day03-受控组件-组件通信

1. 受控组件-组件通信

### 表单处理

#### 特别说明

HTML 中的表单元素是可输入的，也就是有自己的可变状态, 但是默认表单元素的值不受所在组件state控制, 也就是表单元素所在组件无法实时获取最新的表单元素值

#### 非受控组件

表单元素值不受所在组件状态的控制, 我们将这样的表单元素称作: 非受控组件.

#### 受控组件

受控组件 : 值受到 React 组件状态控制的表单元素

一般是通过value属性, onChange事件配合将非受控组件变为受控组件.

1. 文本框、富文本框、下拉框 操作value属性

2. 复选框 操作checked属性

<input type="text" value={this.state.txt} onChange={e => this.setState({ txt: e.target.value })} />

#### 多表单元素操作

1. 给表单元素添加name属性，名称与 state 相同

2. 根据表单元素类型获取对应值

3. 在 change 事件处理程序中通过 [name] 来修改对应的state

### 组件通信

#### 父组件向子组件传值

每个组件都可以通过props来接收外界的数据，正好，子组件的外部是暴露在父组件内部的，所以我们可以直接通过props进行数据传递.

**父组件:**

export default class App extends Component{

render(){

return(

<div>

<Sub title = "向子组件通信" />

</div>

)

}

}

**子组件:**

const Sub = (props) => {

return(

<h1>

{ props.title }

</h1>

)

}

#### 子组件向父组件传值

利用回调函数，可以实现子组件向父组件通信：父组件将一个函数作为 props 传递给子组件，子组件调用该回调函数，便可以向父组件通信.

**父组件:**

export default class App extends Component{

callback( msg ){

console.log(msg);

}

render(){

return(

<div>

<Sub callback = { this.callback.bind(this) } />

</div>

)

}

}

**子组件:**

const Sub = (props) => {

return(

<div>

<button onClick = { ()=>props.callback("向父组件通信") }>点击我</button>

</div>

)

}

#### 嵌套组件通信

**React.createContext**

Import React from ‘react’;

const MyContext = React.createContext();

创建一个 Context 对象。当 React 渲染一个订阅了这个 Context 对象的组件，这个组件会从组件树中离自身最近的那个匹配的 Provider 中读取到当前的 context 值.

**MyContext.Provider**

<MyContext.Provider value={/\* 某个值 \*/}>

每个 Context 对象都提供一个 Provider React 组件，它允许内部的消费组件订阅 context ( value 数据 ) 的变化。

Provider 接收一个 value 属性，传递给内部的消费组件。一个 Provider 可以和多个消费组件有对应关系。多个 Provider 也可以嵌套使用，里层的会覆盖外层的数据。

当 Provider 的 value 值发生变化时，它内部的所有消费组件都会重新渲染。Provider 及其内部 consumer 组件都不受制于 shouldComponentUpdate 函数，因此当 consumer 组件在其祖先组件退出更新的情况下也能更新

**Class.contextType**

class MyClass extends React.Component {

componentDidMount() {

let value = this.context;

}

}

MyClass.contextType = MyContext;

#### 非嵌套组件间通信

对于非嵌套组件，可以采用下面两种方式：

1.利用二者共同父组件进行通信

2.使用自定义事件的方式

如果采用组件间共同的父级来进行中转，会增加子组件和父组件之间的耦合度，如果组件层次较深的话，找到二者公共的父组件不是一件容易的事，当然也不是不可以...

这里我们采用自定义事件的方式来实现非嵌套组件间的通信。

我们需要使用一个 events 包

npm install events --save

新建一个 ev.js，引入 events 包，并向外提供一个事件对象，供通信时使用

import { EventEmitter } from "events";

export default new EventEmitter( );

组件一:

export default class Foo extends Component{

constructor(props) {

super(props);

this.state = {

msg:null,

};

}

componentDidMount(){

// 声明一个自定义事件

this.eventEmitter = emitter.addListener("callMe",(msg)=>{

this.setState({

msg

})

});

}

}

组件二:

export default class Boo extends Component{

render(){

const cb = (msg) => {

return () => {

// 触发自定义事件

emitter.emit("callMe","Hello")

}

}

return(

<div>

我是非嵌套 2 号

<button onClick = { ()=>emitter.emit("callMe","Hello") }>点击我</button>

</div>

);

}

}

自定义事件是典型的发布/订阅模式，通过向事件对象上添加监听器和触发事件来实现组件间通信。

**总结 React 中组件的几种通信方式，分别是：**

父组件向子组件通信：使用 props

子组件向父组件通信：使用 props 回调

跨级组件间通信：使用 context 对象

非嵌套组件间通信：使用事件订阅

1. 综合案例

<cdf-会员购> 分类页-顶部搜索功能

<cdf-会员购> 分类页-一级/二级分类切换