# 环境及项目搭建

1.全局安装create-react-app（React脚手架的名称）

npm i -g create-react-app

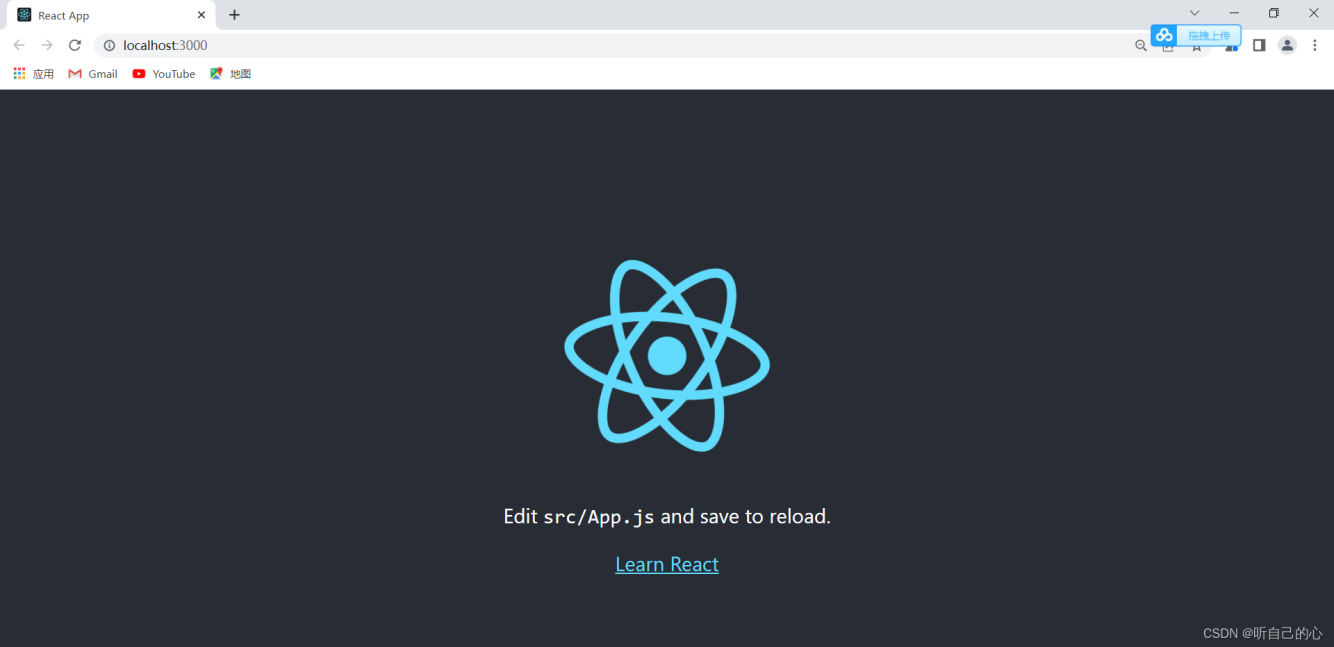
2.初始化项目

create-react-app react-demo

3.启动项目

npm start

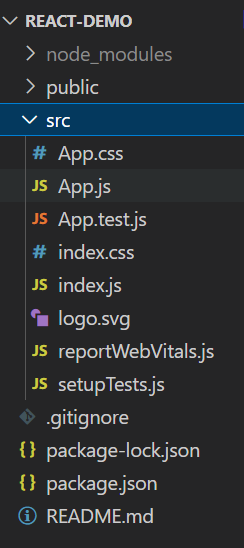
4.启动后页面



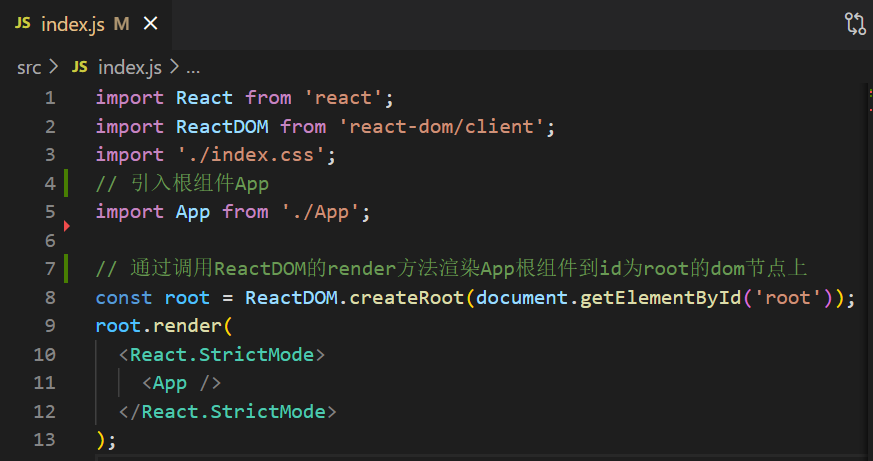
# 目录及项目

Src：项目代码开发目录

Package.json 中俩个核心库：react 、react-dom



入口文件index.js说明



# 3.JSX基础

JSX： JavaScript XML(HTML)的缩写，表示在 JS 代码中书写 HTML 结构

作用：在React中创建HTML结构（页面UI结构）

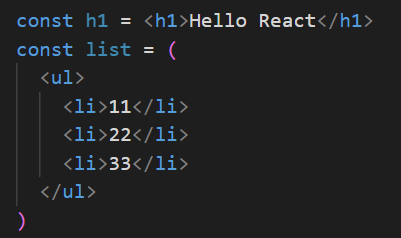
优势：

* 采用类似于HTML的语法，降低学习成本，会HTML就会JSX
* 充分利用JS自身的可编程能力创建HTML结构

注意：

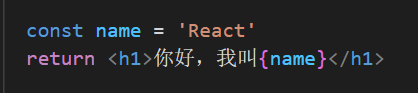
JSX并不是标准的JS语法，是JS的语法扩展，浏览器默认是不识别的，脚手架中内置的@babel/plugin-transform-react-jsx 包，用来解析该语法。

## 3.1 JSX声明UI



## 3.2 JSX中使用js表达式

语法：{ js 表达式 }



## 3.3 JSX列表渲染

实现：使用数组map方法



注意：遍历项需要添加 key 属性，否则会有错误警告

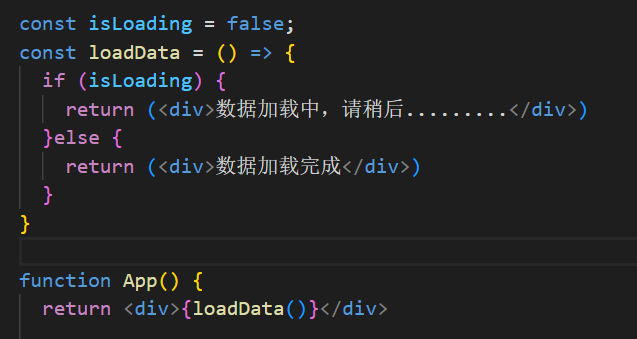
* key在HTML结构中是看不到的，是React内部用来性能优化时使用
* key在当前列表中要唯一的字符串或者数值（String/Number）

## 3.4 JSX条件渲染

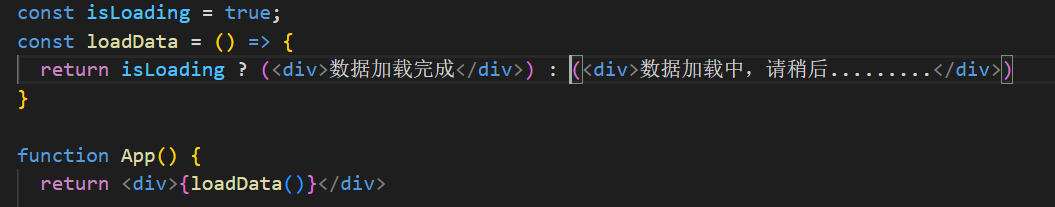
实现：

* 使用if/else
* 三目运算符
* 短路运算符
  + &&（第一个为false则终止执行）
  + ||（第一个为true则终止执行）

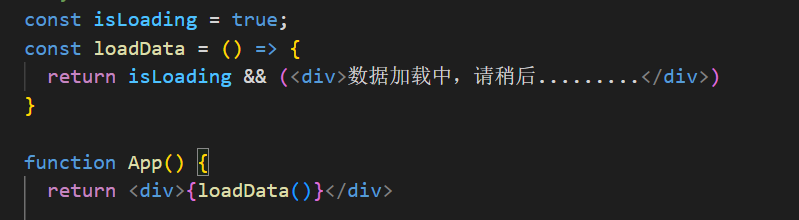
1. 使用if/else



1. 使用三目运算



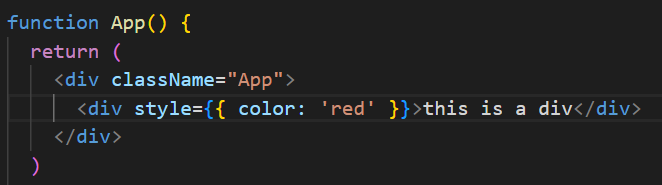
1. 使用短路运算符



## 3.5 JSX样式处理

1. 行内样式 - style

写法一



写法二



2. 类名 - className（推荐）

## 3.6 JSX注意事项

* JSX必须有一个根节点，如果没有根节点，可使用<></>（幽灵节点）替代
* 所有标签必须形成闭合，成对闭合或者自闭合
* JSX中的语法更加贴近JS语法，属性名采用驼峰命名法class->className
* JSX支持多行（换行），如果需要换行，需使用()包裹，防止bug出现

# 4.React 组件

## 1.函数组件

实现：使用 JS 的函数（或箭头函数）创建的组件，就叫做函数组件



注意点：

1.名称必须首字母大写开头，react根据这个来判断是组件还是普通HTML标签；

2.函数组件必须有返回值，表示该组件的UI结构，如果不需要渲染任何内容，则返回null；

3.使用函数名称作为组件标签名称，可以成对出现也可以自闭合；

## 2.类组件

实现：使用 class 创建的组件，叫做类（class）组件



注意点：

1.类名称必须以大写字母开头;

2.必须继承React.Component这个类，从而使用父类中提供的方法或属性;

3.必须提供render方法,render方法必须有返回值，表示该组件的UI结构

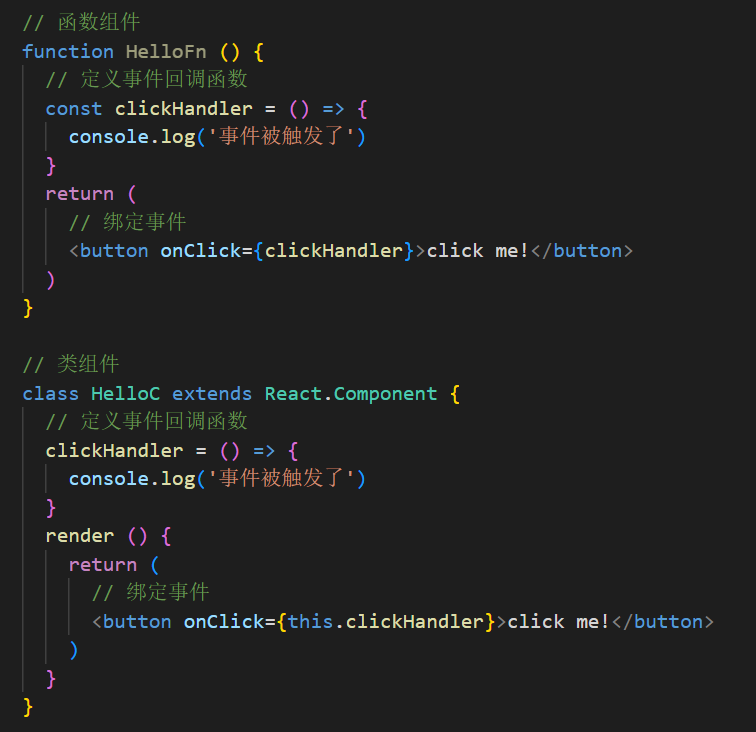
# 5.事件绑定

## 5.1绑定事件

语法：on+事件名称= {事件处理程序}

比如：<div onClick={()=>{}}></div>

注意点：react事件采用驼峰命名法，比如：onMouseEnter、onFocus



## 5.2获取事件对象

实现：通过事件处理程序的参数获取事件对象**e**



# 组件状态

注意：函数式组件是没有自己的状态的，仅类组件有自己的状态

## 6.1初始化状态

实现：通过class的实例属性state来初始化

说明：state的值是一个对象结构，表示一个组件可以有多个数据状态

## 6.2读取状态

实现：通过this.state来获取状态

## 6.3修改状态

实现：数据驱动视图，只要修改数据状态,页面就会自动刷新,无需手动操作dom

语法：this.setState({ 要修改的部分数据 })

setState方法作用：

* 修改state中的数据状态
* 更新UI

注意：不要直接修改state中的值，必须通过setState方法进行修改

**代码示例**



# 7.React组件通信

## 7.1父子组件

**父传子实现：**

1.父组件提供要传递的数据 - msg

2.给子组件标签添加属性值为msg中的数据

3.子组件中通过 props 接收父组件中传过来的数据

* + 类组件使用this.props获取props对象
  + 函数式组件直接通过参数获取props对象

**子传父实现：**

1.父组件提供一个回调函数 - 用于接收数据

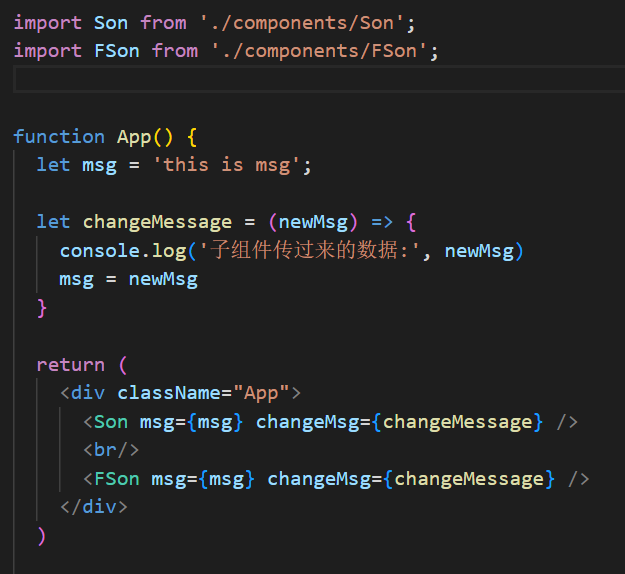
2.将函数作为属性的值，传给子组件

3.子组件通过props调用 回调函数

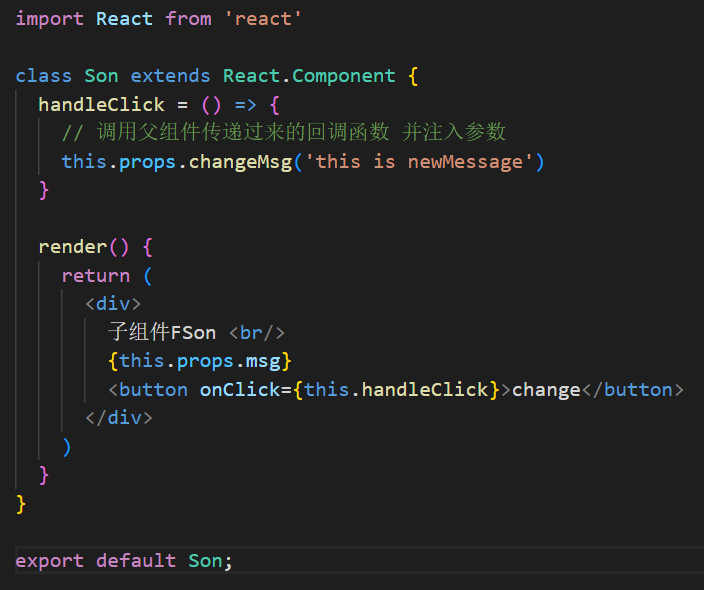
4.将子组件中的数据作为参数传递给回调函数

**代码示例：**

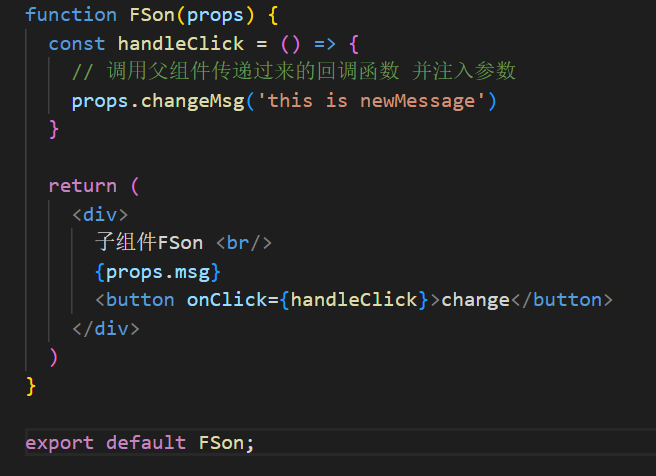
父组件



子组件Son



子组件FSon



## 7.2 兄弟组件

思路：通过状态提升机制，利用共同的父组件实现兄弟通信

实现步骤：

1.将共享状态提升到最近的公共父组件中，由公共父组件管理这个状态

* 提供共享状态
* 提供操作共享状态的方法

2.要接收数据状态的子组件通过 props 接收数据

3.要传递数据状态的子组件通过props接收方法，调用方法传递数据

## 7.3跨组件通信Context

思路：若采取的方式就是一层一层的props往下传，显然很繁琐；所以Context 提供了一个无需为每层组件手动添加 props，就能在组件树间进行数据传递的方法。

**实现步骤：**

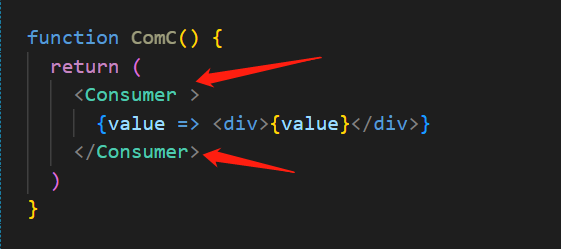
1. 创建Context对象 导出 Provider 和 Consumer对象



1. 使用Provider包裹根组件提供数据



1. 需要用到数据的组件使用Consumer包裹获取数据



# 8.React生命周期

## 8.1挂载阶段

| **钩子函数** | **触发时机** | **作用** |
| --- | --- | --- |
| constructor | 创建组件时，最先执行，初始化的时候只执行一次 | 1.初始化state  2.创建 Ref  3.使用bind解决this指向问题等 |
| render | 每次组件渲染都会触发 | 渲染UI（**注意： 不能在里面调用setState()** ） |
| componentDidMount | 组件挂载（完成DOM渲染）后执行，初始化的时候执行一次 | 1. 发送网络请求   2.DOM操作 |

## 8.2更新阶段

| **钩子函数** | **触发时机** | **作用** |
| --- | --- | --- |
| render | 每次组件渲染都会触发 | 渲染UI（与 挂载阶段 是同一个render） |
| componentDidUpdate | 组件更新后（DOM渲染完毕） | DOM操作，可以获取到更新后的DOM内容，**不要直接调用setState** |

## 8.3卸载阶段

| **钩子函数** | **触发时机** | **作用** |
| --- | --- | --- |
| componentWillUnmount | 组件卸载(从页面中消失） | 执行清理工作（如:清理定时器等） |

# 9.Hooks基础

说明：一套能够使函数组件更强大，更灵活的“钩子”

注意点：

1.有hooks后，为兼容老版本，类组件并没有被移除，俩者都可以使用

2.有hooks后，函数组件不再是无状态组件，因为hooks为它提供了状态

3.hooks只能在函数组件中使用

## 9.1 UseState

**作用：**useState为函数组件提供状态（state）

**参数：**状态初始值比如,传入 0 表示该状态的初始值为 0

**返回值：**数组，包含两个值：1 状态值（state） 2 修改该状态的函数（setState）

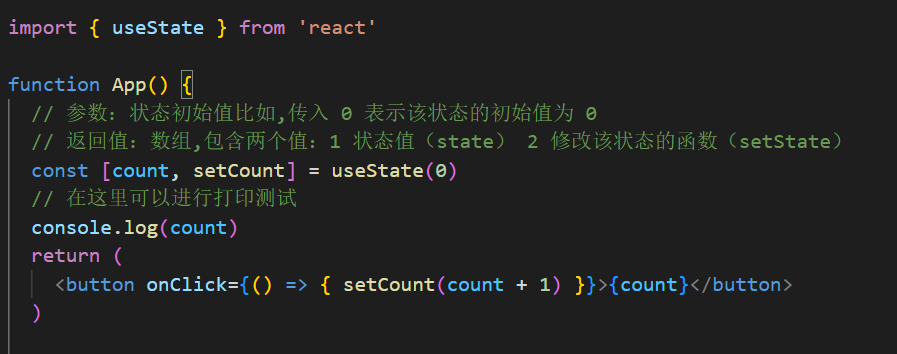
**基础使用：**

1. 导入 useState 函数
2. 调用 useState 函数，并传入状态的初始值
3. 从useState函数的返回值中，拿到状态和修改状态的方法
4. 在JSX中展示状态
5. 调用修改状态的方法更新状态

**状态的读取和修改：**

1. setState是一个函数，参数表示最新的状态值
2. 调用该函数后，将使用新值替换旧值
3. 修改状态后，由于状态发生变化，会引起视图变化

**代码示例：**



执行过程：

组件第一次渲染

* 从头开始执行该组件中的代码逻辑
* 调用 useState(0) 将传入的参数作为状态初始值，即：0
* 渲染组件，此时，获取到的状态 count 值为： 0

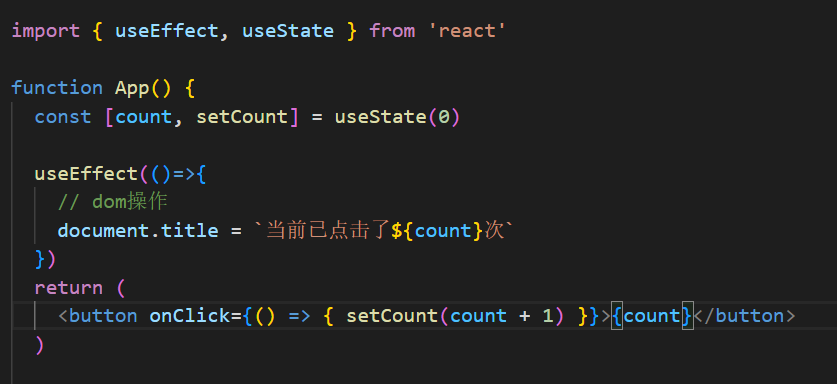
组件第二次渲染

* 点击按钮，调用 setCount(count + 1) 修改状态，因为状态发生改变，所以，该组件会重新渲染
* 组件重新渲染时，会再次执行该组件中的代码逻辑
* 再次调用 useState(0)，此时 React 内部会拿到最新的状态值而非初始值，比如，该案例中最新的状态值为 1
* 再次渲染组件，此时，获取到的状态 count 值为：1

## 9.2 UseEffect

实现步骤

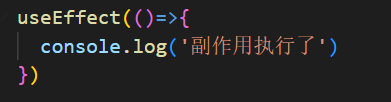
1. 导入 useEffect 函数
2. 调用 useEffect 函数，并传入回调函数
3. 在回调函数中编写副作用处理（dom操作）
4. 修改数据状态
5. 检测副作用是否生效



依赖项（可控制执行时机）

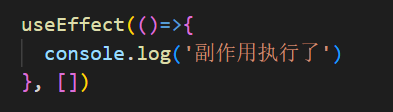
1. 不添加依赖项

* 组件初始渲染
* 组件更新都会重新执行（不管是哪个状态引起的组件更新）



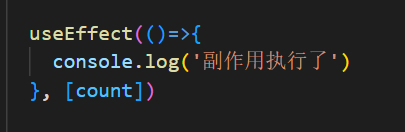
1. 添加空数组

* 组件只在首次渲染时执行一次



1. 添加特定依赖项

* 副作用函数在首次渲染时执行，在依赖项发生变化时重新执行



注意：

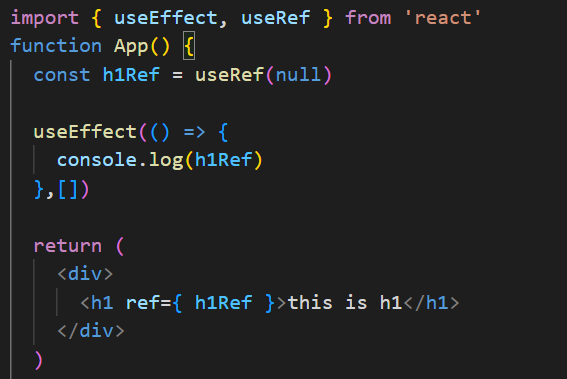
useEffect 回调函数中用到的数据（比如，count）就是依赖数据，须出现在依赖项数组中，若不添加依赖项就会有bug出现

## 9.3 UseRef

场景：在函数组件中获取真实的dom元素对象或者是组件对象

实现步骤：

1. 导入useRef函数
2. 执行useRef函数并传入null，返回值为一个对象，内部有一个current属性存放拿到的dom对象/组件实例
3. 通过ref 绑定，要获取的元素或者组件



注意：函数组件由于没有实例，不能使用ref获取，如果想获取组件实例，必须是类组件