组件的通讯：

导语：在React中，数据流动是单向的，父组件向子组件的通讯也是最常见的；父组件主要通过props向子组件传递需要的信息。本节内容我们就来看一下父组件向子组件的通讯方式-属性传值。

回顾上节内容，我们定义了Person.js组件

const person = () => {

    return <p>React前端教程</p>

}

如果我们要显示 “React前端教程第xx节”，这里我们定义一个随机数，进行随机显示，这里我们需要使用JSX的{}运算符，定义js即可，具体实现形式如下：

const person = () => {

    return <p>React前端教程 第{Math.round(Math.random() \* 30)}节</p>

}

效果图如下:



可以看到当前组件中显示的内容都是写死的，这并不符合我们组件化的思想，如果我们想复用这个组件，我们希望调用组件时动态的传递需要显示的内容。来到App.js文件，在调用时传递需要显示的数据。具体实现形式如下：

<Person name="Mary" age='20'></Person >

<Person name="Lucy" age='18'></Person >

<Person name="Lily" age='30'></Person >

<Person name="Candy" age='25'></Person >

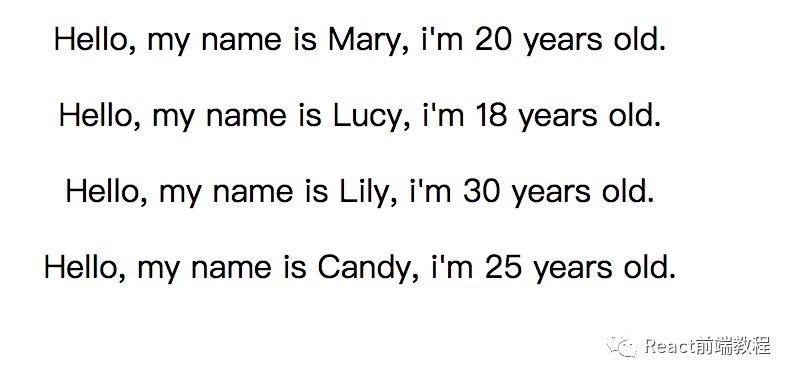
来到Person.js中接收传递过来的数据，这里我们定义一个propos，接收属性数据，具体实现代码如下：

const person = (props) => {

    return <p>Hello, my name is {props.name}, i'm {props.age} years old.</p>

}

效果图如下：



这里我们已将完成了属性传值，另外需要注意 propos  只是形参，我们可以替换为任意名字。

接下来如果我们的App.js代码修改为如下形式

<Person name="Mary" age='20'></Person >

<Person name="Lucy" age='18'></Person >

<Person name="Lily" age='30'></Person >

<Person name="Candy" age='25'>非常感谢大家支持我们的课程</Person >

如果要显示标签中嵌套的内容，我们可以使用propos.children拿到，为了更好的显示，我们将代码定义为如下形式：

const person = (props) => {

    return (

        <div>

            <p>Hello, my name is {props.name}, i'm {props.age} years old.</p>

            <p>{props.children}</p>

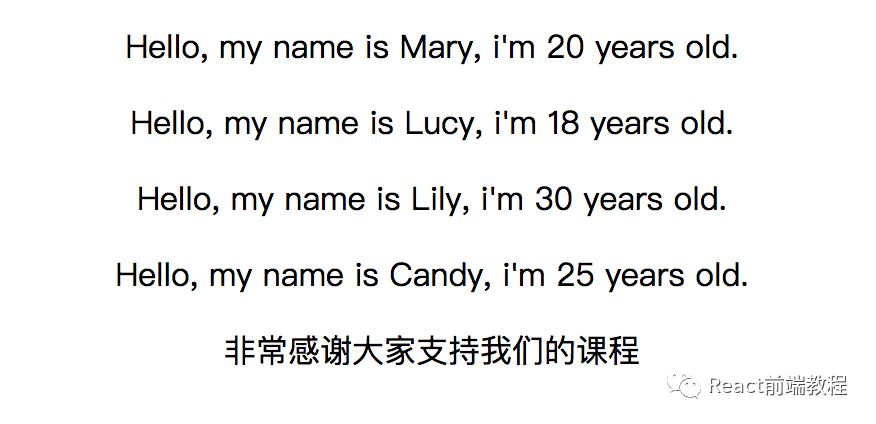
        </div>

      )

}

注意：组件中只能有一个根标签，这里我们使用div作为根标签。

效果图如下：



React-State状态的使用

原创 清音 [React前端教程](javascript:void(0);) 2018-07-25

    导语：在React中，核心的思想就是组件化思想,而组件中最重要的概念是Sate（状态)，state是组件渲染时的数据依据。

    上节我们使用的props也可以为组件提供数据，它们之间的区别是State是可变的，Props对于使用它的组件来说，是只读的，要想修改Props，只能通过该组件的父组件修改。

    本节内容我们来看一下state是如何使用的。

回顾上节内容，我们再向组件传值时，在调用时定义内容是在字符串中写死的，只能手动进行修改：

<Person name="Mary" age='20'></Person >

<Person name="Lucy" age='18'></Person >

<Person name="Lily" age='30'></Person >

<Person name="Candy" age='25'>非常感谢大家支持我们的课程</Person >

如果要动态修改组件内容，我们就要使用state。

**注意：要想定义state，需要在继承自Component的组件的类中。目前我们有App.js和Person.js两个组件， 来看一下各自定义形式：**

class App extends Component {}

const person = (props) => {}

我们可以在App.js组件中进行定义state,Person.js组件如要需要定义，需要引入Component并继承。

**定义state，首先来到App.js,具体实现代码如下：**

class App extends Component {

    state = {

        persons: [

            { name: 'Lucy', age: 18 },

            { name: 'Lily', age: 28 },

            { name: 'Mary', age: 38 },

            { name: 'Candy', age: 22 }]

    }

    render() {

        return (...);

    }

}

export default App;

**接下来是State的使用, 具体定义代码如下：**

render() {

    return (

    <div>

        <Person name={this.state.persons[0].name} age={this.state.persons[0].age}></Person >

        <Person name={this.state.persons[1].name} age={this.state.persons[1].age}></Person >

        <Person name={this.state.persons[2].name} age={this.state.persons[2].age}></Person >

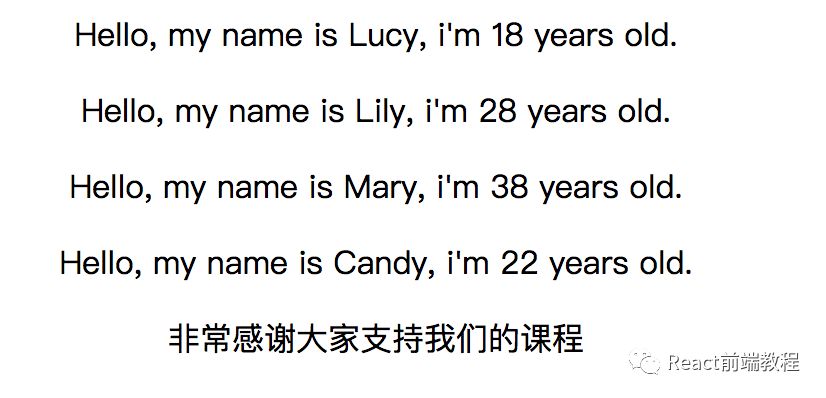
        <Person name={this.state.persons[3].name} age={this.state.persons[3].age}>非常感谢大家支持我们的课程</Person >

        </div>

    )

}

效果图如下：



**如果要修改state，首先定义button并添加点击事件，具体代码定义如下：**

<button>更改状态值</button>

// 在render上面定义方法，用来更改state

switchNameHandler = () => {

    console.log('hello')

}

调用方法

<button onClick={this.switchNameHandler}>更改状态值</button>

这里调用时，方法名后面并没有添加'()',如果添加括号，事件会自执行一次。接下来点击按钮并查看效果：



**因此事件可以正常执行，接下来在事件中进行state更改，在react中，并不能直接更改state,需要通过setState(),具体代码定义如下：**

switchNameHandler = () => {

    // console.log('hello')

    this.setState({

        persons: [

            { name: ' 露西 ', age: 100 },

            { name: 'Lily', age: 28 },

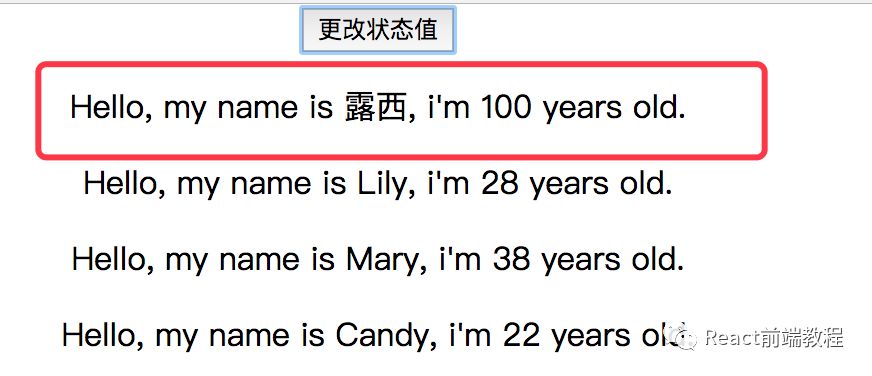
            { name: 'Mary', age: 38 },

            { name: 'Candy', age: 22 }]

     })

}

点击按钮，效果图如下：



**最后我们可以总结一下state和props的使用区别：state用于改变组件内容状态的值，props用于组件通讯进行传值。**

React-属性传值（传事件）

原创 清音 [React前端教程](javascript:void(0);) 2018-07-26

导语:本节内容我们主要介绍两个知识点，一个是如何给事件传参，另一个是如何通过属性传递事件。

回顾上节内容，我们定义了state并动态修改state的内容：

switchNameHandler = () => {

    this.setState({

        persons: [

            { name: '露西', age: 100 },

            { name: 'Lily', age: 28 },

            { name: 'Mary', age: 38 },

            { name: 'Candy', age: 22 }]

    })

}

在实际开发过程中，我们需要在触发事件时通过传递的参数来动态更改state的值。因此我们将上面代码更改为如下形式：

switchNameHandler = (newName) => {

    this.setState({

        persons: [

            { name: newName, age: 100 },

            { name: 'Lily', age: 28 },

            { name: 'Mary', age: 38 },

            { name: 'Candy', age: 22 }]

    })

}

上面的 newName 是传递过来的形参。因此在触发事件时，需要携带一个实参，将调用方法的代码定义为如下形式：

<button

    onClick={() => this.switchNameHandler('露西')}>

    更改状态值

</button>

或者

<button

    onClick={this.switchNameHandler.bind(this, '露西')}>

    更改状态值

</button>

在方法调用时如果要携带参数，有上面两种写法，推荐第二种写法。注意：在react中不能写为this.switchNameHandler('露西') 的形式。

上面我们在更改状态值时，直接触发的是父组件中的点击事件，如果我们希望点击子组件中的p标签来触发方法，该如何实现？

在React中，可以通过属性传递事件。具体实现代码形式如下：

首先App.js中定义事件属性

<Person

    name={this.state.persons[0].name}

    age={this.state.persons[0].age}

    myClick={this.switchNameHandler.bind(this, '露西Miss')}

></Person >

然后在Person.js中触发属性事件：

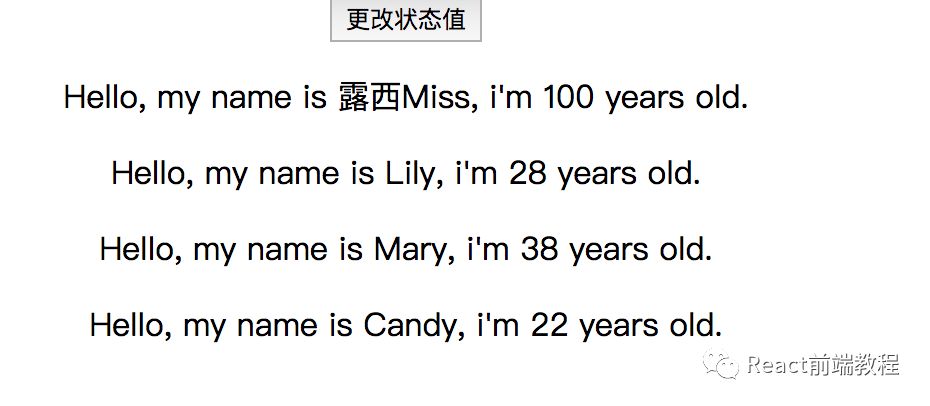
<p

    onClick={props.myClick}>

    Hello, my name is {props.name}, i'm {props.age} years old.

</p>

当我们点击第一个Person组件，效果图如下:



React-数据双向绑定

原创 清音 [React前端教程](javascript:void(0);) 2018-07-27

导语：与Vue、Angular自带的数据双向绑定指令相比，React是单向的数据更新。本节内容我们就来介绍一下如何在react中实现数据的双向绑定。

回顾之前内容，我们如果要更新数据，其实就是更新state。所以我们双向绑定的原理就是实时更新state值。

首先在Person.js中定义input，代码如下：

<input type="text"/>

效果图：

我们想实现的效果是，当在输入框输入内容时，name 也及时更新为输入的内容。

首先，我们需要定义onChange方法，来到Person.js，代码如下：

<input type="text" onChange={props.changed} />

接着，需要在App.js中 定义changed 属性来调用方法，代码如下：

<Person

    changed={this.nameCangedHandler}

    name={this.state.persons[0].name}

    age={this.state.persons[0].age}

    myClick={this.switchNameHandler.bind(this, '露西Miss')}

></Person >

接下来实现 nameCangedHandler 方法，代码如下：

nameCangedHandler = (event) => {

    this.setState({

        persons: [

            { name: event.target.value, age: 100 },

            { name: 'Lily', age: 28 },

            { name: 'Mary', age: 38 },

            { name: 'Candy', age: 22 }]

    })

}

通过 event.target.value  可以获取输入框输入的内容，效果图如下：

注意：现在只给第一个输入框添加的事件，如果是多个情况，我们会在循环的时候来实现。

如果我们希望输入框默然显示name 数据，在react中，需要定义defaultValue 属性，而不是value。具体定义代码形式如下：

<input type="text" onChange={props.changed} defaultValue={props.name} />

效果图如下：

React-组件样式的两种方式

原创 白清音 [React前端教程](javascript:void(0);) 2018-07-31

导语：与传统使用CSS给HTML添加样式的方式相比，React在给元素添加样式的时候方式有所不同。React的核心哲学之一就是让可视化的组件自包含，并且可复用。这就是为什么HTML元素和Javascript放在一起组成了Component（组件）。本节内容我们将介绍React定义样式的方式。

**第一种方式：选择器样式**

首先创建Person.css，定义我们所需要的样式，具体样式代码定义为如下形式：

.Person{ width: 60%; margin: 16px auto; border: 1px solid #eee; box-shadow: 0 2px 3px #ccc; padding: 16px; text-align: center;}

input {width: 200px;border: 1px solid #eee; outline: none;border-radius:10px;padding: 16px;}

如果要使用样式，需要在Person.js中引入：

import './Person.css'

这时可以查看页面变化：

可以看到我们定义的input标签选择器样式已经生效，如果要使用类选择器，需要使用className 属性：

这时样式就渲染上了，效果图如下：

**第二种方式：内联样式**

React推崇的是内联的方式定义样式。这样做的目的就在于让你的组件更加的容易复用。下面给按钮添加一个内联样式：

来到App.js文件，将button 按钮定义为如下形式：

<button

    style={{ backgroundColor: 'white', border: '1px solid blue', padding:'8px', cursor: 'pointer' }}>

    更改状态值

</button>

需要注意的是，JSX中使用样式对象定义内联样式，复合样式使用驼峰命名法，对象属性直接使用逗号隔开。

为了书写方法，我们通常将样式对象定义在render中，可以将上面代码改为如下形式：

render() {

    //  定义行内样式

    const style = {

        backgroundColor: 'white',

        border: '1px solid blue',

        outline: 'none',

        padding: '8px',

        cursor: 'pointer'

    }

    return (

        <div >

            <button  style={style}> 更改状态值</button>

        </div>

    )

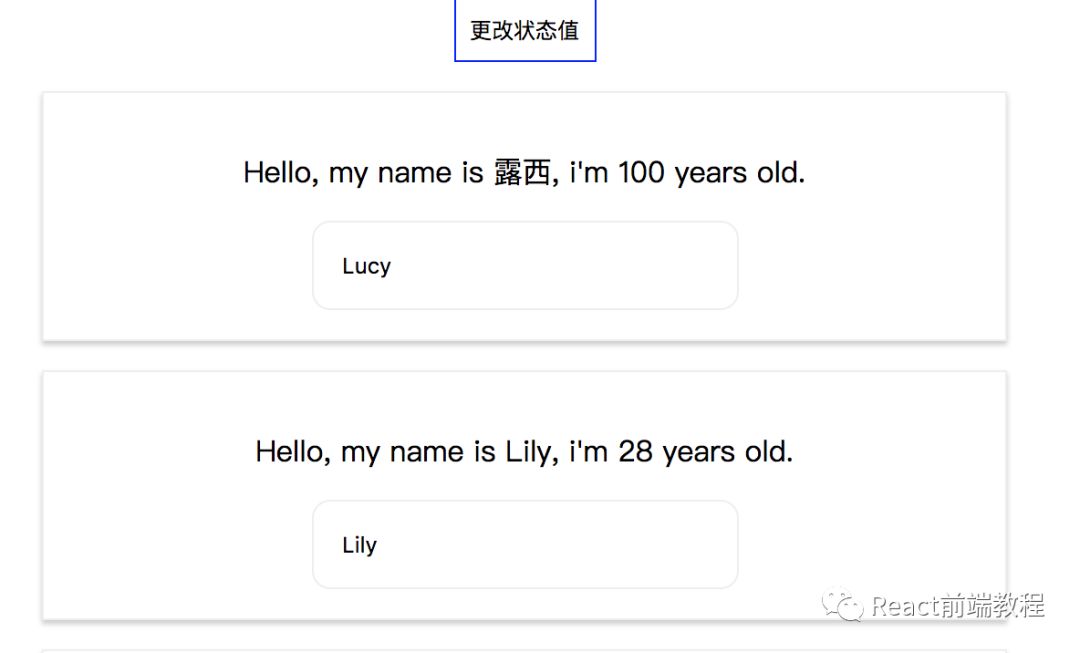
}

React-条件渲染

原创 白清音 [React前端教程](javascript:void(0);) 2018-08-01

导语：在React中，常用的条件语句有if语句、三元操作符、逻辑(&& ||)语句、switch...case等。本节内容我们将介绍在React中条件语句的使用。

查看当前页面的效果图:



我们要实现的功能为：通过点击按钮控制下面内容的显示隐藏。

第一步，定义一个state用来控制Person组件是否显示：

showPersons: false

第二步，给Person组件一个容器，并判断状态进行渲染，在 { } 中定义逻辑代码：

return (

{

    this.state.showPersons ?

        <div>

            <Person

            changed={this.nameCangedHandler}

            name={this.state.persons[0].name}

            age={this.state.persons[0].age}></Person >

            <Person name={this.state.persons[1].name} age={this.state.persons[1].age}></Person >

            <Person name={this.state.persons[2].name} age={this.state.persons[2].age}></Person >

            <Person name={this.state.persons[3].name} age={this.state.persons[3].age}></Person >

        </div> : null

} )

上面这段代码的书写形式我们在实际开发中并不主张，这时return 中的内容过于冗余，可以将以上代码修改为如下形式：

render() {

    let persons = null;

    if (this.state.showPersons) {

        persons = (

            < div >

            <Person

            changed={this.nameCangedHandler}

            name={this.state.persons[0].name}

            age={this.state.persons[0].age} ></Person >

            <Person name={this.state.persons[1].name} age={this.state.persons[1].age}></Person >

            <Person name={this.state.persons[2].name} age={this.state.persons[2].age}></Person >

            <Person name={this.state.persons[3].name} age={this.state.persons[3].age}></Person >

            </div >

        )

    }

    return (

        <div >

        {    persons}

        </div >

    )

}

上面定义了一个persons变量，根据状态修改persons的值。

第三步，定义事件修改状态值：

togglePersonsHandler = () => {

    this.setState({

        showPersons: !this.state.showPersons

    })

}

第四步，绑定事件：

<button  onClick={this.togglePersonsHandler}>更改状态值</button>

React-有状态组件和无状态组件

原创 白清音 [React前端教程](javascript:void(0);) 2018-08-07

导语：众所周知，组件化是React的核心理念之一。在React中组件分为两种类型-有状态组件和无状态组件。本节内容我们就来看一下二者之间的区别。

**区别：**

首先回顾我们当前项目代码，其中定义的App.js是有状态组件，Person.js为无状态组件。

// 有状态组件

class App extends Component {

    render() {

        return (......);

    }

}

export default App;

// 无状态组件

const person = (props) => {

    return (......)

}

export default person

从上面定义我们看到的明显区别是：有状态组件其实就是一个类，而无状态组件是一个函数。

从数据管理和存储来看:有状态组件可以使用State ,而无状态组件不能使用state，而是使用props来接收数据。

从是否有生命周期方法来说，有状态组件可以使用生命周期方法，无状态组件则不可以使用生命周期方法。原因在于我们数据进行更新主要是更新state,然而state发生变化时会触发生命周期的方法。

**使用场景：**

区别完两个组件之后，那么我们应该在什么情况下使用有状态组件，什么情况下使用无状态组件呢？

通常情况下，如果我们需要存储数据并且需要对数据做一些增删改查的操作，这时我们需要定义有状态组件；如果只是通过接收数据进行渲染操作，可以使用无状态组件。

在实际开发中，建议更多的使用无状态组件，因为有状态组件带有生命周期函数，会在不同时刻触发更新。所以更多的使用无状态组件可以提高整体的渲染性能。

最后我们将两个组件区别总结为如下：

React-生命周期钩子函数（一）

原创 白清音 [React前端教程](javascript:void(0);) 2018-08-11

导语：在React中，生命周期钩子函数主要是针对于有状态的组件。我们主要分为三节内容带大家了解这些钩子函数的具体使用。本节内容我们将简单介绍React中的生命周期钩子函数有哪些。

首先来看一下React中钩子函数有哪些：

上面我们看到的钩子函数中，有的只会触发一次，有的会反复触发。我们在开发过程中需要我们重点关注的钩子函数有：

上面四个钩子函数我们是我们在组件开发过程中常用的方法，接下来我们来看一下它们的使用流程以及各自用途：

constructor(): 构造函数，在创建组件时调用一次。这是es6提供的方法。

componentWillMount(): 组件挂载之前调用一次。可以在这里初始化state，这时会在render中看到。

render(): 这是React组件所必不可少的核心函数（上面的其它函数都不是必须的），在这个方法中进行创建虚拟dom并更新dom。注意不能在rander修改state,会引起死循环。并不是一次setState会触发一次render，React可能会合并操作，再一次性进行render。

componentDidMount() ：在这个方法中，如果我们更新state,就会重新render(),render更新完后又会执行componentDidMount(),从而引起死循环，所以在这个方法中不能更新状态。

最后我们来看一下react的生命周期钩子函数的执行流程图：

## React-生命周期钩子函数（二）

原创 白清音 [React前端教程](javascript:void(0);) 2018-08-27

导语：上节内容我们介绍了在react中的生命钩子函数，这节我们具体来看一下如何使用这些生命周期钩子函数。

一、组件构造时触发的生命钩子函数

只要组件存在constructor,就必要要写super,否则this指向会错误

constructor(props,context) {    
    super(props,context)    
    console.log(this.props,this.context) *// 在内部可以使用props和context*}

componentWillMount(){  
    console.log('componentsWillMount is running...')  
}  
render () {    
    console.log('render is running...')    
    return (         
        ...

   )

}  
  
componentDidMount(){  
 console.log('componentDidMount is running...')  
}

接下来我们把App.js中的子组件Person.js也改为有状态组件，这时钩子函数执行顺序如图所示：

可以看到当所有子组件加载完成之后父组件才会执行 componentDidMount()

二、组件状态更新时触发的生命周期钩子函数

当有状态组件的state值发生变化时触发的生命周期钩子函数及执行顺序如图所示：

**componentWillReceiveProps(**nextProps**)**

在接受父组件改变后的props需要重新渲染组件时用到的比较多

### ****shouldComponentUpdate(nextProps,nextState)****

### 唯一用于控制组件重新渲染的生命周期。因为react父组件的重新渲染会导致其所有子组件的重新渲染，这个时候其实我们是不需要所有子组件都跟着重新渲染的，因此需要在子组件的该生命周期中做判断。在这里return false可以阻止组件的更新

### ****componentWillUpdate (nextProps,nextState)****

shouldComponentUpdate返回true以后，组件进入重新渲染的流程，进入componentWillUpdate,这里同样可以拿到nextProps和nextState

### ****componentDidUpdate(prevProps,prevState)****

组件更新完毕后，react只会在第一次初始化成功会进入componentDidmount,之后每次重新渲染后都会进入这个生命周期，这里可以拿到prevProps和prevState，即更新前的props和state。