# 一 组件简介与JSX

## 1 组件定义方式

import React, {Component} from 'react';  
import ReactDOM from 'react-dom';  
  
require('./common/style/main.css');  
  
class Nav extends React.Component{  
 render(){  
 return (  
 <ul>  
 <li>nav1</li>  
 <li>nav2</li>  
 <li>nav3</li>  
 </ul>  
 )  
 }  
}  
  
ReactDOM.render(  
 <Nav/>,  
 document.getElementById('root')  
);  
  
if(module.hot){   
 module.hot.accept()  
}

注意事项

（1）你的React组件名称的首字母应当大写，关于大小写的差异你会在后面发现。

（2）你应该会注意到div元素的样式类是用 className而不是class声明的，这是因为class 是JavaScript的保留字，渲染后，真实的DOM还会是：

<div class="ez-led">Hello, React!</div>

1. 如果要引入图片 <img src={require(’./common/img/1.jpg’)}/>，后续的案例中出现了类似src={require(’/common/img/1.jpg’)} 类似绝对路径的设置，是因为webpack增加了如下设置：

resolve: {  
 modules: [  
 'node\_modules',  
 path.resolve(\_\_dirname, 'src/common'),  
 path.resolve(\_\_dirname, 'src/components')  
 ]  
},

在查找资源时，先从node\_modules中查找，如果找不到去\_\_dirname 下的src/coomon中查找，依次类推。之前的案例中webpack没有这样的配置，所以使用了相对路径。

1. import React, {Component} from 'react'; 这句话导出了两个对象，那么在使用时候，不需要像上述案例那样书写，直接可以写：

class Nav extends Component{}

## 2 jsx语法详解

### 2.1 JSX语法案例

|  |
| --- |
| **import** React **from 'react'**; **import** ReactDOM **from 'react-dom'**;  require(**'./common/style/main.css'**);  **let** w1 = **'Hello'**; **let** sp = <**span**>World</**span**>  ReactDOM.render(  <**div**>你好，{w1} <**span className="Welcome" data-myprop="999"**>{sp}</**span**>{1+1}</**div**>,  **document**.getElementById(**'root'**) );  注意：render里有多个标签，必须有一个闭合标签将其包裹住 |

如果在 JSX 中使用的属性不存在于 HTML 的规范中，这个属性会被忽略。如果要使用自定义属性，可以用 data- 前缀。

所有的属性都是驼峰命名的，class 属性和 for 属性分别改为 className 和 htmlFor，来符合 DOM API 规范。

### 2.2 属性扩散与覆盖

有时候你需要给组件设置多个属性，你不想一个个写下这些属性，或者有时候你甚至不知道这些属性的名称，这时候 spread attributes 的功能就很有用了。

|  |
| --- |
| **import** React,{Component} **from 'react' import** ReactDOM **from 'react-dom'  import** Hello **from '../components/Hello.js'  const** obj = {  **age**:33,  **prop1**:**'ddd'** } ReactDOM.**render**(  <**div**>  <**Hello name="lisi"** {...obj}/>  </**div**>,  **document**.getElementById(**'app'**) ); |

属性也可以被覆盖，写在后面的属性值会覆盖前面的属性。

var props = { foo: 'default' };

var component = <Component {...props} foo={'override'} />;

console.log(component.props.foo); // 'override'

### 2.3 注释

在 JSX 里使用注释也很简单，就是沿用 JavaScript，唯一要注意的是在一个组件的子元素位置使用注释要用 {} 包起来

var content = (

<Nav>

{/\* child comment, put {} around \*/}

<Person

name={window.isLoggedIn ? window.name : ''} // end of line comment

/>

</Nav>

);

## 3 视图相关概念

* props（属性，就是element上的attrs，换个名字property，变成复数，即props）从父组件传递进来的props属性，都可以被监听到！
* state（写过view组件的基本都会知道，按钮有三态，Normal，Highlight，Selected，包括extjs，jquery里的大部分ui框架都是有状态的。）
* event（其实还应该算一个dom事件，下面的例子就把onChange的handler编译后的handleChange方法）

了解了上面这些，就可以写代码了，因为

* 属性，解决了view的定义问题，即语义描述
* 状态，是view的有穷状态机，根据状态决定组件ui和行为
* 事件，是view里元素的行为

### 3.1 props传递数据

从入口函数处，或者父组件处传递属性值：

|  |
| --- |
| **import** React,{**Component**} **from 'react' import** ReactDOM **from 'react-dom' import** Hello **from '../components/Hello.js'** ReactDOM.**render**(  <**div**>  <**Hello name="lisi"**/>  </**div**>, **document**.getElementById(**'app'**) ); |

然后在组件中就被传递到了该name：

|  |
| --- |
| **import** React, {**Component**} **from 'react' export default class** Hello **extends Component** {  render() {  **console**.log(**this**.**props**);  **return** (  <**div**>  <**h1**>我是Hello组件</**h1**>  <**img**/>  </**div**>  )  } } |

### 3.2 props案例

父组件：Father.js

import React,{Component} from 'react';  
import Son from './Son';  
  
let arr = [  
 {  
 name:'ls',  
 imgSrc:'http://img.taopic.com/uploads/allimg/120727/201995-120HG1030762.jpg'  
 },  
 {  
 name:'zs',  
 imgSrc:'http://img.taopic.com/uploads/allimg/120727/201995-120HG1030762.jpg'  
 },  
 {  
 name:'ww',  
 imgSrc:'http://img.taopic.com/uploads/allimg/120727/201995-120HG1030762.jpg'  
 }  
];  
  
export default class Father extends Component{  
 render(){  
  
 let sonArray = arr.map((elt,i)=>{  
 return (  
 <Son  
 key = {i} *//react要求每个都需要一个key，否则会有警告！* name = {elt.name}  
 imgSrc = {elt.imgSrc}  
 />  
 )  
 })  
  
 return(  
 <div>  
 {sonArray}  
 </div>  
 )  
  
 }  
}

子组件：Son.js

import React,{Component} from 'react';  
  
export default class Son extends Component{  
 render(){  
 let {name,imgSrc} = this.props;  
 return(  
 <div>  
 <div>{name}</div>  
 <img src={imgSrc}></img>  
 </div>  
 )  
 }  
}

### 3.3 props属性验证

有时候，后台请求过来的属性并不一定符合前台要求，需要对属性值类型进行验证。我们需要使用propTypes验证：

import React,{Component,PropTypes} from 'react';

新版的react移除了这个设置，需要使用npm安装第三方包 propTypes来验证：

### 3.4 context传递数据

props传递数据只能一个组件一个组件传递，context可以实现跨组件传递。注意：使用context必须进行属性验证。

class App extends Component{  
 *//给所有子类提供一个属性* getChildContext(){  
 return {  
 msg:'heihei'  
 }  
 }  
 render(){  
 return (  
 <div>  
 <Father/>  
 </div>  
 )  
 }  
}  
  
App.childContextTypes = {  
 msg:PropTypes.string  
}

我们可以跨过父组件Father，直接将数据传递给子组件Son

import React,{Component} from 'react';  
import PropTypes from 'prop-types';  
*//isRequired表示必须有这个属性值*let propTypes = {  
 imgSrc:PropTypes.string,  
 name:PropTypes.string.isRequired  
}  
let contextTypes = {  
 msg:PropTypes.string  
}  
export default class Son extends Component{  
 render(){  
 let {name,imgSrc} = this.props;  
 console.log(this.context)  
 return(  
 <div>  
 <div>{name}</div>  
 <img src={imgSrc}></img>  
 </div>  
 )  
 }  
}  
Son.propTypes = propTypes;  
Son.contextTypes = contextTypes;

### 3.3 state入门

在构造函数中定义，在组件中使用，类似Angular中的$scope

|  |
| --- |
| **import** React, {**Component**} **from 'react' export default class** Hello **extends Component** {   constructor(props,context){  **super**(props,context);  **this**.**state** = {  **name**:**"lisi"**,  **isShowLoading** :**true**,  **movieDetailData** : []  }  }   render() {  **console**.log(**this**.**props**);  **return** (  <**div**>  <**h1**>我是Hello组件,{**this**.**state**.**name**}</**h1**>  <**img**/>  </**div**>  )  } } |

如果要给state里的属性赋值，需要使用this.setState

### 3.4 this指向

在react绑定this指向推荐的做法是在构造函数中绑定：

class App extends Component{

constructor(props){

super(props);

this.changeView = this.changeView.bind(this);

}

changeView(view){

}

}

## 4 组件的生命周期

组件的生命周期，另外的名字是状态回调，和上面讲的状态的唯一差别，上面的状态是它里面的元素，而组件的生命周期是它自己。

### 组件的生命周期分成三个状态：

* Mounting：已插入真实 DOM
* Updating：正在被重新渲染
* Unmounting：已移出真实 DOM

### 处理函数

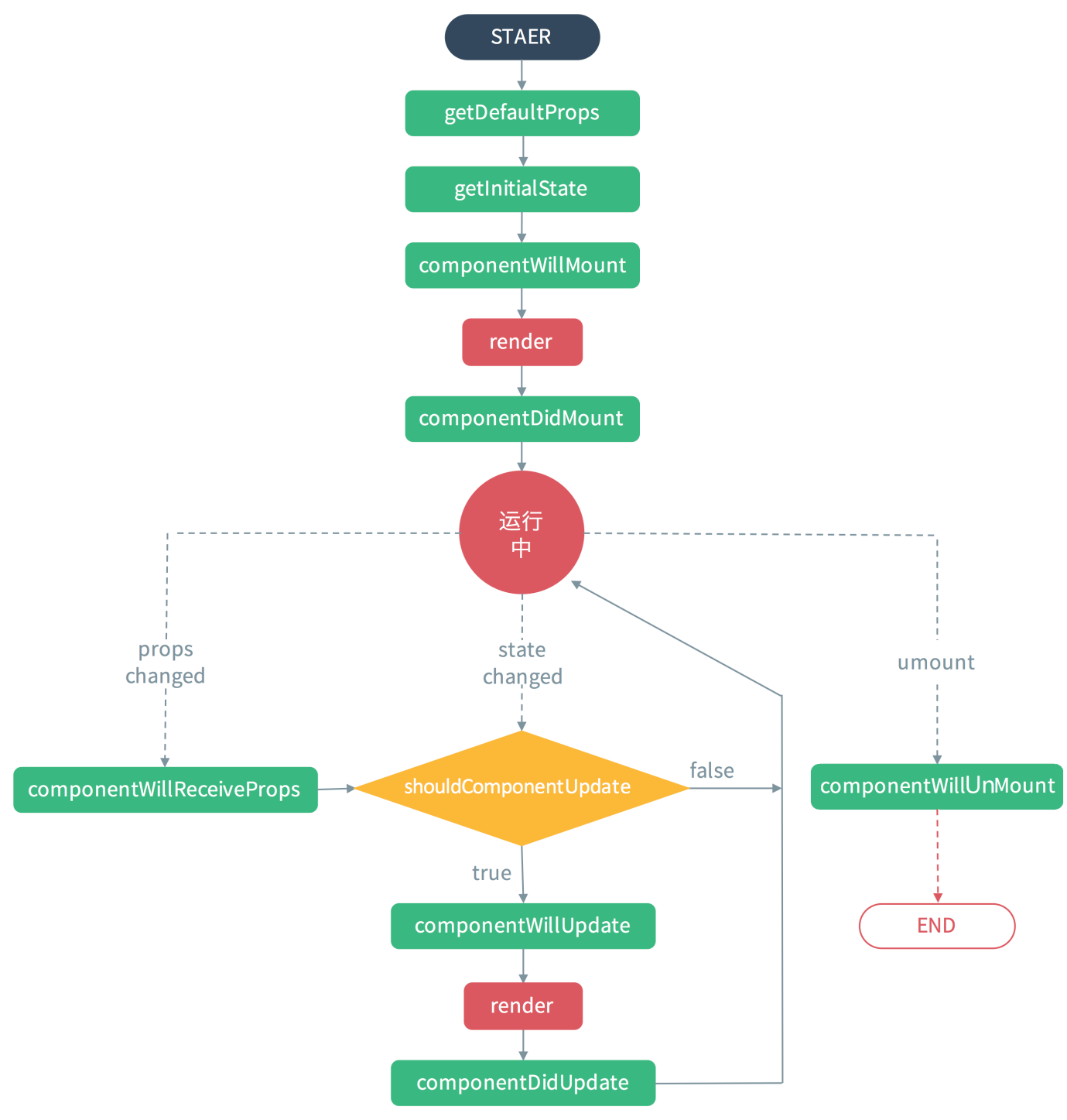
React 为每个状态都提供了两种处理函数，will函数在进入状态之前调用，did 函数在进入状态之后调用，三种状态共计五种处理函数。

* componentWillMount()
* componentDidMount()
* componentWillUpdate(object nextProps, object nextState)
* componentDidUpdate(object prevProps, object prevState)
* componentWillUnmount()

此外，React 还提供两种特殊状态的处理函数。

* 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

### 函数调用顺序图



从上图中我们可以看出来，组件在初始化一次之后就不会再运行上图运行中文字以上的方法，反而里面会有事件监听，从而执行shouleComponentUpdate事件。

## 事件处理

### 使用

onClick这种进行驼峰命名ES5和ES6的写法不一样，在ES6中要用bind方法绑定this(具体可参照ES5和ES6写法对照表)

import React, { Component } from 'react';  
import { render } from 'react-dom';  
  
export default class LinkButton extends Component {  
 constructor(props) {  
 super(props);  
 this.**state** = {**liked**: false};  
 }  
  
 handleClick(e) {  
 this.setState({ **liked**: !this.**state**.**liked** });  
 }  
  
 render() {  
 const text = this.**state**.**liked** ? 'like' : 'haven\'t liked';  
 return (  
 <p onClick={this.handleClick.bind(this)}>  
 You {text} this. Click to toggle.  
 </p>  
 );  
 }  
}

### 参数传递

ES6写法：给事件处理函数传递额外参数的方式：bind(this, arg1, arg2, ...)

render: function() {

return <p onClick={this.handleClick.bind(this, param1,param2,param3)}>;

},

handleClick: function(param1,param2,param3, event) {

// handle click

}

## Dom操作

### 方法一：findDOMNode()方法（了解）

首先我们要了解 ReactDOM.render 组件返回的是对组件的引用也就是组件实例（对于无状态状态组件来说返回 null），注意 JSX 返回的不是组件实例，它只是一个 ReactElement 对象。

当组件加载到页面上之后（mounted），你都可以通过 react-dom 提供的 findDOMNode() 方法拿到组件对应的 DOM 元素。

import { findDOMNode } from 'react-dom';

// Inside Component class

componentDidMound() {

constel = findDOMNode(this);

}

findDOMNode() 不能用在无状态组件上。

此种获取方式过于老旧，不用学习。

### 方法二：refs属性

另外一种方式就是通过在要引用的 DOM 元素上面设置一个 ref 属性指定一个名称，然后通过 this.refs.name 来访问对应的 DOM 元素。

如果 ref 是设置在原生 HTML 元素上，它拿到的就是 DOM 元素，如果设置在自定义组件上，它拿到的就是组件实例，这时候就需要通过 findDOMNode 来拿到组件的 DOM 元素。

因为无状态组件没有实例，所以 ref 不能设置在无状态组件上，一般来说这没什么问题，因为无状态组件没有实例方法，不需要 ref 去拿实例调用相关的方法，但是如果想要拿无状态组件的 DOM 元素的时候，就需要用一个状态组件封装一层，然后通过 ref 和 findDOMNode 去获取。

import React, { Component } from 'react';  
  
export default class MyInputFocus extends Component {  
 constructor(props) {  
 super(props);  
 this.**state**={ **userInput**: '' };  
 }  
  
 handleChange(e) {  
 **console**.log(this.**refs**.theInput.**value**);  
 this.setState({ **userInput**: e.**target**.**value** });  
 }  
  
 clearAndFocusInput() {  
 this.setState({ **userInput**: '' }, () => {  
 this.**refs**.theInput.focus();  
 });  
 }  
  
 render() {  
 return (  
 <div>  
 <div onClick={this.clearAndFocusInput.bind(this)}>  
 Click to Focus and Reset  
 </div>  
 <input  
 ref="theInput"  
 value={this.**state**.**userInput**}  
 onChange={this.handleChange.bind(this)}  
 />  
 </div>  
 );  
 }  
}  
  
MyInputFocus.**defaultProps**={  
 **autoPlay**:false,  
 **maxLoops**:10,  
}  
MyInputFocus.**propTypes** = {  
 **autoPlay**: React.**PropTypes**.**bool**.**isRequired**,  
 **maxLoops**: React.**PropTypes**.**number**.**isRequired**,  
}

### 注意事项

* 你可以使用 ref 到的组件定义的任何公共方法，比如 this.refs.myTypeahead.reset()
* Refs 是访问到组件内部 DOM 节点唯一**可靠**的方法
* Refs 会自动销毁对子组件的引用（当子组件删除时）
* 不要在 render 或者 render 之前访问 refs
* 不要滥用 refs，比如只是用它来按照传统的方式操作界面 UI：找到 DOM -> 更新 DOM

## 和其他库配合使用

<http://reactjs.cn/react/tips/use-react-with-other-libraries.html>

## 组件的 DOM 事件监听

这篇文章是讲如何给 DOM 元素绑定 React 未提供的事件

var Box = React.createClass({

getInitialState: function() {

return {windowWidth: window.innerWidth};

},

handleResize: function(e) {

this.setState({windowWidth: window.innerWidth});

},

componentDidMount: function() {

window.addEventListener('resize', this.handleResize);

},

componentWillUnmount: function() {

window.removeEventListener('resize', this.handleResize);

},

render: function() {

return <div>Current window width: {this.state.windowWidth}</div>;

}

});

React.render(<Box />, mountNode);

<http://reactjs.cn/react/tips/dom-event-listeners.html>

1、注意添加dom事件的位置

2、在组件退出的时候，取消监听事件

## 数据获取

http://facebook.github.io/react/tips/initial-ajax.html

## 表单

表单不同于其他 HTML 元素，因为它要响应用户的交互，显示不同的状态，所以在 React 里面会有点特殊。

### 状态属性

表单元素有这么几种属于状态的属性：

* value，对应 <input> 和 <textarea> 所有
* checked，对应类型为 checkbox 和 radio 的 <input> 所有
* selected，对应 <option> 所有

*在 HTML 中 <textarea> 的值可以由子节点（文本）赋值，但是在 React 中，要用 value 来设置。*

表单元素包含以上任意一种状态属性都支持 onChange 事件监听状态值的更改。

针对这些状态属性不同的处理策略，表单元素在 React 里面有两种表现形式。

### 受控组件

对于设置了上面提到的对应“状态属性“值的表单元素就是受控表单组件，比如：

render: function() {

return <input type="text" value="hello"/>;

}

一个受控的表单组件，它所有状态属性更改涉及 UI 的变更都由 React 来控制（状态属性绑定 UI）。比如上面代码里的 <input> 输入框，用户输入内容，用户输入的内容不会显示（输入框总是显示状态属性 value 的值 hello），这有点颠覆我们的认知了，所以说这是**受控**组件，不是原来默认的表单元素了。

如果你希望输入的内容反馈到输入框，就要用 onChange 事件改变状态属性 value 的值：

getInitialState: function() {

return {value: 'hello'};

},

handleChange: function(event) {

this.setState({value: event.target.value});

},

render: function() {

var value = this.state.value;

return <input type="text" value={value} onChange={this.handleChange} />;

}

使用这种模式非常容易实现类似对用户输入的验证，或者对用户交互做额外的处理，比如截断最多输入140个字符：

handleChange: function(event) {

this.setState({value: event.target.value.substr(0, 140)});

}

### 非受控属性

和受控组件相对，如果表单元素没有设置自己的“状态属性”，或者属性值设置为 null，这时候就是非受控组件。

它的表现就符合普通的表单元素，正常响应用户的操作。

同样，你也可以绑定 onChange 事件处理交互。

如果你想要给“状态属性”设置默认值，就要用 React 提供的特殊属性 defaultValue，对于 checked 会有 defaultChecked，<option> 也是使用 defaultValue。

### 为什么要有受控组件

引入受控组件不是说它有什么好处，而是因为 React 的 UI 渲染机制，对于表单元素不得不引入这一特殊的处理方式。

在浏览器 DOM 里面是有区分 attribute 和 property 的。attribute 是在 HTML 里指定的属性，而每个 HTML 元素在 JS 对应是一个 DOM 节点对象，这个对象拥有的属性就是 property（可以在 console 里展开一个 DOM 节点对象看一下，HTML attributes 只是对应其中的一部分属性），attribute 对应的 property 会从 attribute 拿到初始值，有些会有相同的名称，但是有些名称会不一样，比如 attribute class 对应的 property 就是 className。（详细解释：[.prop](http://api.jquery.com/prop/" \t "_blank)，[.prop() vs .attr()](http://stackoverflow.com/questions/5874652/prop-vs-attr" \t "_blank)）

回到 React 里的 <input> 输入框，当用户输入内容的时候，输入框的 value property 会改变，但是 value attribute 依然会是 HTML 上指定的值（attribute 要用 setAttribute 去更改）。

React 组件必须呈现这个组件的状态视图，这个视图 HTML 是由 render 生成，所以对于

render: function() {

return <input type="text" value="hello"/>;

}

在任意时刻，这个视图总是返回一个显示 hello 的输入框。

### <select>的处理

在 HTML 中 <select> 标签指定选中项都是通过对应 <option> 的 selected 属性来做的，但是在 React 修改成统一使用 value。

**所以没有一个 selected 的状态属性。**

<select value="B">

<option value="A">Apple</option>

<option value="B">Banana</option>

<option value="C">Cranberry</option>

</select>

你可以通过传递一个数组指定多个选中项：<select multiple={true} value={['B', 'C']}>

## 参数传递的判断

http://facebook.github.io/react/docs/transferring-props.html

## 组合组件

使用组件的目的就是通过构建模块化的组件，相互组合组件最后组装成一个复杂的应用。

在 React 组件中要包含其他组件作为子组件，只需要把组件当作一个 DOM 元素引入就可以了。

<http://reactjs.cn/react/docs/multiple-components.html>

### 循环插入子元素

如果组件中包含通过循环插入的子元素，为了保证重新渲染 UI 的时候能够正确显示这些子元素，每个元素都需要通过一个特殊的 key 属性指定一个唯一值。为了内部 diff 的效率。

var TodoList = React.**createClass**({  
 render: function() {  
 var createItem = function(item) {  
 return <li key={item.**id**}>{item.**text**}</li>;  
 };  
 return <ul>{this.**props**.**items**.map(createItem)}</ul>;  
 }  
});  
module.**export**=TodoList

（1）当 React 校正带有 key 的子级时，它会确保它们被重新排序（而不是破坏）或者删除（而不是重用）。 务必 把 key 添加到子级数组里组件本身上，而不是每个子级内部最外层 HTML 上。

（2）也可以传递 object 来做有 key 的子级。object 的 key 会被当作每个组件的 key。但是一定要牢记 JavaScript 并不总是保证属性的顺序会被保留。实际情况下浏览器一般会保留属性的顺序，**除了** 使用 32位无符号数字做为 key 的属性。数字型属性会按大小排序并且排在其它属性前面。一旦发生这种情况，React 渲染组件的顺序就是混乱。可能在 key 前面加一个字符串前缀来避免：

render: function() {

var items = {};

this.props.results.forEach(function(result) {

// 如果 result.id 看起来是一个数字（比如短哈希），那么

// 对象字面量的顺序就得不到保证。这种情况下，需要添加前缀

// 来确保 key 是字符串。

items['result-' + result.id] = <li>{result.text}</li>;

});

return (

<ol>

{items}

</ol>

);

}

### 子级

组件标签里面包含的子元素会通过父元素的props.children 传递进来。

HTML 元素会作为 React 组件对象、JS 表达式结果是一个文字节点，都会存入 Parent 组件的 props.children。

props.children 通常是一个组件对象的数组，但是当只有一个子元素的时候，props.children 将是这个唯一的子元素，而不是数组了

var NotesList = React.createClass({

render: function() {

return (

<ol>

{

React.Children.map(this.props.children, function (child) {

return <li>{child}</li>;

})

}

</ol>

);

}

});

ReactDOM.render(

<NotesList>

<span>hello</span>

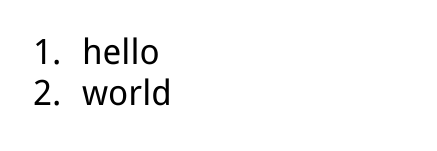
<span>world</span>

</NotesList>,

document.body

);

上面代码的 NoteList 组件有两个 span 子节点，它们都可以通过 this.props.children 读取，运行结果如下。



这里需要注意， this.props.children 的值有三种可能：如果当前组件没有子节点，它就是 undefined ;如果有一个子节点，数据类型是 object ；如果有多个子节点，数据类型就是 array 。所以，处理 this.props.children 的时候要小心。

React 提供一个工具方法 [React.Children](https://facebook.github.io/react/docs/top-level-api.html" \l "react.children" \t "_blank) 来处理 this.props.children 。我们可以用 React.Children.map 来遍历子节点，而不用担心 this.props.children 的数据类型是 undefined 还是 object。更多的 React.Children 的方法，请参考[官方文档](https://facebook.github.io/react/docs/top-level-api.html" \l "react.children" \t "_blank)。

## propsType

<http://www.reactjs.cn/react/docs/reusable-components.html>

## Context

http://facebook.github.io/react/docs/context.html

## 动画

<http://facebook.github.io/react/docs/animation.html>

<http://blog.csdn.net/lihongxun945/article/details/46778723>

https://zhuanlan.zhihu.com/p/20419592

## 获取react常用插件的网址

<https://js.coach/react/react-infinite>

<https://react.parts/native>

## diff算法

http://blog.csdn.net/lihongxun945/article/details/46640503

http://reactjs.cn/react/docs/reconciliation.html

<http://blog.csdn.net/yczz/article/details/49585283>

<http://blog.csdn.net/yczz/article/details/49886061>

## Web Components

http://www.oschina.net/p/polymer

http://facebook.github.io/react/docs/webcomponents.html

## 服务器渲染

<http://zhuanlan.zhihu.com/p/20669111?from=groupmessage&isappinstalled=0>

## 组件间通信

### 非父子组件间的通信

使用全局事件 Pub/Sub 模式，在 componentDidMount 里面订阅事件，在 componentWillUnmount 里面取消订阅，当收到事件触发的时候调用 setState 更新 UI。

这种模式在复杂的系统里面可能会变得难以维护，所以看个人权衡是否将组件封装到大的组件，甚至整个页面或者应用就封装到一个组件。

一般来说，对于比较复杂的应用，推荐使用类似 Flux 这种单项数据流架构，参见[Data Flow](https://hulufei.gitbooks.io/react-tutorial/content/data-flow.html)。Flux和redux

### 数据流Flux

Github地址：<https://github.com/facebook/flux>

React redux react-redux react-router webpack+gulp ES6 babel

Mocha+chai node

React native Flex fetch 原生 找插件

# 四 react-router

## 学习网址

github地址：<https://github.com/reactjs/react-router>

最好是去github上看文档，因为随着时间的推移，版本的变更，市面上的一些文档已经没有及时性和准确性了。Github上已经有好几个版本的了，我们学习最新版本的。以后如果有问题先检查版本是不是升级导致的问题。

gitbook地址：<http://react-guide.github.io/react-router-cn/>

https://segmentfault.com/a/1190000006063554

## react-router简介

React Router 是一个基于 React之上的强大路由库，它可以让你向应用中快速地添加视图和数据流，同时保持页面与 URL 间的同步。

## 之前页面跳转的实现（了解）

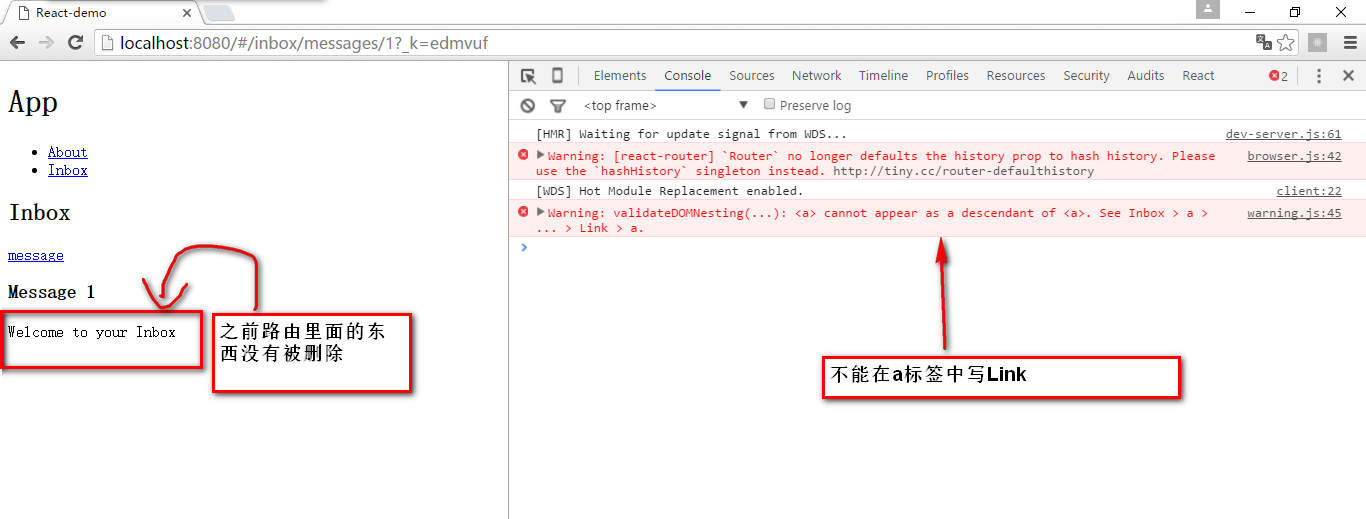
<http://react-guide.github.io/react-router-cn/docs/Introduction.html>

## 基础配置

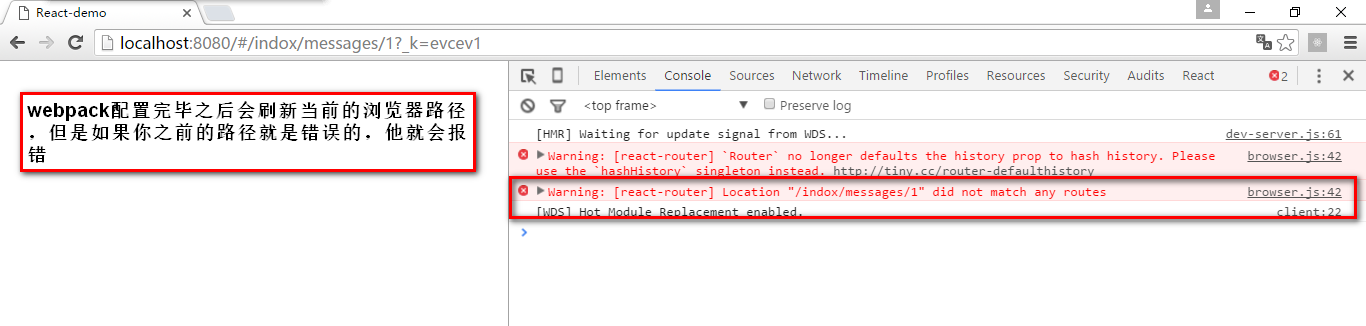
1、他会把route的第一个组件加载到根组件的this.chileren里面

2、下面这种写法会报错

<a><Link to="/indox/messages/1">message</Link></a>



3、刷新的错误



# 五 React-redux