## 用create-react-app快速构建react开发环境

$ cnpm install -g create-react-app

$ create-react-app my-app

$ cd my-app/

$ npm start

Example：

<body>

<div id="example"></div>

<script type="text/babel">

//函数Clock接收参数

function Clock(props) {

return (

<div>

<h1>Hello, world!</h1>

<h2>现在是 {props.date.toLocaleTimeString()}.</h2>

</div>

);

}

function tick() {

ReactDOM.render(

//此处给Clock传参数

<Clock date={new Date()} />,

document.getElementById('example')

);

}

setInterval(tick, 1000);

</script>

</body>

ReactDOM.render()此方法用来将元素渲染到页面上

除了函数外我们还可以创建一个 React.Component 的 ES6 类，该类封装了要展示的元素，需要注意的是在 render() 方法中，需要使用this.props替换props

<body>

<div id="example"></div>

<script type="text/babel">

class Clock extends React.Component {

render() {

return (

<div>

<h1>Hello, world!</h1>

<h2>现在是 {this.props.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

function tick() {

ReactDOM.render(

<Clock date={new Date()} />,

document.getElementById('example')

);

}

setInterval(tick, 1000);

</script>

</body>

## 2.Jsx

const element = <h1>Hello, world!</h1>

这是一个jsx语法

在jsx中可以使用javascript表达式 表达式写在花括号中{}

在JSX 中不能使用 if else 语句，但可以使用 conditional (三元运算) 表达式来替代以下实例中如果变量 i 等于 1 浏览器将输出 true, 如果修改 i 的值，则会输出 false.

var i = 1;

ReactDOM.render(

<div>

<h1>{i == 1 ? 'True!' : 'False'}</h1>

</div>

,

document.getElementById('example')

);

#### 内联样式

React 推荐使用内联样式。我们可以使用 camelCase 语法来设置内联样式. React 会在指定元素数字后自动添加 px 。以下实例演示了为 h1 元素添加 myStyle 内联样式：

<script type="text/babel">

var myStyle = {

fontSize: 100,

color: '#FF0000'

};

ReactDOM.render(

<h1 style = {myStyle}>菜鸟教程</h1>,

document.getElementById('example')

);

</script>

#### 注释

注释需要写在花括号中{}

#### 数组

JSX 允许在模板中插入数组，数组会自动展开所有成员

<script type="text/babel">

var arr = [

<h1>菜鸟教程</h1>,

<h2>学的不仅是技术，更是梦想！</h2>,

];

ReactDOM.render(

<div>{arr}</div>,

document.getElementById('example')

);

</script>

## 3.组件

1. 可以使用函数定义组件

function HelloMessage(props) {

return <h1>Hello World!</h1>;}

1. 使用es6 class来定义一个组件

class Welcome extends React.Component {

render() {

return <h1>Hello World!</h1>;

}}

const element = <HelloMessage /> 为用户自定义的组件

向组件传递参数

<script type="text/babel">

//用class来定义组件

class HelloMessage extends React.Component {

render () {

return <h1>Hello {this.props.name} !</h1>

}

}

//------------------------------------------//

//用函数来定义组件

function HelloMessage(props) {

return <h1>Hello {props.name}!</h1>;

}

const element = <HelloMessage name="Runoob"/>;

ReactDOM.render(

element,

document.getElementById('example')

);

</script>

#### 复合组件

我们可以通过创建多个组件来合成一个组件，即把组件的不同功能点进行分离。

<script type="text/babel">

//通过函数来定义组件

function Name(props) {

return <h1>网站名称：{props.name}</h1>;

}

function Url(props) {

return <h1>网站地址：{props.url}</h1>;

}

function Nickname(props) {

return <h1>网站小名：{props.nickname}</h1>;

}

function App() {

return (

<div>

<Name name="菜鸟教程" />

<Url url="http://www.runoob.com" />

<Nickname nickname="Runoob" />

</div>

);

}

//将元素渲染到页面上

ReactDOM.render(

<App />,

document.getElementById('example')

);

</script>

#### 4.React State(状态)

React 把组件看成是一个状态机（State Machines）。通过与用户的交互，实现不同状态，然后渲染 UI，让用户界面和数据保持一致。

React 里，只需更新组件的 state，然后根据新的 state 重新渲染用户界面（不要操作 DOM）。

#### 5.React Props

state和props主要的区别在于props是不可变的，而state可以根据与用户交互来改变，这就是为什么有些容器组件需要定义state来更新和修改数据，而子组件只能通过props来传递数据

通过函数来创建组件时 用props.xx来调用参数

通过class类来创建组件时 用this.props.xx来调用参数

###### 使用state和props的用例

<script type="text/babel">

//创建父组件 给父组件state赋值

class WebSite extends React.Component {

constructor() {

super();

this.state = {

name: "菜鸟教程",

site: "http://www.runoob.com"

}

}

render() {

return (

<div>

// 在父组件中调用Name 和 Link 子组件 并把state的值作为参数

// 传给子组件

<Name name={this.state.name} />

<Link site={this.state.site} />

</div>

);

}

}

// 创建Name子组件

class Name extends React.Component {

render() {

return (

// 获取父组件调用时所传来的参数

<h1>{this.props.name}</h1>

);

}

}

// 创建Link子组件

class Link extends React.Component {

render() {

return (

// 获取父组件调用Link时所传来的参数

<a href={this.props.site}>

{this.props.site}

</a>

);

}

}

// 在页面上渲染组件

ReactDOM.render(

<WebSite />,

document.getElementById('example')

);

</script>

##### Props验证

如果要进行props验证需要额外引入prop-types库

<script src="https://cdn.bootcss.com/prop-types/15.6.1/prop-types.js"></script>

<script type="text/babel">

var title = "菜鸟教程";

// var title = 123;

class MyTitle extends React.Component {

render() {

return (

<h1>Hello, {this.props.title}</h1>

);

}

}

// MyTitle有一个propTypes方法 设置title参数的类型只能为字符串

MyTitle.propTypes = {

title: PropTypes.string

};

ReactDOM.render(

<MyTitle title={title} />,

document.getElementById('example')

);

</script>

Props更多验证器说明





#### 6.React事件处理

事件命名采用驼峰式命名法

需要传入一个函数 作为事件处理函数

<script type="text/babel">

class Toggle extends React.Component {

constructor(props) {

super(props);

this.state = {isToggleOn: true};

// 这边绑定是必要的，这样 `this` 才能在回调函数中使用

// ？？？？ 不太明白这里为什么要绑定this

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

this.setState(prevState => ({

isToggleOn: !prevState.isToggleOn

}));

}

render() {

return (

<button onClick={this.handleClick}>

{this.state.isToggleOn ? 'ON' : 'OFF'}

</button>

);

}

}

ReactDOM.render(

<Toggle />,

document.getElementById('example')

);

</script>

##### 向事件处理程序传递参数

<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button><button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>

参数e作为React事件对象将会被作为第二个参数进行传递，通过箭头函数的方式，事件对象必须显式的进行传递，但是通过bind的方式，事件对象以及更多的参数将会被隐式的进行传递

通过bind方式向监听函数传参，在类组件中定义的监听函数，事件对象e要排

在所传递的参数的后面



#### Constructor() 和 super() 的意义

Constructor（）是组件的构造函数 可以理解为给组件初始化用的，

Constructor可以有也可以没有

但是 如果有Constructor() 就一定有super()

当需要在constructor中访问this.props时 就需要调用super(props)

-------Constructor() 构造方法

这是ES6对类的默认方法，通过new命令生成对象实例时自动调用该方法，并且，该方法是类中必须有的，如果没有显示定义，则会默认添加空的constructor()

--------Super() 继承

在calss方法中，继承是使用extends关键字来实现的，子类必须在constructor()调用super() 方法 ，否则新建实例时会报错

报错原因：子类是没有自己的this对象的，它只能继承父类的this对象，然后对其进行加工，而super() 就是将父类中的this对象继承给子类，没有super 子类就得不到this对象

#### 7.React条件渲染

例子：

class LoginControl extends React.Component {

// 创建一个LoginControl组件 给这个组件一个构造函数初始化

constructor(props) {

// super使构造函数内可以使用this

super(props);

// 给this.handleLoginClick 绑定当前组件的

this.handleLoginClick = this.handleLoginClick.bind(this);

this.handleLogoutClick = this.handleLogoutClick.bind(this);

this.state = {isLoggedIn: false};

}

handleLoginClick() {

this.setState({isLoggedIn: true});

}

handleLogoutClick() {

this.setState({isLoggedIn: false});

}

render() {

const isLoggedIn = this.state.isLoggedIn;

let button;

if (isLoggedIn) {

// 因为在上文中 this.handleLoginClick 被绑定了父组件

// 所以子组件在调用this.handleLoginClick时

// 函数内部的this指向的还是父组件而不是此时调用的子组件

button = <LogoutButton onClick={this.handleLogoutClick} />;

} else {

button = <LoginButton onClick={this.handleLoginClick} />;

}

return (

<div>

<Greeting isLoggedIn={isLoggedIn} />

{button}

</div>

);

}

}

function UserGreeting(props) {

return <h1>欢迎回来!</h1>;

}

function GuestGreeting(props) {

return <h1>请先注册。</h1>;

}

function Greeting(props) {

const isLoggedIn = props.isLoggedIn;

if (isLoggedIn) {

return <UserGreeting />;

}

return <GuestGreeting />;

}

function LoginButton(props) {

return (

<button onClick={props.onClick}>

登陆

</button>

);

}

function LogoutButton(props) {

return (

<button onClick={props.onClick}>

退出

</button>

);

}

ReactDOM.render(

<LoginControl />,

document.getElementById('example')

);

</script>

#### 8.React 列表 和 keys

Keys在react中一般给到调用时的组件上 可以在dom中某些元素被增加或者删除的时候帮助react识别哪些元素发生了变化 因此应当给数组中的每一个元素赋予一个确定的标识

例子：

<script type="text/babel">

function ListItem(props) {

// 对啦！这里不需要指定key:

return <li>{props.value}</li>;

}

function NumberList(props) {

const numbers = props.numbers;

const listItems = numbers.map((number) =>

// 又对啦！key应该在数组的上下文中被指定

<ListItem key={number.toString()}

value={number} />

);

return (

<ul>

{listItems}

</ul>

);

}

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(

<NumberList numbers={numbers} />,

document.getElementById('example')

);

</script>

###### 元素的key在他的兄弟元素之间应该唯一

但是在全局之间 例如用两个组件渲染两个不同的对象时，不同组件之间的id可以是相同的

#### 9.React组件API

###### 设置状态：setState setState(object nextState[, function callback])

不能在组件内部通过this.state修改状态，因为该状态会调用setState() 后被替换

setState() 并不会立即改变this.state 而是创建一个即将处理的state setState()并不一定是同步的，为了提升性能 React会批量执行state和DOM渲染

setState()总是会触发一次组件重绘，除非在shouldComponentUpdate()中实现了一些条件渲染逻辑

###### 替换状态:replaceState replacState(object nextState[, function callback])

replaceState()方法与setState()类似，但是方法hi会保留nextState中状态，原state不再nextState中的状态都会被删掉

###### 设置属性：setProps setProps(object nextProps[, function callback])

设置组件属性 并重新渲染组件

Props相当与组件的数据流，他总是会从父组件向下传递至所有的子组件中，当和一个外部的javascript应用集成时，我们可能会需要像组件传递数据或通知React.Render()组件需要重新渲染 可以使用setProps()

更新组件我可以在节点上再次调用React.render() 也可以通过setProps()方法改变组件属性，触发组件重新渲染

###### 替换属性：replaceProps

###### replaceProps(object nextProps[, function callback])

replaceProps()方法与setProps类似，但是它会删除原有props

###### 强制更新：forceUpdate forceUpdate([function callback])

forceUpdate() 方法会使组件调用自身的render()方法重新渲染组件，组件的子组件也会调用自己的render() 但是，组件重新渲染时，依然会读取this.props 和 this.state 如果状态没有改变，那么React只会更新DOM

forceUpdate方法适用于this.props和this.state之外的组件重绘（如：修改了this.state后）通过该方法通知React需要调用render()

一般来说，应该尽量避免使用forceUpdate() 而仅仅从this.props和this.state中读取状态并由React触发render()调用

###### 获取DOM节点：findDOMNode DOMElement findDOMNode()

返回值：DOM元素DOMElement

如果组件已经挂载到DOM中，该方法返回对应的本地浏览器DOM元素

当render返回null或false时，this.findDOMNode()也会返回null

###### 判断组件挂在状态：isMounted bool isMounted()

返回值：true或false 表示组件是否已挂载到DOM中

isMounted()方法用于判断组件是否已挂在到DOM中，可以使用该方法保证了setState() 和 forceUpdate()在异步场景下的调用不会出错

#### React组件生命周期

组件生命周期 分为三个状态

###### Mounting 已插入真实DOM

React Components被render解析 生成对应的DOM节点，并被插入浏览器的DOM结构的一个过程

getInitailProps() 通过组件调用方在调用组件时指定的 props一般是不变的

对于被调用组件来说，props的拥有者是调用方

getInitialState() 用来初始化react components的state

componentWillMount() componets将要被解析插入浏览器

componentDidMount() components已经被解析插入浏览器

###### Updating 正在被重新渲染

一个mounted的React Components被重新render的过程

一个已经被render解析过了生成了对应的DOM节点，并被插入浏览器的DOM结构后又重新被render的过程

并不是说重新被render一次就一定会改变，react会把这个components的当前state和最近一次state对比一下，如果当前state发生改变，并影响到了当前dom结构时，才会改变

componentWillReceiveProps() 当一个mounted的component将要接收新的props 时会调用这个函数，其参数就是新的props对象 可以用在比较这个函数和this.props是否相同从而 修改state

ShouldComponentUpdate() 当一个mounted的component已经接收到新的

props和state之后，判断是否有必要去更新dom

结构，参数有两个，第一个：新的props对象 ，

第二个：新的state对象 分别对比来判断是否需

要更新dom 函数返回true 表示更新 返回false

不更新

componentWillUpdate()

componentDidUpdate()

###### 3.Unmounting 已移除真实DOM

一个mounted的React Components对应的DOM节点被从DOM结构中移除的这样一个过程

componentWillUnmount() 可以在此函数中执行释放图片资源，内存资源等

操作

#### 11.React AJAX

React组件的数据可以通过componentDidMount方法中的ajax来获取，当从服务器获取数据时可以将数据存储在state中，再用this.setState方法重新渲染UI

当使用异步加载数据时，在组件卸载前使用componentWillUnmount来取消未完成的请求

<script type="text/babel">

class UserGist extends React.Component {

constructor(props) {

super(props);

this.state = {username: '', lastGistUrl: ''};

}

componentDidMount() {

this.serverRequest = $.get(this.props.source, function (result) {

var lastGist = result[0];

this.setState({

username: lastGist.owner.login,

lastGistUrl: lastGist.html\_url

});

}.bind(this));

}

componentWillUnmount() {

this.serverRequest.abort();

}

render() {

return (

<div>

{this.state.username} 用户最新的 Gist 共享地址：

<a href={this.state.lastGistUrl}>{this.state.lastGistUrl}</a>

</div>

);

}

}

ReactDOM.render(

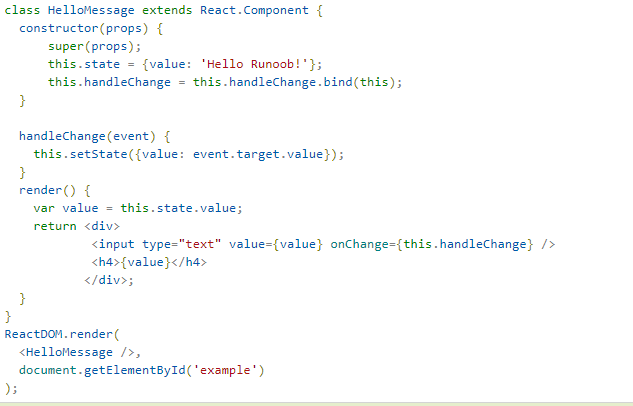
<UserGist source="https://api.github.com/users/octocat/gists" />,

document.getElementById('example')

);

</script>

## 12.React表单与事件





#### Select下拉菜单



## 13.React Refs

React支持一种非常特殊的属性Ref，你可以用来绑定到render()输出的任何组件上，这个特殊属性允许你引用render()返回的相应的支撑实例，这样就可以确保在任何时间总是拿到正确的实例

<script type="text/babel">

class MyComponent extends React.Component {

handleClick() {

// 使用原生的 DOM API 获取焦点

this.refs.myInput.focus();

}

render() {

// 当组件插入到 DOM 后，ref 属性添加一个组件的引用于到 this.refs

return (

<div>

<input type="text" ref="myInput" />

<input

type="button"

value="点我输入框获取焦点"

onClick={this.handleClick.bind(this)}

/>

</div>

);

}

}

ReactDOM.render(

<MyComponent />,

document.getElementById('example')

);

</script>