# React01

前端三大框架:

* vue: 适合中小型项目 -- 开源社区
* angular: 适合大型项目 -- Google
* react: 中小型项目 -- Facebook

## React

官方网站: https://react.docschina.org/

**开发方式分两种**

* 脚本方式: 初学者入门
* 脚手架方式: 项目的开发

## 脚本

**官方提供脚本**: 外网,有些同学可能无法使用.

把 src 的路径放浏览器测试下就知道了

<script crossorigin src="https://unpkg.com/react@16/umd/react.production.min.js"></script>  
<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>

**国内提供的脚本**:

<script src="https://cdn.staticfile.org/react/16.4.0/umd/react.development.js"></script>  
<script src="https://cdn.staticfile.org/react-dom/16.4.0/umd/react-dom.development.js"></script>

## React基础写法

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Document</title>  
 </head>  
 <body>  
 <div id="app"></div>  
  
 <!-- 使用 React 方式, 类似于 jQuery 需要引入脚本之后使用 -->  
 <!-- 注!意!引!入!顺!序! 有依赖关系, 先短后长 -->  
 <script src="./vendor/react.development.js"></script>  
 <script src="./vendor/react-dom.development.js"></script>  
  
 <script>  
 let now = new Date().toLocaleTimeString(); //当前时间  
  
 // 使用原生DOM方式, 把 <b id='b1' class="danger">当前时间:xxxx</b>  
 // 放到 id='app' 的 div 中  
  
 // React 把DOM 操作分为两类: 创建 和 渲染, 使用不同的类处理  
 // 1. React 类: 负责创建元素  
 // 2. ReactDOM 负责渲染元素到页面上  
  
 // 新技术的出现原因: 懒; 利用原生的类方式进行了封装操作 达到简化目的!  
 let b = React.createElement(  
 "b",  
 { id: "b1", className: "danger" },  
 "当前时间:" + now  
 );  
  
 ReactDOM.render(b, document.getElementById("app"));  
 </script>  
 </body>  
</html>

## JSX语法

Facebook工程师, 为了简化代码, 创建的新语法

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Document</title>  
 </head>  
 <body>  
 <div id="app"></div>  
  
 <!-- 使用 React 方式, 类似于 jQuery 需要引入脚本之后使用 -->  
 <!-- 注!意!引!入!顺!序! 有依赖关系, 先短后长 -->  
 <script src="./vendor/react.development.js"></script>  
 <script src="./vendor/react-dom.development.js"></script>  
  
 <!-- 引入babel 编译工具 -->  
 <script src="./vendor/babel.min.js"></script>  
  
 <!-- 声明type, 告诉浏览器 这里的代码需要babel编译 -->  
 <script type="text/babel">  
 let now = new Date().toLocaleTimeString(); //当前时间  
  
 // 体验 JSX 语法: JS + XML 代表 在 JS 中书写 HTML  
 // jsx: 长得跟 html 一样, 所以使用者容易适应, 本质是 DOM 操作的 语法糖写法  
  
 // 浏览器不认 JSX 代码. babel工具 专门做翻译工作  
 let b = (  
 <b id="b1" className="danger">  
 当前时间: {now}  
 </b>  
 );  
  
 ReactDOM.render(b, document.getElementById("app"));  
 </script>  
 </body>  
</html>

## babel工具

**官方地址**:

<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

**本地地址**:

<script src="https://cdn.staticfile.org/babel-standalone/6.26.0/babel.min.js"></script>

使用babel编译的 script , 必须添加 type

<script type="text/babel">

因为babel工具**不会多管闲事**, 只编译指定类型的脚本

## 组件

组件: 组成页面的零件, 英文 component

特点: **复用性**

React组件制作的方式有两种:

* 函数方式: 适合简单组件, **无状态**
* 类方式: 适合复杂组件, **有状态**

### 函数组件

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Document</title>  
 <!-- 为什么vue和angular 用 {{}}, 因为是写在 html 中, 而 html 中存在 {} -->  
 <!-- 假设是 {}, 则 下方是 {color:red} 是什么? 就会有歧义! -->  
 <!-- jsp的语法 {% %} php <?php ?> 还有: {? ?} -->  
 <style>  
 h1 {  
 color: red;  
 }  
 </style>  
 </head>  
 <body>  
 <div id="app"></div>  
  
 <script src="./vendor/react.development.js"></script>  
 <script src="./vendor/react-dom.development.js"></script>  
 <script src="./vendor/babel.min.js"></script>  
  
 <script type="text/babel">  
 // 组件: 组成页面的零件, 特征是复用  
 // 在JS中,能够实现代码复用的: 函数 和 类  
 // 组件名: 为了区分原始标签, 和vue策略相同, 要求 大驼峰写法  
  
 // 函数方式: 适合简单组件  
 function Hello() {  
 return <h1>Hello React!</h1>;  
 }  
  
 let h = Hello(); //返回值是: <h1>Hello React!</h1>  
  
 // 类似于 vue, angular. JS代码要放 {} 中  
 let more = (  
 <div>  
 {Hello()}  
 {Hello()}  
 {Hello()}  
 {Hello()}  
 </div>  
 );  
  
 // 语法糖: 可以让 组件的是写法更好看  
 more = (  
 <div>  
 <Hello />  
 <Hello />  
 <Hello />  
 <Hello />  
 </div>  
 );  
  
 ReactDOM.render(more, document.getElementById("app"));  
 </script>  
 </body>  
</html>

### 函数组件传参

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Document</title>  
 </head>  
 <body>  
 <div id="app"></div>  
  
 <script src="./vendor/react.development.js"></script>  
 <script src="./vendor/react-dom.development.js"></script>  
 <script src="./vendor/babel.min.js"></script>  
  
 <script type="text/babel">  
 // 组件的父子传参  
 // 必须大驼峰写法  
 function HelloName(props) {  
 return <h1>Hello, {props.name}</h1>;  
 }  
  
 let b = HelloName({ name: "东东" });  
 // 语法糖写法: 看着更像组件  
 b = <HelloName name="东东" />;  
  
 let more = (  
 <div>  
 <HelloName name="亮亮" />  
 <HelloName name="然然" />  
 {HelloName({ name: "吉吉" })}  
 </div>  
 );  
  
 ReactDOM.render(more, document.getElementById("app"));  
 </script>  
 </body>  
</html>

### 类方式的组件

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Document</title>  
 </head>  
 <body>  
 <div id="app"></div>  
  
 <script src="./vendor/react.development.js"></script>  
 <script src="./vendor/react-dom.development.js"></script>  
 <script src="./vendor/babel.min.js"></script>  
  
 <script type="text/babel">  
 // 类方式的组件: 拥有更多的特性, 点击事件, 生命周期..  
  
 // 类名: 必须大驼峰写法. 即 首字母大写! 与vue要求相同  
 // extends: 继承. 只有继承了父类 才会有强大的特性.  
 class Hello extends React.Component {  
 // 固定的方法名: 与 语法糖有关  
 render() {  
 return <h1>Hello React</h1>;  
 }  
 }  
  
 let h = new Hello().render();  
 // 语法糖写法: 被babel编译成 new Hello().render(); 固定调用 render()  
 h = <Hello />;  
  
 ReactDOM.render(h, document.getElementById("app"));  
 </script>  
 </body>  
</html>

### 类方式的传参

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Document</title>  
 </head>  
 <body>  
 <div id="app"></div>  
  
 <script src="./vendor/react.development.js"></script>  
 <script src="./vendor/react-dom.development.js"></script>  
 <script src="./vendor/babel.min.js"></script>  
  
 <script type="text/babel">  
 // 类方式的组件 父子传参  
 // 继承带来的问题: 好多属性方法 隐藏在父类中, 对于阅读代码 比较困难.  
 class HelloName extends React.Component {  
 // 父类中的代码  
 // 父类的构造方法: 会把收到的参数 存储在 成员属性 this.props 中  
 // constructor(props) {  
 // this.props = props;  
 // }  
  
 render() {  
 // 面向对象中, 属性必须添加 this 前缀才能调用  
 return <h1>Hello, {this.props.name}</h1>;  
 }  
 }  
  
 let h = new HelloName({ name: "东东" }).render();  
 // 语法糖写法: 与上方代码等价 -- babel工具会自动编译  
 h = <HelloName name="然然" />;  
  
 let more = (  
 <div>  
 <HelloName name="亮亮" />  
 <HelloName name="凯凯" />  
 <HelloName name="铭铭" />  
 {new HelloName({ name: "西西" }).render()}  
 </div>  
 );  
  
 ReactDOM.render(more, document.getElementById("app"));  
 </script>  
 </body>  
</html>

## 脚手架方式

**安装脚手架**: 非必备步骤

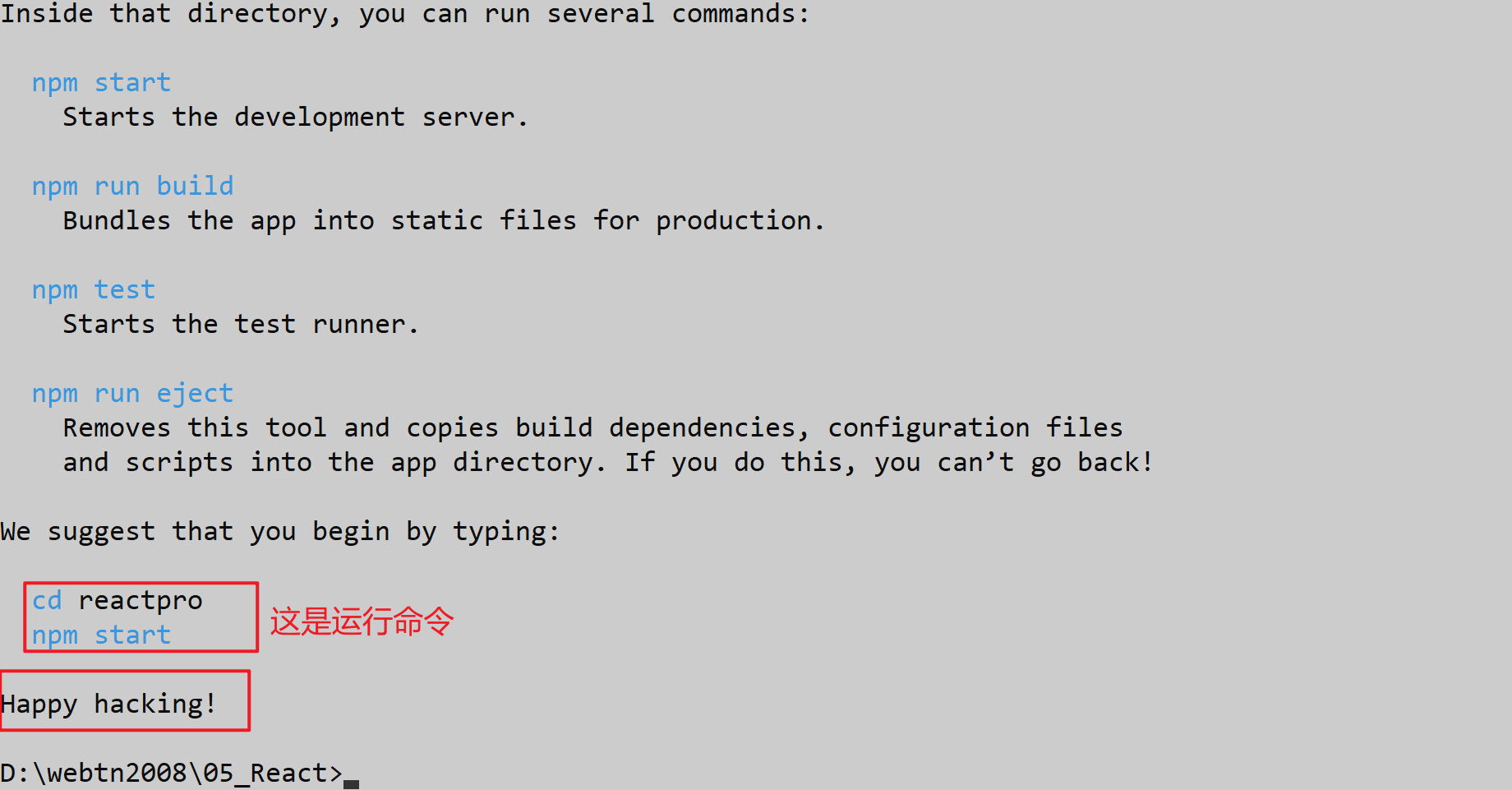
npm i -g create-react-app

**生成项目包**:

注意 cmd 执行所在的目录下, 生成包

* 安装了脚手架
* create-react-app 包名
* 没安装脚手架
* npx 会临时去下载脚手架使用: 优点是不用安装全局 缺点:每次都要下载一次
* npx create-react-app 包名

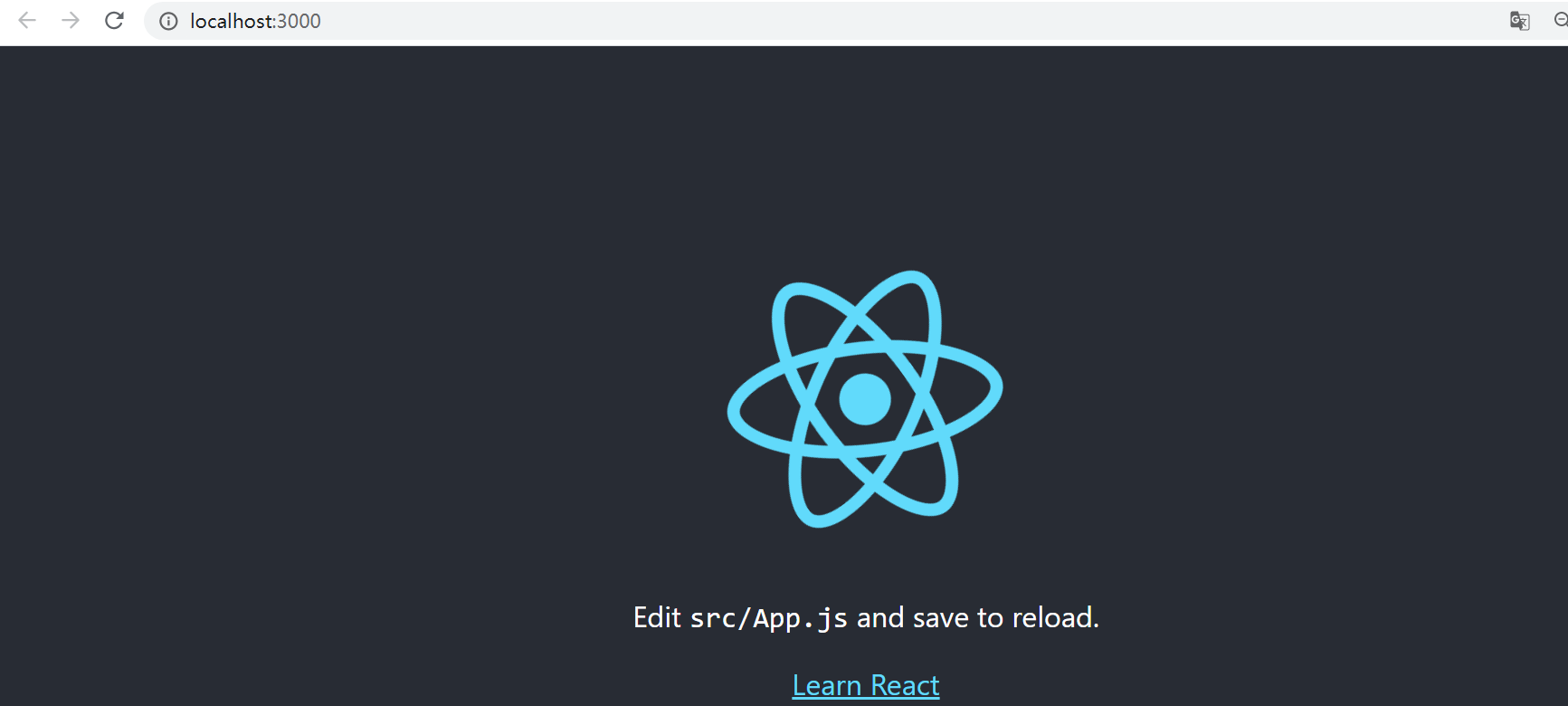
例如:npx create-react-app reactpro



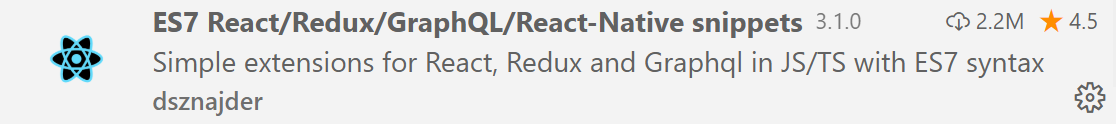
**启动命令**:

注意: 必须在项目目录下

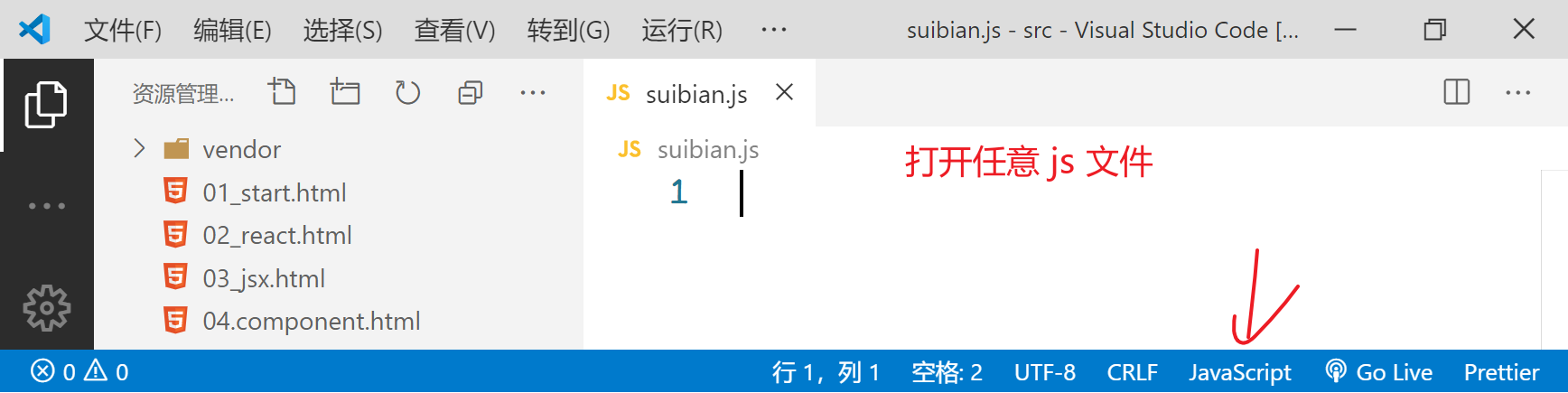
npm start



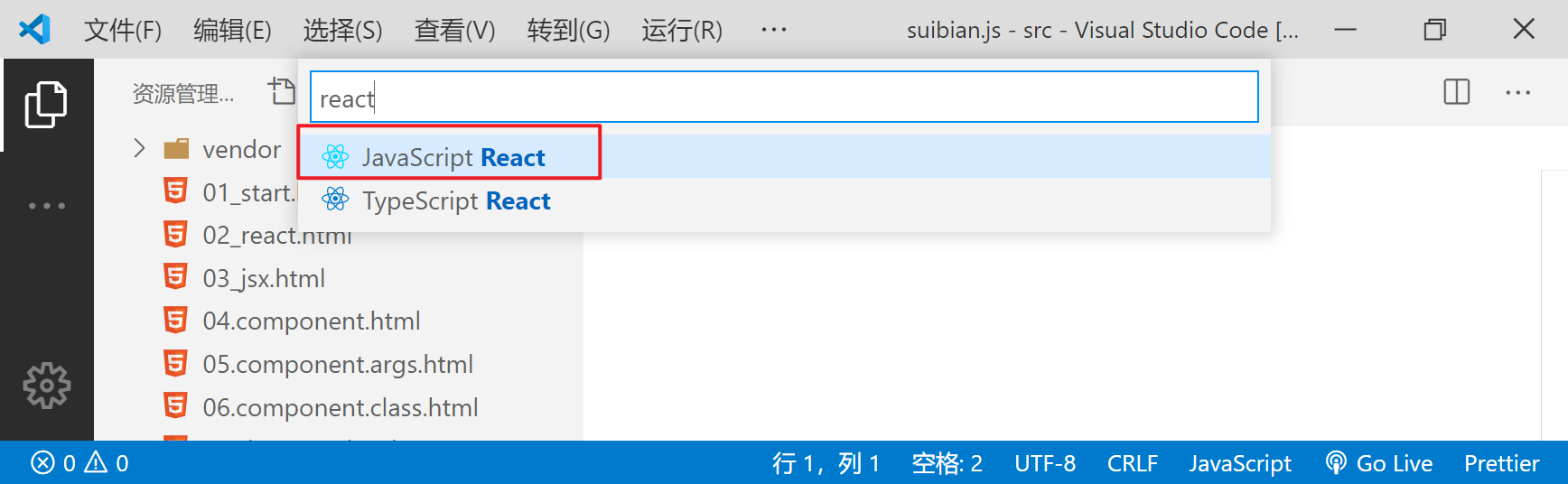
## 插件



配置: 告诉vscode把 JS 文件当成JSX来处理. 将会拥有更多的代码提示









## 启动流程

当在浏览器中输入 localhost:3000 发生了什么

* localhost 域名, 代表当前计算机
* **:3000** : 端口号. 程序在计算机上的唯一标识
* localhost:3000: 访问当前计算机上 唯一标识是 **3000** 的程序
* React的服务器默认端口号是3000, 使用 npm start 就可以启动这个服务器
* 服务器软件的设定: 来访人员 一律访问 index.html 文件: 此文件称为 **入口文件**
* index.html: 包含一个 <div id="root"></div>
* webpack: 此工具会自动打包 index.js 文件并引入到 index.html 中
* 根组件的加载: index.js
* 渲染了 App.js 到 id='root' 的标签中
* ReactDOM.render(
    
   <React.StrictMode>
    
   <App />
    
   </React.StrictMode>,
    
   document.getElementById('root')
    
  );

## 事件

// index.html => index.js => App.js
  
// React属于 单页应用 SPA项目
  
   
// 快捷代码块 rcc
  
import React, { Component } from "react";
  
   
export default class App extends Component {
  
 render() {
  
 return (
  
 <div>
  
 <h1>Hello World!</h1>
  
 {/\*
  
 vue中: @click="" v-on:click=""
  
 ng中: (click)=""
  
 原生: onclick=""
  
 小程序: bindtap="" 或 catchtap=""
  
 react: onClick=""
  
 \*/}
  
   
 {/\* {}中的代码在 页面显示时, 会自动执行 \*/}
  
 <button onClick={this.show}>点我</button>
  
 {/\*
  
 关于事件中的方法名是否写()结尾
  
 不带参数时:
  
 vue: 都可以 @click="show" @click="show()"
  
 angular: 必须带() (click)="show()"
  
 react: 不带() onClick={this.show}
  
 \*/}
  
 </div>
  
 );
  
 }
  
   
 show() {
  
 alert("点击事件");
  
 }
  
}

## 事件的this指向

// 事件中的 this 指向问题
  
   
/\*\*
  
 \* 函数分两种:
  
 \* 普通函数: function(){}
  
 \* \* this指向 函数调用者. xxx.show(); show()中的this 指向xxx
  
 \* \* 3个与this有关的方式 bind apply call
  
 \* \*\* xxx.bind(this): 只改指向 但不会执行
  
 \* \*\* apply, call: 修改指向的同时 会执行. 两者的差异是 参数2不同
  
 \* 箭头函数: ()=>{}
  
 \* \* this指向 声明时所在环境 的this
  
 \*
  
 \*
  
 \*/
  
   
// rcc
  
import React, { Component } from "react";
  
   
export default class App extends Component {
  
 name = "东东";
  
   
 /\*\*
  
 \* 普通的函数的this指向调用者. obj.show(); obj就是this
  
 \* window 触发的函数, this是 undefined
  
 \*/
  
 show() {
  
 console.log(this.name);
  
 }
  
   
 // 实际工作: 更加偏爱 箭头函数. 杜绝this的指向问题
  
 show1 = () => {
  
 console.log(this.name);
  
 };
  
   
 render() {
  
 return (
  
 <div>
  
 <button onClick={this.show1}>箭头函数</button>
  
   
 {/\* 相当于: 点击按钮之后 打然然的媳妇 \*/}
  
 {/\* 点击后 执行 当前对象的show方法: 事件由window触发! \*/}
  
 {/\* bind: 替换 普通函数的 this 指向 \*/}
  
 <button onClick={this.show.bind(this)}>普通函数</button>
  
   
 {/\* 利用箭头函数, 也可以保持this指向 \*/}
  
 <button onClick={() => this.show()}>箭头+普通</button>
  
   
 {/\* 箭头函数的格式: ()=>{ xxx } 如果 {} 只有一行代码, 语法糖: ()=> xxx \*/}
  
 {/\* 点击之后, 执行的是箭头函数, 而 箭头函数 再执行其中的 show() 方法 \*/}
  
 <button
  
 onClick={() => {
  
 this.show();
  
 }}
  
 >
  
 箭头+普通
  
 </button>
  
 </div>
  
 );
  
 }
  
}

## 函数传参

// 事件传参
  
   
// rcc
  
import React, { Component } from "react";
  
   
export default class App extends Component {
  
 show(name) {
  
 alert(name);
  
 }
  
   
 // 箭头函数能替换this么? 答案: 不能
  
 show1 = (name) => {
  
 alert(name, this.name);
  
 };
  
   
 render() {
  
 return (
  
 <div>
  
 {/\* bind: 执行后, 返回一个新的函数, 其中的参数都已经指定. 只待执行 \*/}
  
 <button onClick={this.show.bind(this, "东东")}>东东</button>
  
 {/\* 点击时, 执行箭头函数, 箭头函数再执行其中的show \*/}
  
 <button onClick={() => this.show("亮亮")}>亮亮</button>
  
 {/\* 箭头函数只有一种方案: 箭头 套 \*/}
  
 <button onClick={() => this.show1("然然")}>然然</button>
  
 </div>
  
 );
  
 }
  
}

## State

// 状态值 State
  
   
/\*\*
  
 \* 请描述 react 和 vue 的差异: 为什么大型项目不适合vue?
  
 \*
  
 \* 数据绑定的差异:
  
 \* -- vue 依赖为每个数据项增加 set 和 get 方法. 通过方法来监听数据的变化 并 刷新DOM
  
 \* 大型项目 数据量很大, 假设50 变量 就要额外新增 100个方法. 性能会降低
  
 \*
  
 \* -- react 和 小程序相同: 通过 setState/setData 来更新数据. 不需要实时监听每个变量的变更, 性能更高.
  
 \*/
  
   
//rcc
  
import React, { Component } from "react";
  
   
export default class App extends Component {
  
 // 小程序中有一个特殊的属性: data, 需要配合 setData() 进行更新操作, 才能刷新到页面上.
  
 // react中 是 state 属性 和 setState 属性配合
  
 state = { num: 1 };
  
   
 count = 100;
  
   
 // 习惯: 事件触发的函数 带有 \_ 前缀
  
 \_change() {
  
 // //后台会爆黄: state中的值不应该直接修改, 应该使用 setState 修改
  
 this.state.num++;
  
   
 this.setState({ num: this.state.num + 1 });
  
   
 // setState() 有两个作用
  
 // 1. 更新数据
  
 // 2. 更新UI -- 不限于state中的数据项. 例如 count 也会变化
  
 }
  
   
 render() {
  
 return (
  
 <div>
  
 <button onClick={this.\_change.bind(this)}>{this.state.num}</button>
  
 {/\* react 每个属性, 没有监听. 不会因为修改而自动变化 \*/}
  
 <button onClick={() => this.count++}>{this.count}</button>
  
 </div>
  
 );
  
 }
  
}

## State的异步性

// setState 的异步性
  
   
//rcc
  
import React, { Component } from "react";
  
   
export default class App extends Component {
  
 state = { num: 1 };
  
   
 \_change = () => {
  
 // a++ 和 ++a:
  
 // 按照从左向右读: 先读到哪个用哪个. a++ 先用a; ++a 先加
  
   
 // 假设 num=1； 此时a就是 {num:1} this.state.num 是2
  
 let a = { num: this.state.num++ };
  
   
 // setState(): 需要刷新UI, 刷新UI属于耗时操作->cpu 和 gpu
  
 // 防止阻塞线程: 官方设计到异步执行. 每个线程能做一件事. cpu的12线程就是 能同时做12件事
  
 // 参数2: 回调函数, 在页面刷新完毕后执行
  
 this.setState(a, () => {
  
 console.log("UI更新完毕, num变为:", this.state.num);
  
 }); // {num:1}
  
   
 console.log(this.state.num);
  
   
 // 在vue中, 有没有同样的方式, 能够监听到DOM 刷新结束的时机??
  
 // $nextTick() -- 详见 FTP 的 99 面试题
  
 };
  
   
 render() {
  
 return (
  
 <div>
  
 <button onClick={this.\_change}>{this.state.num}</button>
  
 </div>
  
 );
  
 }
  
}

## 动态样式

// 动态样式
  
/\*\*
  
 \* vue中: :style="{样式名:值, 样式名:值}" :class="{样式类: true/false}"
  
 \*
  
 \* ng 中: [ngStyle]="{样式名:值, 样式名:值}" [ngClass]="{样式类: true/false}"
  
 \*/
  
   
//rcc
  
import React, { Component } from "react";
  
   
export default class App extends Component {
  
 state = { size: 18, br: 0, title: "变圆" };
  
   
 \_big() {
  
 this.setState({ size: this.state.size + 1 });
  
 }
  
   
 \_radius = () => {
  
 if (this.state.br == 50) {
  
 // 还原
  
 this.setState({ br: 0, title: "变圆" });
  
 } else {
  
 this.setState({ br: this.state.br + 10 }, () => {
  
 //回调:更新完毕后
  
 if (this.state.br == 50) {
  
 this.setState({ title: "还原" });
  
 }
  
 });
  
 }
  
 };
  
   
 render() {
  
 return (
  
 <div>
  
 <button onClick={this.\_big.bind(this)}>变大</button>
  
 {/\* style必须是对象类型 \*/}
  
 <div style={{ color: "red", fontSize: this.state.size + "px" }}>
  
 Hello World!
  
 </div>
  
   
 <button onClick={this.\_radius}>{this.state.title}</button>
  
 <div
  
 style={{
  
 backgroundColor: "green",
  
 width: "100px",
  
 height: "100px",
  
 borderRadius: this.state.br + "px",
  
 }}
  
 ></div>
  
 </div>
  
 );
  
 }
  
}

## 外部样式

// 外部 css 样式的写法
  
   
/\*\*
  
 \* css样式有3种写法:
  
 \* 1. 内联样式: style
  
 \* 2. 内部样式: html的head种, <style></style>
  
 \* 3. 外部样式: .css文件, 然后引入到代码中
  
 \*/
  
   
// rcc
  
import React, { Component } from "react";
  
   
/\*\*
  
 \* 引入外部的css文件:
  
 \* 1. html 引入: <link rel="stylesheet" href="style.css">
  
 \* 2. css 引入: @import '路径.css';
  
 \* 3. js 引入: import 'css文件路径'
  
 \*/
  
   
// 最常见的报错: import 'xxx.css'
  
// import 可以引入模块 或 文件
  
// 系统会根据 引号里的 是文件路径, 才会当做文件引入; 否则会当成模块引入
  
// ./ / ../ :这些都是文件路径特有的标识. 要求必须添加这些标识
  
import "./App.css";
  
   
export default class App extends Component {
  
 render() {
  
 return (
  
 <div>
  
 {/\* JSX语法并不是html. 本质上是 DOM 操作的语法糖写法. 而DOM操作中的 就是className \*/}
  
 <div className="danger">吉吉国王</div>
  
 {/\* 没有vue那种 @class="{xxx: true}" 写法 \*/}
  
 </div>
  
 );
  
 }
  
}

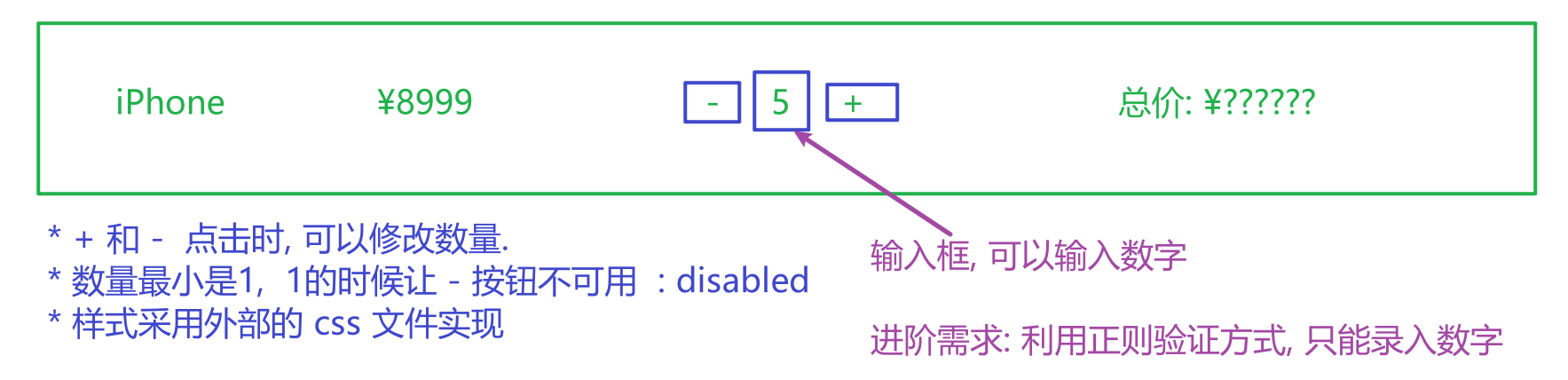
## 双向数据绑定

// 双向数据绑定
  
   
/\*\*
  
 \* vue 和 react 的差异:
  
 \* vue 是 双向数据绑定
  
 \* react 是 单向数据绑定, 需要配合 onChange 才能实现双向效果
  
 \*
  
 \*/
  
   
//rcc
  
import React, { Component } from "react";
  
   
export default class App extends Component {
  
 state = { uname: "dongdong" };
  
   
 render() {
  
 return (
  
 <div>
  
 {/\* react 不存在 v-model 这种简化写法. 必须手写 双向绑定的过程 \*/}
  
 <input type="text" value={this.state.uname} onChange={this.\_change} />
  
   
 <br />
  
 <input
  
 type="text"
  
 value={this.state.uname}
  
 onChange={(event) => this.setState({ uname: event.target.value })}
  
 />
  
   
 <p>{this.state.uname}</p>
  
 </div>
  
 );
  
 }
  
   
 // 事件触发的函数, 事件会作为参数传入.
  
 \_change = (event) => {
  
 // console.log(event);
  
 console.log(event.target.value);
  
   
 // 更新到数据中
  
 this.setState({ uname: event.target.value });
  
 };
  
}

## 今日内容回顾

* **原生DOM操作** -> **React基础操作** -> **JSX语法** -> **函数组件** -> **类组件**
* **安装脚手架**-> **生成项目包** -> **事件** -> **this指向** -> **事件参数**
* **状态值**-> **setState的异步性**
* **动态样式**
* **双向绑定**

## 作业



再次提醒: 下载 RN 需要的资源, 周三使用!!!

https://pan.baidu.com/s/16jrKVaIZ\_10H47t1rh4qBw
  
提取码：lqc6