**React day01**

面试题：

1)前端三大框架的区别有哪些？

2)前端可用的异步请求方案有哪些？

|  |
| --- |
| 响应消息的Content-Type：  text/html  text/css  application/javascript  application/json  image/jpeg  audio/mpeg3  ..... |
| 请求消息的Content-Type：  text/plain：请求数据是未经编码的普通文本  application/x-www-form-urlencoded：请求数据是经过编码的数据  multipart/form-data：请求数据包含文件上传内容  application/json：请求数据是JSON格式的字符串 |

1.React框架概述

官网：https://reactjs.org/ 最新版V16.10

中文网：https://zh-hans.reactjs.org/

中文社区网：https://react-1251415695.cos-website.ap-chengdu.myqcloud.com/

由Facebook维护的MVVM框架，官方定义自己是一个“用于构建UI的JS库”——核心概念很少；但生态圈扩展非常广泛，如

React-Router：路由、

Redux： 状态保持、

ReactVR/360： 虚拟现实、

ReactNative：使用JS做App原生开发...

|  |
| --- |
| 项目中使用Vue.js中的方法：  ①SCRIPT标签直接引入  ②脚手架方式 |
| 项目中使用Angular中的方法：  ②脚手架方式 |
| 项目中使用React中的方法：  ①SCRIPT标签直接引入  ②脚手架方式 |

2.HTML元素属性(Attribute) vs JS DOM对象的属性(Property)

任何一个HTML元素都对应一个JSDOM对象，有两套属性系统

|  |  |
| --- | --- |
| **HTML元素属性** | **JS DOM对象的属性** |
| Attribute | Property |
| <img src="" id="" title=""> | img.src = ""  img.id = ""  img.title = "" |
| <img class=""> | img.className = "" |
| <label for=""> | label.htmlFor = "" |
| <td colspan=""> | 没有对应项 |
| 没有对应项 | div.innerHTML = ""  div.innerText = "" |

3.使用SCRIPT方式引入React

<div id="box"></div>

<script src="js/react.js"></script> 提供React对象

<script src="js/react-dom.js"></script> 提供ReactDOM对象

<script>

let el = React.createElement(标签名, 属性列表, 内容/子元素)

ReactDOM.render( el, box )

</script>

练习：使用React在#box中添加如下的DOM结构：

<div class="login">

<label for="uname">用户名</label>

<input type="text" id="uname"></input>

</div>

4.JSX —— 简化元素的创建过程 (取代React.createElement)

JavaScript XML**：形式上是XML，本质是JS对象**——可以看做是一种JS的变种（类似于TS）——浏览器无法理解JSX语法，必须使用编译器转化为JS才能被浏览器执行

JSX语法规范：

①JSX不是字符串！最外侧不能有引号！

②JSX形式上不是HTML语法，而是XML语法；故<br>必须写作<br/>；属性值必须用引号；有且只能有一个根元素

③JSX中可以使用HTML标签，但严格区分大小写——任何HTML标签都必须全小写，自定义组件名必须使用大驼峰法则，如<MyHeader></MyHeader>

④因为JSX中都是JS对象，所以属性都要使用DOM属性，而不是HTML属性，例如：<p className="..">

⑤JSX片段中还可以出现JSX表达式： { }

⑥JSX中的注释为： {/\* .... \*/}

练习：使用React在#box中添加如下的DOM结构——使用JSX语法代替React.create( )

<div class="login">

<label for="uname">用户名</label>

<input type="text" id="uname">

</div>

|  |
| --- |
| Babel：是一个第三方提供的JS变种编译器，可以把ES/TS/JSX转化为标准的JS代码，使用方法：  <script src="js/babel.js"></script>  <script type="text/babel">  //此处的代码将由Babel编译器处理  </script> |

5.JSX中的{ }表达式

提示：在JSX中，还可以使用{ }进行数据绑定或者运算，可以使用在两种情形下：

① <p>{表达式}</p> —— React中的内容绑定

② <p title={表达式}></p> —— React中的属性绑定！注意与Angular中的不同

测试：JSX表达式中可以出现哪些语句：

1)算术运算：可以

2)比较运算：可以，但true/false都不显示

3)逻辑运算：可以，但true/false都不显示

4)三目运算：可以

5)调用函数：可以，对象成员函数或全局函数或匿名自调函数都可以

6)创建对象：只要是可以转换为字符串的对象就可以——Angular中完全禁止

7)调用全局对象方法(JSON.stringify())：可以——Angular中不可以

6.React中如何实现各种数据绑定

①内容绑定：<p>{内容}</p>

②属性绑定：<img src={imgUrl} title={myTitle} />

③指令绑定：React中没有指令的概念

④事件绑定：

⑤双向数据绑定：

注意：因为React中没有指令，所以类似于ngFor/ngIf的实现有些特别——很重要！具体见手册“核心概念>第7/8章”

|  |
| --- |
| **React中的条件渲染：**  function check(){  if() return JSX  else return JSX  }  let element = <div>{ check( ) }</div> |
| **React中的列表渲染：**  注意：{ 数组 } 绑定语句会把数组中的每个元素展开添加为当前元素的子节点  let list = [....]  let element = <ul>  {  list.map((e, i)=>{  return <li key={i}>{e}</li>  })  }  </ul> |

练习：创建一个变量表示用户当前的积分，若积分大于0，可以显示一个视频播放区(<video>)，否则显示一段警告消息(<p>积分不足无法观看</p>)

练习：创建一个数组，保存如下员工信息：

[

{eid: 101, ename: 'dingding', salary: 5000, sex: 1},

{eid: 102, ename: 'doudou, salary: 6000, sex: 1},

{eid: 103, ename: 'yaya', salary: 7000, sex: 0},

]

把上述数据渲染在一个TABLE中，每个员工信息对应一个TR，内部包含四个TD

7.React中的自定义组件

概念：组件是一段可以反复使用的页面片段

组件 = 模板 + 脚本(数据) + 样式

React提供了两种自定义组件的方法：

**①function式组件——一般只用于创建简单组件(只有模板)**

function MyC01( ){

return ( JSX )

}

<MyC01></MyC01>

**②class式组件——适合于创建复杂组件(模板+脚本数据)**

class MyC02 extends React.Component {

render( ){

return ( JSX )

}

}

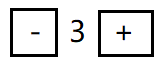
<MyC02></MyC02>

提示：React中要求组件名必须使用“大驼峰命名法”！

课后任务：

①阅读React官网手册“核心概念”第1~8章内容，学习事件绑定的方法和模型数据的声明方法——大约需要2小时

②创建自定义的class组件BuyCount，其中的模板内容如下：



要求实现点击减号按钮购买数量-1，点击加号按钮购买数量+1

**Day02**

复习:

见思维导图

面试题:三大框架中数据绑定实现上有何绑定?

1.React的第二种使用方法 —— 脚手架方式

①前提: Node.js > 8.10

②下载全局脚手架工具

npm i -g create-react-app

③运行全局脚手架工具,创建空白项目

create-react-app 项目名

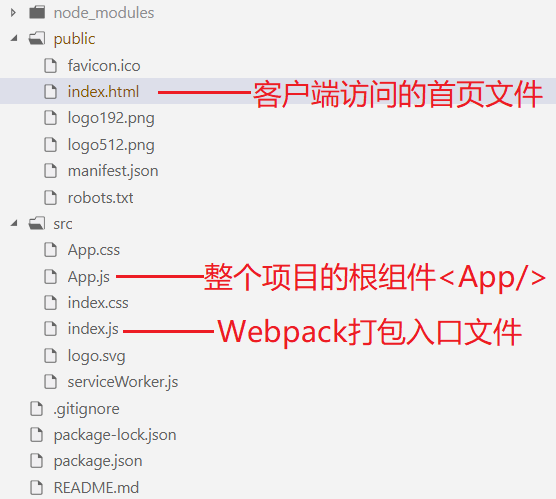
④进入空白项目,运行开发服务器

cd 项目名

npm start

⑤客户端访问测试

http://127.0.0.1:3000



练习: 创建自定义组件 MyC01BuyCount.js (购物计数器), 在根组件App.js中使用该组件.

2.React中的数据绑定

①内容绑定 —— <p>{表达式}</p>

②属性绑定 —— <p title={表达式}></p>

③指令绑定 —— React中没有指令

④事件绑定 —— 与其它框架区别较大！

⑤双向数据绑定 ——与其它框架区别较大！

React中的事件句柄不同于HTML中的事件句柄(handler)：

onclick —— onClick

ondblclick —— onDoubleClick

onmouseover —— onMouseOver

onblur —— onBlur

......

|  |  |  |
| --- | --- | --- |
| Vue.js中的事件绑定  <p @click="f2">  <p @click="f2()"> | Angular中的事件绑定  <p (click)="f2()"> | React中的事件绑定  <p onClick={this.f2}> |

总结：React中事件绑定需要注意：

①事件句柄名称必须用“小驼峰命名法”

②事件处理函数不能用引号

③事件处理方法前必须加 { }

④当前class内部的方法做事件处理方法，则必须加 this

⑤事件处理方法名后不能加 ( )

面试题1：React中事件处理方法中为什么this默认指向undefined？

JSX中，事件处理方法赋值时不是函数调用，而是函数的引用赋值 —— this指向不再指向当前组件对象！而是指向当前ES6模块中的全局this——值为undefined

面试题2：如何解决React事件处理方法中无法使用this的问题？——用三种方法

**①用匿名函数封装立即调用的函数**

f2( ){

console.log(this)

}

<p onClick={ ( )=>{ this.f2( ) } } >

**②用箭头匿名函数代替function匿名函数**

f2 = ( )=>{

console.log(this)

}

<p onClick={ this.f2 } >

**③在构造方法中固定this的指向**

constructor( ){

super( )

this.f2 = this.f2.bind( this )

}

f2( ){

}

<p onClick={ this.f2 } >

3.React中组件里的Model数据

React中class组件可以声明两种形式的Model数据：

**①class属性数据 —— 可以一次性绑定到模板，数据再变则不会再次修改视图模板**

class MyC02{

count =3

}

**②组件状态数据 —— 可以绑定到模板，未来Model变则View变**

使用步骤：

1)创建状态数据——只能在构造方法中

constructor(){

super()

this.state = { age: 20 }

}

2)绑定状态数据

<p>{this.state.age}</p>

3)修改状态数据

this.setState({

age: 21

})

注意：①setState()会修改虚拟DOM树 ②setState()不会清空已有的状态数据，只会把新修改的数据合并到已有数据中 ③setState()是异步操作，不会立即执行——底层会把连续的多次setState合并为一次操作，以减少虚拟DOM/真实DOM的修改；如果希望查看修改后的值，可以使用setState({ }, callback)中的回调函数 ④不推荐直接在setState中使用之前的this.state数值，使用临时变量代替

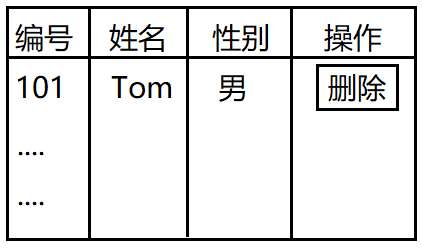
练习：创建一个组件MyC04，其中有状态数据：empList = [

{eid: 101, ename: 'Tom', sex:1},

{eid: 102, ename: 'Mary', sex:1},

{eid: 103, ename: 'Jerry', sex:1},

]，把这些数据绑定到一个TABLE中，点击删除可以删除该记录，如下所示：



4.React中的双向数据绑定

|  |  |  |
| --- | --- | --- |
| Vue.js中的双向数据绑定 | Angular中的双向数据绑定 | React中的双向数据绑定 |
| v-model | [(ngModel)] | 受控组件 |

提示：双向数据绑定只能用于“表单元素”，如input、textarea、select等

**方向1：Model => View(value)**

constructor(){

this.state = { uname: '' }

}

<input value={this.state.uname} />

至此，输入框中会呈现状态数据，但是无法再修改——受控组件

**方向2：View(value) => Model**

<input onChange={this.doChange}/>

doChange = (event)=>{

this.setState({

uname: event.target.value

})

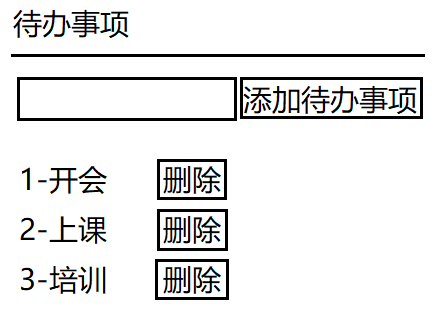
}

|  |
| --- |
| 受控组件：一旦将输入元素的value绑定为状态数据，用户在其中的输入都会被重置为状态数据的当前值（感觉像是只读输入框一样）—— 只有通过setState修改了状态数据，输入框的内容才能改变；  接下来只需要监听输入框的onChange事件，捕获到事件源当前的最新值(用户的最新输入)，修改为状态数据即可。 |

课后任务：

(1)总结比较三大框架中数据绑定、事件绑定、双向数据绑定实现上的异同

(2)完成“待办事项列表”应用：



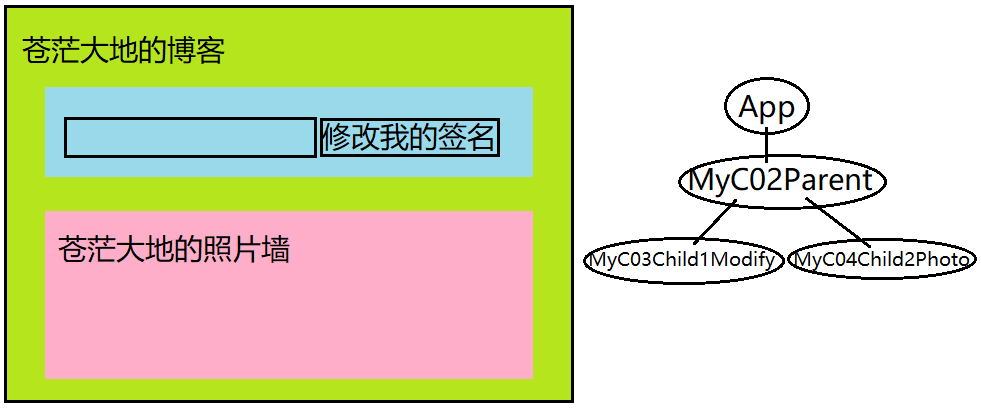
**Day 03**

复习：

见思维导图

|  |
| --- |
| Vue.js中父子组件数据传递：Props Down，Events Up  Angular中父子组件数据传递：Props Down，Events Up  React中父子组件数据传递：Props Down，**Props Up** |

1.React中父子组件数据传递



**父=>子：父亲通过子组件的自定义属性，把自己的数据传递下去**

Parent.js：

<Child myNameInChild={this.state.myNameInParent}/>

Child.js：

<h3>{**this.props.**myNameInChild}的照片墙</h3>

**子=>父：父亲通过子组件的自定义属性，把自己的方法传递下去；子组件体内调用此方法，传递实参**

Parent.js：

doModifyMyName=(newName)=>{ }

<Child modifyName={this.doModifyMyName}/>

Child.js：

this.props.modifyName('ABC')

|  |
| --- |
| “Props Up”的原理：  Parent p = new Parent()  p.doModifyMyName = function(newName){ ... }  Child c = new Child()  c.modifyMyName = p.doModifyMyName //父的方法传给子  c.modifyMyName('ABC') |

React中没有直接的兄弟间数据传递机制，只能借助于父组件：兄弟1 => 父组件 => 兄弟2

React中父组件想获得子组件的所有成员，有简便方法：

Parent.js：

constructor(){

super()

this.c3 = React.createRef( ) //创建子组件的引用

}

--------------------------------------

<h3/MyC03 ref={this.c3}> //子组件的引用绑定到某个孩子

--------------------------------------

this.c3.current 就是孩子对象

|  |
| --- |
| Vue.js中父组件获得子组件的引用：  <Child ref="c2" />  this.$refs.c2 |
| Angular中父组件获得子组件的引用：  <Child #c2/>  @ViewChild('c2', {static: true})  private child2 |
| React中父组件获得子组件的引用：  <Child ref={ } /> |

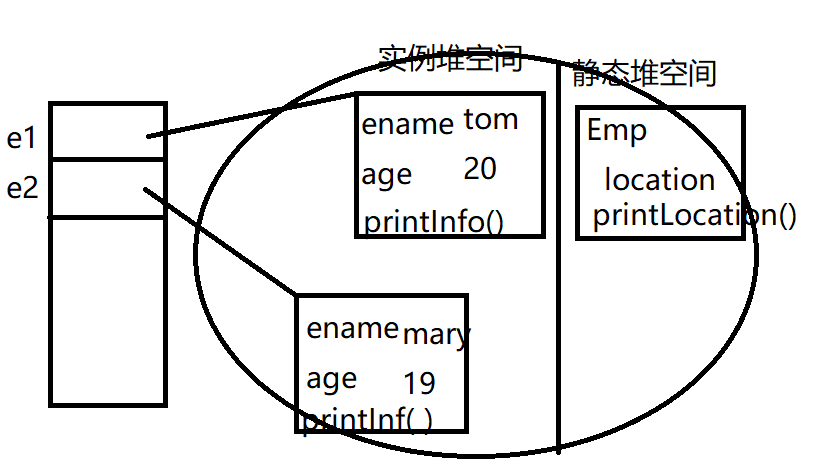
2.理论知识补充：class中的静态(static)成员

类(class)中的成员分为两大类：

①实例成员：实例属性、实例方法

②静态成员：静态属性、静态方法

概念：如果某个class的每一个实例都具备的成员，且值各不相同（例如员工的姓名、年龄），应该声明为实例成员。 如果某个class的所有实例都共用的成员，值在内存中只需保存一份（例如达内所有员工的所在地），应该声明为静态成员。



注意：一般情况下，class内部，静态属性通过静态方法来访问，实例成员通过实例方法来访问。class外部，实例成员需要通过对象的引用来访问；静态成员通过类名来访问，例如：Emp.location、 Emp.printLocation( )

**强化记忆：三大框架中，组件加载完成和即将卸载对应的钩子函数**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Vue.js** | **Angular** | **React** |
| 组件加载完成 | mounted(){ } | ngOnInit(){ } | componentDidMount(){ } |
| 组件即将卸载 | beforeDestroy(){ } | ngOnDestroy(){ } | componentWillUnmount(){ } |

3.面试题：React中组件的声明周期钩子函数 —— 比较凌乱

提示：React不同版本中生命周期钩子函数各不相同！React16.3和之前版本中的都不相同！

React组件的生命周期钩子函数分为三组：

**1)首次渲染相关函数**

①contstructor()

②getDerivedStateFromProps() 用于将this.props转为this.state

③render()

④componentDidMount() 用于初始化组件中的数据，如异步获取服务器端数据

**2)二次渲染相关函数(props属性更改、setState状态修改等)**

①getDerivedStateFromProps() 需要返回转换得到state对象或null

②shouldComponentUpdate() 需要返回true或false

③render()

④componentDidUpdate()

**3)组件卸载相关函数**

①componentWillUnmount() 用于销毁组件创建的长期存在数据，如定时器

4.复习：前端技术中创建移动App的技术有哪些？

①原生开发： Android下用Java/Kotlin，iOS下用OC/Swift

②H5/WebView：使用浏览器内核打开特定的页面 —— Vue.js/Mint-UI

③混编开发：H5/WebView+Cordova/PhoneGap —— Angular/Ionic

④JS Bridge：代码使用JS，使用桥转换，运行时是Java或OC —— React/RN

⑤Flutter：使用全新的Dart语言，直接在手机GPU上绘图

5.RN概述

中文网：https://reactnative.cn/

ReactNative：使用JS语法编写移动App应用，RN会把JS转换为底层Java或OC，最终运行于手机——完全不依赖于浏览器或者WebView组件！

搭建RN应用开发环境：

①下载并安装全局脚手架工具

npm i -g react-native-cli

②运行脚手架工具，创建出空白项目

react-native init 项目名

③下载并安装原生Android/iOS开发软件，创建客户端安装程序

安卓开发需要：Windows + Android Studio + Android SDK

iOS开发需要：Mac OS + XCode

具体过程参考：https://reactnative.cn/docs/getting-started/

最后在项目根目录下得到：android\app\build\outputs\apk\debug\app-debug.apk文件——AndroidApp的安装程序，需要把此APK文件安装到真实手机或者模拟器手机中

④进入空白项目，运行其中的开发服务器——PC机目前是服务器

cd 项目目录

npm start

⑤使用真实手机/模拟器访问测试——Android系统是客户端

Windows服务器的IP地址:8081

|  |
| --- |
| 查看Windows服务器的IP地址：  cmd > ipconfig |

课后任务：

①总结三大框架中的组件生命周期钩子函数

②总结三大框架中父子组件间数据传递的方法

③使用自己的Windows当RN服务器，真实安卓手机或者安卓模拟器(如夜神)当RN客户端，搭建RN开发环境，见到Welcome界面就算成功 —— 中间可能碰到各种错误，试着重新启动、重新安装一下

**Day 04**

复习：

WebView方案：Vue.js+MintUI

混编方案：Angular+Ionic

JSBridge方案：React+ReactNative

1.ReactNative概述

RN本质是一种JSBridge方案，代码使用JS/React来编写，RN会将这些代码转换为Android/Java或者iOS/OC，最后运行在手机端

优势：一套代码到处运行；运行效率高；功能丰富；

不足：原生Android和iOS默认组件效果不一致，非要相同只能深度定制

**提示：语法是React语法，采用组件化编程；可以使用的组件要么是用户自定义的组件，要么是RN预定义的组件，如<View>、<Text >、<Image>等，不能使用任何HTML标签组件——底层没有浏览器！**

官网：www.reactnative.com

中文网：reactnative.cn

搭建RN开发运行环境：

前提：

Windows系统(服务器) + Android手机或模拟器(客户端)

MacOS系统(服务器) + iOS手机或模拟器(客户端)

①下载并安装全局脚手架工具

npm i -g react-native-cli

②运行脚手架工具创建空白项目

react-native init 项目名

③在PC上下载并安装Android或iOS开发环境，编译当前项目得到APP安装程序包——详情参见手册

此步骤会生成：项目目录\android\app\build\outputs\apk\debug\app-debug.apk

电脑ip地址: window + r ipconfig

④进入空白项目并运行其中的开发服务器

cd 项目名

npm start

⑤在手机上安装APP安装包，远程访问PC服务器

|  |
| --- |
| 注意：开发服务器命令行界面千万不要用鼠标选中任何内容！如果选中了客户端手机APP无法再得到更新后的内容！！ |

2.RN开发常见的错误

①点击APP图标，立即弹出对话框“此应用程序已关闭”

原因：APP安装失败，或与当前手机不兼容；

可以卸载并重新安装；

不行就重启一下模拟器再试试；

再不行换一个其它版本的模拟器

②点击APP图标，一片空白

原因：APP启动失败了，或者无法从服务器获取更新后的内容；

保证服务器端处于运行状态，并且控制台没有用鼠标选中任何内容

关闭手机APP程序，重新启动；

③点APP图标，显示黑色底红色错误信息

原因：APP设置有问题，或者服务器端代码执行出错；

仔细查看红色错误信息，从中找到解决方案

常见APP红色错误信息：

|  |
| --- |
| Unable to load Script ...  原因：APP无法从服务器加载更新后的内容  解决办法：给APP重新设置服务器的主机名和端口号 176.204.11.17:8081  摇一摇 > Dev Settings > Debug Server host & port for device > 输入完成后记得重启APP即可 |
| Could not connect to development server  原因：APP不能连接到开发服务器；可能原因：①开发服务器没有启动 ②没有正确查找到服务器的IP地址（正确的地址应该形如 172.x.x.x或者192.168.x.x） ③APP没有正确的设置服务器IP和端口 |

3.ReactNative中的组件样式编写

提示：RN应用中没有浏览器，没有CSS解释器！RN样式有些名称有点类似CSS样式名称，但是本质完全不一样！！而且名称细节也不一样，例如不存在float、存在borderColor但是不存在border等。

①RN中的尺寸都是数字类型，不能赋值为字符串！也没有单位！如fontSize: 14

②RN中父元素的样式，不会继承给子元素——每个元素的样式都由自己的style完全控制，不需要叠加计算父元素的样式

**方法1：行内样式**

<Any style={ {color:'red', padding: 20} }>

**方法2：外部样式**

let ss = StyleSheet.create({

success: { color: 'red' }

})

<Any style={ ss.success }>

也可以给一个组件指定多个样式对象——多个对象组成的数组：

<Any style={ [ ss.success, ss.textRight, {fontSize:30} ] }>

|  |
| --- |
| CSS：Cascading StyleSheet，层叠/级联样式表，子元素可以继承父元素的样式 |
| RN SS： StyleSheet，样式表，子元素不会继承父元素的样式 |

4.ReactNative中的组件布局

提示：RN中的组件没有“块级元素”和“行内元素”之分；

所有元素的尺寸由width和height两个属性控制；

若没有指定宽和高，默认有布局系统来控制尺寸——一套很类似于CSS重点FlexBox布局系统。常用属性：

**①flex：指定当前元素在主轴上的尺寸占比**

number

**②flexDirection：子元素排列方向**

column：默认值，纵向排列

column-reverse：纵向排列（反向）

row：横向排列

row-reverse：横向排列（反向）

**③alignItems：子元素在交叉轴方向上的对齐方式**

flex-start：子元素对齐到容器的开始位置

flex-end：子元素对齐到容器的结束位置

center：子元素对齐到容器的中部

stretch：默认值，子元素在交叉轴方向上拉伸撑满容器

**④justifyContent：调整所有的子元素（即内容）在主轴上的分布方式**

flex-start：子元素码放到容器的主轴开始位置

flex-end：子元素码放到容器的主轴结束位置

center：子元素码放到容器的主轴中央

space-between：第一个孩子在开始，最后一个孩子在结尾，其它孩子之间平均分配空白空间

space-around：每个孩子上下都包裹着相同的空白空间——相邻两个子元素间的空白是开头和结尾空白的2倍

space-evenly：所有的空白在孩子之间以及上下顶部平均分配

练习：使用RN的布局系统实现如下内容：

flex、flexDirection、alignItems、justifyContent



分析：三行两列

<View grid>

<View row>

<View col>Text x 3</View>

<View col>Text x 3</View>

</View> x3

</View>

5.ReactNative提供的常用组件

**①View：**最简单的容器，默认没有高度，要靠内容撑起来，内容溢出后将不显示

<View style={ }>子组件</View>

注意：View中不能直接放置文本，文本只能放置在Text中

**②ScrollView：**可以滚动的容器，默认没有高度，要靠内容撑起来，内容溢出后将出现滚动条

<ScrollView style={ }>子组件</ScrollView>

**③Text：**显示单行或多行文本

<Text style={} onPress={} onLongPress={} numberOfLines={显示出来的行数} ellipsizeMode='省略号显示的位置'>文本{'\n'}内容</Text>

注意：RN允许Text中嵌套Text，而且子Text可以继承父Text的样式！

换行{'\n'}

练习：点击一段文本，实现单击计数器

**④Button**：按钮

<Button title="按钮上的文字" color="按钮背景色" onPress={ } disabled={true/false} />

**⑤TextInput：**文本输入框(单行/多行)

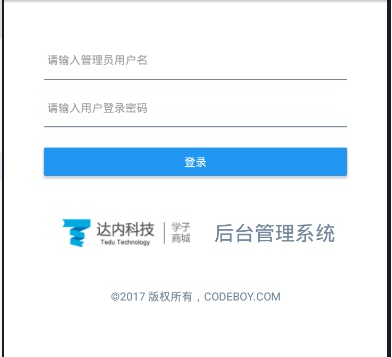
<TextInput placeholder="提示文字" secureTextEntry={true} multiline={true} numberOfLines={3} value={this.state.uname} onChangeText={this.doChange}/>

**⑥Image**：图片

课后任务：

①读RN手册，掌握Image组件的两种用法

②完成“RN阶段项目”管理员登录页的页面布局效果



提示：LOGO图片可以下载后保存在当前项目目录下，使用<Image>元素进行渲染

**Day05**

复习：

见思维导图

1.RN中的常用组件——图片

**本地图片：**

<Image source={require('../src/assets/x.jpg')}/>

本地图片可以无需指定尺寸（因为导入/打包在服务器端进行），默认是原始尺寸；如果指定了width，height不会自动的修改，也必须手工赋值

**远程图片：**

<Image source={{uri:'http://t.cn/logo.png'}} style={{width:x,height:x}}/>

注意：Image的source可以赋值为“远程图片的URL地址”或者“本地的图片**数据**”——本地图片不能使用文件路径名字符串！远程图片需要异步请求，初始时图片尺寸为0；请求完成后默认尺寸仍然为0——必须手工设置远程图片的尺寸（防止页面重排）

Image组件的resizeMode决定了图片尺寸与组件尺寸不等时如何缩放，可取值：

①cover：覆盖，等比例缩放图片，保证可以完全覆盖组件的宽和高，溢出部分不显示（默认值）

②contain：包含，等比例缩放图片，保证容器可以完全容纳图片的全部内容，可能出现留白

③stretch：拉伸，不等比例缩放图片，保证容器可以完全容纳图片的全部内容，不会留白但图片会变形

④repeat：重复，等比例缩放图片，在容器中平铺图片，保证没有留白

⑤center：图片不拉伸直接放置在容器正中央

|  |
| --- |
| 重点：如何获取远程图片的原始尺寸？从而实现图片内容全部显示且无留白  Image.getSize( uri, (w, h)=>{ }, (err)=>{ } )  import React from 'react'  import {View,Image} from 'react-native'  export default class App extends React.Component{    constructor(){      super()      // 如何获取图片的原始宽高      this.state = {        uri: 'http://cdn.tmooc.cn/bsfile//imgad///1a7a90030ff2458799fa83f366f901c8.jpg',        originalWidth: 0,   //图片原始宽        originalHeight: 0   //图片原始高      }    }    // 组件挂载完成后，异步请求图片的尺寸      componentDidMount(){        Image.getSize(this.state.uri,(w, h)=>{          // 异步获取到的图片的尺寸          this.setState({            originalWidth: w,            originalHeight: h          })        })      }    render(){      return (        <View>          {/\* 方法一:本地图片  可以自动识别宽高 \*/}          <Image source={require('./src/assets/1.jpg')} style={{width:100,height:100}} />          {/\* 方法二：远程图片 无法自动识别宽高 \*/}          {/\* 获取图片原始宽高 \*/}          <Image source={{uri:this.state.uri}} style={{width:this.state.originalWidth,height:this.state.originalHeight}} />        </View>      )    }  } |
| 午间任务：RN中如何读取手机屏幕的尺寸？从而实现图片铺满屏幕宽度 |

|  |
| --- |
| 小知识：url、uri、urn三者间的关系？  Unified Resource Identifier：统一的资源识别符  Unified Resource Locator：统一的资源定位符，如http://b.com/logo.jpg  Unified Resource Naming：统一的资源命名符，如tel: 13312345676、mailto:tom@t.cn  **URI = URL + URN** |

|  |
| --- |
| 使用本地图片时的经典错误：  this.state.list = [  './src/asset1/0.jpg',  './src/assets/1.jpg'  ]  <View>  {  this.state.list.map((e, i)=>{  return <Image source={require(e)}/>  })  }  </View> |
| 错误提示： Invalid call at line xx： require(e)  错误原因： require放在循环/选择中，变为动态导入，Webpack打包时不支持动态导入！  解决方法： 把动态导入变为静态导入： |
| this.state.list = [  require('./src/asset1/0.jpg'),  require('./src/assets/1.jpg')  ]  <View>  {  this.state.list.map((e, i)=>{  return <Image source={ e }/>  })  }  </View> |

2.RN中的常用组件——活动指示器

<ActivityIndicator size="small/large" color="#f00" />

显示一个“加载中”的圆环动画图片；默认为小号，可以设置为大号；

注意：页面中最后一个“活动指示器”的color会影响前面所有指示器的颜色

3.RN中的常用组件——开关

<Switch value={ 布尔变量 } onValueChange={事件处理函数} />

属于需要进行“双向数据绑定”的受控组件，value是boolean数据

4.RN中的常用组件——触摸反馈

<TouchableOpacity>

其它组件(文字/图片等)

</TouchableOpacity>

提示：RN中提供了四个触摸反馈组件，其中TouchableOpacity是触摸后发生透明度的改变

5.RN中的常用组件——简单列表——难点&重点

高性能的列表组件——底层值挂载当前需要显示的列表项，以及上方/下方若干条即将被显示的列表项，更大范围内列的表项不进行挂载

<FlatList data={数组} renderItem={ this.\_renderItem } keyExtractor={this.\_keyExtractor} />

\_renderItem = ( obj )=>{

//obj.item：是一个数据 obj.index：是该数据的下标

return (表示一个列表项的JSX)

}

|  |
| --- |
| FlatList组件的属性：  ①data：必需的，数组类型，指定列表要渲染的数据  ②renderItem：必需的，函数类型，指定如何渲染一个列表项  ③keyExtractor：函数类型，指定如何提取每个列表项的key  ④ListHeaderComponent：函数类型，指定列表头部内容  ⑤ListFooterComponent：函数类型，指定列表尾部内容  ⑥ItemSeparatorComponent：函数类型，指定列表项间的分隔条内容  ⑦onEndReached：事件，指代滚动到了列表尾部  ⑧onEndReachedThreshold：数字，0~1之间，指定滚动到距离底部还剩多远时触发onEndReached事件 |
| export default class App extends React.Component{  constructor(){    super()    this.state = {      list: ['今天','昨天','前天','加油']    }  }  \_renderItem = (obj)=>{//如何渲染一个列表项    // obj中：{item:'今天',index:0}    return(      <TouchableOpacity>        <Text>{obj.index+1} - {obj.item}</Text>      </TouchableOpacity>    )  }  \_keyExtractor = (item,index)=>{//返回每一个列表项的key    return String(index) //每一个列表项的key必须是string  }  render(){    return(      <View>        <FlatList data={this.state.list} renderItem={this.\_renderItem} keyExtractor={this.\_keyExtractor} />      </View>    )  }  } |

练习：假设有如下数据：laptopList : [

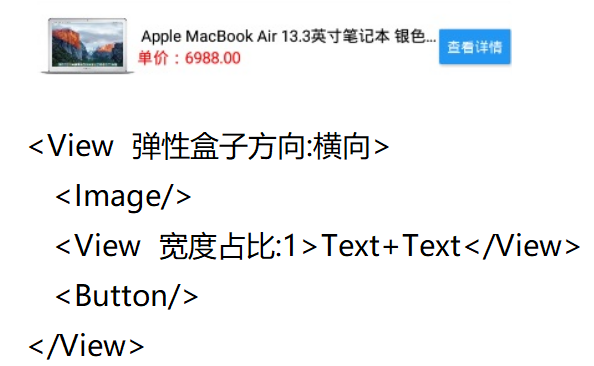
{lid: 100, title:'笔记本标题0', pic: './src/assets/0.jpg',price:500},

{lid: 101, title:'笔记本标题1', pic: './src/assets/1.jpg',price:600},

{lid: 102, title:'笔记本标题2', pic: './src/assets/2.jpg',price:700},

{lid: 103, title:'笔记本标题3', pic: './src/assets/3.jpg',price:800},

]，将这些数据渲染在一个FlatList中，其中每个列表项形如：



|  |
| --- |
| import React from 'react'  import {View, Image, FlatList,Button,Text,ActivityIndicator} from 'react-native'  export default class App extends React.Component{    constructor(){      super()      this.state = {        laptopList: [        {lid:100, title:'笔记本标题1', pic:require('./src/assets/0.png'),price:3400},        {lid:200, title:'笔记本标题2', pic:require('./src/assets/1.png'),price:5000},        {lid:300, title:'笔记本标题3', pic:require('./src/assets/2.png'),price:5400},        {lid:400, title:'笔记本标题4', pic:require('./src/assets/3.png'),price:5200}      ]      }    }    \_renderItem = (obj)=>{      // console.log(obj)      return (        <View style={{flexDirection:'row'}}>          <Image source={obj.item.pic} style={{width:80,height:80}} />          <View style={{flex:1}}>            <Text>{obj.item.title}</Text>            <Text>{obj.item.price}</Text>          </View>          <Button title='查看详情' />        </View>      )    }    \_keyExtractor = (item,index)=>{      return  String(index)    }    \_ListHeaderComponent = ()=>{      return (        <Text style={{textAlign:'center'}}>笔记本列表      </Text>      )    }    \_ListFooterComponent = ()=>{      return(        <ActivityIndicator size='large' />      )    }    \_ItemSeparatorComponent = ()=>{      return(        // 分割线        <View style={{borderTopWidth:1,height:0,marginVertical:10}}></View>      )    }    render(){      return(          <FlatList ItemSeparatorComponent={this.\_ItemSeparatorComponent} ListFooterComponent={this.\_ListFooterComponent} ListHeaderComponent={this.\_ListHeaderComponent} data={this.state.laptopList} renderItem={this.\_renderItem} keyExtractor={this.\_keyExtractor} />      )    }  } |

6.RN中路由和导航实现

提示：RN的运行不依赖于浏览器！没有window.navigator/history对象！甚至没有Page的概念！——RN中切换是Screen，不是Page。

RN官方没有提供导航组件，推荐使用第三方导航组件：**React Navigation.**

React Navigation的官网：https://reactnavigation.org/

ReactNavigation的使用步骤：

①在当前项目下载必需的NPM包

npm i react-navigation

npm i react-native-gesture-handler

npm i react-native-reanimated

②创建路由词典 —— 称为“栈式路由导航器”

//App.js

let routes = createStackNavigator({

'login': LoginScreen, //排名第一的组件就是默认APP的首屏内容

'main': MainScreen,

})

③创建根组件，注册路由词典

//App.js

export default createAppContainer( routes )

④访问测试

摇一摇 > 重新加载即可

提示：RN应用完全不涉及浏览器，所以无法在任何地址栏中输入路由地址！只能通过ReactNavigation提供的路由跳转机制来跳转

课后任务：

①根据“RN阶段项目设计图”完成如下四屏组件的静态布局内容：

LoginScreen、MainScreen、ProductListScreen、ProductDetailScreen

注意：页头和标题栏由导航组件自动提供，无需手工编写

②阅读ReactNavigation手册，掌握页面跳转及页面传参的相关实现代码

|  |
| --- |
| 前端三大难点：  ①浏览器/设备兼容性  ②性能优化 —— 列表组件的性能优化问题？  ③Web安全 |

**Day06**

复习：

见思维导图

1.RN阶段项目——“学子商城后台管理系统”重构

①创建导航屏幕组件：

src/components/LoginScreen：管理员登录屏幕

src/components/MainScreen：主功能菜单屏幕

src/components/ProductListScreen：商品列表屏幕

src/components/ProductDetailScreen：商品详情屏幕

src/components/UserListScreen：用户列表屏幕

src/components/UserDetailScreen：用户详情屏幕

.....

②在App.js中创建路由词典(即栈式导航器)

③创建App根组件并注册路由词典

④完善LoginScreen组件的内容，并实现登录验证

View >

TextInput

TextInput

Button

View >

Image

Text

Text

重点：布局系统、异步POST请求

⑤完善MainScreen组件的内容，实现其中的组件布局

View >

View > 统计信息

View.row > View.col + View.col

View.row > View.col + View.col

View.row > View.col + View.col

View > 功能菜单

View.row > View.col + View.col

View.row > View.col + View.col

重点：布局系统、样式的使用

⑥完善ProductListScreen组件的内容，实现无限滚动加载以及路由跳转传参

<FlatList data={this.state.productList} renderItem={this.\_renderItem}, keyExtractor={this.\_keyExtractor} onEndReached={this.loadMore} onEndReachedThrashold={0.1}/>

重点：无限滚动组件的使用、异步数据获取

⑦完善ProductDetailScreen组件的内容，实现服务器端数据的异步获取及转换

自定义“轮播广告”组件——重点

自定义“商品详情”组件——难点

2.ReactNavigation中如何实现页面跳转

因为每个屏幕组件(具有路由地址的组件)都是有App根组件自动创建并挂载的，App组件在创建屏幕组建时，会自动传递进来一个props： navigation

从当前屏幕跳转到其它屏幕：

this.props.navigation.navigate( '屏幕地址' )

注意：默认情况下，Image没有onPress事件，如果想被点击实现跳转，需要封装到“触摸反馈组件”，如

<TouchableOpacity onPress={}>

<Image/>

</TouchableOpacity>

3.ReactNavigation中如何实现跳转传参

回忆：基于浏览器的Web应用中如何传参？

Angular中：

路由词典：{path: 'product/detail/:pid', component: ...}

执行跳转：<a routerLink="product/details/15">

注意：RN应用根本没有浏览器，没有地址栏，没有/15！

RN中的路由传参不需要修改路由词典；

**前一个组件跳转时提供路由参数：**

this.props.navigation.navigate('目标地址', {pid:13,age:30})

**后一个组件加载完成时读取路由参数：**

this.props.navigation.state.params —— {pid:13, age:30}

或者

this.props.navigation.getParam('pid', 99) //99表示默认参数值

|  |
| --- |
| class App extends React.Component{  render(){  return (  <View>  <ScreenHeader>  { 读取LoginScreen.navigationOptions来填充页头 }  </ScreenHeader>  <ContentOutlet>  {  if(当前客户端想访问login地址的话){  return <LoginScreen navigation={...}/>  }....  }  </ContentOutlet>  </View>  )  }  } |

4.ReactNavigation中如何配置页头

①仅指定标题栏文字

static navigationOptions = {

**title: '标题栏文字'**

}

②指定复杂的标题栏组件

static navigationOptions = {

**headerTitle: <Any />,**

**headerLeft: <Any />**

**headerRight: <Any />,**

}

可以为headerTitle、headerLeft、headerRight属性赋值为任意组件/自定义组件，实现页头自定义的功能；

此外，如果需要在页头中使用navigation对象，不能直接使用this.props.navigation（静态成员中不能使用this），可以使用

static navigationOptions = ((obj)=>{

return **{**

**title: '商品编号：'+obj.navigation.getParam('pid')+'的详情'**

**}**

})

5.前端可用的异步请求技术有哪些？

①原生XHR：直接， 但可能产生回调地狱

②jQuery.ajax( )：简单，但可能产生回调地狱

③Axios：功能强大，基于Promise

④HttpClient：功能强大，基于观察者模式

----------------------------------------------

⑤Fetch：W3C的新标准，有望取代XHR

|  |
| --- |
| RN应用中没有浏览器，没有原生XHR技术；但是RN底层根据W3C Fetch标准，提供了一套完整的Fetch API。 |

Fetch API的使用方法：

**发起GET请求：**

fetch(url).then( (res)=>{

return res.json( ) //等待下载完全部响应消息主体，解析为JS对象

} ).then( (body)=>{

//body就是响应主体

} ).catch((err)=>{

})

**发起POST请求：**

let options = {

method: 'POST',

headers: { },

body: 'k=v&k=v'

}

fetch(url , options ).then( (res)=>{

return res.json( ) //等待下载完全部响应消息主体，解析为JS对象

} ).then( (body)=>{

//body就是响应主体

} ).catch((err)=>{

})

课后任务：

(1)完成“RN阶段项目”——“商品列表”界面组件；

注意：使用FlatList组件实现“无限滚动”效果

(2) 完成“RN阶段项目”——“商品详情”界面组件；

注意：需要自定义“轮播广告”组件，以及“商品详情”组件

**Day07**

复习：

见思维导图

1.RN阶段项目——“学子商城后台管理系统”重构

**①创建导航屏幕组件：**

src/components/LoginScreen：管理员登录屏幕

src/components/MainScreen：主功能菜单屏幕

src/components/ProductListScreen：商品列表屏幕

src/components/ProductDetailScreen：商品详情屏幕

src/components/UserListScreen：用户列表屏幕

src/components/UserDetailScreen：用户详情屏幕

.....

**②在App.js中创建路由词典(即栈式导航器)**

//App.js

let routes = createStackNavigator({

login: LoginScreen,

main: MainScreen

})

**③创建App根组件并注册路由词典**

//App.js

export default createAppContainer( routes )

**④完善LoginScreen组件的内容，并实现登录验证**

View >

TextInput

TextInput

Button

View >

Image

Text

Text

重点：布局系统、异步Fetch POST请求

**⑤完善MainScreen组件的内容，实现其中的组件布局**

View >

View > 统计信息

View.row > View.col + View.col

View.row > View.col + View.col

View.row > View.col + View.col

View > 功能菜单

View.row > View.col + View.col

View.row > View.col + View.col

重点：布局系统、样式的使用

**⑥完善ProductListScreen组件的内容，实现无限滚动加载以及路由跳转传参**

<FlatList data={this.state.productList} renderItem={this.\_renderItem}, keyExtractor={this.\_keyExtractor} onEndReached={this.loadMore} onEndReachedThrashold={0.1}/>

重点：无限滚动组件的使用、异步Fetch GET数据获取

**⑦完善ProductDetailScreen组件的内容，实现服务器端数据的异步获取及转换**

自定义“轮播广告”组件——重点

自定义“商品详情”组件——难点

2.小知识：如何在海量的代码快速定位错误 —— 调试技巧？

增量调试：注释掉最新添加的功能，如果没有错误了，说明最新添加的代码有问题；如果仍旧有错，说明错误存在于之前的代码中；继续注释掉次新添加的功能...

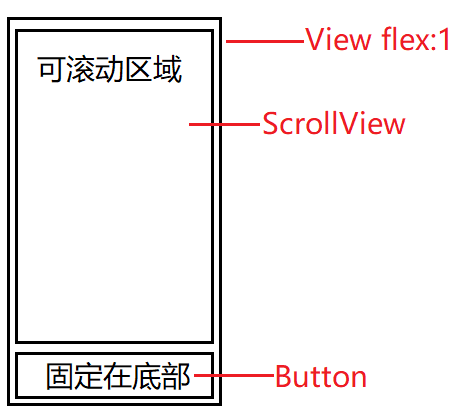
功能1

功能2

//功能3

//功能4

3.如何在RN中实现类似于Web中顶部/底部固定效果？



要点：

①父容器<View>必须设定height、或者设定高度占比flex

②ScrollView天然就会占用所有可用空间，故可省略flex

③固定在底部的元素只需要声明在ScrollView下方

4.RN中如何获取当前手机屏幕的尺寸？

import {Dimensions} from 'react-native'

获取屏幕的宽： Dimensions.get('screen').width

获取屏幕的高： Dimensions.get('screen').height

获取窗口的宽： Dimensions.get('window').width

获取窗口的高： Dimensions.get('window').height

提示：手机中window的宽和screen的宽；screen的高等于window的高+状态栏的高度

5.如何原生App中渲染HTML片段

提示：原生App不是浏览器的，没有HTML解释器，不能解释HTML/CSS/JS代码。

**方法1：在App中嵌入现成的浏览器内核组件——WebView**

<WebView source={{uri: 'https://www.baidu.com'}} />

<WebView source={{html: '<h1>Hello world</h1>' }} />

因为该组件体积过于庞大，RN官方已经从核心中移除该组件

**方法2：使用正则表达式把HTML标签转换为RN的组件，如<img>转换为<Image/>**

|  |
| --- |
| 原始的HTML片段： |
| <div>  <img class="" src="img/product/detail/586f7483N695168a2.jpg">  </div>  <div>  <img class="" src="img/product/detail/586f7486Nf809b915.jpg">  </div>  <div>  <img class="" src="img/product/detail/586f748bN28dbcbb2.jpg">  </div> |
| 转换后的目标组件： |
| <Image source={{uri: 'http://www.codeboy.com/'+"img/product/detail/586f7483N695168a2.jpg"}}/>  <Image source={{uri: 'http://www.codeboy.com/'+"img/product/detail/586f7486Nf809b915.jpg"}}/>  <Image source={{uri: 'http://www.codeboy.com/'+"img/product/detail/586f748bN28dbcbb2.jpg"}}/> |

课后任务：

(1)完成“RN阶段项目”——商品列表和商品详情

(2)完成“RN阶段项目”——用户列表和用户详情

(3)在自己的电脑上安装Python运行环境——Python解释器

<https://www.python.org/>

下载安装文件，开始安装，注意勾选：Add Python to PATH！

Python的两种执行方法：

①交互式执行： python 回车 输入一行执行一行

②脚本式执行： python 2.py 回车 一次性执行其中所有代码

**Redux**

1.下载

npm i redux -S

**React-router**

1. 下载

npm i react-router-dom -S

1. 使用
2. 根目录的index.js

注意router分BrowserRouter和HashRouter

引入 Router 并使用<Router></Router>对根组件进行包裹

|  |
| --- |
|  |

HashRouter 会有 /#/出现

|  |
| --- |
|  |

此外还提供了Route组件，当使用了Route组件进行渲染的组件，会自动注入history等

|  |
| --- |
|  |

1. 相关的组件或属性
2. Router分为HashRouter和BrowerRouter
3. Route 有许多api

|  |
| --- |
|  |

1. Switch 包裹在其他路由的外面

|  |
| --- |
|  |

1. Redirect
2. Link和NavLink 后者会自动加上了class=’active’ 与 aria-current=’page’

|  |
| --- |
|  |

1. exact属性
2. 动态路由

|  |
| --- |
|  |

8)渲染组件有两个方法 component（优先起作用）和render（当组件需要传递参数等时使用）

|  |
| --- |
|  |

9)编程式导航

|  |
| --- |
|  |

1. withRouter

|  |
| --- |
|  |

1. react Link组件 路由传参的几种方式
2. Query 需要手动解析

|  |
| --- |
|  |

2）隐式路由传参 使用state 隐式传参

|  |
| --- |
|  |

3）动态路由 /path/:param => params

|  |
| --- |
|  |

附：向后端发送数据的方式

1. ajax
2. img: 如： const img=new Image()  
    img.src=’https://www.domainnme.com/button-01.gif?x=1&y=2’
3. sendBeacon 支持数据发送了才跳转 兼容性不好 成功率高

补充：react全家桶 = react react-router redux

**Ant-Desgin (**react UI框架，中后台产品**)**

手机端：mobile.ant.desigin

1.安装

npm i antd -S 为了配置的更好还需要安装

npm i react-app-rewired customize-cra -D(某盘) （这两个要同时安装）

2.rewired 的设置 修改package.json 里面scripts的 react-scripts改为 react-app-rewired

|  |
| --- |
|  |

在根目录下新建第一个 config-overrides.js 用于修改配置

|  |
| --- |
|  |

3.下载babel-plugin-import

npm i babel-plugin-import -D（某盘）

config-overrides.js

|  |
| --- |
|  |

4.使用 实例

|  |
| --- |
|  |

5.国际化 语言设置为中文

|  |
| --- |
|  |

**React 配置 less （结合antd）**

1.安装 less 和 less-loader

npm i less less-loader -D

2.修改config-override.js

引入 addLessLoader

|  |
| --- |
|  |

3.测试less安装

|  |
| --- |
|  |

4.当想修改 antd的主题颜色时

方法一： 在addLessLoader 的 modifyVars:{ .....} 进行设置

方法二： 单独写一个文件 如 theme.js

|  |
| --- |
| 在config-overrides.js中 引用 |

解决：antd写的元素class中带有ant-xx前缀

**配置装饰器**

1.测试高阶组件

|  |
| --- |
|  |