菜鸟教程

React 是一个用于构建用户界面的 JAVASCRIPT 库。

React主要用于构建UI，很多人认为 React 是 MVC 中的 V（视图）。

React 起源于 Facebook 的内部项目，用来架设 Instagram 的网站，并于 2013 年 5 月开源。

React 拥有较高的性能，代码逻辑非常简单，越来越多的人已开始关注和使用它。

React安装：1. 安装nodejs(最新版本自带npm更加方便8.9.2以上)；2. 安装cnpm，因为国内使用npm速度很慢： $ npm install -g cnpm --registry=https://registry.npm.taobao.org

$ npm config set registry <https://registry.npm.taobao.org>

接下来使用create-react-app快速构建react 开发环境

create-react-app 是来自于 Facebook，通过该命令我们无需配置就能快速构建 React 开发环境。

create-react-app 自动创建的项目是基于 Webpack + ES6 。

执行以下命令创建项目：

$ cnpm install -g create-react-app

$ create-react-app my-app

$ cd my-app/

$ npm start

在浏览器中打开 **http://localhost:3000/** ，尝试修改 src/App.js 文件代码。

**React 特点**

* **1.声明式设计** −React采用声明范式，可以轻松描述应用。
* **2.高效** −React通过对DOM的模拟（Document Object Model，也就是文档对象模型 就是针对 HTML 和 XML 提供的一个API。什么意思？就是说为了能以编程的方法操作这个 HTML 的内容（比如添加某些元素、修改元素的内容、删除某些元素），我们把这个 HTML 看做一个对象树（DOM树），它本身和里面的所有东西比如 <div></div> 这些标签都看做一个对象，每个对象都叫做一个节点（node），节点可以理解为 DOM 中所有 Object 的父类。），最大限度地减少与DOM的交互。
* **3.灵活** −React可以与已知的库或框架很好地配合。
* **4.JSX** − JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX ，但我们建议使用它。它允许将HTML和JS混写而不用引号，<>用HTML规则解析，{}用JS规则解析。
* **5.组件** − 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。
* **6.单向响应的数据流** − React 实现了单向响应的数据流，从而减少了重复代码，这也是它为什么比传统数据绑定更简单。

React通过createClass新建标签分支，之后可以作为标签直接使用，里面主函数是render。通过reactDOM.render调用这些标签，标签所在位置通过document.getNameByElement获得。大部分props和state下的变量需要通过花括号加以区别。

**HTML5特性：**

HTML5添加了许多新的语法特征，其中包括<video>、<audio>和<canvas>元素。这些元素是为了更容易的在网页中添加和处理多媒体和图片内容而添加的。其它新的元素如<section>、<article>、<header>和<nav>则是为了丰富文档。同时也有一些属性和元素被移除掉了。一些元素，像<a>、<cite>和<menu>被修改，重新定义或标准化了。

# React JSX

它有以下优点：

* JSX 执行更快，因为它在编译为 JavaScript 代码后进行了优化。
* 它是类型安全的，在编译过程中就能发现错误。
* 使用 JSX 编写模板更加简单快速。
* 我们可以在 JSX 中使用 JavaScript 表达式。表达式写在花括号 **{}** 中。{1+1}
* 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**. {i == 1 ? 'True!' : 'False'}
* 变量也需要写在花括号内。 例如：var myStyle = { fontSize: 100, color: '#FF0000' }; ReactDOM.render( <h1 style = {myStyle}>菜鸟教程</h1>, document.getElementById('example') );
* 注释需要写在花括号中。1、在标签内部的注释需要花括号。2、在标签外的的注释不能使用花括号
* JSX 允许在模板中插入数组，数组会自动展开所有成员。也写在大括号内（同变量）
* 要渲染 React 组件，只需创建一个大写字母开头的本地变量。React 的 JSX 使用大、小写的约定来区分本地组件的类和 HTML 标签。

# React 组件

**React.createClass** 方法用于生成一个组件类 **HelloMessage**。

**<HelloMessage />** 实例组件类并输出信息。

*注意，原生 HTML 元素名以小写字母开头，而自定义的 React 类名以大写字母开头，比如 HelloMessage 不能写成 helloMessage。除此之外还需要注意组件类只能包含一个顶层标签，否则也会报错。*

如果我们需要向组件传递参数，可以使用 **this.props** 对象,实例如下：

**React 实例**

var HelloMessage = React.createClass({ render: function() { return <h1>Hello {this.props.name}</h1>; } }); ReactDOM.render( <HelloMessage name="Runoob" />, document.getElementById('example') );

*注意，在添加属性时， class 属性需要写成 className ，for 属性需要写成 htmlFor ，这是因为 class 和 for 是 JavaScript 的保留字。*

# 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。

React.createClass({

propTypes: {

// 可以声明 prop 为指定的 JS 基本数据类型，默认情况，这些数据是可选的

optionalArray: React.PropTypes.array,

optionalBool: React.PropTypes.bool,

optionalFunc: React.PropTypes.func,

optionalNumber: React.PropTypes.number,

optionalObject: React.PropTypes.object,

optionalString: React.PropTypes.string,

// 可以被渲染的对象 numbers, strings, elements 或 array

optionalNode: React.PropTypes.node,

// React 元素

optionalElement: React.PropTypes.element,

// 用 JS 的 instanceof 操作符声明 prop 为类的实例。

optionalMessage: React.PropTypes.instanceOf(Message),

// 用 enum 来限制 prop 只接受指定的值。

optionalEnum: React.PropTypes.oneOf(['News', 'Photos']),

// 可以是多个对象类型中的一个

optionalUnion: React.PropTypes.oneOfType([

React.PropTypes.string,

React.PropTypes.number,

React.PropTypes.instanceOf(Message)

]),

// 指定类型组成的数组

optionalArrayOf: React.PropTypes.arrayOf(React.PropTypes.number),

// 指定类型的属性构成的对象

optionalObjectOf: React.PropTypes.objectOf(React.PropTypes.number),

// 特定 shape 参数的对象

optionalObjectWithShape: React.PropTypes.shape({

color: React.PropTypes.string,

fontSize: React.PropTypes.number

}),

// 任意类型加上 `isRequired` 来使 prop 不可空。

requiredFunc: React.PropTypes.func.isRequired,

// 不可空的任意类型

requiredAny: React.PropTypes.any.isRequired,

// 自定义验证器。如果验证失败需要返回一个 Error 对象。不要直接使用 `console.warn` 或抛异常，因为这样 `oneOfType` 会失效。

customProp: function(props, propName, componentName) {

if (!/matchme/.test(props[propName])) {

return new Error('Validation failed!');

}

}

},

/\* ... \*/

});

this.props 对象的属性与组件的属性一一对应，但是有一个例外，就是 this.props.children 属性。它表示组件的所有子节点

# React 组件 API

### 关于setState

不能在组件内部通过this.state修改状态，因为该状态会在调用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。 设置属性：setProps

**props**相当于组件的数据流，它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，我们可能会需要向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

# React 组件生命周期

在本章节中我们将讨论 React 组件的生命周期。

组件的生命周期可分成三个状态：

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

**componentDidMount** : 在第一次渲染后调用，只在客户端。之后组件已经生成了对应的DOM结构，可以通过this.getDOMNode()来进行访问。 如果你想和其他JavaScript框架一起使用，可以在这个方法中调用setTimeout, setInterval或者发送AJAX请求等操作(防止异部操作阻塞UI)。和reactDOM不同的是，它具有一次性和持续性，可用来开发timer。

componentDidMount: function () {

this.timer = setInterval(function () {

var opacity = this.state.opacity;

opacity -= .05;

if (opacity < 0.1) {

opacity = 1.0;

}

this.setState({

opacity: opacity

});

}.bind(this), 100);

},

# React AJAX

React 组件的数据可以通过 componentDidMount 方法中的 Ajax 来获取，当从服务端获取数据库可以将数据存储在 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。

以下实例演示了获取 Github 用户最新 gist 共享描述，注意$.get(this.props.source,function(result))中source是以json存储的格式，二result就是source里的东西:

var UserGist = React.createClass({

getInitialState: function() {

return {

username: '',

lastGistUrl: ''

};

},

componentDidMount: function() {

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: function() {

this.serverRequest.abort();

},

render: function() {

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" />,

mountNode

);

# React 表单与事件

本章节我们将讨论如何在 React 中使用表单。

### 一个简单的实例

在实例中我们设置了输入框 input 值**value = {this.state.data}**。在输入框值发生变化时我们可以更新 state。我们可以使用 **onChange** 事件来监听 input 的变化，并修改 state。

## var HelloMessage = React.createClass({

## getInitialState: function() {

## return {value: 'Hello Runoob!'};

## },

## handleChange: function(event) {

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

## },

## render: function() {

## var value = this.state.value;

## return <div>

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

## <h4>{value}</h4>

## </div>;

## }

## });

## ReactDOM.render(

## <HelloMessage />,

## document.getElementById('example')

## );

用户在表单填入的内容，属于用户跟组件的互动，所以不能用 this.props 读取（查看 [demo9](https://github.com/ruanyf/react-demos/blob/master/demo09/index.html) ）。

var Input = React.createClass({

getInitialState: function() {

return {value: 'Hello!'};

},

handleChange: function(event) {

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

},

render: function () {

var value = this.state.value;

return (

<div>

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

<p>{value}</p>

</div>

);

}

});

ReactDOM.render(<Input/>, document.body);

上面代码中，文本输入框的值，不能用 this.props.value 读取，而要定义一个 onChange 事件的回调函数，通过 event.target.value 读取用户输入的值。textarea 元素、select元素、radio元素都属于这种情况

# 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。

这个特殊的属性允许你引用 render() 返回的相应的支撑实例（ backing instance ）。这样就可以确保在任何时间总是拿到正确的实例。

### 使用方法

绑定一个 ref 属性到 render 的返回值上：

<input ref="myInput" />

在其它代码中，通过 this.refs 获取支撑实例:

var input = this.refs.myInput;

var inputValue = input.value;

var inputRect = input.getBoundingClientRect();