# Reactjs code snippets 代码提示

|  |
| --- |
| <https://marketplace.visualstudio.com/items?itemName=xabikos.ReactSnippets>  <https://blog.csdn.net/qq_39969226/article/details/101279697> |

# 搭建环境

|  |
| --- |
| 第一种（老-现在推荐）  1.安装脚手架工具 npm install -g create-react-app  2.创建项目 create-react-app reactdemo  3. cd reactdemo  4. npm start 或yarn start运行项目  npm run build 或yarn build 生成项目  第二种（新-未来推荐）  1. npx create-react-app reactdemo  2. cd reactdemo  3. npm start 运行项目 npm run build 生成项目 |

# react的介绍

|  |
| --- |
| React来自于Facebook公司的开源项目  React *可以开发单页面应用* *=>* spa（单页面应用）  react 组件化模块化  开发模式  React通过对DOM的模拟(虚拟dom)，最大限度地减少与DOM的交互  （数据绑定）  react灵活  React可以与已知的库或框架很好地配合。  react 基于jsx的语法，JSX是React的核心组成部分，它使用XML标记的方式去直接声明界面，  html  js混写模式 |

# npx介绍

|  |
| --- |
| <http://www.phonegap100.com/thread-4910-1-1.html>  npm v5.2.0引入的一条命令（npx），引入这个命令的目的是为了提升开发者使用包内提供的命令行工具的体验  npx create-react-app my-app  这条命令会临时安装 create-react-app 包，命令完成后create-react-app 会删掉，不会出现在 global 中。下次再执行，还是会重新临时安装。  npx 会帮你执行依赖包里的二进制文件  npx http-server 可以一句话帮你开启一个静态服务器 |

# React目录结构

|  |
| --- |
| manifest.json 文件简介：  https://lavas.baidu.com/mip/doc/engage-retain-users/add-to-home-screen/introduction  允许将站点添加至主屏幕，是 PWA 提供的一项重要功能，当前 manifest.json 的标准仍属于草案阶段，Chrome 和 Firefox 已经实现了这个功能，微软正努力在 Edge 浏览器上实现，Apple 目前仍在考虑中  super关键字：  参考：http://www.phonegap100.com/thread-4911-1-1.html  Es6中的super可以用在类的继承中，super关键字，它指代父类的实例（即父类的this对象）。子类必须在constructor方法中调用super方法，否则新建实例时会报错。这是因为子类没有自己的this对象，而是继承父类的this对象，然后对其进行加工。如果不调用super方法，子类就得不到this对象。  class Person {  constructor (name) {  this.name = name;  }  }  class Student extends Person {  constructor (name, age) {  super(); // 用在构造函数中，必须在使用this之前调用  this.age = age;  }  }  为什么官方的列子里面写个super(props)：  只有一个理由需要传递props作为super()的参数，那就是你需要在构造函数内使用this.props  那官方提供学习的例子中都是写成super(props)，所以说写成super(props)是完全没问题的，也建议就直接这样写。 |

## index.js

|  |
| --- |
| // React是 React 的核心库，  import React from 'react';  // ReactDOM 是提供与 DOM 相关的功能，  import ReactDOM from 'react-dom';  // css可以删掉  import './index.css';  // 引入App.js这个组件  import App from './App';  //不用管  加快react运行速度的一个js文件  import registerServiceWorker from './registerServiceWorker';  ReactDOM.render(     <*App* />,     document.getElementById('root')  );  registerServiceWorker(); |

|  |
| --- |
| {  *"short\_name"*: "React App",  *"name"*: "Create React App Sample",  *"icons"*: [      {  *"src"*: "favicon.ico",  *"sizes"*: "64x64 32x32 24x24 16x16",  *"type"*: "image/x-icon"      }    ],  *"start\_url"*: "./index.html",  *"display"*: "standalone",  *"theme\_color"*: "#000000",  *"background\_color"*: "#ffffff"  } |

# React创建组件

|  |
| --- |
|  |

# ReactJSX语法

|  |
| --- |
|  |

# React绑定数据

|  |
| --- |
|  |

# React绑定对象

|  |
| --- |
|  |

# React绑定属性( 绑定class 绑定style 图片)

|  |
| --- |
|  |

# React目录结构

|  |
| --- |
|  |

# React目录结构

|  |
| --- |
|  |

# React目录结构

|  |
| --- |
|  |

# React目录结构

|  |
| --- |
|  |

# React目录结构

|  |
| --- |
|  |

# React目录结构

|  |
| --- |
|  |

# index.js

|  |
| --- |
| import React from 'react'; // React是 React 的核心库，  import ReactDOM from 'react-dom'; // ReactDOM 是提供与 DOM 相关的功能，  import './index.css';  import App from './App';  //不用管 加快react运行速度的一个js文件  import registerServiceWorker from './registerServiceWorker';  或  import \* as serviceWorker from './serviceWorker';  ReactDOM.render(<*App* />, document.getElementById('root'));  registerServiceWorker();  或  serviceWorker.unregister(); |

# App.js

|  |
| --- |
| *class* App extends *Component* {  *constructor*(*props*){  super(props); // 用于父子组件传值 固定写法  this.state = { //定义数据  name:'张三',  logo:'./images/logo.png'  }  }  render() {  return (  <div className="App">  {/\* 注释 \*/}  {  // 注释  }  {this.state.name}  <img src={logo} alt="logo" />  </div>  );  }  } |

# 虚拟dom

|  |
| --- |
|  |

# 生命周期

|  |
| --- |
|  |

|  |
| --- |
|  |

# React写法

## 函数方式

|  |
| --- |
| *const* index = () *=>* {    return (     <div>哈哈哈</div>    );  }  export default index |

## cass方式

|  |
| --- |
| import React, { Component } from 'react'  export default *class* FileName extends Component {    render() {      return <div>$2</div>    }  } |

# dvajs

|  |
| --- |
|  |

# 高级指南

## React.StrictMode

|  |
| --- |
|  |

# Hook

|  |
| --- |
| 增强函数组件的功能：状态、引用、生命周期(hook并不能提供全部生命周期的功能) |

## useState

|  |
| --- |
| import './App.css';  import { useState } from 'react';  import { Button } from 'antd';  *function* App() {  *const* [count, setCount] = useState(0);  *const* [obj, setObj] = useState({name: 'zhangsan'});  *const* [arr, setArr] = useState([1,2,3]);  *const* [func, setFunc] = useState(() *=>* {      return {name: 'wangwu'};    })    return (      <div className="App">        <header className="App-header">          <div>{count}</div>          <*Button*  onClick={() *=>* {            setCount(count+1)          }} type="primary">count</*Button*>          <div>{obj.name} {obj.age}</div>          <*Button*  onClick={() *=>* {            setObj({              ...obj,              age: 18            })          }} type="primary">obj</*Button*>          <div>{arr}</div>          <*Button*  onClick={() *=>* {            setArr(() *=>* {              arr.push(4);              return [...arr];            })          }} type="primary">arr</*Button*>          <div>{func.name}</div>          <*Button*  onClick={() *=>* {            setFunc({name: 'lisi'})          }} type="primary">arr</*Button*>        </header>      </div>    );  }  export default App; |

## useEffect

|  |
| --- |
| 1.componentDidMount、componentDidUpdate 和 componentWillUnmount   2.副作用(DOM操作、数据请求、组件更新)   3.useEffect为什么在组件函数内部执行？可以获取props和state，采用闭包的形式   4.无阻塞更新   5.useEffect(回调函数，数组(可不写))   6.多个useEffect()  import './App.css';  import { useState, useEffect } from 'react';  import { Button } from 'antd';  *function* App() {  *const* [count, setCount] = useState(0);  *const* [num, setNum] = useState(0);    // 不传[],监听所以状态    // 传[count], 只监听count，  count每次更新都会执行useEffect    // 传[],不监听状态，只在初始化时执行一次    useEffect(() *=>* {      console.log(count, '-count-')      console.log(num, '-num-')    })    useEffect(() *=>* {      console.log(count)      console.log(num)      return () *=>* {        console.log('componentWillUnmount')      }    }, [count])    return (      <div className="App">        <div>{count}</div>        <*Button*  onClick={() *=>* {          setCount(count+1)        }} type="primary">count</*Button*>        <div>{num}</div>        <*Button*  onClick={() *=>* {          setNum(num+1)        }} type="primary">num</*Button*>      </div>    );  }  export default App; |

## useRef

|  |
| --- |
| import './App.css';  import { useRef } from 'react';  import { Button, Input } from 'antd';  *function* App() {  *const* inpPutEl = useRef(null);  *const* save = useRef({value: '789'})    return (      <div className="App">        <*Input* ref={inpPutEl}/>        <*Button*  onClick={() *=>* {          console.log(inpPutEl)          console.log(inpPutEl.current)          console.log(inpPutEl.current.state.value)          save.current.value = inpPutEl.current.state.value          console.log(save)        }} type="primary">count</*Button*>      </div>    );  }  export default App; |

## useContext

|  |
| --- |
| import './App.css';  import { useRef, useState, createContext, useContext } from 'react';  import { Button, Input } from 'antd';  *const* MyContext = createContext();  *const* ChildContent = () *=>* {  *let* count = useContext(MyContext);    return (      <h3>子组件：{count}</h3>    )  }  *function* App() {  *const* [count, setCount] = useState(0);  *const* inpPutEl = useRef(null);      return (      <div className="App">        <*MyContext.Provider* value={count}>          <*ChildContent*></*ChildContent*>        </*MyContext.Provider*>        <*Input* ref={inpPutEl}/>        <*Button*  onClick={() *=>* {          // setCount(count+1)          setCount(inpPutEl.current.state.value)        }} type="primary">count</*Button*>      </div>    );  }  export default App; |

### App.js

|  |
| --- |
| import './App.css';  import { useRef, useState, useContext } from 'react';  import { Button, Input } from 'antd';  import { MyContext } from './MyContext';  import ChildContent from './ChildContent'  *function* App() {  *const* [count, setCount] = useState(0);  *const* inpPutEl = useRef(null);      return (      <div className="App">        <*MyContext.Provider* value={count}>          <*ChildContent*></*ChildContent*>        </*MyContext.Provider*>        <*Input* ref={inpPutEl}/>        <*Button*  onClick={() *=>* {          // setCount(count+1)          setCount(inpPutEl.current.state.value)        }} type="primary">count</*Button*>      </div>    );  }  export default App; |

### MyContext.js

|  |
| --- |
| import { createContext } from 'react';  export *let* MyContext = createContext(); |

### ChildContent.js

|  |
| --- |
| import { useContext } from 'react';  import { MyContext } from './MyContext';  export default () *=>* {  *let* count = useContext(MyContext);      return (          <h3>子组件：{count}</h3>      )  } |

## useMemo

|  |
| --- |
| import './App.css';  import {  useState, , useEffect } from 'react';  import { Button, Input } from 'antd';  *function* App() {  *const* [count, setCount] = useState(0);  *const* [num, setNum] = useState(0);    // useMemo(() => {    //   console.log('useMemo');    // })      // useEffect(() => {    //   console.log('useEffect');    // })  *let* res = useMemo(() *=>* {      console.log('useMemo', count, num);      return {count, num}    }, [count])      return (      <div className="App">        <h3>{res.count}</h3>        <*Button*  onClick={() *=>* {          setCount(count+1)          console.log('count：', count)        }} type="primary">count</*Button*>        <h3>{res.num}</h3>        <*Button*  onClick={() *=>* {          setNum(num+1)          console.log('num：',num)        }} type="primary">num</*Button*>      </div>    );  }  export default App; |

### App.js

|  |
| --- |
| import './App.css';  import {  useState, useEffect } from 'react';  import { Button, Input } from 'antd';  import ChildMemo from './ChildMemo';  *function* App() {  *const* [count, setCount] = useState(0);  *const* [num, setNum] = useState(0);      return (      <div className="App">        <*ChildMemo* c={count} n={num} />        <div>父组件：count{count}---num{num}</div>        <*Button*  onClick={() *=>* {          setCount(count+1)          console.log('count：', count)        }} type="primary">count</*Button*>        <*Button*  onClick={() *=>* {          setNum(num+1)          console.log('num：',num)        }} type="primary">num</*Button*>      </div>    );  }  export default App; |

### ChildMemo.js

|  |
| --- |
| import { useMemo } from 'react';  export default ({*c*, *n*}) *=>* {    *let* res = useMemo(() *=>* {          console.log('ChildMemo', *c*, *n*);          return {c, n}      })      return (          <div>子组件：count{res.c}---num{res.n}</div>      )  } |

## useCallback

|  |
| --- |
| import './App.css';  import {  useState, useCallback } from 'react';  import { Button } from 'antd';  *function* App() {  *const* [count, setCount] = useState(0);  *const* [num, setNum] = useState(0);  *let* callback = useCallback(() *=>* {      console.log(count)      return count;    }, [])      return (      <div className="App">          <h3>状态count：{count}</h3>        <h3>状态num：{num}</h3>        <h3>callback：{callback()}</h3>        <*Button*  onClick={() *=>* {          setCount(count+1)          console.log('count：', count)        }} type="primary">count</*Button*>        <*Button*  onClick={() *=>* {          setNum(num+1)          console.log('num：',num)        }} type="primary">num</*Button*>      </div>    );  }  export default App; |

## useImperativeHandle

### forwardRef

|  |
| --- |
| import './App.css';  import { forwardRef, useRef, useImperativeHandle } from 'react';  *const* Forward = forwardRef((*props*, *ref*) *=>* {    return (      <>        <h3 ref={*ref*}>123</h3>        <h4>456</h4>      </>    )  })  *function* App() {  *const* el = useRef(null);    return (      <div className="App">        <*Forward* ref={el}/>        <button  onClick={() *=>* {          console.log(el.current)        }} >获取子组件dom</button>      </div>    );  }  export default App; |

### useImperativeHandle

|  |
| --- |
| import './App.css';  import { forwardRef, useRef, useState, useImperativeHandle } from 'react';  *const* Imperative = forwardRef((*props*, *refa*) *=>* {  *const* inputRef = useRef();  *const* [count, setCount] = useState(0);  *const* [num, setNum] = useState(0);    // 暴露给父组件自定义属性和方法    // 只有num属性变化时，才把count更新给父组件    useImperativeHandle(*refa*, () *=>* ({      name: 'zhangsan',      focus: () *=>* {        inputRef.current.focus();      },      count    }), [num]);    return (      <>      <h3>count{count}</h3>      <h3>num{num}</h3>      <input type="text" ref={inputRef}/>      <button onClick={() *=>* {setCount(count+1)}}>setCount</button>      <button onClick={() *=>* {setNum(num+1)}}>setNum</button>      </>    )  })  *function* App() {  *const* el = useRef(null);    return (      <div className="App">        <*Imperative* ref={el}/>        <button  onClick={() *=>* {          el.current.focus()          console.log(el)        }} >获取子组件自定义方法或属性</button>      </div>    );  }  export default App; |

## useLayoutEffect

|  |
| --- |
| import './App.css';  import { useState, useEffect, useLayoutEffect } from 'react';  *function* App() {  *const* [count, setCount] = useState(0);    useEffect(() *=>* {      console.log('useEffect')      return () *=>* {        console.log('useEffect-return')      }    })    useLayoutEffect(() *=>* {      console.log('useLayoutEffect')      return () *=>* {        console.log('useLayoutEffect-return')      }    })    return (      <div className="App">        <div>{count}</div>        <button  onClick={() *=>* {          setCount(count+1)          console.log(count)        }}>btn</button>      </div>    );  }  export default App; |

## 自定义hook

|  |
| --- |
| import './App.css';  import { useState } from 'react';  *const* useCus = (*val*, *num*) *=>* {  *let* [count, setCount] = useState(*val*);  *const* add = () *=>* {      setCount(count + *num*)    }    return { count, add}  }  *function* App() {    *let* { count, add } = useCus(10, 2)    return (      <div className="App">        <div>{count}</div>        <button  onClick={() *=>* {          add()        }}>btn</button>      </div>    );  }  export default App; |

## useReducer

|  |
| --- |
| import './App.css';  import { useReducer } from 'react';  *function* App() {  *const* [state, dispatch] = useReducer((*state*, *action*) *=>* {      switch (*action*.type) {        case 'setname':          return {            ...*state*,            name: *action*.name          }        case 'setage':          return {            ...*state*,            age: *action*.age          }        default:         return *state*;      }    }, {name: '张三', age: 18})      return (      <div className="App">        <div>姓名：{state.name}---年龄：{state.age}</div>        <button  onClick={() *=>* {          dispatch({            type: 'setname',            name: '李四'          })        }}>setname</button>        <button  onClick={() *=>* {          dispatch({            type: 'setage',            age: 20          })        }}>setage</button>      </div>    );  }  export default App; |