# 组件通信

## 1.父亲传递数据给儿子

### 最简单的方式是子组件使用props属性接收父组件传递过来是数据

### 我们还是使用reactdemo3-component项目，在src目录下面新建一个文件夹：11react-props在这个文件夹里面新建一个FunctionProps.jsx组件文件，用快捷键rsc生成骨架，并且添加一些渲染代码

|  |
| --- |
| **import *React* from 'react'**;  **const** *FunctionProps* = (props) => {  **return** (  <**div**>  <**h4**>name : {props.**name**}</**h4**>  <**h4**>age: {props.**age**}</**h4**>  <**h4**>gender:{props.**gender**}</**h4**>  <**h4**>money:{props.money}</**h4**>  </**div**>  ); };  **export default** *FunctionProps*; |

### 然后在App.jsx文件里面添加对组件的引用，注意此时必须传递参数

|  |
| --- |
| **import** { *useState* } **from 'react'** *// import reactLogo from './assets/react.svg' // import viteLogo from '/vite.svg' // import './App.css'* **import** *FuncComponent* **from "./01-compoment-basic/FuncComponent"**; **import** ClassComponent **from "./01-compoment-basic/ClassComponent"**; **import** ClassEvent **from "./02-class-component-events/ClassEvent"**; **import** ClassState **from "./03class-component-state/ClassState"**; **import** ClassComputed **from "./04class-computed-property/ClassComputed" import** ClassLifeCycle **from "./05class-comp-life-cycle/ClassLifeCycle"**; *// import SideMenu from "./side-menu/SideMenu";* **import** SideMenu2 **from "./side-menu-v2/SideMenu2"**; **import** ComponentStyle **from "./06component-style/ComponentStyle"**; **import** *FunctionState* **from "./07readct-hook-useState/FunctionState"**; **import** *FunctionLifeCycle* **from "./08react-Hook-useEffect/FunctionLifeCycle"**; **import** *FunctionComputed* **from "./09react-Hook-useMemo/FunctionComputed"**; **import** *FunctionComputed2* **from "./09react-Hook-useMemo/FunctionComputed2"**; **import** *FunctionRef* **from "./10react-Hook-useRef/FunctionRef"**; **import** *TodoList* **from "./exercise-todoList/TodoList"**; **import** *FunctionProps* **from "./11react-props/FunctionProps"**;  **function** *App*() {   **return** (  <>  {*/\*<h1>React demo3</h1>\*/*}  {*/\*<FuncComponent/>\*/*}  {*/\*<ClassComponent/>\*/*}  {*/\*<ClassEvent/>\*/*}  {*/\* <ClassState></ClassState>\*/*}  {*/\* <ClassComputed></ClassComputed>\*/*}  {*/\* <ClassLifeCycle></ClassLifeCycle>\*/*}  {*/\* <SideMenu></SideMenu>\*/*}  {*/\* <SideMenu2/>\*/*}  {*/\* <ComponentStyle></ComponentStyle>\*/*}  {*/\* <FunctionState></FunctionState>\*/*}  {*/\* <FunctionLifeCycle></FunctionLifeCycle>\*/*}  {*/\* <FunctionComputed></FunctionComputed>\*/*}  {*/\* <FunctionComputed2></FunctionComputed2>\*/*}  {*/\* <FunctionRef/>\*/*}  {*/\*<TodoList/>\*/*}  <**FunctionProps name = 'jack' age=**{30} **gender='male' money=**{100000} />  </>  ) }  **export default** *App* |

### 效果

|  |
| --- |
|  |

### App.jsx还可以怎么写

|  |
| --- |
| **import** { *useState* } **from 'react'** *// import reactLogo from './assets/react.svg' // import viteLogo from '/vite.svg' // import './App.css'* **import** *FuncComponent* **from "./01-compoment-basic/FuncComponent"**; **import** ClassComponent **from "./01-compoment-basic/ClassComponent"**; **import** ClassEvent **from "./02-class-component-events/ClassEvent"**; **import** ClassState **from "./03class-component-state/ClassState"**; **import** ClassComputed **from "./04class-computed-property/ClassComputed" import** ClassLifeCycle **from "./05class-comp-life-cycle/ClassLifeCycle"**; *// import SideMenu from "./side-menu/SideMenu";* **import** SideMenu2 **from "./side-menu-v2/SideMenu2"**; **import** ComponentStyle **from "./06component-style/ComponentStyle"**; **import** *FunctionState* **from "./07readct-hook-useState/FunctionState"**; **import** *FunctionLifeCycle* **from "./08react-Hook-useEffect/FunctionLifeCycle"**; **import** *FunctionComputed* **from "./09react-Hook-useMemo/FunctionComputed"**; **import** *FunctionComputed2* **from "./09react-Hook-useMemo/FunctionComputed2"**; **import** *FunctionRef* **from "./10react-Hook-useRef/FunctionRef"**; **import** *TodoList* **from "./exercise-todoList/TodoList"**; **import** *FunctionProps* **from "./11react-props/FunctionProps"**;  **function** *App*() {  **let** props={  **name** :**'jack'** ,  **age**:35,  **gender**:**'male'** ,  **money**:9000000  }  **return** (  <>  {*/\*<h1>React demo3</h1>\*/*}  {*/\*<FuncComponent/>\*/*}  {*/\*<ClassComponent/>\*/*}  {*/\*<ClassEvent/>\*/*}  {*/\* <ClassState></ClassState>\*/*}  {*/\* <ClassComputed></ClassComputed>\*/*}  {*/\* <ClassLifeCycle></ClassLifeCycle>\*/*}  {*/\* <SideMenu></SideMenu>\*/*}  {*/\* <SideMenu2/>\*/*}  {*/\* <ComponentStyle></ComponentStyle>\*/*}  {*/\* <FunctionState></FunctionState>\*/*}  {*/\* <FunctionLifeCycle></FunctionLifeCycle>\*/*}  {*/\* <FunctionComputed></FunctionComputed>\*/*}  {*/\* <FunctionComputed2></FunctionComputed2>\*/*}  {*/\* <FunctionRef/>\*/*}  {*/\*<TodoList/>\*/*}  {*/\* <FunctionProps name = 'jack' age={30} gender='male' money={100000} />\*/*}  <**FunctionProps** {...props} />  </>  ) }  **export default** *App* |

### 实例

### 在11react-props文件夹里面新建两个组件文件：FatherProp.jsx和ChildProp.jsx文件，用快捷键rsc回车创建文件骨架，并且在父组件中添加对子组件的引用，

### FatherProp.jsx

|  |
| --- |
| **import *React* from 'react'**; **import** *ChildProp* **from "./ChildProp"**;  **const** *FatherProp* = () => {  **return** (  <**div**>  <**h3**>FatherProp</**h3**>  <**ChildProp**/>  </**div**>  ); };  **export default** *FatherProp*; |

### ChildProp.jsx

|  |
| --- |
| **import *React* from 'react'**;  **const** *ChildProp* = () => {  **return** (  <**div**>  <**h3**>ChildProp</**h3**>  </**div**>  ); };  **export default** *ChildProp*; |

### 然后在App.jsx中添加对FatherProp.jsx的引用

|  |
| --- |
| **import** { *useState* } **from 'react'** *// import reactLogo from './assets/react.svg' // import viteLogo from '/vite.svg' // import './App.css'* **import** *FuncComponent* **from "./01-compoment-basic/FuncComponent"**; **import** ClassComponent **from "./01-compoment-basic/ClassComponent"**; **import** ClassEvent **from "./02-class-component-events/ClassEvent"**; **import** ClassState **from "./03class-component-state/ClassState"**; **import** ClassComputed **from "./04class-computed-property/ClassComputed" import** ClassLifeCycle **from "./05class-comp-life-cycle/ClassLifeCycle"**; *// import SideMenu from "./side-menu/SideMenu";* **import** SideMenu2 **from "./side-menu-v2/SideMenu2"**; **import** ComponentStyle **from "./06component-style/ComponentStyle"**; **import** *FunctionState* **from "./07readct-hook-useState/FunctionState"**; **import** *FunctionLifeCycle* **from "./08react-Hook-useEffect/FunctionLifeCycle"**; **import** *FunctionComputed* **from "./09react-Hook-useMemo/FunctionComputed"**; **import** *FunctionComputed2* **from "./09react-Hook-useMemo/FunctionComputed2"**; **import** *FunctionRef* **from "./10react-Hook-useRef/FunctionRef"**; **import** *TodoList* **from "./exercise-todoList/TodoList"**; **import** *FunctionProps* **from "./11react-props/FunctionProps"**; **import** *FatherProp* **from "./11react-props/FatherProp"**;  **function** *App*() {  **let** props={  **name** :**'jack'** ,  **age**:35,  **gender**:**'male'** ,  **money**:9000000  }  **return** (  <>  {*/\*<h1>React demo3</h1>\*/*}  {*/\*<FuncComponent/>\*/*}  {*/\*<ClassComponent/>\*/*}  {*/\*<ClassEvent/>\*/*}  {*/\* <ClassState></ClassState>\*/*}  {*/\* <ClassComputed></ClassComputed>\*/*}  {*/\* <ClassLifeCycle></ClassLifeCycle>\*/*}  {*/\* <SideMenu></SideMenu>\*/*}  {*/\* <SideMenu2/>\*/*}  {*/\* <ComponentStyle></ComponentStyle>\*/*}  {*/\* <FunctionState></FunctionState>\*/*}  {*/\* <FunctionLifeCycle></FunctionLifeCycle>\*/*}  {*/\* <FunctionComputed></FunctionComputed>\*/*}  {*/\* <FunctionComputed2></FunctionComputed2>\*/*}  {*/\* <FunctionRef/>\*/*}  {*/\*<TodoList/>\*/*}  {*/\* <FunctionProps name = 'jack' age={30} gender='male' money={100000} />\*/*}  {*/\* <FunctionProps {...props} />\*/*}  <**FatherProp**/>  </>  ) }  **export default** *App* |

### 测试一下，正常

|  |
| --- |
|  |

### 好我们在父组件里面调用子组件，我们想给子组件传递一些值，如下

|  |
| --- |
| //FatherPop.jsx  **import *React***, {*useState*} **from 'react'**; **import** *ChildProp* **from "./ChildProp"**;  **const** *FatherProp* = () => {  **let** [gender,setGender] = *useState*(**'男'**)  **return** (  <**div**>  <**h2**>FatherProp</**h2**>  <**ChildProp name='Jack' age=**{25} **gender=**{gender}/>  </**div**>  ); };  **export default** *FatherProp*; |

### 此时不会报错，但是子组件没有接收到数据，因为子组件压根没有此时

### 我们可以给子组件添加一个形参props对象来接受父组件传递过来的值，然后使用

|  |
| --- |
| **import *React* from 'react'**;  **const** *ChildProp* = (props) => {  **return** (  <**div**>  <**h3**>ChildProp</**h3**>  <**p**>name:{props.**name**}</**p**>  <**p**>age:{props.**age**}</**p**>  <**p**>gender:{props.**gender**}</**p**>  </**div**>  ); };  **export default** *ChildProp*; |

### 效果

|  |
| --- |
|  |

### 注意传递数据时，只有字符串类型可以直接写，其他类型都需要花括号，也可以使用解构表达式来获取数据

|  |
| --- |
| //ChildProp.jsx  **import *React* from 'react'**;  **const** *ChildProp* = (props) => {  **let** {name,age,gender} = props  **return** (  <**div**>  <**h3**>ChildProp</**h3**>  <**p**>name:{name}</**p**>  <**p**>age:{age}</**p**>  <**p**>gender:{gender}</**p**>  </**div**>  ); };  **export default** *ChildProp*; |

### 效果是一样的

### 还可以怎么写

|  |
| --- |
| //ChildProp.jsx  **import *React* from 'react'**;  **const** *ChildProp* = ({name,age,gender}) => {  **return** (  <**div**>  <**h3**>ChildProp</**h3**>  <**p**>name:{name}</**p**>  <**p**>age:{age}</**p**>  <**p**>gender:{gender}</**p**>  </**div**>  ); };  **export default** *ChildProp*; |

### 不过不建议怎么写

### Props默认值

### 我们可以给props设置默认值，注意这个默认值写在组件外面，使用的是组件的defaultProps

|  |
| --- |
| //ChildProp.jsx  **import *React* from 'react'**;  **const** *ChildProp* = (props) => {  **let** {name,age,gender} = props  **return** (  <**div**>  <**h3**>ChildProp</**h3**>  <**p**>name:{name}</**p**>  <**p**>age:{age}</**p**>  <**p**>gender:{gender}</**p**>  </**div**>  ); }; *ChildProp*.**defaultProps**={  **gender**:**'女'** } **export default** *ChildProp*; |

### 此时如果父组件不传递gender，也可以正常显示

|  |
| --- |
| **import *React***, {*useState*} **from 'react'**; **import** *ChildProp* **from "./ChildProp"**;  **const** *FatherProp* = () => {  **let** [gender,setGender] = *useState*(**'男'**)  **return** (  <**div**>  <**h2**>FatherProp</**h2**>  <**ChildProp name='Jack' age=**{25} />  </**div**>  ); };  **export default** *FatherProp*; |

### 效果

|  |
| --- |
|  |

### props的验证

### 可以在需要接收数据的组件里面指定需要接收数据的数据类型，如果外部组件传递的数据类型不匹配就会抛异常

### 需要一个插件：prop-types

#### 我们使用yarn安装：yarn add prop-types

#### 我们在子组件里面添加验证代码，还是在组件外面设置

|  |
| --- |
| **import *React* from 'react'**; **import** pt **from 'prop-types' const** *ChildProp* = (props) => {  **let** {name,age,gender} = props  **return** (  <**div**>  <**h3**>ChildProp</**h3**>  <**p**>name:{name}</**p**>  <**p**>age:{age}</**p**>  <**p**>gender:{gender}</**p**>  </**div**>  ); }; *ChildProp*.**defaultProps**={  **gender**:**'女'** } *ChildProp*.**propTypes** = {  **name**:pt.***string***,  **age**:pt.***number***,  **gender**:pt.***string*** } **export default** *ChildProp*; |

### 此时我们在FatherProp的组件引用里面把age的值改为“25”

|  |
| --- |
| **import *React***, {*useState*} **from 'react'**; **import** *ChildProp* **from "./ChildProp"**;   **const** *FatherProp* = () => {  **let** [gender,setGender] = *useState*(**'男'**)  **return** (  <**div**>  <**h2**>FatherProp</**h2**>  <**ChildProp name='Jack' age='25'** />  </**div**>  ); };  **export default** *FatherProp*; |

### 运行，发现报错了

|  |
| --- |
|  |

### 说明验证生效了

### 我们把age重新改为number，它有正常了

|  |
| --- |
| **import *React***, {*useState*} **from 'react'**; **import** *ChildProp* **from "./ChildProp"**;   **const** *FatherProp* = () => {  **let** [gender,setGender] = *useState*(**'男'**)  **return** (  <**div**>  <**h2**>FatherProp</**h2**>  <**ChildProp name='Jack' age=**{30} />  </**div**>  ); };  **export default** *FatherProp*; |
|  |

### prop-types常见类型

### 以pt来代表prop-types

|  |  |
| --- | --- |
| 类型 | 说明 |
| **pt.string** | **字符串** |
| **pt.number** | **数字** |
| **pt.bool** | **布尔值** |
| **pt.array** | **数组** |
| **pt.object** | **对象** |
| **pt.func** | **函数** |
| **pt.oneOfType([pt.type1,pt.type2,pt.type3])** | **是指定类型的其中一种** |

## 2.儿子传递数据给父亲

思路：父组件传递一个方法(回调函数)给子组件，子组件通过props接收到父组件的方法，然后调用子组件的方法并传值给父组件，也就是说子传父是通过回调函数来实现的

### 实例，

### 还是上面的，我们来实现子传父功能，我们在父组件里面添加一个getChildData方法然后传递给子组件

#### FatherProp.jsx

|  |
| --- |
| **import *React***, {*useState*} **from 'react'**; **import** *ChildProp* **from "./ChildProp"**;   **const** *FatherProp* = () => {  **let** [gender,setGender] = *useState*(**'男'**)  **const** getChildData = (childData)=>{  ***console***.log(**'childData'**,childData)  }  **return** (  <**div**>  <**h2**>FatherProp</**h2**>  <**ChildProp name='Jack' age=**{30} **getChildData=**{getChildData}/>  </**div**>  ); };  **export default** *FatherProp*; |

### 然后儿子需要调用这个方法传递数据

#### ChildProp.jsx

|  |
| --- |
| **import *React* from 'react'**; **import** pt **from 'prop-types' const** *ChildProp* = (props) => {  **let** {name,age,gender,getChildData} = props  *// let passToDad = ()=>{  // getChildData({  // name:'Son',  // age:20,  // gender:'male'  // })  // }* **return** (  <**div**>  <**h3**>ChildProp</**h3**>  <**p**>name:{name}</**p**>  <**p**>age:{age}</**p**>  <**p**>gender:{gender}</**p**>  <**button onClick=**{()=> getChildData({  **name**:**'Son'**,  **age**:20,  **gender**:**'male'** })}>传递数据给父亲</**button**>   </**div**>  ); }; *ChildProp*.**defaultProps**={  **gender**:**'女'** } *ChildProp*.**propTypes** = {  **name**:pt.***string***,  **age**:pt.***number***,  **gender**:pt.***string***,   } **export default** *ChildProp*; |

### 效果

|  |
| --- |
|  |

## 3.兄弟组件传递数据

### 兄弟之间其实可以间接传递数据其实可以通过父组件来实现，不过现在有另外一个办法，就是事件总线。事件总线event bus需要安装一个第三方库events

### 安装：yarn add events 或者npm install events

### 实例

### 还是使用reactdemo3项目，我们在src下面新建一个文件夹12comp-communication，然后在这个文件夹下面新建3个组件文件FatherEvent.jsx,ChildA.jsx,ChildB.jsx,使用快捷键rsc插件骨架，

FatherEvent.jsx

|  |
| --- |
| **import *React* from 'react'**; **import** *ChildA* **from "./ChildA"**; **import** *ChildB* **from "./ChildB"**;  **const** *FatherEvent* = () => {  **return** (  <**div**>  <**ChildA**/>  <**ChildB**/>  </**div**>  ); };  **export default** *FatherEvent*; |

ChildA.jsx

|  |
| --- |
| **import *React* from 'react'**;  **const** *ChildA* = () => {  **return** (  <**div**>    </**div**>  ); };  **export default** *ChildA*; |

ChildB.jsx

|  |
| --- |
| **import *React* from 'react'**;   **const** *ChildB* = () => {  **return** (  <**div**>   </**div**>  ); };  **export default** *ChildB*; |

### 然后在App.jsx在添加对父组件的引用

|  |
| --- |
| **import** { *useState* } **from 'react'** *// import reactLogo from './assets/react.svg' // import viteLogo from '/vite.svg' // import './App.css'* **import** *FuncComponent* **from "./01-compoment-basic/FuncComponent"**; **import** ClassComponent **from "./01-compoment-basic/ClassComponent"**; **import** ClassEvent **from "./02-class-component-events/ClassEvent"**; **import** ClassState **from "./03class-component-state/ClassState"**; **import** ClassComputed **from "./04class-computed-property/ClassComputed" import** ClassLifeCycle **from "./05class-comp-life-cycle/ClassLifeCycle"**; *// import SideMenu from "./side-menu/SideMenu";* **import** SideMenu2 **from "./side-menu-v2/SideMenu2"**; **import** ComponentStyle **from "./06component-style/ComponentStyle"**; **import** *FunctionState* **from "./07readct-hook-useState/FunctionState"**; **import** *FunctionLifeCycle* **from "./08react-Hook-useEffect/FunctionLifeCycle"**; **import** *FunctionComputed* **from "./09react-Hook-useMemo/FunctionComputed"**; **import** *FunctionComputed2* **from "./09react-Hook-useMemo/FunctionComputed2"**; **import** *FunctionRef* **from "./10react-Hook-useRef/FunctionRef"**; **import** *TodoList* **from "./exercise-todoList/TodoList"**; **import** *FunctionProps* **from "./11react-props/FunctionProps"**; **import** *FatherProp* **from "./11react-props/FatherProp"**; **import** *FatherEvent* **from "./12comp-communication/FatherEvent"**;  **function** *App*() {  **let** props={  **name** :**'jack'** ,  **age**:35,  **gender**:**'male'** ,  **money**:9000000  }  **return** (  <>  {*/\*<h1>React demo3</h1>\*/*}  {*/\*<FuncComponent/>\*/*}  {*/\*<ClassComponent/>\*/*}  {*/\*<ClassEvent/>\*/*}  {*/\* <ClassState></ClassState>\*/*}  {*/\* <ClassComputed></ClassComputed>\*/*}  {*/\* <ClassLifeCycle></ClassLifeCycle>\*/*}  {*/\* <SideMenu></SideMenu>\*/*}  {*/\* <SideMenu2/>\*/*}  {*/\* <ComponentStyle></ComponentStyle>\*/*}  {*/\* <FunctionState></FunctionState>\*/*}  {*/\* <FunctionLifeCycle></FunctionLifeCycle>\*/*}  {*/\* <FunctionComputed></FunctionComputed>\*/*}  {*/\* <FunctionComputed2></FunctionComputed2>\*/*}  {*/\* <FunctionRef/>\*/*}  {*/\*<TodoList/>\*/*}  {*/\* <FunctionProps name = 'jack' age={30} gender='male' money={100000} />\*/*}  {*/\* <FunctionProps {...props} />\*/*}  {*/\* <FatherProp/>\*/*}  <**FatherEvent**/>  </>  ) }  **export default** *App* |

### 注意：使用父组件间接传递的方法比用event bus好

### 然后我们再创建一个events.js文件，在这里创建事件总线对象

#### reactdemo3-components/src/12comp-communication/events.js

|  |
| --- |
| **import** {EventEmitter} **from 'events' export default new** EventEmitter() |

### 在需要接受数据的组件中给事件总线添加一个方法，如ChildB

|  |
| --- |
| **import *React***, {*useEffect*} **from 'react'**; **import** events **from './events'  const** *ChildB* = () => {  *useEffect*(()=>{  events.addListener(**'getData'**,(data)=>{  ***console***.log(**"组件A数据："**,data);  })  },[])  **return** (  <**div**>  childb  </**div**>  ); };  **export default** *ChildB*; |

### 在需要传递数据的组件中调用其他组件给事件总线添加的方法

|  |
| --- |
| **import *React* from 'react'**; **import** events **from './events'  const** *ChildA* = () => {  **return** (  <**div**>  <**button onClick=**{  ()=> events.emit(**'getData'**,{**from**:**'ChildA'**,**msg**:**'Hello ChildB'**,**value**:1000})  }>传递数据</**button**>  </**div**>  ); };  **export default** *ChildA*; |

### 效果

#### 点击按钮前

|  |
| --- |
|  |

#### 点击按钮后

|  |
| --- |
|  |

## 4.多级嵌套组件之间的数据传递

### 这个如果使用父传子，子传孙的方法比较麻烦，可以使用一个叫做useContext的Hook来实现

### 我们在src下面新建一个文件夹13multilayer-comp-communication，在里面新建GrandFather.jsx，Father.jsx和Son.jsx,用快捷键rsc创建骨架

|  |
| --- |
| //GrandFather.jsx  **import *React* from 'react'**; **import** *Father* **from "./Father"**;  **const** *GrandFather* = () => {  **return** (  <**div style=**{{**border**:**'1px solid deeppink'**}}>  <**h3**>GrandFather</**h3**>  <**Father**/>  </**div**>  ); };  **export default** *GrandFather*; |
| //Father.jsx  **import *React* from 'react'**; **import** *Son* **from "./Son"**;  **const** *Father* = () => {  **return** (  <**div style=**{{**border**:**'1px dashed skyblue'**}}>  <**h4**>father</**h4**>  <**Son**/>  </**div**>  ); };  **export default** *Father*; |
| //Son.jsx  **import *React* from 'react'**;  **const** *Son* = () => {  **return** (  <**div style=**{{**background**:**"#cccccc"**,**color**:**'deeppink'**}}>  son  </**div**>  ); };  **export default** *Son*; |

### 然后在App.jsx中添加对爷爷组件的引用

|  |
| --- |
| **import** { *useState* } **from 'react'** *// import reactLogo from './assets/react.svg' // import viteLogo from '/vite.svg' // import './App.css'* **import** *FuncComponent* **from "./01-compoment-basic/FuncComponent"**; **import** ClassComponent **from "./01-compoment-basic/ClassComponent"**; **import** ClassEvent **from "./02-class-component-events/ClassEvent"**; **import** ClassState **from "./03class-component-state/ClassState"**; **import** ClassComputed **from "./04class-computed-property/ClassComputed" import** ClassLifeCycle **from "./05class-comp-life-cycle/ClassLifeCycle"**; *// import SideMenu from "./side-menu/SideMenu";* **import** SideMenu2 **from "./side-menu-v2/SideMenu2"**; **import** ComponentStyle **from "./06component-style/ComponentStyle"**; **import** *FunctionState* **from "./07readct-hook-useState/FunctionState"**; **import** *FunctionLifeCycle* **from "./08react-Hook-useEffect/FunctionLifeCycle"**; **import** *FunctionComputed* **from "./09react-Hook-useMemo/FunctionComputed"**; **import** *FunctionComputed2* **from "./09react-Hook-useMemo/FunctionComputed2"**; **import** *FunctionRef* **from "./10react-Hook-useRef/FunctionRef"**; **import** *TodoList* **from "./exercise-todoList/TodoList"**; **import** *FunctionProps* **from "./11react-props/FunctionProps"**; **import** *FatherProp* **from "./11react-props/FatherProp"**; **import** *FatherEvent* **from "./12comp-communication/FatherEvent"**; **import** *GrandFather* **from "./13multilayer-comp-communication/GrandFather"**;  **function** *App*() {  **let** props={  **name** :**'jack'** ,  **age**:35,  **gender**:**'male'** ,  **money**:9000000  }  **return** (  <>  {*/\*<h1>React demo3</h1>\*/*}  {*/\*<FuncComponent/>\*/*}  {*/\*<ClassComponent/>\*/*}  {*/\*<ClassEvent/>\*/*}  {*/\* <ClassState></ClassState>\*/*}  {*/\* <ClassComputed></ClassComputed>\*/*}  {*/\* <ClassLifeCycle></ClassLifeCycle>\*/*}  {*/\* <SideMenu></SideMenu>\*/*}  {*/\* <SideMenu2/>\*/*}  {*/\* <ComponentStyle></ComponentStyle>\*/*}  {*/\* <FunctionState></FunctionState>\*/*}  {*/\* <FunctionLifeCycle></FunctionLifeCycle>\*/*}  {*/\* <FunctionComputed></FunctionComputed>\*/*}  {*/\* <FunctionComputed2></FunctionComputed2>\*/*}  {*/\* <FunctionRef/>\*/*}  {*/\*<TodoList/>\*/*}  {*/\* <FunctionProps name = 'jack' age={30} gender='male' money={100000} />\*/*}  {*/\* <FunctionProps {...props} />\*/*}  {*/\* <FatherProp/>\*/*}  {*/\* <FatherEvent/>\*/*}  <**GrandFather**/>  </>  ) }  **export default** *App* |

### 测试一下，效果如下

|  |
| --- |
|  |

### 需要新建一个叫做context.js的js文件

reactdemo3-components/src/13multilayer-comp-communication/context.js

|  |
| --- |
| **import** {*createContext*} **from "react"**; **export default** *createContext*(**null**) |

### 在爷爷组件里面添加代码把数据传递给爸爸，需要使用<Context.Provider>和</Context.Provider>把爸爸包裹起来，然后给这个Provider添加value属性实现数据传递，注意这个数据是一个对象，可以传递多个状态

|  |
| --- |
| **import *React***, {*useState*} **from 'react'**; **import** *Father* **from "./Father"**; **import** Context **from './context'  const** *GrandFather* = () => {  **let** [count ,setCount] = *useState*(0)  **return** (  <**div style=**{{**border**:**'1px solid deeppink'**}}>  <**h3**>GrandFather</**h3**>  <**Context.Provider value=**{{count,setCount}}>  <**Father**/>  </**Context.Provider**>  </**div**>  ); };  **export default** *GrandFather*; |

### 注意，只需要把father组件作为提供者，标签father组件里面包含son子组件，son就可以 使用Context里面的状态，提供者不能作为Context的使用者，它在页面中不可见

### 我们在Son里面添加接受数据的代码，需要用到一个useContext的Hook

|  |
| --- |
| **import *React***, {*useContext*} **from 'react'**; **import** Context **from "./context"**;  **const** *Son* = () => {  **let** {count,setCount} = *useContext*(Context)  **return** (  <**div style=**{{**background**:**"#cccccc"**,**color**:**'deeppink'**}}>  <**h5**>Son</**h5**>  <**button onClick=**{()=>setCount(--count)}>-</**button**>  <**span** >{count}</**span**>  <**button onClick=**{()=>setCount(++count)}>+</**button**>  </**div**>  ); };  **export default** *Son*; |

#### 效果：点击+会增加计数，点击-会减小计数

|  |
| --- |
|  |

### 我们可以把代码修改一下，再做另外一个案例

#### GrandFather.jsx

|  |
| --- |
| **import *React***, {*useState*} **from 'react'**; **import** Context **from './context.js' import** *Father* **from "./Father"**;  **const** *GrandFather* = () => {  **let** [count,setCount] = *useState*(0)  **let** [num,setNum] = *useState*(0)   **return** (  <**div**>  <**Context.Provider value=**{{count, setCount,num,setNum}}>  <**Father**></**Father**>  </**Context.Provider**>   </**div**>  ); };  **export default** *GrandFather*; |

#### Father.jsx

|  |
| --- |
| **import *React* from 'react'**; **import** *Child1* **from "./Child1"**; **import** *Child2* **from "./Child2"**;  **const** *Father* = () => {  **return** (  <**div**>  <**Child1**></**Child1**>  <**Child2**></**Child2**>  </**div**>  ); };  **export default** *Father*; |

#### Child1.jsx

|  |
| --- |
| **import *React***, {*useContext*} **from 'react'**; **import** Context **from './context'  const** *Child1* = () => {  **let** {num,setNum}= *useContext*(Context)  **return** (  <**div**>   <**button onClick=**{()=>setNum(++num)}>+</**button**>  {num}  <**button onClick=**{()=>setNum(--num)}>-</**button**>  </**div**>  ); };  **export default** *Child1*; |

#### Child2.jsx

|  |
| --- |
| **import *React***, {*useContext*} **from 'react'**; **import** Context **from './context'  const** *Child2* = () => {  **let** {count,setCount} = *useContext*(Context)  **return** (  <**div**>   <**button onClick=**{()=>setCount(++count)}>+</**button**>  {count}  <**button onClick=**{()=>setCount(--count)}>-</**button**>  </**div**>  ); };  **export default** *Child2*; |

## 总结：在react中组件间数据传递就有这么4种方法

|  |  |
| --- | --- |
| **方式** | **实现** |
| 父亲传递给儿子 | props |
| 儿子传递数据给父亲 | 使用父亲提供的回调函数 |
| 兄弟之间传递数据（3种方式） | 使用父组件或者events或者状态机 |
| 祖先传递给后的 | 使用createContext方法返回Context，然后使用useContext这个Hook来获取Context |

# 函数组件的优化

## 1.函数组件存在哪些问题需要优化？

## 答：有太多渲染，有时候数据没有变化它也渲染

## 我们创建一个实例

### 在src文件夹下面新建一个文件夹：14func-component-optimize，然后在它下面新建一个FatherMemo.jsx和一个ChildMemo.jsx组件文件，用快捷键rsc创建骨架在父组件里面引用子组件

#### FatherMemo.jsx

|  |
| --- |
| **import *React* from 'react'**; **import** *ChildMemo* **from "./ChildMemo"**;  **const** *FatherMemo* = () => {  **return** (  <**div**>  <**ChildMemo**/>  </**div**>  ); };  **export default** *FatherMemo*; |

#### ChildMemo.jsx

|  |
| --- |
| **import *React* from 'react'**;  **const** *ChildMemo* = () => {  **return** (  <**div**>    </**div**>  ); };  **export default** *ChildMemo*; |

### 在App.jsx中添加对父组件的引用

|  |
| --- |
| **import** { *useState* } **from 'react'** *// import reactLogo from './assets/react.svg' // import viteLogo from '/vite.svg' // import './App.css'* **import** *FuncComponent* **from "./01-compoment-basic/FuncComponent"**; **import** ClassComponent **from "./01-compoment-basic/ClassComponent"**; **import** ClassEvent **from "./02-class-component-events/ClassEvent"**; **import** ClassState **from "./03class-component-state/ClassState"**; **import** ClassComputed **from "./04class-computed-property/ClassComputed" import** ClassLifeCycle **from "./05class-comp-life-cycle/ClassLifeCycle"**; *// import SideMenu from "./side-menu/SideMenu";* **import** SideMenu2 **from "./side-menu-v2/SideMenu2"**; **import** ComponentStyle **from "./06component-style/ComponentStyle"**; **import** *FunctionState* **from "./07readct-hook-useState/FunctionState"**; **import** *FunctionLifeCycle* **from "./08react-Hook-useEffect/FunctionLifeCycle"**; **import** *FunctionComputed* **from "./09react-Hook-useMemo/FunctionComputed"**; **import** *FunctionComputed2* **from "./09react-Hook-useMemo/FunctionComputed2"**; **import** *FunctionRef* **from "./10react-Hook-useRef/FunctionRef"**; **import** *TodoList* **from "./exercise-todoList/TodoList"**; **import** *FunctionProps* **from "./11react-props/FunctionProps"**; **import** *FatherProp* **from "./11react-props/FatherProp"**; **import** *FatherEvent* **from "./12comp-communication/FatherEvent"**; **import** *GrandFather* **from "./13multilayer-comp-communication/GrandFather"**; **import** *FatherMemo* **from "./14func-component-optimize/FatherMemo"**;  **function** *App*() {  **let** props={  **name** :**'jack'** ,  **age**:35,  **gender**:**'male'** ,  **money**:9000000  }  **return** (  <>  {*/\*<h1>React demo3</h1>\*/*}  {*/\*<FuncComponent/>\*/*}  {*/\*<ClassComponent/>\*/*}  {*/\*<ClassEvent/>\*/*}  {*/\* <ClassState></ClassState>\*/*}  {*/\* <ClassComputed></ClassComputed>\*/*}  {*/\* <ClassLifeCycle></ClassLifeCycle>\*/*}  {*/\* <SideMenu></SideMenu>\*/*}  {*/\* <SideMenu2/>\*/*}  {*/\* <ComponentStyle></ComponentStyle>\*/*}  {*/\* <FunctionState></FunctionState>\*/*}  {*/\* <FunctionLifeCycle></FunctionLifeCycle>\*/*}  {*/\* <FunctionComputed></FunctionComputed>\*/*}  {*/\* <FunctionComputed2></FunctionComputed2>\*/*}  {*/\* <FunctionRef/>\*/*}  {*/\*<TodoList/>\*/*}  {*/\* <FunctionProps name = 'jack' age={30} gender='male' money={100000} />\*/*}  {*/\* <FunctionProps {...props} />\*/*}  {*/\* <FatherProp/>\*/*}  {*/\* <FatherEvent/>\*/*}  {*/\* <GrandFather/>\*/*}  <**FatherMemo**/>  </>  ) }  **export default** *App* |

## 我们在父组件里面使用useState生成count和setCount，添加一个按钮，在点击事件里面调用setCount然后渲染到h3标签中

|  |
| --- |
| **//FatherMemo.jsx**  **import *React***, {*useState*} **from 'react'**; **import** *ChildMemo* **from "./ChildMemo"**;  **const** *FatherMemo* = () => {  ***console***.log(**'父组件'**)  **let** [count,setCount] = *useState*(0)  **return** (  <**div**>  <**h3**>{count}</**h3**>  <**button onClick=**{()=>setCount(++count)}>add</**button**>  <**ChildMemo**/>  </**div**>  ); };  **export default** *FatherMemo*; |

## 运行项目，发现当我们点击按钮时，即使子组件数据没有更新，也一起渲染了

|  |
| --- |
|  |

## 2.解决办法？

## 答：先使用React.memo把组件处理后，再把结果返回而不是直接返回组件

### 我们把上面的子组件的代码修改一下,

|  |
| --- |
| **//ChildMemo.jsx**  **import *React*** ,{*memo*} **from 'react'**;  **const** ChildMemo = () => {  ***console***.log(**'================子组件'**)  **return** (  <**div**>  <**h3**>child memo</**h3**>  </**div**>  ); };  **export default** *memo*(ChildMemo); |

### 先把memo函数解构出来，然后把子组件作为参数传递给memo函数，再把返回值导出

### 然后测试，发现点击按钮就只有父组件在渲染

|  |
| --- |
|  |

### 注意：当子组件的数据发生改变，它还是需要被渲染的，如我们修改一下父组件的代码，个体子组件传递一个count

|  |
| --- |
| **import *React***, {*useState*} **from 'react'**; **import** ChildMemo **from "./ChildMemo"**;  **const** *FatherMemo* = () => {  ***console***.log(**'父组件'**)  **let** [count,setCount] = *useState*(0)  **return** (  <**div**>  <**h3**>{count}</**h3**>  <**button onClick=**{()=>setCount(++count)}>add</**button**>  <**ChildMemo count=**{count}/>  </**div**>  ); };  **export default** *FatherMemo*; |

### 子组件代码做相应的修改

|  |
| --- |
| **import *React*** ,{*memo*} **from 'react'**;  **const** ChildMemo = (props) => {  **let** {count} = props  ***console***.log(**'================子组件count：'**,count)  **return** (  <**div**>  <**h3**>child memo</**h3**>  <**p**>Count:{count}</**p**>  </**div**>  ); };  **export default** *memo*(ChildMemo); |

### 此时每点击一次按钮，两个组件都会渲染，因为数据的确发生了变化

|  |
| --- |
|  |

### 好，我们把代码还原。现在不传数据我们给父组件添加一个方法并且传递一个给子组件，

|  |
| --- |
| **import *React***, {*useState*} **from 'react'**; **import** ChildMemo **from "./ChildMemo"**;  **const** *FatherMemo* = () => {  ***console***.log(**'父组件'**)  **let** [count,setCount] = *useState*(0)  **let** sayHello = ()=>{  ***console***.log(**'hello from dad'**)  }  **return** (  <**div**>  <**h3**>{count}</**h3**>  <**button onClick=**{()=>setCount(++count)}>add</**button**>  <**ChildMemo sayHello=**{sayHello} />  </**div**>  ); };  **export default** *FatherMemo*; |

### 我们发现此时点击按钮，即使子组件没有接收父组件的方法，子组件也会一起渲染

|  |
| --- |
|  |

### 这是为什么?我们只修改了count的值并没有修改函数，那它为什么会把子组件也渲染？

### 答：因为我们修改了父组件的数据，父组件就会重新渲染，每一次渲染sayHello函数的地址都不一样，所以传递给子组件的数据其实也是改变了，需要子组件要更新，需要重新渲染。

### 那这个问题这么解决？

### 需要使用一个叫做useCallback的Hook，他是专门用来缓存函数的，我们把父组件的代码修改一下

|  |
| --- |
| **import *React***, {*useCallback*, *useState*} **from 'react'**; **import** ChildMemo **from "./ChildMemo"**;  **const** *FatherMemo* = () => {  ***console***.log(**'父组件'**)  **let** [count,setCount] = *useState*(0)  **let** sayHello = ()=>{  **let** msg = **'hello from dad'**;  ***console***.log(msg)  **return** msg  }  **var** sayHelloCb = *useCallback*(sayHello,[]);  **return** (  <**div**>  <**h3**>{count}</**h3**>  <**button onClick=**{()=>setCount(++count)}>add</**button**>  <**ChildMemo sayHello=**{sayHelloCb} />  </**div**>  ); };  **export default** *FatherMemo*; |

### 在子组件中接受函数并且使用

|  |
| --- |
| **import *React*** ,{*memo*} **from 'react'**;  **const** ChildMemo = (props) => {   ***console***.log(**'================子组件'**)  **return** (  <**div**>  <**h3**>child memo</**h3**>  <**span**>Message:{props.**sayHello**()}</**span**>  </**div**>  ); };  **export default** *memo*(ChildMemo); |

### 注意：我们在父组件中用useCallback这个Hook给函数包装了一层，但是到了子组件里面看不见包装层，只有原来的函数

### 此时的确可以避免不必要的渲染

|  |
| --- |
|  |

### 总结：

### 默认情况下，当父组件更新，不管子组件里面的数据有没有改变子组件也会更新

### 函数组件的性能优化有2步，1.在子组件里面用React的memo函数把子组件出来一下在返回，其实就是用memo来缓存子组件。2.在父组件里面使用useCallback把需要传递和子组件的函数进行缓存处理。然后把处理后的结果传递给子组件，在子组件内部仍然只能看到原来的函数

### react有3个Hook可以用来缓存东西的

#### useMemo用来缓存数值数据

#### memo用来缓存组件

#### useCallback用来缓存函数