React 开发环境搭建：

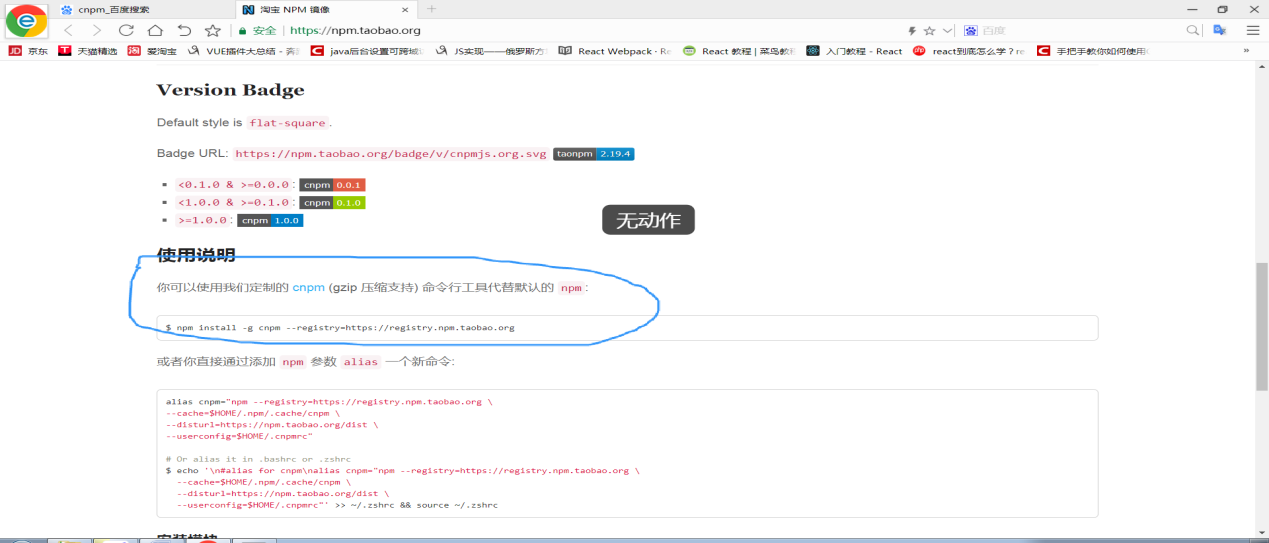
1. 安装nodejs （安装nodejs稳定版本

<https://nodejs.org/dist/v8.11.2/node-v8.11.2-x64.msi>

[odejs.org/dist/v8.11.2/node-v8.11.2-x86.msi](https://nodejs.org/dist/v8.11.2/node-v8.11.2-x86.msi)）

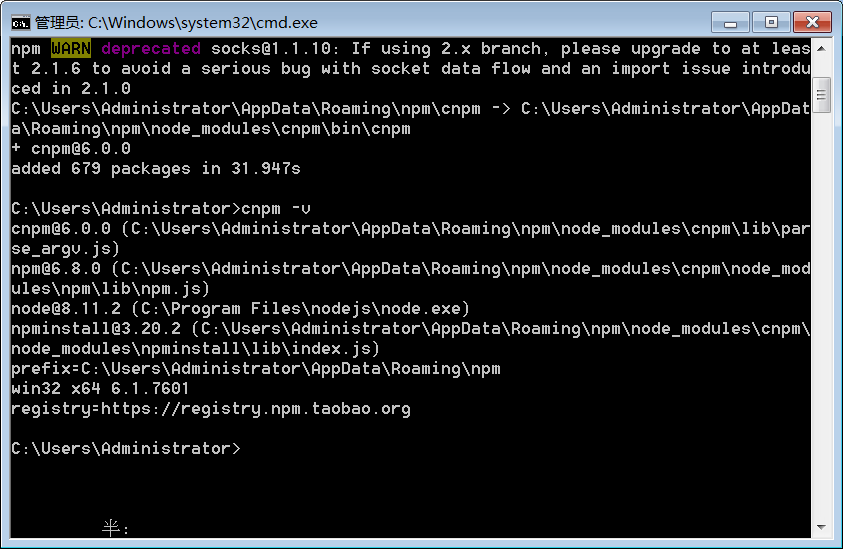
1. 安装好node之后 在cmd检查版本 node –v 在安装完node之后npm页会自动安装 检查版本 npm –v
2. 安装cnpm 代替npm 在百度上搜索cnpm 找淘宝镜像（<http://npm.taobao.org/>）



在当前页面上下拉找：

复制去cmd安装cnpm

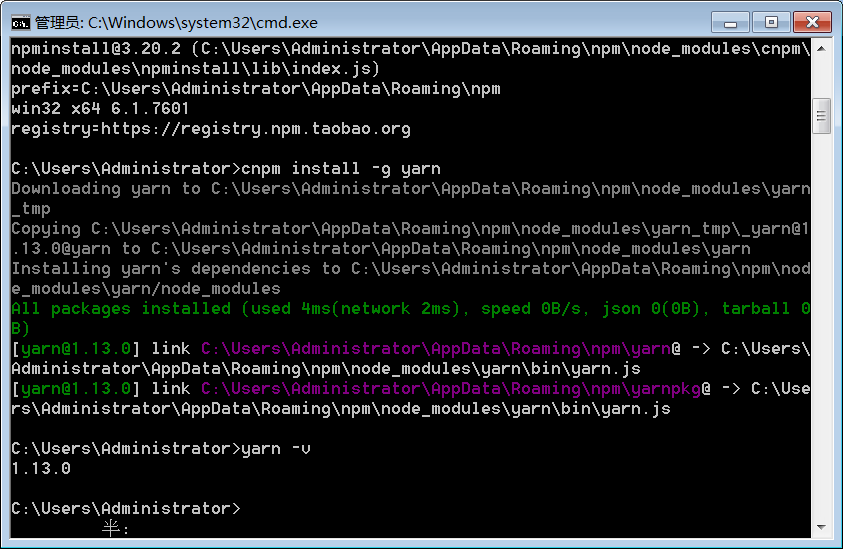
1. 安装完成之后 检查版本 cnpm –v 输出以下为安装完成：



安装yarn 依赖工具（<https://yarn.bootcss.com/>）

在中文官网里面有windows安装yarn的方式。

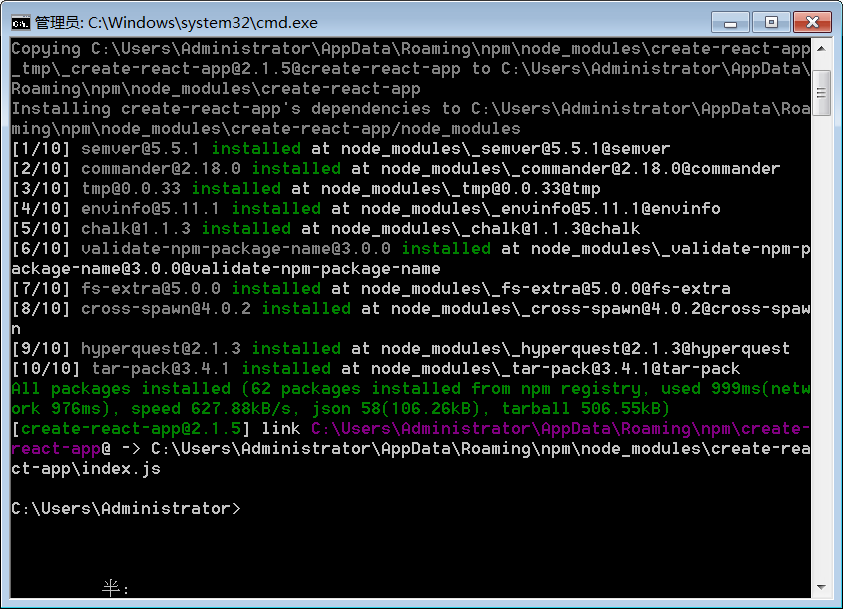
1. 简单安装yarn cnpm install -g yarn 或者 npm install –g yarn
2. 安装完成之后 检查yarn的版本 yarn –v



1. 创建React 开发环境 官方文档（<https://reactjs.org/>）

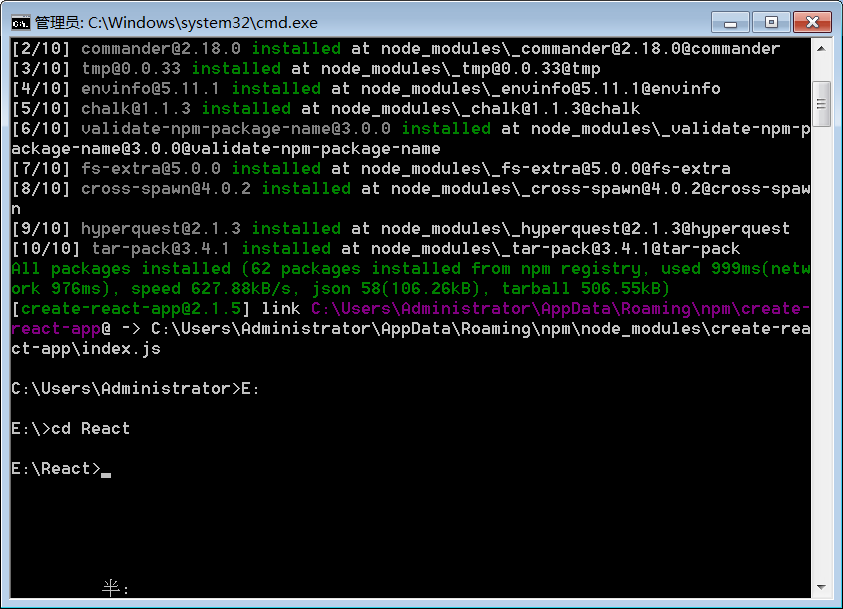
第一种：安装脚手架

cnpm install -g create-react-app 或者 npm install –g create-react-app



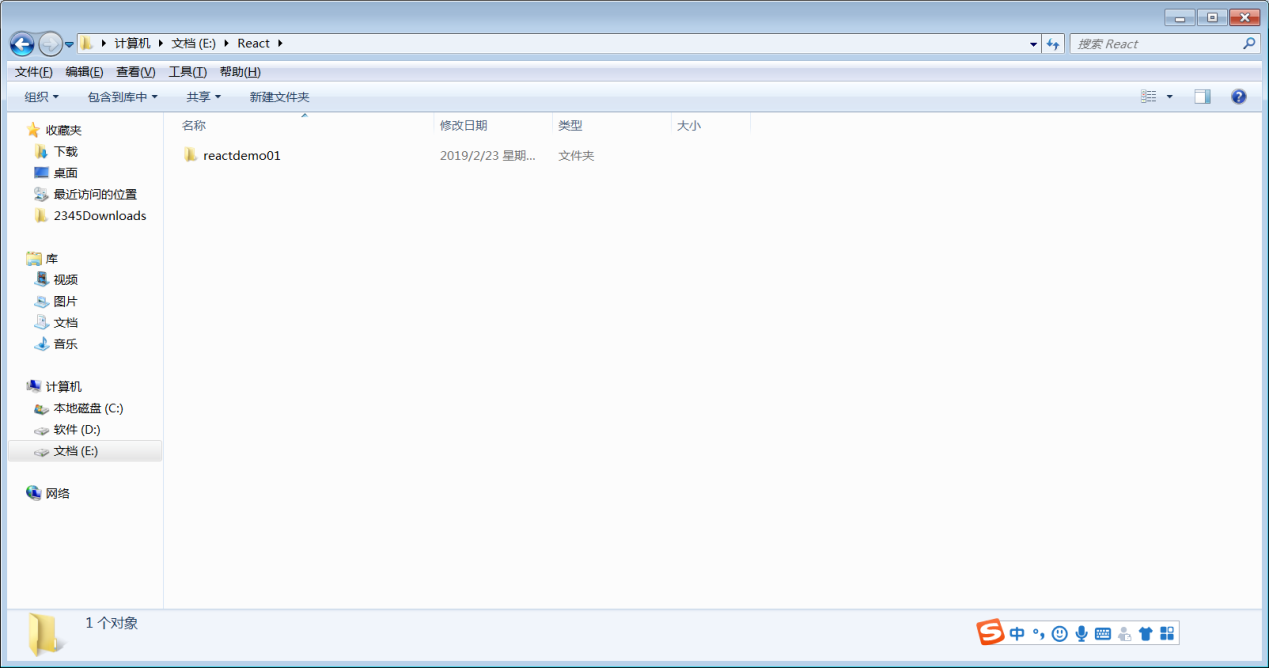
脚手架安装完成之后 创建项目 （自己选择文件的位置）（我默认项目文件盘）

1. cmd命令窗口切换到E盘：进入你创建的目录

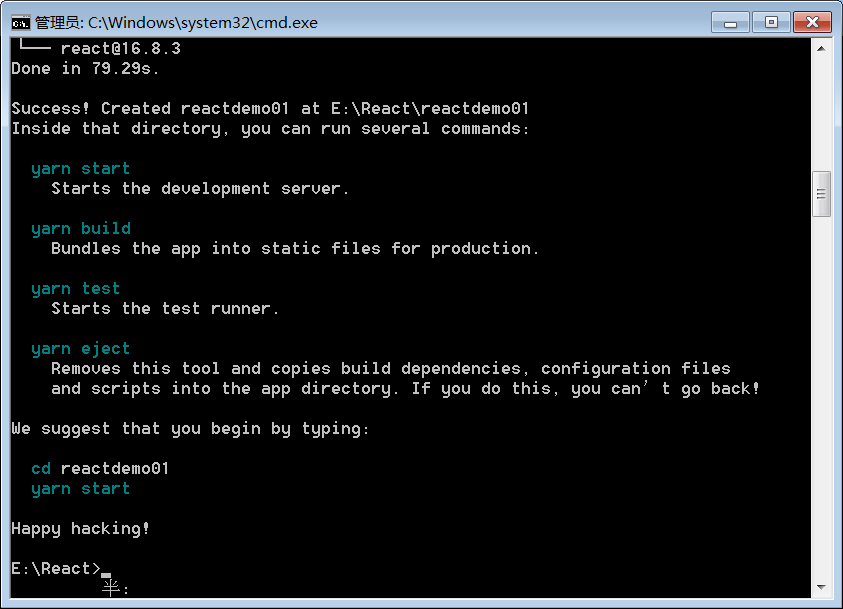


1. 输入创建项目的命令 create-react-app reactdemo

项目创建完成之后，会在你的项目目录里面生成你创建的文件



Cmd命令框下拉 下面的指令提示你项目文件的运行、编译等等功能



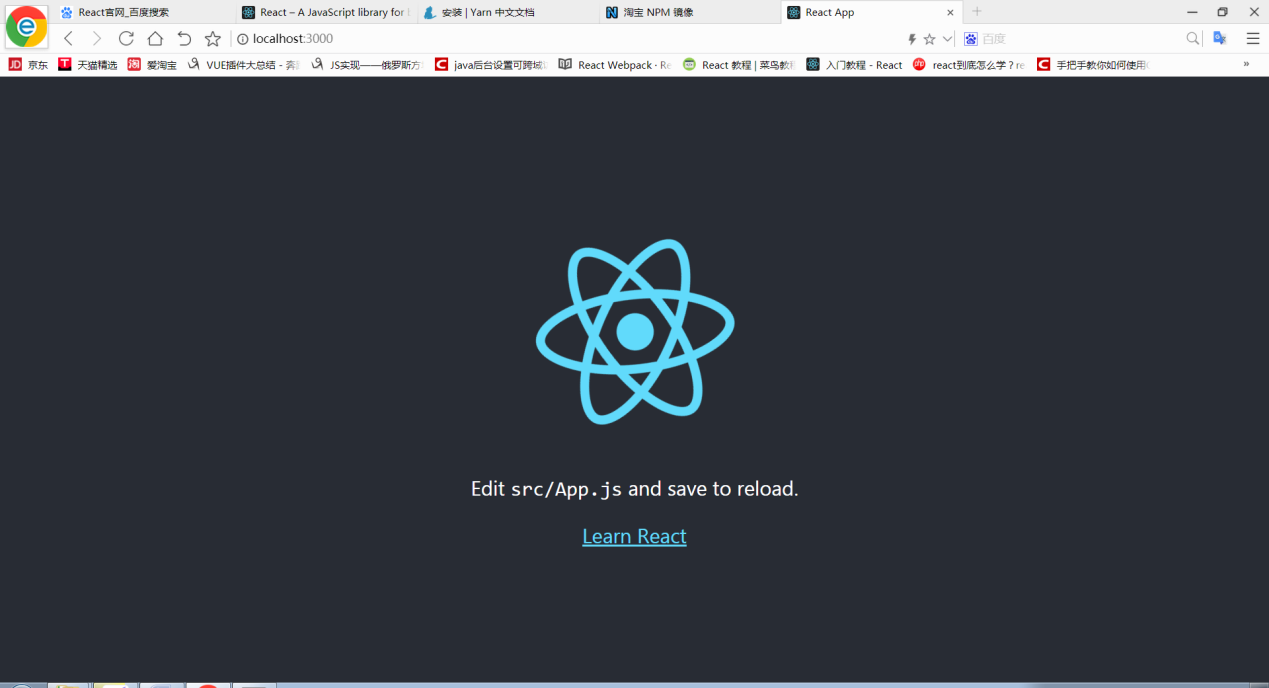
1. Cd到项目里面

cd reactdemo01

npm start 或者yarn start 运行

npm run build 或者yarn build 生成项目

1. 运行之后



React项目创建成功

1. 目录分析

Src文件为开发文件（对src文件进行开发整理）

其他文件可以先不管

1. 声明组件

创建组件文件

import React, { Component } from 'react';

*//组件的类声明*

class **Home** extends **Component**{

**render**(){

return <div>我是组件</div>;

}

}

*//组件暴露*

export default Home;

//组件里面有多个元素的时候注意：在return里面返回元素只有一个父元素 且在（）里面写标签

import React,{Component} from 'react'

*//组件的类*

class **Cart** extends **Component**{

**render**(){

return(

<div>

<span>我是2组件</span>

<span>多个元素写法</span>

</div>

);

}

}

*//暴露*

export default Cart;

组件的挂载：

import React, { Component } from 'react';

import logo from './assets/image/logo.svg';

import './assets/css/App.css';

*//组件的引入*

import Home from './components/Home'

import Cart from './components/Cart'

*//jsx语法 js+html混写*

class **App** extends **Component** {

**render**() {

return (

<div *className*="App">

你好React

<**Home**></**Home**>

<**Cart**/>

</div>

);

}

}

export default App;

1. 在vsCode种写代码没智能提示 ：添加typings扩展组件
2. 在组件里面声明数据

在组件的class种声明类的构造函数constructor 在构造函数里面必须写super（）否则报错

import React, { Component } from 'react';

*//组件的类声明*

class **Home** extends **Component**{

*//声明类的构造函数*

constructor(){

*//类似于Java里面的继承*

super();

*//数据写在*

*//React 定义数据*

this.state={

name:"张三",

age:"20",

userdata:{

user:"猜猜"

}

}

}

**render**(){

return (

<div>

{*/\* 数据的绑定 \*/*}

<p>姓名：{this.state.name}</p>

<p>年龄{this.state.age}</p>

<p>object：{this.state.userdata.user}</p>

</div>

);

}

}

*//组件暴露*

1. 在组件里面属性的绑定
2. import React,{Component} from 'react'
3. import "../assets/css/index.css"
4. *//组件的类*
5. class **Cart** extends **Component**{
6. constructor(){
7. super();
8. *//数据*
9. this.state={
10. title:"绑定title",
11. color:"color",
12. style:{
13. color:"red",
14. fontSize:"20px"
15. }
16. }
17. }
18. **render**(){
19. return(
20. <div>
21. <span>在这里绑定属性</span>
22. <br/>
23. {*/\* 普通属性的绑定 \*/*}
24. <div *title*={this.state.title}>给我绑定title属性</div>
25. <br/>
26. {*/\* 绑定特殊属性 和 js里面冲突的一些属性 \*/*}
27. {*/\* class属性和上面类的关键字冲突 必须换为className \*/*}
28. <div *className*="color">绑定class属性</div>
29. <br/>
30. {*/\* 动态绑定className \*/*}
31. <div *className*={this.state.color}>动态绑定class</div>
32. <br/>
33. {*/\* for属性和js里面的for循环一样 \*/*}
34. <label *htmlFor*="name">绑定for</label>
35. <input *id*="name" *type*="text"/>
36. {*/\* 绑定style属性 \*/*}
37. <div *style*={{"color": "red"}}>绑定style属性</div>
38. <div *style*={this.state.style}>绑定style属性</div>
39. {*/\* 绑定冲突属性换被的方式其他的属性一样 \*/*}
40. </div>
41. );
42. }
43. }
44. *//暴露*
45. export default Cart;

17） 图片的src引入问题和数组的循环

import React from 'react';

*// 图片路径的引入*

import imginfo from '../assets/image/my\_2.png';

class **News** extends **React**.**Component**{

constructor(){

super();

this.state={

news:"新闻"

}

}

**render**(){

return (

<div *className*="news">

{this.state.news}

{*/\* 图片路径的引入 \*/*}

{*/\* 这里的路径不能直接写 \*/*}

<img *src*={imginfo} *alt*=""/>

{*/\* 或者src使用es5里面的require \*/*}

<img *src*={**require**('../assets/image/my\_2.png')} *alt*=""/>

{*/\* 引入远程图片 直接将路径写上去\*/*}

<img *src*="" *alt*=""/>

</div>

);

}

}

export default News;

数组遍历和元素的绑定问题

import React from "react";

class **User** extends **React**.**Component**{

constructor(){

super();

this.state={

list:["1","2","3"],

listele:[<div *key*="1">1</div>,<div *key*="2">2</div>,<div *key*="3">3</div>],

*// 对象的转化*

liobject:[

{txtvalue:"你好"},

{txtvalue:"你好"},

{txtvalue:"你好"},

{txtvalue:"你好"},

{txtvalue:"你好"}

]

}

}

**render**(){

*// let eles=this.state.list.map(function (value,key){*

*// return <div key={key}>{value}</div>*

*// });*

let eles=this.state.list.**map**(function (value,key){

return (<li *key*={key}>{value}</li>);

});

return (

<div>

遍历数组和绑定元素

<br/>

{this.state.listele}

<br/>

{this.state.list}

{*/\* 也就是说现在只需要将值转化为带标签的数组就可以 \*/*}

<ul>

{eles}

</ul>

<hr/>

{*/\* 或者 \*/*}

<ul>

{this.state.list.**map**(function (value,key){

return <li *key*={key}>{value}</li>

})}

</ul>

<hr/>

<ul>

{

this.state.liobject.**map**(function (value,key){

return <li *key*={key}>{value.txtvalue}</li>

})

}

</ul>

</div>

);

}

}

export default User;

18） React事件方法

1） 方法的声明 和事件的写法

2） 修改state里面的数据

3） 事件调用方法传参

import React from 'react';

class **Userdata** extends **React**.**Component**{

constructor (){

super();

this.state={

msg:"我是消息",

username:"李四"

};

*//第二种修改在构造函数里面直接修改this的问题*

this.getData=this.getData.**bind**(this);

}

*// 方法的声明在这一块同级位置*

*// 使用es6语法声明*

**getmsg**(){

*//获取数据*

*//直接这样获取 this的指针不同 故报错*

**console**.**log**(this.state.msg);

}

**getData**(){

**console**.**log**(this.state.msg);

}

**getmessage**=()=>{

**console**.**log**(this.state.msg);

}

**setdata**=()=>{

*//修改数据使用setState方法*

this.**setState**({

msg:"修改之后的值"

});

}

**setName**=(str)=>{

this.**setState**({

username:str

});

}

**render**(){

return (

<div>

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

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

演示方法的声明+事件的绑定

<br/>

{*/\* 修改该方法的this指针为当前组件的this \*/*}

<button *onClick*={this.getmsg.**bind**(this)}>第一种获取数据的方式</button>

<br/>

{*/\* 修改该方法的this指针为当前组件的this \*/*}

<button *onClick*={this.getData}>第二种获取数据的方式</button>

<br/>

{*/\* 修改该方法的this指针为当前组件的this \*/*}

<button *onClick*={this.getmessage}>第三种获取数据的方式</button>

<br/>

{*/\* 修改数据 \*/*}

<button *onClick*={this.setdata}>修改state里面的值</button>

<br/>

{*/\* 调用方法传参 \*/*}

<button *onClick*={this.setName.**bind**(this,"张三")}>调用方法传值</button>

</div>

);

}

}

export default Userdata;

19） 键盘事件+表单事件以及事件对象 react种ref获取dom元素节点和实现vue中数据的双向问题

1） 事件对象event 类似js event

import React from 'react';

class **Eventinfo** extends **React**.**Component**{

constructor (){

super();

this.state={

msg:"事件对象"

}

}

**getEvent**=(event)=>{

*//事件的参数 event*

*// alert(this.state.msg);*

*//获取当前元素节点*

let btn=event.target;

*//操作当前元素节点的属性（自定义属性等等）*

**console**.**log**(btn.**getAttribute**("data-user"));

}

**render**(){

return (

<div>

<h1>{this.state.msg}</h1>

<br/>

{*/\* 事件对象的说明 \*/*}

<button *onClick*={this.getEvent} *data-user*="123">事件对象</button>

</div>

);

}

}

export default Eventinfo;

2）表单事件以及获取表单元素的值

import React from 'react';

class **Eventinfo** extends **React**.**Component**{

constructor (){

super();

this.state={

msg:"表单事件",

iptval:""

}

}

**getInputChange**=(e)=>{

this.**setState**({

iptval:e.target.value

});

}

**getInputValue**=()=>{

**console**.**log**(this.state.iptval);

}

**render**(){

return (

<div>

<h1>{this.state.msg}</h1>

<br/>

{*/\* 表单事件以及获取表单元素的值 \*/*}

{*/\**

*1.监听表单元素的文本变化*

*2.赋值给state上的变量*

*3.事件获取*

*\*/*}

<input *onChange*={this.getInputChange}/>

<button *onClick*={this.getInputValue}>获取表单元素的值</button>

</div>

);

}

}

export default Eventinfo;

ref获取dom元素节点：

1. 先给元素设置ref=“name”
2. 在去直接写this.refs.name.value;

键盘事件和以上事件写法类似

React 实现类似于vue的数据双向绑定

import React,{Component} from 'react';

class **Datato** extends **Component** {

constructor(props) {

super(props);

this.state = {

user:"123"

};

}

**setChange**=(e)=>{

this.**setState**({

user:e.target.value

});

}

**render**() {

return (

<div>

<p>实现数据的双向绑定</p>

<input *onChange*={this.setChange} *value*={this.state.user}/>

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

</div>

);

}

}

export default Datato;

20） React获取表单元素 设置表单元素（select,input,checkbox,textarea）

限制性约束 （value（通过react实现mv 方式绑定的值））和非限制性约束（defaultvalue 相当于原生的value）

import React from 'react';

class **Fromtxt** extends **React**.**Component**{

constructor(){

super();

this.state={

*//获取一个人的信息*

name:"",

sex:0,

city:"",

citys:[

"宝鸡",

"西安",

"延安"

],

hobby:[

{title:"篮球",check:false},

{title:"足球",check:true},

{title:"羽毛球",check:false}

],

content:""

}

}

*//设置姓名的方法*

**setName**=(e)=>{

*//通过事件对象获取当前的dom操作节点*

*//页可以通过refs*

this.**setState**({

*// name:e.target.value*

name:this.refs.username.value

});

}

*//设置性别的方法*

**setSex**(e){

this.**setState**({

sex:**parseInt**(e.target.value)

});

}

*//设置下拉菜单的值*

**setCity**=(e)=>{

this.**setState**({

city:e.target.value

});

}

*//设置兴趣爱好*

**setHobby**=(key)=>{

let hobby=this.state.hobby;

hobby[key].check=!hobby[key].check;

this.**setState**({

hobby:hobby

});

}

*//设置备注*

**setContent**=(e)=>{

this.**setState**({

content:e.target.value

});

}

*// 表单提交的方法*

**handleSubmit**=(e)=>{

**console**.**log**(this.state.name+"/"+this.state.sex+"/"+this.state.city+"/"+this.state.hobby+"/"+this.state.content);

*//阻止事件的提交 阻止事件的默认行为*

e.**preventDefault**();

}

**render**(){

return (

<div>

我是表单组件

<br/>

<br/>

<form *onSubmit*={this.handleSubmit}>

<ul>

<li>姓名：<input *ref*="username" *type*="text" *value*={this.state.name} *onChange*={this.setName}/></li>

<li>性别：<input *type*="radio" *value*="0" *checked*={this.state.sex===0} *onChange*={this.setSex.**bind**(this)}/> 男

<input *type*="radio" *value*="1" *checked*={this.state.sex===1} *onChange*={this.setSex.**bind**(this)}/> 女</li>

<li>城市：

<select *value*={this.state.city} *onChange*={this.setCity}>

{

this.state.citys.**map**(function (value,key){

return <option *key*={key}>{value}</option>

})

}

</select>

</li>

<li>

兴趣：

{

*//这里得注意this的指向 使用箭头函数修改this指向 保持上下文this统一*

this.state.hobby.**map**((value,key)=>{

return (

<span *key*={key}>

<input *type*="checkbox" *onChange*={this.setHobby.**bind**(this,key)} *checked*={value.check} /> {value.title}

</span>

)

})

}

</li>

<li>备注：

<textarea *value*={this.state.content} *onChange*={this.setContent}></textarea>

</li>

</ul>

<input *type*="submit" *defaultValue*="提交"/>

</form>

</div>

);

}

}

export default Fromtxt;

21）todolist（待办事项列表）



以下代码比较多，可以通过一个集合的属性进行循环绑定产生效果（代码量较少）

import React,{Component} from "react";

import '../assets/css/ToDoLIST.css';

class **Todolist** extends **Component** {

constructor(props) {

super(props);

this.state = {

title:"ToDOList",

content:"",

now:[

{

isover:false,

title:"录制React教程",

ismove:false

}

],

over:[]

};

this.**enterKey**();

}

*//设置值*

**setContent**=(e)=>{

this.**setState**({

content:e.target.value

});

}

*// 按键事件*

**enterKey**=()=>{

let that=this;

document.**addEventListener**("keypress",function(e){

switch(e.charCode){

case 13:

let now= that.state.now;

now.**push**( {

isover:false,

title:that.state.content,

ismove:false

});

*//注意数据的双向问题*

that.**setState**({

now:now

});

break;

default: break;

}

})

}

*//设置移除*

**setMove**=(key)=>{

let now=this.state.now;

now[key].ismove=!now[key].ismove;

if(now[key].ismove)

{

now.**splice**(key,1);

}

this.**setState**({

now:now

});

}

*//设置完成*

**setOver**=(key)=>{

let now=this.state.now;

let over=this.state.over;

now[key].isover=!now[key].isover;

if(now[key].isover)

{

*// now.splice(key,1);*

over.**push**(now[key]);

now.**splice**(key,1);

}

this.**setState**({

over:over,

now:now

});

}

**setOverW**=(key)=>{

let now=this.state.now;

let over=this.state.over;

over[key].isover=!over[key].isover;

if(!over[key].isover)

{

*// now.splice(key,1);*

now.**push**(over[key]);

over.**splice**(key,1);

}

this.**setState**({

over:over,

now:now

});

}

**setMoveW**=(key)=>{

let over=this.state.over;

over[key].ismove=!over[key].ismove;

if(over[key].ismove)

{

over.**splice**(key,1);

}

this.**setState**({

over:over

});

}

**render**() {

return (

<div *className*="todo">

<div *className*="title">

<span *className*="titleinfo">{this.state.title}</span>

<input *className*="addcontent" *placeholder*="添加ToDO" *value*={this.state.content} *onChange*={this.setContent}/>

</div>

<div *className*="list">

<div *className*="listnow">

<h3>正在进行</h3>

<ul *className*="menu">

{

this.state.now.**map**( (value,key)=>{

return (

<li *key*={key}>

<input *type*="checkbox" *checked*={value.isover} *onChange*={this.setOver.**bind**(this,key)}/>

{value.title}

<input *type*="radio" *checked*={value.ismove} *onChange*={this.setMove.**bind**(this,key)}/>

</li>

);

})

}

</ul>

</div>

<div *className*="listnow">

<h3>已经完成</h3>

<ul *className*="menu">

{

this.state.over.**map**( (value,key)=>{

return (

<li *key*={key} *style*={{backgroundColor:"rgba(177, 173, 173,1)"}}>

<input *type*="checkbox" *checked*={value.isover} *onChange*={this.setOverW.**bind**(this,key)}/>

{value.title}

<input *type*="radio" *checked*={value.ismove} *onChange*={this.setMoveW.**bind**(this,key)}/>

</li>

);

})

}

</ul>

</div>

</div>

</div>

);

}

}

export default Todolist;

修改代码之后：

import React,{Component} from "react";

import '../assets/css/ToDoLIST.css';

class **Todolist** extends **Component** {

constructor(props) {

super(props);

this.state = {

title:"ToDOList",

content:"",

now:[

{

isover:false,

title:"录制React教程",

ismove:false

}

],

over:[]

};

this.**enterKey**();

}

*//设置值*

**setContent**=(e)=>{

this.**setState**({

content:e.target.value

});

}

*// 按键事件*

**enterKey**=()=>{

let that=this;

document.**addEventListener**("keypress",function(e){

switch(e.charCode){

case 13:

let now= that.state.now;

now.**push**( {

isover:false,

title:that.state.content,

ismove:false

});

that.state.content="";

*//注意数据的双向问题*

that.**setState**({

now:now

});

break;

default: break;

}

})

}

*//设置移除*

**setMove**=(key)=>{

let now=this.state.now;

now[key].ismove=!now[key].ismove;

if(now[key].ismove)

{

now.**splice**(key,1);

}

this.**setState**({

now:now

});

}

*//设置完成*

**setOver**=(key)=>{

let now=this.state.now;

now[key].isover=!now[key].isover;

this.**setState**({

now:now

});

}

**render**() {

return (

<div *className*="todo">

<div *className*="title">

<span *className*="titleinfo">{this.state.title}</span>

<input *className*="addcontent" *placeholder*="添加ToDO" *value*={this.state.content} *onChange*={this.setContent}/>

</div>

<div *className*="list">

<div *className*="listnow">

<h3>正在进行</h3>

<ul *className*="menu">

{

this.state.now.**map**( (value,key)=>{

if(!value.isover)

{

return (

<li *key*={key}>

<input *type*="checkbox" *checked*={value.isover} *onChange*={this.setOver.**bind**(this,key)}/>

{value.title}

<input *type*="radio" *checked*={value.ismove} *onChange*={this.setMove.**bind**(this,key)}/>

</li>

);

}

else{

return ""

}

})

}

</ul>

</div>

<div *className*="listnow">

<h3>已经完成</h3>

<ul *className*="menu">

{

this.state.now.**map**( (value,key)=>{

if(value.isover)

{

return (

<li *key*={key}>

<input *type*="checkbox" *checked*={value.isover} *onChange*={this.setOver.**bind**(this,key)}/>

{value.title}

<input *type*="radio" *checked*={value.ismove} *onChange*={this.setMove.**bind**(this,key)}/>

</li>

);

}

else{

return ""

}

})

}

</ul>

</div>

</div>

</div>

);

}

}

export default Todolist;

22）React 种数据缓存（Storage）

使用的是Html5 中的storage 缓存

在以上案例中写入缓存：

localStorage.**setItem**("todolist",JSON.**stringify**(now));

在react生命周期函数中加载缓存

*//react 生命周期函数*

*// 页面加载就会触发 渲染页面*

**componentDidMount**(){

*// 读取缓存*

let list=localStorage.**getItem**("todolist");

if(list)

{

this.**setState**({

now:JSON.**parse**(list)

});

}

}

23）react 封装自定义模块优化代码性能（意思就是将某个功能封装为一个模块在任何组件里面都可以被使用）

封装的模块：

*//自定义模块的封装*

let Storage={

**setStorage**(key,value){

localStorage.**setItem**(key,JSON.**stringify**(value));

},

**getStorage**(key){

return JSON.**parse**(localStorage.**getItem**(key));

},

**removeStorage**(key){

localStorage.**removeItem**(key);

}

}

*//模块暴露*

export default Storage;

组件调用使用：

import React,{Component} from "react";

import '../assets/css/ToDoLIST.css';

*//引入自定义的模块*

import Storage from '../model/Storage.js'

class **Todolist** extends **Component** {

constructor(props) {

super(props);

this.state = {

title:"ToDOList",

content:"",

now:[

{

isover:false,

title:"录制React教程",

ismove:false

}

],

over:[]

};

this.**enterKey**();

}

*//设置值*

**setContent**=(e)=>{

this.**setState**({

content:e.target.value

});

}

*// 按键事件*

**enterKey**=()=>{

let that=this;

document.**addEventListener**("keypress",function(e){

switch(e.charCode){

case 13:

let now= that.state.now;

now.**push**( {

isover:false,

title:that.state.content,

ismove:false

});

that.state.content="";

*//注意数据的双向问题*

that.**setState**({

now:now

});

*//添加数据的时候 写入缓存*

Storage.**setStorage**("todolist",now);

break;

default: break;

}

})

}

*//设置移除*

**setMove**=(key)=>{

let now=this.state.now;

now[key].ismove=!now[key].ismove;

if(now[key].ismove)

{

now.**splice**(key,1);

}

this.**setState**({

now:now

});

*//修改数据的时候 写入缓存*

Storage.**setStorage**("todolist",now);

}

*//设置完成*

**setOver**=(key)=>{

let now=this.state.now;

now[key].isover=!now[key].isover;

this.**setState**({

now:now

});

*//修改数据的时候 写入缓存*

Storage.**setStorage**("todolist",now);

}

*//react 生命周期函数*

*// 页面加载就会触发 渲染页面*

**componentDidMount**(){

*// 读取缓存*

let list=Storage.**getStorage**("todolist");

if(list)

{

this.**setState**({

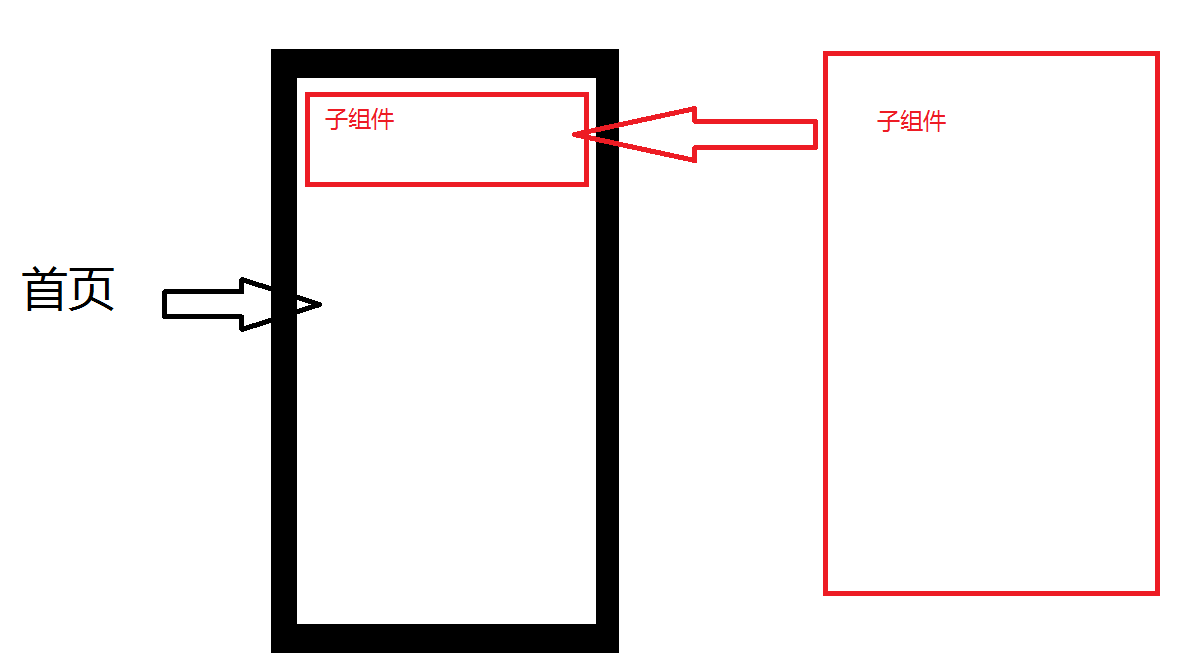
now:list

});

}

}

24）React中的组件，父子组件，父组件给子组件传值（props），子组件给父组件传值，父组件中通过refs获取子组件的属性和方法



组件的声明：

import React,{Component} from 'react';

class **Header** extends **Component** {

constructor(props) {

super(props);

this.state = { };

}

**render**() {

return (

<div>

我是子组件

</div>

);

}

}

export default Header;

父组件给子组件传值：

<**Header** *dataname*={this.state.name}></**Header**>

import React,{Component} from 'react';

class **Header** extends **Component** {

constructor(props) {

super(props);

this.state = { };

}

**render**() {

return (

<div>

{this.props.dataname}

</div>

);

}

}

export default Header;

父组件给子组件传值：可以传值，传方法，传整个组件都可以，在子组件里面实现父组件的方法，属性的执行和调用。

子组件给父组件传值：可以借助父组件传方法给子组件，子组件调用方法传参，到父组件的方法当中。

父：

class **App** extends **Component** {

constructor(props){

super(props);

this.state={

name:"首页组件"

}

}

**getData**=(msg)=>{

**alert**(msg);

}

**render**() {

return (

<div *className*="App">

<hr/>

<**Header** *dataname*={this.state.name} *getmethod*={this.getData}></**Header**>

</div>

);

}

}

子：

import React,{Component} from 'react';

class **Header** extends **Component** {

constructor(props) {

super(props);

this.state = {

name:"给父组件传值"

};

}

*//声明周期加载函数*

**componentDidMount**(){

this.props.**getmethod**(this.state.name);

}

**render**() {

return (

<div>

{this.props.dataname}

</div>

);

}

}

export default Header;

父组件页可以直接获取子组件对象

给子组件添加属性ref 在父组件里面直接this.refs.name 可以拿到子组件

**componentDidMount**(){

let child=this.refs.header.state.name;

**alert**(child);

}

**render**() {

return (

<div *className*="App">

<hr/>

<**Header** *ref*="header" *dataname*={this.state.name} *getmethod*={this.getData}></**Header**>

</div>

);

}

25）父组件给子组件传值：

defaultProps:父组件传值中，如果父组件调用子组件不传值，可以在子组件中使用defaultProps定义默认的值；

propTypes:验证传值的合法性；

这两个都是给子组件使用的；

import React,{Component} from 'react';

*//先导入*

import PropTypes from 'prop-types';

class **Header** extends **Component** {

constructor(props) {

super(props);

this.state = {

name:"给父组件传值"

};

}

*//声明周期加载函数*

**componentDidMount**(){

this.props.**getmethod**(this.state.name);

}

**render**() {

return (

<div>

{this.props.dataname}

</div>

);

}

}

*//定义传值的默认*

Header.defaultProps={

dataname:"默认"

}

*//约束传值的类型*

Header.propTypes={

dataname:PropTypes.string,

numinfo:PropTypes.number,

datalist:PropTypes.bool

}

export default Header;

26） React 获取服务器的数据axios插件（类似cros 跨域） fetch-jsonp插件的使用

<https://www.npmjs.com> 上可以查找js模块

使用axios模块：

1）安装模块 npm install axios --save/cnpm install axios --save

2) 引入模块 import axios from ‘axios’

3) 使用

安装完成之后：

{

"name": "reactdemo01",

"version": "0.1.0",

"private": true,

"dependencies": {

"axios": "^0.18.0",

"react": "^16.8.3",

"react-dom": "^16.8.3",

"react-scripts": "2.1.5"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": "react-app"

},

"browserslist": [

">0.2%",

"not dead",

"not ie <= 11",

"not op\_mini all"

]

}

使用axios请求服务器数据：

import React,{Component} from 'react';

*//引入请求服务器的模块*

import axios from 'axios';

class **Axios** extends **Component** {

constructor(props) {

super(props);

this.state = {

};

}

**componentDidMount**(){

let api="http://www.phonegap100.com/appapi.php?a=getPortalList&catid=20";

//里面是用this 注意this的指向

axios.**get**(api).**then**(function (res){

**console**.**log**(res);

}).**catch**(function (res){

**console**.**log**(res);

});

}

**render**() {

return (

<div>

<span>请求服务器的数据</span>

</div>

);

}

}

export default Axios;

axios 插件用法：

<https://www.npmjs.com/package/axios>

fetch-jsonp 组件使用 jsonp请求

1. 安装
2. 引入
3. 写代码

<https://www.npmjs.com/package/fetch-jsonp>

{

"name": "reactdemo01",

"version": "0.1.0",

"private": true,

"dependencies": {

"axios": "^0.18.0",

"fetch-jsonp": "^1.1.3",

"react": "^16.8.3",

"react-dom": "^16.8.3",

"react-scripts": "2.1.5"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": "react-app"

},

"browserslist": [

">0.2%",

"not dead",

"not ie <= 11",

"not op\_mini all"

]

}

使用fetch-jsonp 请求服务器数据：

import React from 'react'

import fetchjsonp from 'fetch-jsonp'

class **Fetchjsonp** extends **React**.**Component** {

constructor(props) {

super(props);

this.state = {

};

}

**componentDidMount**(){

let api="http://www.phonegap100.com/appapi.php?a=getPortalList&catid=20";

**fetchjsonp**(api).**then**(function (res){

*//这里对数据进行操作 默认的*

return res.**json**();

}).**then**(function (res){

**console**.**log**(res);

}).**catch**(function (res){

**console**.**log**(res);

});

}

**render**() {

return (

<div>

<span>jsonp请求数据</span>

</div>

);

}

}

export default Fetchjsonp;

fetch-jsonp 官网用法：<https://www.npmjs.com/package/fetch-jsonp>

27） React 使用的生命周期函数

import React from 'react';

class **Life** extends **React**.**Component** {

constructor(props) {

**console**.**log**("类的构造函数");

super(props);

this.state = {

msg:"222"

};

}

*//生命周期函数*

**componentWillMount**(){

**console**.**log**("组件将要挂载");

}

*//生命周期函数*

**componentDidMount**(){

**console**.**log**("组件挂载完成");

}

*//在数据更新的时候触发的生命周期函数*

**shouldComponentUpdate**(){

*//是否更新数据的生命周期函数 有返回值*

return true;

}

*//将要更新数据的函数*

**componentWillUpdate**(){

**console**.**log**("数据将要更新");

}

*//数据更新完成*

**componentDidUpdate**(){

**console**.**log**("数据更新完成");

}

*//组件销毁的时候执行的生命周期函数*

**componentWillUnmount**(){

}

**setmsg**=()=>{

this.**setState**({

msg:"333"

});

}

**render**() {

**console**.**log**("渲染虚拟的dom页面");

return (

<div>

生命周期函数

<button *onClick*={this.setmsg}>更新数据</button>

</div>

);

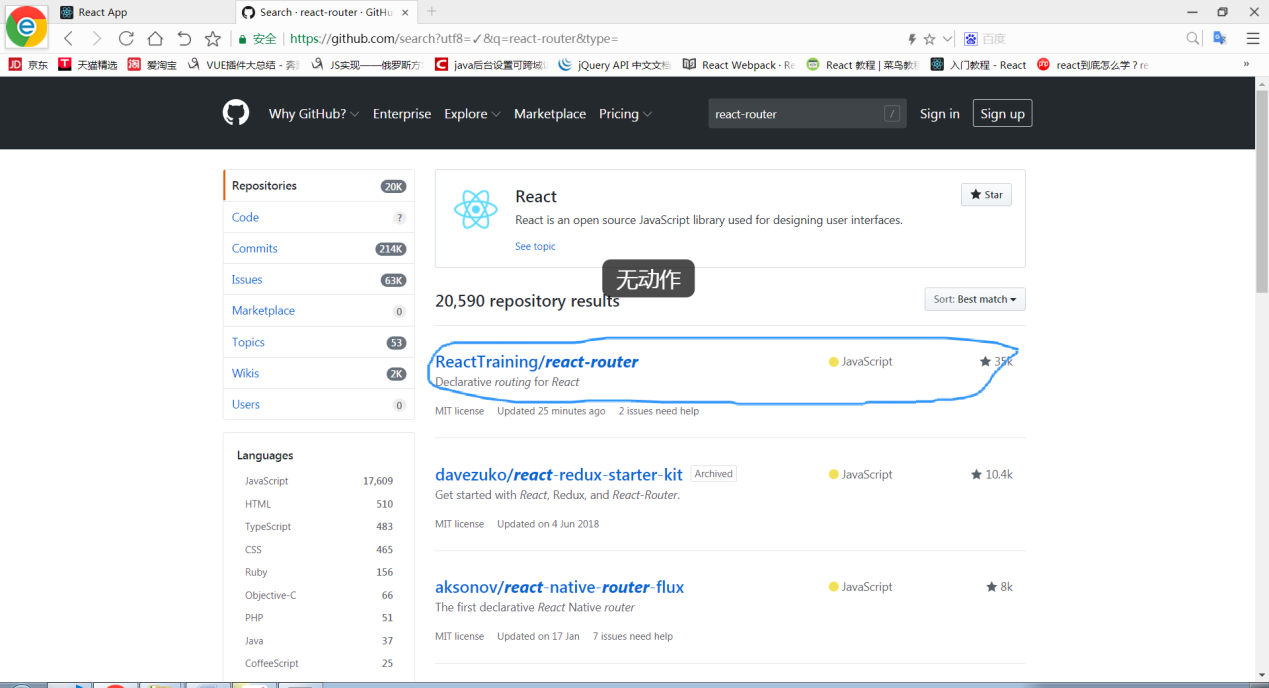
}

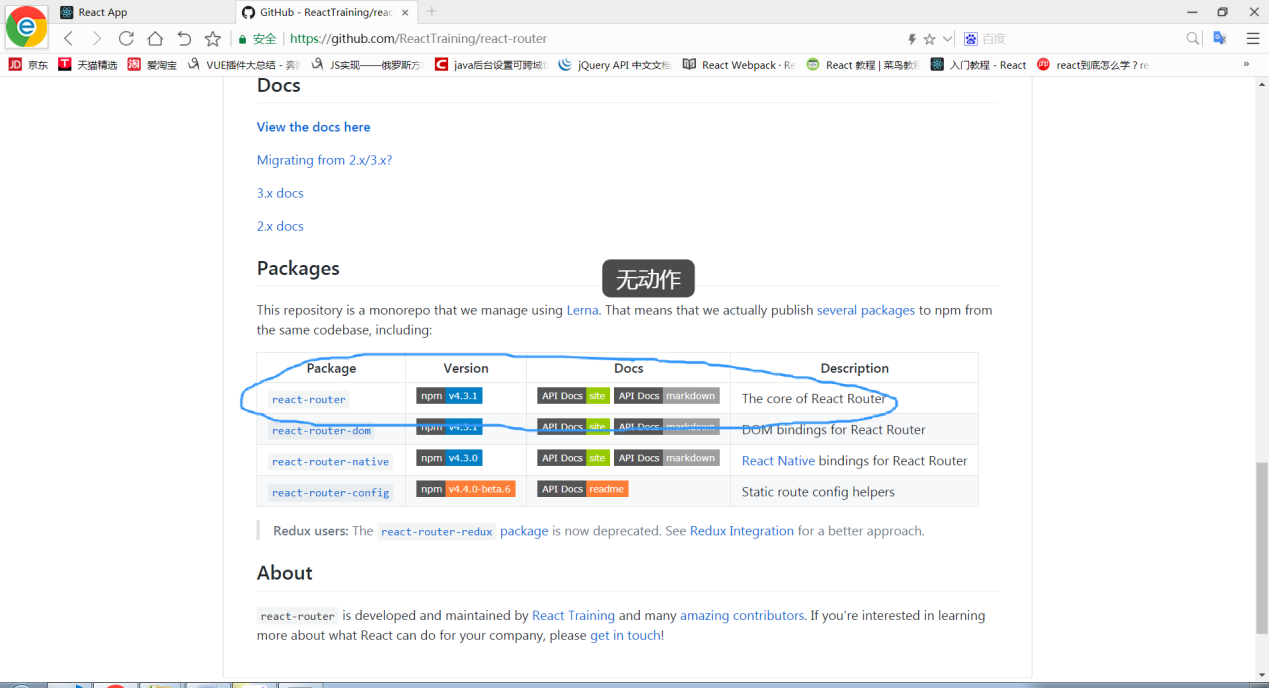
}

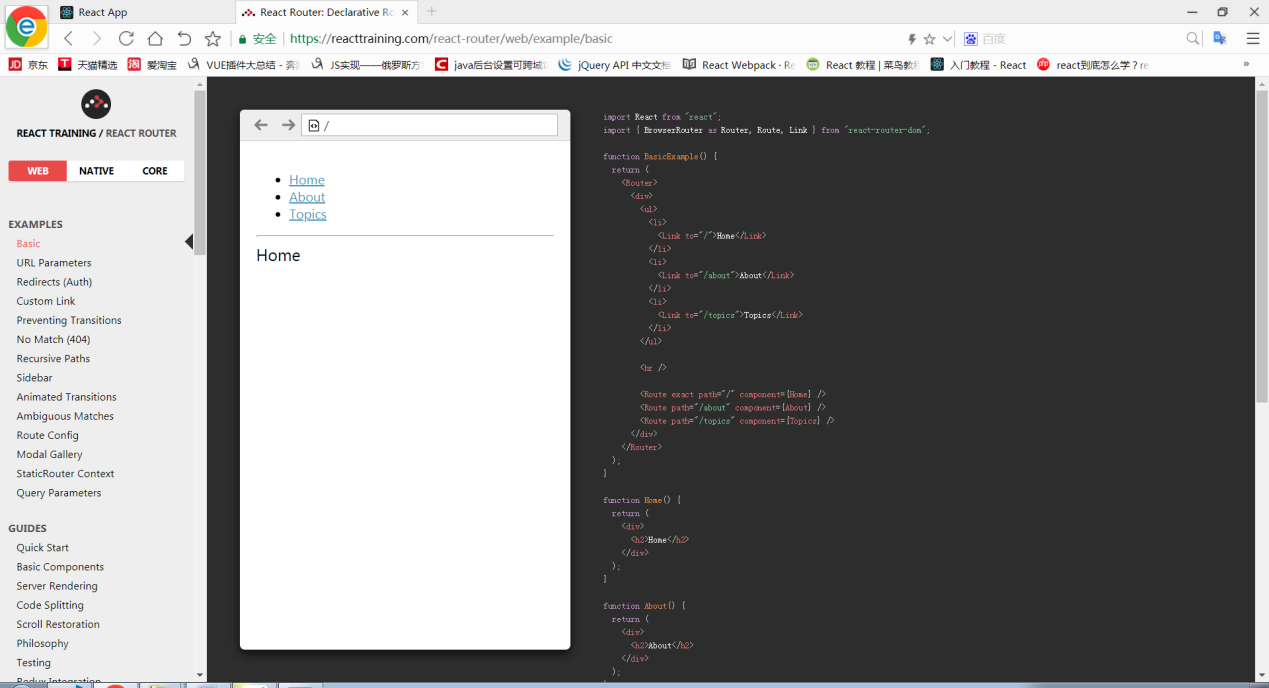
export default Life;

28）React 路由基本配置 react-router 4.x

在github官网上搜索react-router







官网上有说明使用方式：

1）react 路由配置： 先安装路由：c

路由安装完毕之后 配置文件

{

"name": "reactdom02",

"version": "0.1.0",

"private": true,

"dependencies": {

"react": "^16.8.3",

"react-dom": "^16.8.3",

"react-router-dom": "^4.3.1",

"react-scripts": "2.1.5"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": "react-app"

},

"browserslist": [

">0.2%",

"not dead",

"not ie <= 11",

"not op\_mini all"

]

}

安装路由建议使用cnpm install react-router-dom –save 不要使用npm以防项目报错

import React, { Component } from 'react';

import './assets/css/App.css';

*// 要在项目里面使用路由先导入路由*

import {BrowserRouter as Router, Route,Link} from 'react-router-dom';

*//导入路由子组件*

import Home from './component/Home'

import News from './component/News'

import Product from './component/Product'

class **App** extends **Component** {

**render**() {

return (

<**Router**>

<div>

<**Link** *to*="/">Home</**Link**>

<br/>

<**Link** *to*="/News">News</**Link**>

<br/>

<**Link** *to*="/Product">Product</**Link**>

<hr/>

<**Route** *exact* *path*="/" *component*={Home}/>

<**Route** *path*="/News" *component*={News}/>

<**Route** *path*="/Product" *component*={Product}/>

</div>

</**Router**>

);

}

}

export default App;

exact 为Route的属性 为bool类型 true为严格匹配 false为普通匹配

29） React里面的动态路由以及get传值：

1）动态路由传值

import React, { Component } from 'react';

import './assets/css/App.css';

*// 要在项目里面使用路由先导入路由*

import {BrowserRouter as Router, Route,Link} from 'react-router-dom';

*//导入路由子组件*

import Home from './component/Home'

import News from './component/News'

import Product from './component/Product'

import Content from './component/content'

class **App** extends **Component** {

**render**() {

return (

<**Router**>

<div>

<**Link** *to*="/">Home</**Link**>

<br/>

<**Link** *to*="/News">News</**Link**>

<br/>

<**Link** *to*="/Product">Product</**Link**>

<hr/>

<**Route** *exact* *path*="/" *component*={Home}/>

<**Route** *path*="/News" *component*={News}/>

<**Route** *path*="/Product" *component*={Product}/>

<**Route** *path*="/Content/:id" *component*={Content}/>

</div>

</**Router**>

);

}

}

export default App;

import React,{Component} from 'react';

import {Link} from 'react-router-dom';

class **News** extends **Component** {

constructor(props) {

super(props);

this.state = {

list:[

{

id:1,

title:"我是新闻1"

},

{

id:2,

title:"我是新闻2"

},

{

id:3,

title:"我是新闻3"

},

{

id:4,

title:"我是新闻4"

}

]

};

}

**render**() {

return (

<div>

{

this.state.list.**map**((value,key)=>{

return (

<li *key*={key}>

{*/\* 在这里实现动态路由的传参 \*/*}

{*/\* 直接写固定值可以传值 也可以写动态值传值 \*/*}

{*/\* <Link to="/Content/11">{value.title}</Link> \*/*}

{*/\* 也可以使用es6里面的模板语法 \*/*}

<**Link** *to*={`/Content/${value.id}`}>{value.title}</**Link**>

</li>

);

})

}

</div>

);

}

}

export default News;

import React,{Component} from 'react';

class **content** extends **Component** {

constructor(props) {

super(props);

this.state = {

id:0

};

}

**componentDidMount**(){

*//动态路由传过来的值在props里面的match上*

this.**setState**({

id:this.props.match.params.id

});

}

**render**() {

return (

<div>

<div>我是新闻详情组件</div>

{this.state.id}

</div>

);

}

}

export default content;

get传值 相当于在路径上加？id=1

import React,{Component} from 'react';

import {Link} from 'react-router-dom';

class **Product** extends **Component** {

constructor(props) {

super(props);

this.state = {

list:[

{

id:1,

title:"我是商品1"

},

{

id:2,

title:"我是商品2"

},

{

id:3,

title:"我是商品3"

},

{

id:4,

title:"我是商品4"

}

]

};

}

**render**() {

return (

<div>

{

this.state.list.**map**((value,key)=>{

return (

<li *key*={key}>

<**Link** *to*={`/productcontent?id=${value.id}`}>{value.title}</**Link**>

</li>

);

})

}

</div>

);

}

}

export default Product;

import React,{Component} from 'react';

import url from 'url';

class **Productcontent** extends **Component** {

constructor(props) {

super(props);

this.state = {

id:0

};

}

**componentDidMount**(){

*// 通过get方式获取路由传参*

*// ?id=2 对于这样的值可以自己写方法解析 也可以导入node的url模块进行解析*

*// 值就在query 上*

*// 安装url 模块 cnpm install url --save*

**console**.**log**(url.**parse**(this.props.location.search,true));

}

**render**() {

return (

<div>

<div>我是新闻详情组件</div>

{this.state.id}

</div>

);

}

}

export default Productcontent;

获取出来的值？id=2 可以自己写方法解析，也可以直接使用node 的url模块解析

安装url模块 cnpm install url –save

官网：[Material-UI](https://material-ui.com/)

<https://material-ui.com/getting-started/installation/>

// 用npm安装

npm install @material-ui/core

// 用yarn安装

yarn add @material-ui/core