染函数

    return <ListItem
                imgSrc={rowData.img}
                title={rowData.title}
                des={rowData.des}
                />
}
```

listPage.less中：

```
@import 'global.less';

.list-page{
    background-color: #fff;
    color:#333;

    .list-item()
}
```

三、滑动到底部加载更多

components的 MyListView.js中：

修改componentDidMount() 中的setTimeout() ：

```
componentDidMount() {

        setTimeout(() => {
            this.rData = genData();
            this.setState({
                dataSource: this.state.dataSource.cloneWithRows(this.rData),
                isLoading: false,
            });


        }, 600);
      }
```

全局中添加genData函数：

```
const NUM_ROWS = 10; //每次加载多少条数据
let pageIndex = 0;

function genData(pIndex = 0) {
  //循环处理数据，返回数据对象
  const dataBlob = {};
  for (let i = 0; i < NUM_ROWS; i++) {
    const ii = (pIndex * NUM_ROWS) + i;
    dataBlob[`${ii}`] = `row - ${ii}`;
  }
  return dataBlob;
}
```

class MyListView中添加onEndReached函数：

```
onEndReached = (event) => {
        // load new data
        // hasMore: from backend data, indicates whether it is the last page,
here is false
        if (this.state.isLoading && !this.state.hasMore) {
          return;
```

```
        }
        console.log('reach end', event);
        this.setState({ isLoading: true });
        setTimeout(() => {
          this.rData = { ...this.rData, ...genData(++pageIndex) };
          this.setState({
            dataSource: this.state.dataSource.cloneWithRows(this.rData),
            isLoading: false,
          });
        }, 1000);
      }
```

class MyListView中的render函数中添加row常量，并且

组件标签：

```
render() {
        let index = data.length - 1;
        const row = (rowData, sectionID, rowID) => {
            if (index < 0) {
                index = data.length - 1;
            }
            const obj = data[index--];
            return (
                <ListItem
                        imgSrc={obj.img}
                        title={obj.title}
                        des={obj.des}
                    />
            )
        }
        return (
            <ListView
                dataSource={this.state.dataSource}
                renderRow={row}
                // renderHeader={() => <span>header</span>}
                renderFooter={() => (<div style={{ padding: 30, textAlign:
'center' }}>
                {this.state.isLoading ? '加载中...' : '加载完成'}
                </div>)}

                // renderSeparator={separator}
                className="am-list"
                // pageSize={4}
                useBodyScroll
                // onScroll={() => { console.log('scroll'); }}
                // scrollRenderAheadDistance={500}
                // 滚动到底部执行onEndReached方法
                onEndReached={this.onEndReached}
                // onEndReachedThreshold={10}
            />
        )
    }
```

去掉 randerRow函数

四、详情页搭建

新建详情页 DetailPage.js：

```
import React, { Component } from 'react'
import { NavBar, Icon } from 'antd-mobile';
import "../assets/styles/detailPage.less"
import LogoImg from "../assets/images/logo.png"


export default class DetailPage extends Component {
    render() {
        return (
            <div className="detail-page">
                <NavBar
                    mode="light"
                    icon={<Icon type="left" />}
                    onLeftClick={() => window.history.go(-1)}
                ><img src={LogoImg} style={{height:30}} alt="" /></NavBar>

        </div>
        )
    }
}
```

新建样式文件 detailPage.less

```
@import "./global.less";

.detail-page{
    .am-navbar{
        background-color: @base-color;
    }
    .am-navbar-light{
        color:#fff;
    }
}
```

五、详情页文章

```
            <article>
                    <h1>React 用于构建用户界面的 JavaScript 库</h1>
                    <p>
                        <span>作者：<i>天上的鱼</i></span>
                        <span style={{float:"right"}}>2019-11-01</span>
                    </p>
                    <Img src="dt.gif" />
                    <content>
                        <p>
                        Props、State的概念都很清晰，组件的普通属性是指在组件中直接挂载到this下的属性。其实，Props和State也是组件的两个普通属性，因为我们可以通过this.props 和 this.state 直接获取到。那么Props、State 和 组件的其他普通属性，分别应该在什么场景下使用呢？
</p>
```

```
                        <p>Props和State都是用于组件渲染的，也就是说，一个组件最终长成什么样，取决于这个组件的Props和State。Props和State的变化都会触发组件的render方法。但这两者也是有区别的。Props是只读的数据，它是由父组件传递过来的；而State是组件内部自己维护的状态，是可变的。State可以根据Props的变化而变化。如果组件中还需要其他属性，而这个属性又与组件的渲染无关（也就是render方法中不会用到），那么就可以把这个属性直接挂在到this下，而不是作为组件的一个状态。</p>
                        <p>例如，组件中需要一个定时器，每隔几秒改变一下组件的状态，就可以定义一个this.timer属性，以备在componentWillUnmount时，清除定时器。</p>
                        <p>父组件每次render方法被调用，或者组件自己每次调用setState方法，都会触发组件的render方法（前提是shouldComponentUpdate使用默认行为，总是返回true）。那么组件每次render，是不是都会导致实体DOM的重新创建呢？答案是，不是！</p>
                        <p>React之所以比直接操作DOM的JS库快，原因是React在实体DOM之上，抽象出一层虚拟DOM，render方法执行后，得到的是虚拟DOM，React 会把组将当前的虚拟DOM结构和前一次的虚拟DOM结构做比较，只有存在差异性，React才会把差异的内容同步到实体DOM上。如果两次render后的虚拟DOM结构保持一致，并不会触发实体DOM的修改。</p>
                    </content>
                    <p>
                        <i className="iconfont icon-liulan"></i> <span>6325</span> 次观看
                        <div style={{float:"right"}}><i className="iconfont icon-jubao"></i> 我要举报</div>
                    </p>


                </article>
```

文章样式 detailPage.less中：

```
@import "./global.less";

.detail-page{
    background-color: #fff;
    .am-navbar{
        background-color: @base-color;
    }
    .am-navbar-light{
        color:#fff;
    }
    article{
        padding: 15px;
        h1{
            margin-top: 0;
            font-size: 22px;
            text-align: center;
        }
        img{
            width: 100%;
        }
        content{
            text-indent: 2em;
            line-height: 1.666em;
        }
    }
}
```

六、相关资讯小列表

DetailPage.js中：

```
// 准备数据：


...
    constructor(props){
        super(props)

        this.state = {
            sub_list_data: sub_list_data
        }
    }


...

               <div className="sub_list">
                    <div className="sub_list_header">相关资讯</div>

                    {
                        this.state.sub_list_data.map((obj, key)=>{

                            return <div key = {key}>
                                <ListItem
                                        imgSrc={obj.img}
                                        title={obj.title}
                                        des={obj.des}
                                    />
                            </div>
                        })
                    }
                </div>
```

detailPage.less中：

```
.detail-page{
    ...
    .list-item();

}
```

七、热门评论

DetailPage.js中：

```
               <div className="comment">
                    <div className="comment-header">热门评论</div>
                    <Flex  className="comment-box">
                        <Flex.Item className="box-l">
                            <div className ="avatar-box"><Img src="avatar.jpg"/>
</div>
                        </Flex.Item>
```

```
                        <Flex.Item className="box-r">
                            <p>
                                <strong>爱学习的孩子</strong>
                                <span style={{float:"right"}}>3 <i
classNameName="iconfont icon-dianzan"></i></span>
                            </p>
                            <p className="comment-content">
                            我们应该在组件的哪一个生命周期方法中发送网络请求呢？
                            </p>
                        </Flex.Item>
                    </Flex>

                </div>
```

detailPage.less中：

```
.comment{
        padding:5px 15px;
        .comment-header{
            padding: 15px 0 6px;
            font-size: 20px;
            border-bottom: 1px solid #ddd;

        }

        .comment-box{
            .box-l{
                margin-right: 5px;
                flex:0 1 auto;
                width: 50px;
                height: 50px;
                .avatar-box,img{
                    width: 50px;
                    height: 50px;
                }
                .avatar-box{
                    border-radius:50%;
                    overflow: hidden;

                }

            }

            .comment-content{
                line-height: 1.4em;
            }
        }
    }
```

八、路由配置

项目目录下安装路由：

```
yarn add react-router@3.2.0
```

把App.js修改为：

```
import React, { Component } from 'react'
import ReactDOM from 'react-dom'
import {Router, Route, hashHistory} from 'react-router'

import LoginPage from "./LoginPage"
import HomePage from "./HomePage"
import ListPage from "./ListPage"
import DetailPage from "./DetailPage"


const router2 = <Router history={hashHistory}>

                    <Route path="/" component={LoginPage}/>
                    <Route path="/home" component={HomePage}/>
                    <Route path="/list" component={ListPage}/>
                    <Route path="/detail" component={DetailPage}/>

                </Router>

ReactDOM.render(router2, document.getElementById('root'));
```

再在需要跳转的位置加上a标签(或者Link)：

例如：首页跳转到列表页，在Subject.js中：

```
                    <Flex.Item>
                         <a href="#/list">
                             <i style={{backgroundPositionX:-50}}></i>
                             <p>全栈UI设计</p>
                         </a>
                     </Flex.Item>
                     <Flex.Item>
                         <a href="#/list">
                             <i style={{backgroundPositionX:-100}}></i>
                             <p>H5前端</p>
                         </a>
                     </Flex.Item>
```

一、表单组件值的获取（登录页面）

在LoginPage.js中，我们使用了两个FormInput 组件，我们需要获取这两个FormInput 中input的value值(即用户输入的值)。

在React中，我们要获取组件中表单元素的值，就需要在父组件中操作子组件的数据，这些操作都需要在父组件中去完成。

因此，获取用户名，密码的这些操作，也都会在LoginPage 这个组件中来完成。

接下来开始获取：

把FormInput 做成**受控组件**：

需要给FormInput 组件**添加value和onChange**属性，

其中，value的值交给LoginPage 组件的state数据来控制，

onChange的事件函数中需要修改state数据为用户输入的值。

注意：这两个属性都是FormInput 组件的，为了让它们起作用，要传递到FormInput 的内部中的input元素里面去。

LoginPage.js组件中：

```
export default class LoginPage extends Component {
    constructor(props){
        super(props)

        this.state = {
            username:"",
            password:""
        }
        this.handleChange = this.handleChange.bind(this)
        this.handleChange2 = this.handleChange2.bind(this)
    }
    handleChange(e){
        this.setState({
            username:e.target.value
        })
    }
    handleChange2(e){
        this.setState({
            password:e.target.value
        })
    }


    ...
    ...

                {/* 用户名 */}
                    <FormInput iconClass="icon-shouji" type="text" value=
{this.state.username} onChange={this.handleChange}/>
                    {/* 密码 */}
                    <FormInput  iconClass="icon-mima"  type="password" value=
{this.state.password} onChange={this.handleChange2}/>
```

FormInput.js组件中：

```
                {/* input输入框 */}
                {/* <input type={this.props.type} value={this.props.value}
onChange={this.props.onChange}/> */}
                <input  {...this.props}/>
```

这样就能做到在LoginPage中控制FormInput组件的值。

二、在项目中使用axios完成请求

安装axios：

```
yarn add axios
```

导入：

```
import axios from "axios"
```

下面实现登录按钮的点击发起请求操作：

去掉点击按钮的Link标签，为他添加点击事件

FormInput.js组件中：

```
<button className={"form-btn "+ has_box_class} onClick={this.props.onClick}>
{this.props.children}</button>
```

LoginPage.js组件中：

```
this.handleClick = this.handleClick.bind(this)

...

handleClick(e){
    e.preventDefault();
    console.log("我们将要在这里完成ajax请求。")
}

...
<FormBtn box={true} onClick={this.handleClick}>登录</FormBtn>
```

在handleClick中：

```
    e.preventDefault();
    let params = {
        username:this.state.uname,
        password:this.state.upwd
    }

    axios.get("/server/data.json",params)
        .then((resp)=>{
            if(resp.data.errno =="0"){
                // console.log(resp.data);
                Toast.success(resp.data.errmsg, 1.5);
                this.props.router.push("/home")
            }else{
                Toast.fail(resp.data.errmsg, 1.5);
                // alert(resp.data.errmsg)
            }

        })
        .catch((err)=>{
            console.log(err)
        })
```

如果是前后端分离项目，可以在package.json中设置代理：

```
"proxy": "http://localhost:3003"
// 设置之后要重新启动服务器
```

三、首页学科导航的数据请求

server目录下准备学科数据 subject.json：

```
[
    {
        "id":"1",
        "subjectName":"Java EE"
    },
    {
        "id":"2",
        "subjectName":"全栈UI设计"
    },
    {
        "id":"3",
        "subjectName":"H5前端"
    },
    {
        "id":"4",
        "subjectName":"Python"
    },
    {
        "id":"5",
        "subjectName":"iOS"
    },
    {
        "id":"6",
        "subjectName":"大数据"
    },
    {
        "id":"7",
        "subjectName":"C++"
    }
]
```

在Subject.js组件中：

```
    constructor(props){
        super(props)

        this.state = {
            subject_data:[]
        }
    }
    componentDidMount() {
        axios.get("/server/subject.json")
        .then((resp)=>{
            console.log(resp.data)
            this.setState({
                subject_data : resp.data
            })

        })
    }
render() {
        return (
```

```
            <div className="subject">
                <Flex>
                    {
                        this.state.subject_data.map((v,k)=>{
                            if(k<4){
                                return (
                                    <Flex.Item key={k}>
                                    <a href={"#/list/"+v.id}>
                                        <i style=
{{backgroundPositionX:-50*v.id}}></i>
                                        <p>{v.subjectName}</p>
                                    </a>
                                </Flex.Item>
                                )
                            }
                        })

                    }
                </Flex>
                <Flex>
                    {
                        this.state.subject_data.map((v,k)=>{
                            if(k>=4){
                                return (
                                    <Flex.Item key={k}>
                                    <a href={"#/list/"+v.id}>
                                        <i style=
{{backgroundPositionX:-50*v.id}}></i>
                                        <p>{v.subjectName}</p>
                                    </a>
                                </Flex.Item>
                                )
                            }
                        })

                    }
                    <Flex.Item>

                    </Flex.Item>
                </Flex>
            </div>
        )
    }
```

四、列表页对应路由的调整

既然在上一个步骤中我们的 list路由路径变成 #/list/id值 的格式，那么就需要在对应的路由接收参数

在App.js的路由配置中：

```
<Route path="/list/:subjectId" component={ListPage} />
```

然后在ListPage.js列表页中，挂载完组件后，就可以做ajax请求对应的学科数据列表：

```
    componentDidMount() {
        let subjectId = this.props.routeParams.subjectId
        console.log(subjectId)
        //拿到subjectId之后就拿着这个id去做列表页数据的请求
        // axios.get(`/server/list.json? subjectId=${subjectId}`)
        // .then((resp)=>{
        //     ...
        // })
    }
```

五、Redux介绍(了解)

Redux 是 JavaScript 状态容器，提供项目中的状态管理。

redux的三个重要概念：

store: 数据仓库

action: 组件的动作名和动作的定义（决定了如何修改仓库数据）

dispatch: 执行相应的动作action （决定了何时修改仓库数据）

Redux中提供createStore方法用于生成一个store对象，这个函数接受一个初始值state值和一个reducer函数。当用户发出相应的action时，利用传入的reducer函数计算出一个新的state值，并返回。

使用Redux，我们只获取一次数据并将其存储在一个中心位置，称为 store。然后，任何组件都可以随时使用这些数据。这就像附近有一家超市，我们的厨师可以在那里买到所有的食材。这家超市派卡车从农场大批运回蔬菜和肉类。这比让个别厨师亲自去农场效率高得多。

store 还是唯一的数据源。组件通常从 store 中获取数据，而不是其他地方。这使得 UI 保持高度统一。

注意：

Redux 不只是为 React 而生。一个常见的误解是 Redux 仅用于 React。 听起来Redux在没有React的情况下无法做任何事情。 事实上，正如我们之前所讨论的，Redux在几个重要方面补充了React。 React是最最常见的 Redux 用例。
然而，事实上，Redux可以使用任何前端框架，如Angular、Ember.js 甚至jQuery 或者 普通的JavaScript。

下面看一个redux代码：

（前提：先安装redux　**yarn add redux** 或者 **npm install redux**）

```
/*
   redux小案例:

*/
import React, {Component} from 'react'
import ReactDOM from 'react-dom'

import {createStore} from 'redux'



// 请一个仓库管理员，必须是一个函数
const reducer = (state, action) => {
    console.log("执行了reducer函数")
    console.log(state, action)
```

```
    // 7、回到reducer函数，深拷贝action对象到newState，并返回
    //(此时只是reducer中的参数state发生了变化，而视图组件中的state没有改变，第8步将处理组件中的state)
    if(action.type === "up"){
        let newState = JSON.parse(JSON.stringify(state))
        newState.num1 = action.value
        return newState
    }
    // 1、定义初始state
    return {
        num1: 20
    }
}

//创建一个仓库， 把仓库管理员请来管理这个仓库
// 2、创建store仓库
const store = createStore(reducer)


export default class App extends Component {
    constructor(props){
        super(props)

        // console.log(store.getState())
        // 3、在constructor中，获取初始state
        this.state = store.getState()
        this.changeNumUp = this.changeNumUp.bind(this)

        // 8、store订阅，一旦store数据发生改变，则执行storeChange函数里面的代码
        this.storeChange = this.storeChange.bind(this)
        store.subscribe(this.storeChange)
    }

    render() {
        return (
            <div >
                {/* 4、 书写组件，填入数据 */}
                <p>{this.state.num1} </p>
                {/* 5、业务：点击让num1自增 */}
                <button onClick={this.changeNumUp}>增加</button>
            </div>
        )
    }

    changeNumUp(e){
        // 6、改变数据的时候 需要调用store的dispatch方法，把新的值作为放在对象中传进去
        // 每次调用dispatch，会在内部调用 图书管理员函数reducer
        const action = {
            type:'up',
            value:this.state.num1+1
        }
        store.dispatch(action)  // 每次调用dispatch，会在内部调用 图书管理员函数reducer
    }

    storeChange(e){
        this.setState(store.getState())
    }
```

```
}


ReactDOM.render(
    <App />
    , document.getElementById('root'));
```

六、React-Redux

为了方便使用，Redux 的作者封装了一个 React 专用的库 React-Redux。这个库是可以选用的。实际项目中，你应该权衡一下，是直接使用 Redux，还是使用 React-Redux。后者虽然提供了便利，但是需要掌握额外的 API，并且要遵守它的组件拆分规范。

React-Redux 将所有组件分成两大类：UI 组件（presentational component）和容器组件（container component）。

6.1、UI组件

UI 组件特征：

- 只负责 UI 的呈现，不带有任何业务逻辑
- 没有状态（即不使用`this.state`这个变量）
- 所有数据都由参数（`this.props`）提供
- 不使用任何 Redux 的 API

6.2、容器组件

容器组件特征：

- 负责管理数据和业务逻辑，不负责 UI 的呈现
- 带有内部状态
- 使用 Redux 的 API

6.3、connect()

React-Redux 提供`connect`方法，用于从 UI 组件生成容器组件。`connect`的意思，就是将这两种组件连起来。

6.4、Provider组件

React-Redux 提供`Provider`组件，可以让容器组件拿到`state`状态数据。`Provider`包裹了原来项目的根组件。

案例见进入代码中react-redux文件夹

运行案例前先安装react-redux: **yarn add react-redux 或者 npm install react-redux**

七、项目中使用react-redux

在App.js中：

```
import {createStore} from 'redux'
import { Provider } from 'react-redux'

const defaultState = {
```

```
    subject_data: []
}

// 请一个仓库管理员，必须是一个函数
const reducer = (state = defaultState, action) => {
    if(action.type === "get_subject_data"){
        let newState = JSON.parse(JSON.stringify(state))
        newState.subject_data = action.value
        return newState
    }
    return state
}
//创建一个仓库， 把仓库管理员请来管理这个仓库
const store = createStore(reducer)


const App = (

    <Provider store={store}>
        <Router history={hashHistory}>
            <Route path="/" component={LoginPage} />
            <Route path="/home" component={HomePage} />
            <Route path="/list/:subjectId" component={ListPage} />
            <Route path="/detail" component={DetailPage} />
        </Router>
    </Provider>
)
ReactDOM.render(App, document.getElementById('root'));
```

Subject.js组件中

```
class Subject extends Component {
    // constructor(props){
    //     super(props)

    //     this.state = {
    //         subject_data:[]
    //     }
    // }
    componentDidMount() {
        axios.get("/server/subject.json")
        .then((resp)=>{
            console.log(resp.data)
            this.props.init_sub_data(resp.data)

        })


    }
    render() {
        return (
            <div className="subject">
                <Flex>
                    {
                        this.props.subject_data.map((v,k)=>{
                            if(k<4){
                                return (
```

```
                                    <Flex.Item key={k}>
                                    <a href={"#/list/"+v.id}>
                                        <i style=
{{backgroundPositionX:-50*v.id}}></i>
                                        <p>{v.subjectName}</p>
                                    </a>
                                </Flex.Item>
                                )
                            }
                        })

                    }
                </Flex>
                <Flex>
                    {
                        this.props.subject_data.map((v,k)=>{
                            if(k>=4){
                                return (
                                    <Flex.Item key={k}>
                                    <a href={"#/list/"+v.id}>
                                        <i style=
{{backgroundPositionX:-50*v.id}}></i>
                                        <p>{v.subjectName}</p>
                                    </a>
                                </Flex.Item>
                                )
                            }
                        })

                    }
                    <Flex.Item>

                    </Flex.Item>
                </Flex>
            </div>
        )
    }
}

const mapStateToProps = (state) => {
    return {
        subject_data: state.subject_data,
    }
}


const mapDispatchToProps = (dispatch) => {

    return {

        init_sub_data(resp_data){
            console.log(resp_data)

                let action = {
                    type:'get_subject_data',
                    value:resp_data

                }
```

```
                dispatch(action)
            }
        }
    }

// connect(展示数据的函数,改变数据的函数)(组件类名)
export default connect(mapStateToProps,mapDispatchToProps)(Subject)
```

八、项目部署

8.1、购买服务器

- 选择云服务器:阿里云服务器 https://www.aliyun.com
- 个人免费获取 [https://free.aliyun.com/]

购买之后在首页（右上角）=》控制台 （左边）=》实例，就可以看到我们买到的服务器

8.2、配置服务器

接下来做服务器的配置：

8.3、远程连接到服务器

后续操作：

打开终断输入：

ssh root@公网ip地址
键入密码

看到root@xxxxxxxxxxxxxxxxxxxxxxx:~#　　表示登录成功

8.4、部署项目需要的环境

1、在终端执行以下命令，将自动进行nvm的安装：
root@iZwccccvxw0wcddjrvoeZ:~# wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash

注意：　安装完成后直接关闭终断。打开终断通过ssh重新登录

2、使用nvm安装node版本
root@iZwccccvxw0wcddjrvoeZ:~# NVM_NODEJS_ORG_MIRROR=http://nodejs.org/dist nvm install v10.15.0

3、修改npm为国内镜像
root@iZwccccvxw0wcddjrvoeZ:~# npm config set registry "http://registry.npm.taobao.org/"

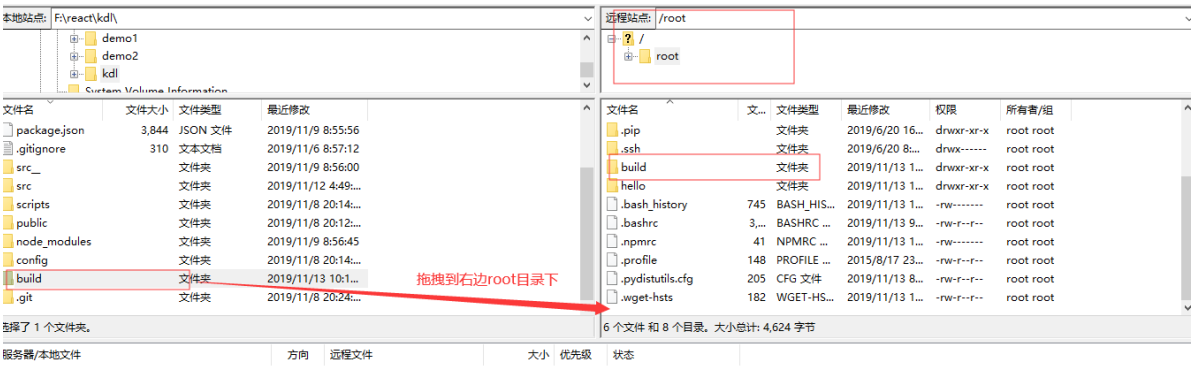
4、安装serve
root@iZwcccccvxw0wcddjrvoeZ:~# npm install -g serve

8.5、上传项目

在本地项目目录下 进行 yarn build 产生一个 build文件夹。

我们要把这个build文件夹上传到服务器上

打开FileZilla之后，按Ctrl+S打开站点管理器



等待上传成功即可

8.6、启动项目

将build文件上传到Linux家目录下，运行程序：

root@iZwcccccvxw0wcddjrvoeZ:~# serve -s ./build