组件

1，了解什么是组件

2，如何创建组件，使用组件

3，组件的结构

4，组件的生命周期

5，组件的嵌套

6，组件的传参

7，在单个组件内使用单向数据流

8，在嵌套组件中使用单向数据流

10，完成一个简单的demo

了解什么是组件

React 允许将代码封装成组件（component），然后像插入普通 HTML 标签一样，在网页中插入这个组件，简单的说，就是具有独立逻辑功能的封装的代码块，

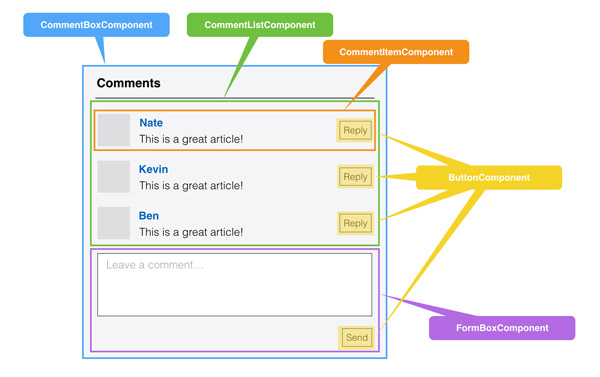
在react中，我们要遵守组件化，模块化的整体架构思路，尽量用[单一责任原则](http://en.wikipedia.org/wiki/Single_responsibility_principle" \t "http://www.cocoachina.com/webapp/20150810/_blank)来设计你的组件，一个组件尽量只完成一个独立的功能逻辑。当你发下你的组件应该做的更多的时候，你可以考虑将其分割为 更小的组件集合

如我们上次介绍过的，他具有如下特性

****（1）可组合（Composeable）****：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有（own）它创建的子组件，通过这个特性，一个复杂的UI可以拆分成多个简单的UI组件；

****（2）可重用（Reusable）****：每个组件都是具有独立功能的，它可以被使用在多个UI场景；

****（3）可维护（Maintainable）****：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护；



如何创建组件，使用组件

var MyComponent = React.createClass({

render: function(){

return (

<h1>Hello, world!</h1>

);

}

});

在es5中 我们用React.createClass 关键字来创建组件

如果是es6 则

class MyComponent extends React.Component {

render(){

return (<div>Hello, world!</div>)

}

}

所有组件类都必须有自己的 render 方法，用于输出组件

注意，组件类的第一个字母必须大写，否则会报错，比如HelloMessage不能写成helloMessage。另外，组件类只能包含一个顶层标签，否则也会报错。

定义好组件后

ReactDOM.render(

<HelloMessage />,

document.getElementById('example'));

);

即可使用

我们下面的代码都将使用es6的写法

组件的结构

一个完整的组件包含了，构造函数，状态，生命周期回调，参数限制，逻辑处理，jsx输出

组件的状态即是存储的组件数据

分为两个部分

一个是组件本身的内部状态

存放在 state中，state则应该被包含在一个构造函数中，state是可变的

class MyComponent extends React.Component {

//构造函数

constructor(props, context) {

super(props, context);

this.state ={

name:"小明"

}

}

render(){

return (<div>{this.state.name}</div>)

}

}

我们通过this.state.xx的语法去获取状态

我们可以使用this.setState({name:“小红”}) 方法来改变状态并刷新ui

另一部分是props 外部状态，这部分状态由外部传入的，不可更改，只能从数据源修改

我们可以通过 this.props.xx 来调用

class MyComponent extends React.Component {

//构造函数

constructor(props, context) {

super(props, context);

this.state ={

name:"小明"

}

}

render(){

return (<div>{this.state.name},{this.props.name},{this.props.children}</div>)

}

}

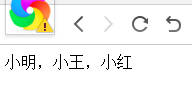
ReactDOM.render(

<HelloMessage name=”小明”><span>小红</span></HelloMessage >,

document.getElementById('example'));

);

输出



这里我们用到了两种方式赋值

一种是 name=”xx” 如同传统的html标签属性一样

另外一种是 在标签之间写入html 这种方式赋值 所有的内容将会赋给 this.props.children

生命周期的回调函数和自定义的逻辑处理的函数都是以类的属性的方式存在

class Template extends Component {

constructor(props, context) {

super(props, context);

this.state ={

}

}

//自定义逻辑处理函数

btnClick(){

Alert(1);

}

//jsx输出

render () {

return (

<div>hi~Template</div>

);

}

//生命周期回调函数

componentWillMount(){}

}

/\*

//属性验证 ， 如果我们设置了属性限制，那么这个属性便只能传递某个类型的数据

Template.propTypes={

//属性名 ： 限制类型

stringProp: React.PropTypes.string.isRequired, //必须是字符串

boolProp:React.PropTypes.bool.isRequired,//必须是布尔

arrayProp:React.PropTypes.array.isRequired,//必须是数组

funcProp:React.PropTypes.func.isRequired,//必须是函数

numberProp:React.PropTypes.number.isRequired,//必须是数字

objectProp:React.PropTypes.object.isRequired,//必须是对象

}

组件的生命周期

组件被创建 引用，再到销毁删除 这个过程就是组件的生命周期

React把这个过程分为3个时期

Mounting：已插入真实 DOM

Updating：正在被重新渲染

Unmounting：已移出真实 DOM

React 为每个状态都提供了两种处理函数，will 函数在进入状态之前调用，did 函数在进入状态之后调用，三种状态共计五种处理函数

componentWillMount(){}//准备：插入真实 DOM

componentDidMount(){}//完成：插入真实 DOM

componentWillUpdate(nextProps, nextState){}//准备：正在被重新渲染

componentDidUpdate(prevProps,prevState){}//完成：正在被重新渲染

componentWillUnmount(){}//已移出真实 DOM

此外还有2个特殊的回调函数

componentWillReceiveProps(object nextProps)：已加载组件收到新的参数时调用

shouldComponentUpdate(object nextProps, object nextState)：组件判断是否重新渲染时调用

他们将会在一个组件不同的状态下被自动调用

组件的嵌套

组件之间的嵌套，如同html标签的嵌套

如

class MyTest extends React.Component{

render(){

return (<div>测试{this.props.name}</div>)

}

}

class MyComponent extends React.Component {

render(){

return (<div><MyTest name=”aaa”/></div>)

}

}

结果输出了MyTest 的内容，

这就是组件的嵌套，组件嵌套没有任何限制，就如同html标签一般,通过定义组件属性传递参数，参数通过props获取

在单个组件内使用单向数据流

在react中，自上而下的数据流动被成为单向数据流

数据总是从组件的state中被定义，然后显示在render中

State改变后 通过setState() 刷新视图 从而完成了一次数据更新

我们来看一个例子，点击按钮使数字+1

class MyComponent extends React.Component {

constructor(props, context) {

super(props, context);

this.state ={

size:1

}

}

sizeClick(){

this.setState({size:this.state.size + 1});

}

render(){

return (<div>

{this.state.size}

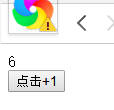
<br />

<input type="button" value="点击+1" onClick={this.sizeClick.bind(this)} />

</div>)

}

}



我们定义了一个数字 一个按钮 按钮绑定了一个点击事件

点击一次则+1 我们看到 我们没有直接操作dom 而是直接操作了状态的值 状态值改变后 再刷新了ui 这就是react的数据驱动开发

在嵌套组件中使用单向数据流

那么我们如何在多层嵌套中使用数据流呢，我们将使用到props传参和回调

而props是无法在组件内改变的，只能改变数据源， 那我们应该如何改变呢，请看下面的例子

我们将定义2个组件 A 和 B B嵌套在A组件内，而B组件通过props传递了name给A显示， 现在我们要在A组件内改变这个name，应该怎么办呢？

class B extends React.Component{

函数执行的时候 我们调用a组件传递过来的函数

bClick(){

this.props.cbk();

}

render(){

return (<div>我是{this.props.name}<br />

我们点击的时候 调用b组件定义好的 处理函数

<input type="button" value="改变父组件属性" onClick={this.bClick.bind(this)} /></div>)

}

}

class A extends React.Component {

constructor(props, context) {

super(props, context);

this.state ={

name:"小明"

}

}

我们在a组件内定义了一个处理函数

aClick(){

当函数被调用时 我们改变了state状态值 并刷新ui

this.setState({name:"小红"});

}

render(){

return (<div>

我们把函数当成参数传递给b组件

<B cbk={this.aClick.bind(this)} name={this.state.name} />

</div>)

}

}

通过上面的例子我们可以看出来

嵌套组件的数据改变，我们要通过改变数据源而改变

这里我们用到了回调的方式来改变父组件的状态值

9，完成一个简单的demo，来实践我们学到的知识