day15笔记

1、react介绍

- 基础介绍
  - 文档地址：https://react.docschina.org/
  - Facebook
  - 版本：v16 稳定！
  - 2013年开源！
- 特点
  - 声明式
  - 组件化
  - 跨终端（ReactNative）
- 描述：**用于构建用户界面的 JavaScript 库**

2、react基础语法

2.1、基础使用

- 引入react
  - 核心方法： **React.createElement(标签名，标签属性，内容)** 创建出来一个React元素标签，一个虚拟DOM节点！
- 引入react-dom
  - 核心方法： **ReactDOM.render(react元素、挂载的节点、回调函数)** 将React元素节点，挂载到真实DOM节点上面！
- 基础使用：

```
let p1 =  React.createElement('p',{key:1},'我是第一段话的内容！')
let p2 = React.createElement('p', {key:2},'我是话的内容')
let h3 = React.createElement('h3',{className:'title',key:3},'中原一点红！')
let div = React.createElement('div',{id:'box'},[h3,p1,p2])

ReactDOM.render(div,document.getElementById("app"))
```

- **由于实际开发不可能使用createElement创建元素，因为很慢。所以有了JSX的语法糖来实现他！**

  需要babel解析一下

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>02、JSX使用</title>
    <!--1、 引入react -->
    <script src="./js/react.development.js"></script>
    <!--2、 引入react-dom -->
    <script src="./js/react-dom.development.js"></script>
    <!--3、引入babel  -->
    <script src="./js/babel.min.js"></script>
</head>
<body>
    <div id="app">
    </div>
</body>
<script type="text/babel">
// 一定要修改type为babel!!!
    // JSX 本质就是做了React.createElement!!! 就是他的语法糖!

    let vdom =  (<div id="box">
                    <h3 className="title">中原一点红! </h3>
                    <p>我是第一段话的内容! </p>
                    <p>我是话的内容!</p>
                </div>)
    ReactDOM.render(vdom,document.getElementById("app"))


</script>
</html>
```

2.2、模板语法

- JSX语法要求

```
//1.渲染变量:  {变量/表达式}
        // 不能输入布尔值!
        // 条件判断可以，只是不能输出布尔值!
        // 对象不能直接输出!
//2.注释语法:  {/* 注释内容 */}
//3.标签一定要闭合，特别注意单标签!  br,hr,img,input
//4.属性绑定
        // <标签 属性名={变量/表达式} />
//5.属性注意
        // class 要换成className
        // label的for换成htmlFor
        // 多个单词组成的属性，要变成驼峰命名!  如果 colspan,rowspan,onclick,等等!
//6.标签的嵌套要遵循规则!
//7.数组的渲染，如果数组的元素都是字符串，那么直接拿去渲染（放在一起，符号都没有）！如果数组的元素是react元素标签，也直接渲染！如果都是react元素标签，每个react元素标签要有key属性!
```

```
//8.渲染HTML字符串：<标签 dangerouslySetInnerHTML={{__html:变量}}></标签>
```

- 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>03、JSX语法</title>
    <!--1、 引入react -->
    <script src="./js/react.development.js"></script>
    <!--2、 引入react-dom -->
    <script src="./js/react-dom.development.js"></script>
    <!--3、引入babel  -->
    <script src="./js/babel.min.js"></script>
</head>
<body>
    <div id="app">

    </div>
</body>
<script type="text/babel">
    let msg = '你好React! '
    let person = {
        name:"张飞",
        age:20
    }
    let mytpl = (<div className="tpl">
                    <h3>我是故事！！</h3>
                </div>)
    function sayName(name){
        return `我的名字是${name}`
    }
    let nums =1000;
    let arr1 = ['小刘','小张','小强','小红','翠花','铁蛋','狗剩']
    let arr2 = [<b key='1'>我是加粗</b>, <i key='2'>我是斜体</i>, <u key='3'>我是下
划线</u>]
    let imgurl = "https://timgsa.baidu.com/timg?
image&quality=80&size=b9999_10000&sec=1600405521930&di=64cc268cce68facb3e5f357f522
4b8f5&imgtype=0&src=http%3A%2F%2Faliyunimg.9ku.com%2Fpic%2Ftoutiao%2F20180803%2Ff2
6db75f65a134e844dba3039ea30310.jpeg%3Fx-oss-
process%3Dimage%2Fresize%2Cw_600%2Cq_60"
    let articleContent = "<div>我是div，<h3>我是h3</h3></div>"

    let vdom =  (<div className="box">
                    <h3>{msg}</h3>
                    <p>{1+1}</p>
```

```
                    <p>{true}</p>
                    <p>{10>20 ? '111':'222' }</p>
                    {/* 不能直接输出对象！ */}
                    {/* <p>{person}</p>*/}
                    {mytpl}
                    {sayName('哈哈')}==========
                    {nums}
                    <hr/>
                    {arr1}
                    <hr/>
                    {arr2}
                    <img src={imgurl}  />
                    <h3 id={'box'+nums}>我是id值为box1000的h3</h3>
                    <label htmlFor=""></label>
                    <table border="1" width="500" cellSpacing="0" cellPadding="0">
                        <tbody>
                            <tr>
                                <td colSpan="2">2</td>
                                <td>3</td>
                            </tr>
                        </tbody>
                    </table>
                    <hr/>
                    <div dangerouslySetInnerHTML={{__html:articleContent}}></div>
                </div>)

    ReactDOM.render(vdom,document.getElementById("app"))
</script>
</html>
```

2.3、样式处理

- 操作class，操作style
- 注意：**style不能直接写样式描述字符串，不能够解析！**
- 样式处理

```
// 处理样式！
    // 处理style   style={{样式属性名1:样式属性值1,样式属性名2:样式属性值2}}   //注意驼峰命名法！
    // 处理class   属性绑定！
```

- 代码

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>04、样式处理</title>
    <!--1、 引入react -->
    <script src="./js/react.development.js"></script>
    <!--2、 引入react-dom -->
    <script src="./js/react-dom.development.js"></script>
    <!--3、引入babel  -->
    <script src="./js/babel.min.js"></script>
    <style>
        *{
            padding: 40px;
        }
        .red{
            color: red;
        }
        .blue{
            color: blue;
        }
    </style>
</head>

<body>
    <div id="app">

    </div>
</body>
<script type="text/babel">
    let c1 = 'red'

    let stylename = {
        border: "2px solid green",
        color: "#fff",
        lineHeight: "40px",
        backgroundColor: "orange"
    }

    let vdom = (<div className='box'>
                    <h3 className={c1}>111</h3>
                    <h3 className={10>20 ? 'red':'blue'}>111</h3>
                    {/*不能直接写样式描述字符串，不能够解析！*/}
                    {/*<h3 style="border:2px solid blue;color:orange">111</h3>*/}
                    {/* style只能接收一个样式描述对象！！！  */}
                    <h3 style={{border:"2px solid
blue",color:"orange",lineHeight:"50px",backgroundColor:"#eee"}}>111</h3>
                     <h3 style={stylename}>111</h3>
                </div>)
```

```
    ReactDOM.render(vdom, document.getElementById("app"))


</script>

</html>
```

2.4、条件渲染

- 本质：**依靠三木运算展示不同的JSX片段！**
- 实现

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>05、条件渲染</title>
    <!--1、 引入react -->
    <script src="./js/react.development.js"></script>
    <!--2、 引入react-dom -->
    <script src="./js/react-dom.development.js"></script>
    <!--3、引入babel  -->
    <script src="./js/babel.min.js"></script>
    <style>
        body{
            padding: 20px;
        }
        .hide{
            display: none;
        }
    </style>
</head>

<body>
    <div id="app">
        <!-- <div class="list">
            <ul>
                <li>111</li>
                <li>222</li>
                <li>333</li>
            </ul>
            <p>暂无数据！ </p>
        </div> -->
    </div>
</body>
```

```
<script type="text/babel">


    let tpl1 = (<ul>
                    <li>111</li>
                    <li>222</li>
                    <li>333</li>
                </ul>)
    let tpl2 = (<p>暂无数据! </p>);

    let state = true;


    let vdom = (<div className="list">
                    {state ? tpl1 : tpl2 }
                    <hr/>
                    { state ?
                        <ul>
                            <li>111</li>
                            <li>222</li>
                            <li>333</li>
                        </ul>
                        :
                        <p>暂无数据! </p>
                    }
                    <hr/>
                    <h3>模拟v-show</h3>
                    <ul className={!state ? 'hide':''}>
                        <li>111</li>
                        <li>222</li>
                        <li>333</li>
                    </ul>
                    <p className={state ? 'hide' : ''}>暂无数据! </p>

                </div>)

    ReactDOM.render(vdom, document.getElementById("app"))


</script>

</html>
```

2.5、列表渲染

- 思路：**react渲染数组！将数据数组变成一个react元素数组，然后再去渲染该react元素数组！**
- 实现：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>06、列表渲染</title>
    <!--1、 引入react -->
    <script src="./js/react.development.js"></script>
    <!--2、 引入react-dom -->
    <script src="./js/react-dom.development.js"></script>
    <!--3、引入babel  -->
    <script src="./js/babel.min.js"></script>
    <style>
        body {
            padding: 20px;
        }

        .hide {
            display: none;
        }
    </style>
</head>

<body>
    <div id="app">
         <!-- <div class="list">
             <ul>
                <li>
                     <h3>姓名：xxx</h3>
                     <p><b>性别：xxx</b><i>年龄：xx</i></p>
                     <ol>
                         <li>学科：xxx==>分数：xxx</li>
                     </ol>
                </li>
             </ul>
         </div> -->
    </div>
</body>
<script type="text/babel">

    let userdata = [
        {
            name:"铁蛋",
            age:20,
```

```
            sex:1,
            score:[
                {type:"语文",num:90},
                {type:"数学",num:100},
                { type: "英语", num: 60 }
            ]
        },
        {
            name: "翠花",
            age: 19,
            sex: 2,
            score: [
                { type: "语文", num: 10 },
                { type: "数学", num: 10 },
                { type: "英语", num: 90 }
            ]
        },
        {
            name: "二狗子",
            age: 30,
            sex: 1,
            score: [
                { type: "语文", num: 80 },
                { type: "数学", num: 30 },
                { type: "英语", num: 60 }
            ]
        },
        {
            name: "狗剩儿",
            age: 25,
            sex: 1,
            score: [
                { type: "语文", num: 70 },
                { type: "数学", num: 30 },
                { type: "英语", num: 20 }
            ]
        }
    ]

    // let listvdom = [<li><h3>姓名：111</h3> <p><b>性别：22</b><i>年龄：33</i></p></li>,<li><h3>姓名：111</h3> <p><b>性别：22</b><i>年龄：33</i></p></li>,<li><h3>姓名：111</h3> <p><b>性别：22</b><i>年龄：33</i></p></li>]
    let listvdom = userdata.map((val,idx)=>{
        return (<li key={idx}>
                    <h3>姓名：{val.name}</h3>
                    <p><b>性别：{val.sex == 1 ? '男' : '女'}</b><i>年龄：{val.age}</i></p>
                    <ol>
                        {val.score.map((v,i)=>{
```

```
                            return <li key={i}>学科：{v.type}==&gt;分数：{v.num}
</li>
                        })}
                    </ol>
                </li>)
    })

    let vdom = (<div className="list">
                    <ul>
                        {listvdom}
                    </ul>
                    <hr/>
                    <ul>
                        {userdata.map((val, idx) => {
                            return (<li key={idx}>
                                <h3>姓名：{val.name}</h3>
                                <p><b>性别：{val.sex == 1 ? '男' : '女'}</b><i>年
龄：{val.age}</i></p>
                                <ol>
                                    {val.score.map((v, i) => {
                                        return <li key={i}>学科：{v.type}==&gt;分
数：{v.num}</li>
                                    })}
                                </ol>
                            </li>)
                        })}
                    </ul>
                </div>)


    ReactDOM.render(vdom, document.getElementById("app"))


</script>

</html>
```

2.6、组件的注册

- 组件的分类
  - 函数式组件

```
function 组件名(){
  return JSX
}
let 组件名 = ()=>(JSX)
```

  - 类组件

```
class 组件名 extends React.Component{
  render(){
      return JSX
  }
}
```

- 组件注意：
  - 组件名首字母大写！
  - 组件只能有一个根标签！
- 组件的使用

```
<组件名/>
```

- 代码

```
  function Box(){
        return (<div className="item">
                    <h3>
                        <span>标题</span>
                        <a href="">更多</a>
                    </h3>
                    <div className="content">
                        <p>这是一个段落</p>
                    </div>
                </div>)
    }
    let Kaixin = ()=> (<div className="kaixin">
                            <h3 className='tit'>
                            我是开心组件！
                            </h3>
                            <p>我是开心组件的内容！ </p>
                        </div>)

    class Banner extends React.Component{
        render(){
            return (<div className='banner'>
                        <h1>我是Banner啊！ </h1>
                        <p>我是Banner啊啊啊！ </p>
                        <Kaixin/>
                    </div>)
        }
    }

    let vdom = (<div id="box">
                    <h1>你好！ ！ ！ </h1>
                    <Box/>
                    <Banner/>
```

```
                    </div>)
    ReactDOM.render(vdom,document.getElementById("app"))
```

3、脚手架

3.1、安装脚手架

- 脚手架名称：create-react-app
- 脚手架官方文档：
  - 英文文档：https://create-react-app.dev/
  - 中文文档：https://www.html.cn/create-react-app/docs/getting-started/
- 安装脚手架：

```
npm  i  create-react-app  -g    // 全局安装
cnpm i  create-react-app  -g
```

- 检测是否安装成功

```
create-react-app -V    // 显示版本号说明安装成功
```

3.2、初始化项目

- npx直接初始化项目，不需要安装脚手架

```
npx create-react-app  项目名称
```

- 脚手架初始项目命令
  - 找到项目存放的目录位置，进入该位置的cmd窗口
  - 执行初始化命令

```
create-react-app 项目名称    // 初始化项目    默认是使用的yarn下载包依赖，如果没有yarn则自动使用npm
```

- 启动项目
  - 进入项目目录

```
cd 项目名称
```

  - 启动项目

```
npm start
// 如果安装了yarn可以执行
yarn start
```

- **题外知识(了解即可！)**
  - yarn的安装 【yarn是一个包管理工具，和npm一样！】
    - 文档地址：https://yarn.bootcss.com/docs/install/#windows-stable
    - 安装yarn

      ```
      cnpm i yarn -g    // 全局安装yarn
      npm i yarn -g
      ```

    - 检测安装是否成功

      ```
      yarn -version   // 输出版本号说明安装成功
      ```

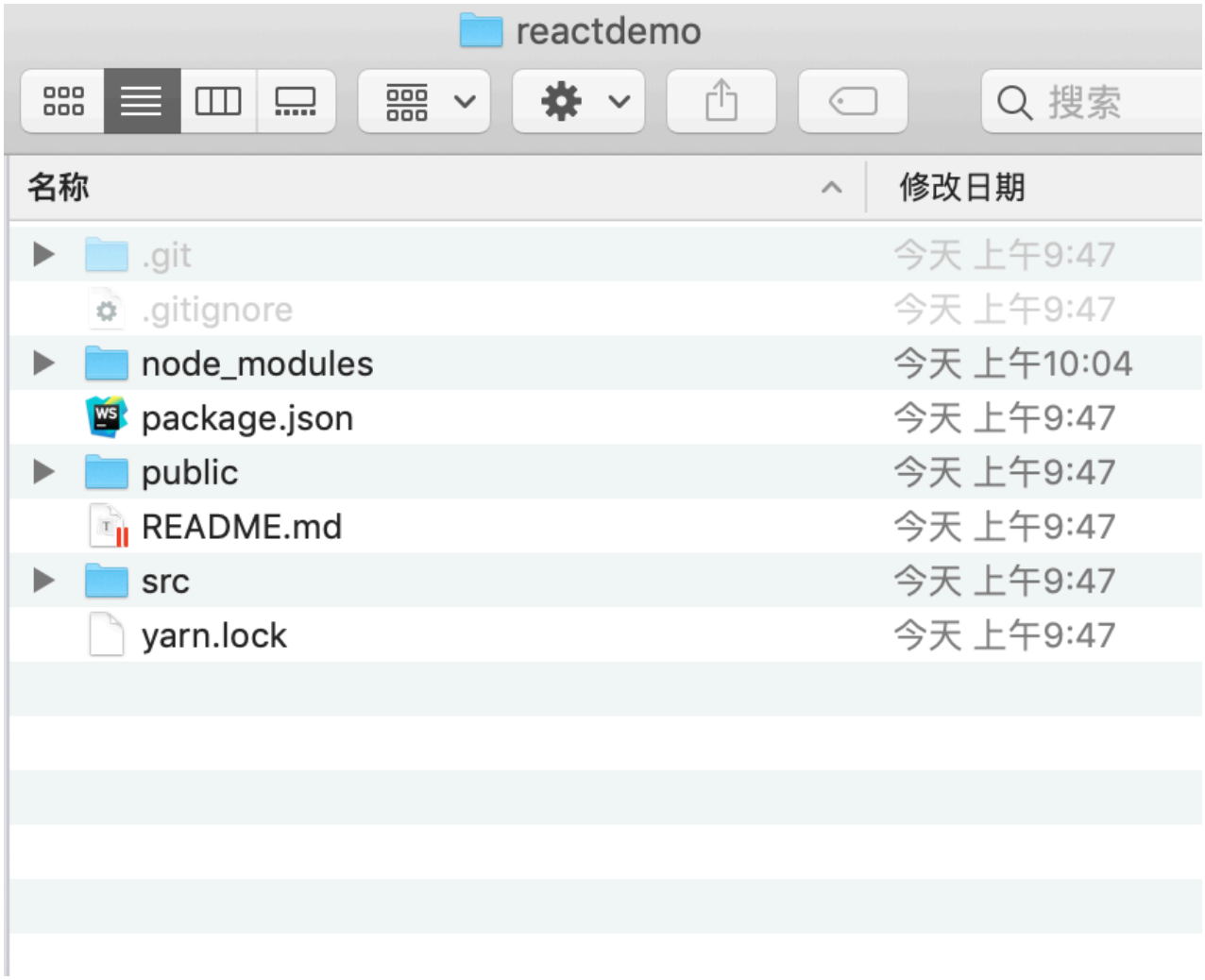
  - yarn的常用指令

    ```
    yarn init   // 初始化项目  类似于 npm init
    yarn add 包名     // 安装一个包   类于  npm install  包名
    yarn upgrade 包名     // 更新一个包   类于  npm update  包名
    yarn remove  包名   // 删除某个包    类似于  npm uninstall 包名

    // 安装项目所有依赖，自动读取项目下面的package.json文件去下载所有依赖包，类似于 npm i
    yarn
    或者
    yarn install
    ```

  - yarn比npm快，但是第一次是一样的，因为yarn他会吧下载过的包缓存起来，下次再下载就会很快！而npm是每次都会重新下载！

3.3、目录结构




```
reactdemo
|== git              //文件夹  不用管，是版本管理文件夹
|== gitignore        //文件    不用管，  是版本管理时候忽略的文件
|== node_modules     //文件夹  项目依赖包文件夹
|== package.json     //文件    包说明文件
|== public           //文件夹   静态文件夹，类似于Vue-cli 项目里面的static
|==|== favicon.ico         // 网页图标  不用管，
|==|== index.html          // 单页面应用的单页面
|==|== manifest.json       // 缓存配置  不用管，
|==|== robots.txt          // 目录抓取  不用管，
|== README.md        // 说明文件，不用管
|== src              //文件夹 【 开发目录！】
|==|== App.css       //文件    根组件样式  ((可以不要，自己写)
|==|== App.js        //文件    根组件      (可以自己写)
|==|== App.test.js   //文件    测试文件    (不用管,可删除)
|==|== index.css     //文件    全局样式文件 (可以不要，自己写)
|==|== index.js      // 项目入口文件，类似于Vue-cli 里面的main.js  【非常重要！】
|==|== serviceWorker// 服务测试文件   (不用管,可删除)
|==|== setupTests.js// 服务测试文件   (不用管,可删除)
```

3.4、文件解释

- index.js 入口文件

```
import React from 'react';  // 引入react
import ReactDOM from 'react-dom';   // 引入react-dom
import './index.css';    //  全局的css文件
import App from './App';  // 导入根组件
import * as serviceWorker from './serviceWorker';   // 离线测试

// 渲染根组件到react实例上面!
ReactDOM.render(<App /> ,document.getElementById('root'));

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();  // 离线测试
```

- App.js 根组件

```
import React from 'react';
import logo from './logo.svg';  // 导入一张svg图
import './App.css';             // 导入app.css样式

// 函数式创建一个App组件!
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
// 暴露组件
export default App;
```

3.5、注意点

- 引入css文件

```
// 把css作为一个模块引入
import "css的路径"
```

- 引入本地图片
  - 使用import作为模块导入

```
import  变量  from "图片的路径"
<img src={ 变量 } />
```

  - 使用require引入

```
<img src={ require("图片路径") } />
```

  - 外链的网络图片没有问题
- 代码

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

// 导入css文件
import "./assets/css/app.css"

// 图片引入解决
// 方式1:
import pq from "./assets/img/pq.jpg";

class App extends Component {
    render() {
        return (
            <div className='app'>
                我是根组件啊!
                {/* 不能按照相对路径去引入本地图片!  */}
                {/* <img src="../assets/img/pq.jpg" /> */}
                {/* 方式1 */}
                <img src={pq} />
                {/* 方式2 */}
                <img src={ require("./assets/img/pq.jpg") } />
                {/* 外链图片没有问题 */}
                <img
src="http://t9.baidu.com/it/u=3363001160,1163944807&fm=79&app=86&size=h300&n=0&g=4n&f=jpeg?sec=1587697249&t=41b2a517a6eaf2499b50f036fc225936" />
            </div>
        );
```

```
    }
}

// 暴露
export default App;
```

- **所有的import语句必须放在最上面**，写完了import语句之后才可以写 常规JS代码。

4、事件绑定

4.1、事件绑定

- **方式1**

```
class Box extends Component {
    fn1(){
        alert("事件绑定1")
    }
    render() {
        return (
            <div className="box">
                <button onClick={this.fn1 }>事件绑定1</button>
            </div>
        );
    }
}
```

这种方式绑定事件，事件函数里面的this是undefined。

不能加括号，加了就是直接调用，所以不能传参

- **方式2**

在绑定的时候加上bind实现this的固定。

```
<标签 onClick="this.事件函数名.bind(this)" />
```

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

class Box extends Component {
    constructor(){
        super()
    }
    fn1(){
        alert("事件绑定1")
        console.log(this)
```

```
    }
    fn2(){
        alert("事件绑定2")
        console.log(this)
    }
    render() {
        return (
            <div className="box">
                <button onClick={this.fn1}>事件绑定1</button>
                {/* 固定this的指向 */}
                <button onClick={this.fn2.bind(this)}>事件绑定2</button>
            </div>
        );
    }
}

export default Box;


// function 函数名(){}
// let 函数名 = function(){}
// new Function()
```

- **方式3**

  在构造器里面就定义好事件函数的this绑定

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

class Box extends Component {
    constructor(){
        super()
        this.fn3 = this.fn3.bind(this)
    }
    fn1(){
        alert("事件绑定1")
        console.log(this)
    }
    fn2(){
        alert("事件绑定2")
        console.log(this)
    }
    fn3() {
        alert("事件绑定3")
        console.log(this)
    }
    render() {
        return (
```

```
            <div className="box">
                <button onClick={this.fn1}>事件绑定1</button>
                {/* 固定this的指向 */}
                <button onClick={this.fn2.bind(this)}>事件绑定2</button>
                <button onClick={this.fn3}>事件绑定3</button>
            </div>
        );
    }
}

export default Box;


// function 函数名(){}
// let 函数名 = function(){}
// new Function()
```

- **方式4**

  定义事件函数的就是用箭头函数从而实现this的固定。

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

class Box extends Component {
    constructor(){
        super()
        this.fn3 = this.fn3.bind(this)
    }
    fn1(){
        alert("事件绑定1")
        console.log(this)
    }
    fn2(){
        alert("事件绑定2")
        console.log(this)
    }
    fn3() {
        alert("事件绑定3")
        console.log(this)
    }
    fn4 = ()=>{
        alert("事件绑定4")
        console.log(this)
    }
    render() {
        return (
```

```
            <div className="box">
                <button onClick={this.fn1}>事件绑定1</button>
                {/* 固定this的指向 */}
                <button onClick={this.fn2.bind(this)}>事件绑定2</button>
                <button onClick={this.fn3}>事件绑定3</button>
                <button onClick={this.fn4}>事件绑定4</button>
            </div>
        );
    }
}

export default Box;


// function 函数名(){}
// let 函数名 = function(){}
// new Function()
```

- 方式5

  间接操作： **在模板上事件绑定的是一个箭头函数，箭头函数里面 执行的我们类里面的事件函数**

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

class Box extends Component {
    constructor(){
        super()
        this.fn3 = this.fn3.bind(this)
    }
    fn1(){
        alert("事件绑定1")
        console.log(this)
    }
    fn2(){
        alert("事件绑定2")
        console.log(this)
    }
    fn3() {
        alert("事件绑定3")
        console.log(this)
    }
    fn4 = ()=>{
        alert("事件绑定4")
        console.log(this)
    }
    fn5(){
        alert("事件绑定5")
        console.log(this)
```

```
    }
    render() {
        return (
            <div className="box">
                <button onClick={this.fn1}>事件绑定1</button>
                {/* 固定this的指向 */}
                <button onClick={this.fn2.bind(this)}>事件绑定2</button>
                <button onClick={this.fn3}>事件绑定3</button>
                <button onClick={this.fn4}>事件绑定4</button>
                {/* <button onClick={() => { console.log(1111) }}>事件绑定
5</button> */}
                <button onClick={() => { this.fn5()  } }>事件绑定5</button>
            </div>
        );
    }
}

export default Box;


// function 函数名(){}
// let 函数名 = function(){}
// new Function()
```

4.2、事件传参

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

class BoxTwo extends Component {
    constructor(){
        super()
        this.fn3 = this.fn3.bind(this)
    }
    fn1(){
        alert("事件绑定1")
        console.log(this)
    }
    fn2(val){  // 可以传参！！！！
        alert(val)
        console.log(this)
    }
    fn3() {
        alert("事件绑定3")
        console.log(this)
    }
```

```
    fn4 = ()=>{
        alert("事件绑定4")
        console.log(this)
    }
    fn5(val){
        alert(val)
        console.log(this)
    }
    render() {
        return (
            <div className="box">
                <button onClick={this.fn1}>事件绑定1===不能传! </button>
                {/* 固定this的指向 */}
                <button onClick={this.fn2.bind(this,2222)}>事件绑定2</button>
                <button onClick={this.fn3}>事件绑定3===不能传! </button>
                <button onClick={this.fn4}>事件绑定4===不能传! </button>
                {/* <button onClick={() => { console.log(1111) }}>事件绑定5</button>
*/}
                <button onClick={() => { this.fn5(2000)  } }>事件绑定5</button>
            </div>
        );
    }
}

export default BoxTwo;

// 模板里面使用bind 和  模板里面使用箭头函数  绑定事件可以传参!   这两种事件绑定使用最多!
```

4.3、事件对象

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

class BoxThree extends Component {
    constructor() {
        super()
        this.fn3 = this.fn3.bind(this)
    }
    fn1(e) {
        console.log(e.target)
        alert("事件绑定1")
        console.log(this)
    }
    fn2(e) {
        console.log(e.target)
        alert("事件绑定2")
        console.log(this)
```

```
    }
    fn3(e) {
        console.log(e.target)
        alert("事件绑定3")
        console.log(this)
    }
    fn4 = (e) => {
        console.log(e.target)
        alert("事件绑定4")
        console.log(this)
    }
    fn5(e) {
        alert("事件绑定5")
        console.log(this)
    }
    render() {
        return (
            <div className="box">
                <button onClick={this.fn1}>事件绑定1</button>
                {/* 固定this的指向 */}
                <button onClick={this.fn2.bind(this)}>事件绑定2</button>
                <button onClick={this.fn3}>事件绑定3</button>
                <button onClick={this.fn4}>事件绑定4</button>
                {/* <button onClick={() => { console.log(1111) }}>事件绑定5</button>
*/}
                <button  onClick={(ev) => { this.fn5(ev)  }}>事件绑定5</button>
            </div>
        );
    }
}

export default BoxThree;


// function 函数名(){}
// let 函数名 = function(){}
// new Function()
```

4.4、既要传参又要获取事件对象

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

class BoxFour extends Component {
    constructor() {
        super()
```

```
        this.fn3 = this.fn3.bind(this)
    }
    fn1() {
        alert("事件绑定1")
        console.log(this)
    }
    fn2(val,e) {  // 可以传参！！！！
        // 如果既要传参又要获取事件对象，那么形参位置的最后一位就是事件对象！
        console.log(e.target)
        alert(val)
        console.log(this)
    }
    fn3() {
        alert("事件绑定3")
        console.log(this)
    }
    fn4 = () => {
        alert("事件绑定4")
        console.log(this)
    }
    fn5(e,num) {
        console.log(e.target)
        console.log(num)
        console.log(this)
    }
    render() {
        return (
            <div className="box">
                <button onClick={this.fn1}>事件绑定1===不能传！ </button>
                {/* 固定this的指向 */}
                <button onClick={this.fn2.bind(this,2222)}>事件绑定2</button>
                <button onClick={this.fn3}>事件绑定3===不能传！ </button>
                <button onClick={this.fn4}>事件绑定4===不能传！ </button>
                {/* <button onClick={() => { console.log(1111) }}>事件绑定5</button>
*/}
                <button onClick={(ev) => { this.fn5(ev,200) }}>事件绑定5</button>
            </div>
        );
    }
}

export default BoxFour;

// 模板里面使用bind 和  模板里面使用箭头函数
    // 绑定事件可以传参且可以同时获取事件对象！   这两种事件绑定使用最多！
```