**React**

**教程**

**V1.0**

**张镇**

**目 录**

[1 简介 1](#_Toc491271451)

[1.1 React背景 1](#_Toc491271452)

[1.2 React特点 1](#_Toc491271453)

[1.3 阅读本教程前，您需要了解的知识 1](#_Toc491271454)

[1.4 React 第一个实例 1](#_Toc491271455)

[2 React 安装 2](#_Toc491271456)

[2.1 使用实例 2](#_Toc491271457)

[2.2 通过 npm 使用 React 3](#_Toc491271458)

[2.3 使用 create-react-app 快速构建 React 开发环境 4](#_Toc491271459)

[3 React JSX 6](#_Toc491271460)

[3.1 使用 JSX 7](#_Toc491271461)

[3.2 独立文件 7](#_Toc491271462)

[3.3 JavaScript 表达式 8](#_Toc491271463)

[3.4 样式 8](#_Toc491271464)

[3.5 注释 9](#_Toc491271465)

[3.6 数组 9](#_Toc491271466)

[3.7 HTML 标签 vs. React 组件 9](#_Toc491271467)

[4 React 组件 10](#_Toc491271468)

[4.1 实例解析 11](#_Toc491271469)

[4.2 复合组件 11](#_Toc491271470)

[5 React State(状态) 12](#_Toc491271471)

[6 React Props 13](#_Toc491271472)

[6.1 使用 Props 13](#_Toc491271473)

[6.2 默认 Props 14](#_Toc491271474)

[6.3 State 和 Props 14](#_Toc491271475)

[6.4 Props 验证 15](#_Toc491271476)

[7 React 组件 API 17](#_Toc491271477)

[7.1 设置状态:setState 18](#_Toc491271478)

[7.1.1 参数说明 18](#_Toc491271479)

[7.1.2 关于setState 18](#_Toc491271480)

[7.1.3 实例 18](#_Toc491271481)

[7.2 替换状态：replaceState 19](#_Toc491271482)

[7.3 设置属性：setProps 19](#_Toc491271483)

[7.4 替换属性：replaceProps 19](#_Toc491271484)

[7.5 强制更新：forceUpdate 20](#_Toc491271485)

[7.6 获取DOM节点：findDOMNode 20](#_Toc491271486)

[7.7 判断组件挂载状态：isMounted 20](#_Toc491271487)

[8 React 组件生命周期 21](#_Toc491271488)

[9 React AJAX 24](#_Toc491271489)

[10 React 表单与事件 25](#_Toc491271490)

[10.1 一个简单的实例 25](#_Toc491271491)

[10.2 实例 2 26](#_Toc491271492)

[10.3 React 事件 27](#_Toc491271493)

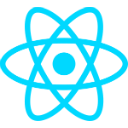
[11 React Refs 28](#_Toc491271494)

[11.1 使用方法 28](#_Toc491271495)

[11.2 完整实例 28](#_Toc491271496)

# 简介

## React背景

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，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。
* 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。

## 阅读本教程前，您需要了解的知识

在开始学习 React 之前，您需要具备以下基础知识：

* HTML5
* CSS
* JavaScript

## React 第一个实例

在每个章节中，您可以在线编辑实例，然后点击按钮查看结果。

本教程使用了 React 的版本为 15.4.2，你可以在官网 <http://facebook.github.io/react/> 下载最新版。

**React 实例**

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

<script type="text/babel">

ReactDOM.render(

<h1>Hello, world!</h1>,

document.getElementById('example')

);

</script>

# React 安装

React 可以直接下载使用，下载包中也提供了很多学习的实例。

本教程使用了 React 的版本为 15.4.2，你可以在官网 <http://facebook.github.io/react/> 下载最新版。

你也可以直接使用 BootCDN 的 React CDN 库，地址如下：

<script src=*"https://cdn.bootcss.com/react/15.4.2/react.min.js"*></script>

<script src=*"https://cdn.bootcss.com/react/15.4.2/react-dom.min.js"*></script>

<script src=*"https://cdn.bootcss.com/babel-standalone/6.22.1/babel.min.js"*></script>

## 使用实例

以下实例输出了 Hello, world!

**React 实例**

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"* />

<title>Hello React!</title>

<script src=*"https://cdn.bootcss.com/react/15.4.2/react.min.js"*></script>

<script src=*"https://cdn.bootcss.com/react/15.4.2/react-dom.min.js"*></script>

<script src=*"https://cdn.bootcss.com/babel-standalone/6.22.1/babel.min.js"*></script>

</head>

<body>

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

<script type=*"text/babel"*>

ReactDOM.render(

<h1>Hello, world!</h1>,

document.getElementById('example')

);

</script>

</body>

</html>

**实例解析：**

实例中我们引入了三个库： react.min.js 、react-dom.min.js 和 babel.min.js：

* **react.min.js** - React 的核心库。
* **react-dom.min.js** - 提供与 DOM[[1]](#footnote-1) 相关的功能。
* **babel.min.js** - Babel 可以将 ES6 代码转为 ES5 代码，这样我们就能在目前不支持 ES6 浏览器上执行 React 代码。Babel 内嵌了对 JSX 的支持。通过将 Babel 和 babel-sublime 包（package）一同使用可以让源码的语法渲染[[2]](#footnote-2)上升到一个全新的水平。

ReactDOM.render( <h1>Hello, world!</h1>, document.getElementById('example') );

以上代码将一个 h1 标题，插入 id="example" 节点中。

***注意：****如果我们需要使用 JSX，则 <script> 标签的 type 属性需要设置为 text/babel。*

## 通过 npm 使用 React

如果你的系统还未安装 Node.js 及 NPM，需要先行安装，具体步骤请上网查询。

我们建议在 React 中使用 CommonJS 模块系统，比如 browserify 或 webpack，本教程使用 webpack对React代码进行打包发布。

**注意：**

* 国内使用 npm 安装Node.js的模块速度很慢，你可以使用淘宝定制的 cnpm (gzip 压缩支持) 命令行工具代替默认的 npm:

$ npm install -g cnpm --registry=https://registry.npm.taobao.org

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

-- 配置后可通过下面方式来验证是否成功

$ npm config get registry

-- 或 npm info express

这样就可以使用 cnpm 命令来安装模块了：

$ cnpm install [name]

更多信息可以查阅：<http://npm.taobao.org/>。

* 打包发布时，系统需要Python2.X支持。Linux版安装包会自动安装Python，但Windows需要单独安装Python。注意，Node.js只支持Python2.X， 建议安装2.7版，不要安装3.X版！！
* 打包发布时，npm自动调用WebPack。Linux自动安装WebPack，但Windows安装包中未包含此部分，需要单独安装。安装时需要明确WebPack版本。建议使用V2版。命令：npm install webpack@2

## 使用 create-react-app 快速构建 React 开发环境

create-react-app 命令行工具是来自于 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：[[3]](#footnote-3)

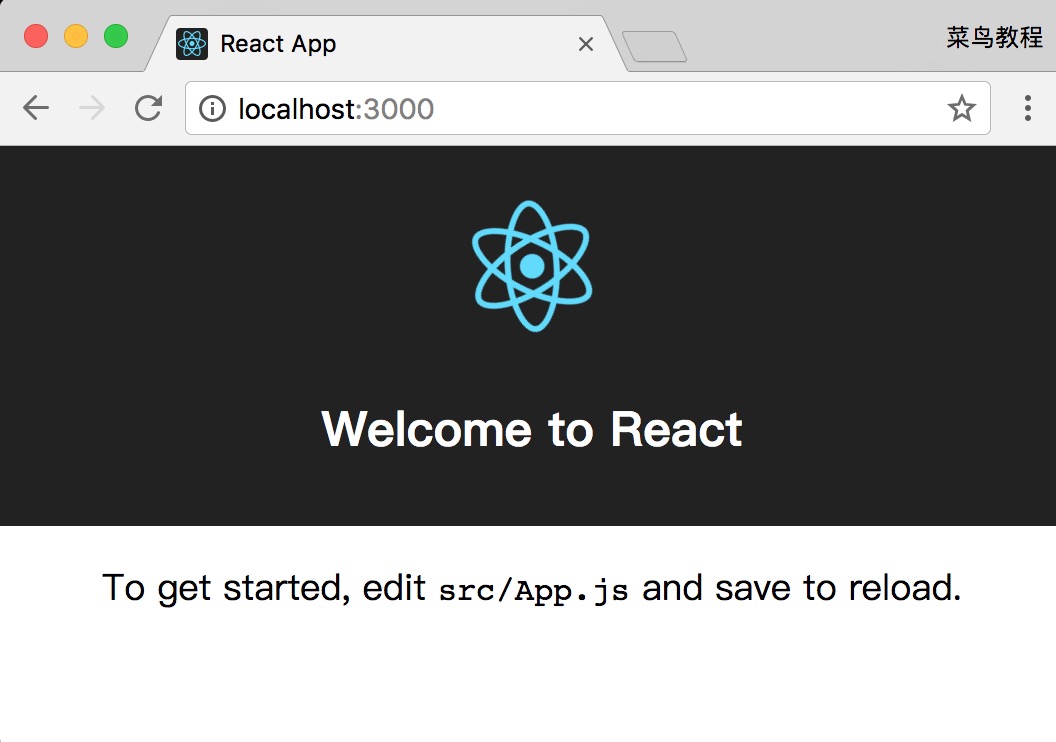
$ cnpm install -g create-react-app

$ create-react-app my-app

$ cd my-app/

$ npm start

在浏览器中打开 **http://localhost:3000/** ，结果如下图所示：



项目的目录结构如下：

my-app/

README.md

node\_modules/

package.json

.gitignore

public/

favicon.ico

index.html

src/

App.css

App.js

App.test.js

index.css

index.js

logo.svg

尝试修改 src/App.js 文件代码：

**src/App.js**

import React, { Component } from 'react';

import logo from './logo.svg';

import './App.css';

**class** App **extends** Component {

render() {

**return** (

<div className="App">

<div className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<h2>欢迎来到菜鸟教程</h2>

</div>

<p className="App-intro">

你可以在 <code>src/App.js</code> 文件中修改。

</p>

</div>

);

}

}

**export** **default** App;

修改后，打开 http://localhost:3000/ （一般自动刷新），输出结果如下：



# React JSX

React 使用 JSX 来替代常规的 JavaScript，JSX 是一个看起来很像 XML 的 JavaScript 语法扩展。我们不需要一定使用 JSX，但它有以下优点：

* 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。

## 使用 JSX

JSX 看起来类似 HTML ，我们可以看下实例:

ReactDOM.render(

<h1>Hello, world!</h1>,

document.getElementById('example')

);

我们可以在以上代码中嵌套多个 HTML 标签，需要使用一个 div 元素包裹它（<div id="example"></div>），实例中的 p 元素添加了自定义属性 data-myattribute，添加自定义属性需要使用 data- 前缀。

**React 实例**

ReactDOM.render(

<div>

<h1>菜鸟教程</h1>

<h2>欢迎学习 React</h2>

<p data-myattribute = "somevalue">这是一个很不错的 JavaScript 库!</p>

</div>

,

document.getElementById('example')

);

## 独立文件

你的 React JSX 代码可以放在一个独立文件上，例如我们创建一个 helloworld\_react.js 文件，代码如下：

ReactDOM.render(

<h1>Hello, world!</h1>,

document.getElementById('example')

);

然后在 HTML 文件中引入该 JS 文件：

**React 实例**

<body>

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

<script type=*"text/babel"* src=*"helloworld\_react.js"*></script>

</body>

## JavaScript 表达式

我们可以在 JSX 中使用 JavaScript 表达式。表达式写在花括号 {} 中。实例如下：

**React 实例**

ReactDOM.render(

<div>

<h1>{1+1}</h1>

</div>

,

document.getElementById('example')

);

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.

**React 实例**

ReactDOM.render(

<div>

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

</div>

,

document.getElementById('example')

);

## 样式

React 推荐使用内联样式[[4]](#footnote-4)。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：

**React 实例**

**var** myStyle = {

fontSize: 100,

color: '#FF0000'

};

ReactDOM.render(

<h1 style = {myStyle}>菜鸟教程</h1>,

document.getElementById('example')

);

## 注释

1、在标签内部的注释需要花括号，实例如下：

**React 实例**

ReactDOM.render(

<div>

<h1>菜鸟教程</h1>

{/\*注释...\*/}

</div>,

document.getElementById('example')

);

2、在标签外的的注释不能使用花括号，实例如下：

ReactDOM.render(

/\* 注释... \*/

<div>

<h1>菜鸟教程</h1>

</div>,

document.getElementById('example')

);

## 数组

JSX 允许在模板中插入数组，数组会自动展开所有成员：

**React 实例**

**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')

);

## HTML 标签 vs. React 组件

React 可以渲染 HTML 标签 (strings) 或 React 组件 (classes)。

要渲染 HTML 标签，只需在 JSX 里使用小写字母的标签名。

**var** myDivElement = <div className="foo" />;

ReactDOM.render(myDivElement, document.getElementById('example'));

要渲染 React 组件，只需创建一个大写字母开头的本地变量。

**var** MyComponent = React.createClass({/\*...\*/});

**var** myElement = <MyComponent someProperty={**true**} />;

ReactDOM.render(myElement, document.getElementById('example'));

React 的 JSX 使用大、小写的约定来区分本地组件的类和 HTML 标签。

*注意:*

*由于 JSX 就是 JavaScript，一些标识符像 class 和 for 不建议作为 XML 属性名。作为替代，React DOM 使用 className 和 htmlFor 来做对应的属性。*

代码中嵌套多个 HTML 标签，需要使用一个标签元素包裹它。

1.错误例子：

ReactDOM.render(

<h1>这是错误的例子</h1>

<span>假设这里是标题下面的内容</span>,

document.getElementById("example")

);

2.正确例子：

ReactDOM.render(

<section>

<h1>这是正确的例子</h1>

<span>假设这里是标题下面的内容</span>

</section>,

document.getElementById("example")

);

# React 组件

本章节我们将讨论如何使用组件使得我们的应用更容易来管理。

接下来我们封装一个输出 "Hello World！" 的组件，组件名为 HelloMessage：

**React 实例**

**var** HelloMessage = React.createClass({

render: **function**() {

**return** <h1>Hello World！</h1>;

}

});

ReactDOM.render(

<HelloMessage />,

document.getElementById('example')

);

## 实例解析

**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')

);

以上实例中 **name** 属性通过 **this.props.name** 来获取。

*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。*

## 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。

以下实例我们实现了输出网站名字和网址的组件：

**React 实例**

**var** WebSite = React.createClass({

render: **function**() {

**return** (

<div>

<Name name={**this**.props.name} />

<Link site={**this**.props.site} />

</div>

);

}

});

**var** Name = React.createClass({

render: **function**() {

**return** (

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

);

}

});

**var** Link = React.createClass({

render: **function**() {

**return** (

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

{**this**.props.site}

</a>

);

}

});

ReactDOM.render(

<WebSite name="菜鸟教程" site=" http://www.runoob.com" />,

document.getElementById('example')

);

实例中 WebSite 组件使用了 Name 和 Link 组件来输出对应的信息，也就是说 WebSite 拥有 Name 和 Link 的实例。

# 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）。

以下实例中创建了 **LikeButton** 组件，getInitialState 方法用于定义初始状态，也就是一个对象，这个对象可以通过 this.state 属性读取。当用户点击组件，导致状态变化，this.setState 方法就修改状态值，每次修改以后，自动调用 this.render 方法，再次渲染组件。

**React 实例**

({liked: !**this**.state.liked});

},

render: **function**() {

**var** text = **this**.state.liked ? '喜欢' : '不喜欢';

**return** (

<p onClick={**this**.handleClick}>

你<b>{text}</b>我。点我切换状态。

</p>

);

}

});

ReactDOM.render(

<LikeButton />,

document.getElementById('example')

);

onClick 等事件，与原生 HTML 不同，on 之后第一个字母是大写的！

比如本章实例中，如果将**onClick={this.handleClick}**换成**onclick={this.handleClick}**则点击事件不再生效

# 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。

## 使用 Props

以下实例演示了如何在组件中使用 props：

**React 实例**

**var** HelloMessage = React.createClass({

render: **function**() {

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

}

});

ReactDOM.render(

<HelloMessage name="Runoob" />,

document.getElementById('example')

);

实例中 name 属性通过 this.props.name 来获取。

## 默认 Props

你可以通过 getDefaultProps() 方法为 props 设置默认值，实例如下：

**React 实例**

**var** HelloMessage = React.createClass({

getDefaultProps: **function**() {

**return** {

name: 'Runoob'

};

},

render: **function**() {

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

}

});

ReactDOM.render(

<HelloMessage />,

document.getElementById('example')

);

## State 和 Props

以下实例演示了如何在应用中组合使用 state 和 props 。我们可以在父组件中设置 state， 并通过在子组件上使用 props 将其传递到子组件上。在 render 函数中, 我们设置 name 和 site 来获取父组件传递过来的数据。

**React 实例**

**var** WebSite = React.createClass({

getInitialState: **function**() {

**return** {

name: "菜鸟教程",

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

};

},

render: **function**() {

**return** (

<div>

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

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

</div>

);

}

});

**var** Name = React.createClass({

render: **function**() {

**return** (

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

);

}

});

**var** Link = React.createClass({

render: **function**() {

**return** (

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

{**this**.props.site}

</a>

);

}

});

ReactDOM.render(

<WebSite />,

document.getElementById('example')

);

## Props 验证

Props 验证使用 propTypes，它可以保证我们的应用组件被正确使用，React.PropTypes 提供很多验证器 (validator) 来验证传入数据是否有效。当向 props 传入无效数据时，JavaScript 控制台会抛出警告。

以下实例创建一个 Mytitle 组件，属性 title 是必须的且是字符串，非字符串类型会自动转换为字符串 ：

**React 实例**

**var** title = "菜鸟教程";

// var title = 123;

**var** MyTitle = React.createClass({

propTypes: {

title: React.PropTypes.string.isRequired,

},

render: **function**() {

**return** <h1> {**this**.props.title} </h1>;

}

});

ReactDOM.render(

<MyTitle title={title} />,

document.getElementById('example')

);

**更多验证器说明如下：**

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!');

}

}

},

/\* ... \*/

});

# React 组件 API

在本章节中我们将讨论 React 组件 API。我们将讲解以下7个方法:

**设置状态：setState**

* 替换状态：replaceState
* 设置属性：setProps
* 替换属性：replaceProps
* 强制更新：forceUpdate
* 获取DOM节点：findDOMNode
* 判断组件挂载状态：isMounted

## 设置状态:setState

setState(object nextState[, function callback])

### 参数说明

* **nextState**，将要设置的新状态，该状态会和当前的**state**合并
* **callback**，可选参数，回调函数。该函数会在**setState**设置成功，且组件重新渲染后调用。

合并nextState和当前state，并重新渲染组件。setState是React事件处理函数中和请求回调函数中触发UI更新的主要方法。

### 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。

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。

### 实例

**React 实例**

**var** Counter = React.createClass({

getInitialState: **function** () {

**return** { clickCount: 0 };

},

handleClick: **function** () {

**this**.setState(**function**(state) {

**return** {clickCount: state.clickCount + 1};

});

},

render: **function** () {

**return** (<h2 onClick={**this**.handleClick}>点我！点击次数为: {**this**.state.clickCount}</h2>);

}

});

ReactDOM.render(

<Counter />,

document.getElementById('message')

);

实例中通过点击 h2 标签来使得点击计数器加 1。

## 替换状态：replaceState

replaceState(object nextState[, function callback])

* **nextState**，将要设置的新状态，该状态会替换当前的**state**。
* **callback**，可选参数，回调函数。该函数会在**replaceState**设置成功，且组件重新渲染后调用。

**replaceState()**方法与**setState()**类似，但是方法只会保留**nextState**中状态，原**state**不在**nextState**中的状态都会被删除。

## 设置属性：setProps

setProps(object nextProps[, function callback])

* **nextProps**，将要设置的新属性，该状态会和当前的**props**合并。
* **callback**，可选参数，回调函数。该函数会在**setProps**设置成功，且组件重新渲染后调用。

设置组件属性，并重新渲染组件。

**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，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])

* **nextProps**，将要设置的新属性，该属性会替换当前的**props**。
* **callback**，可选参数，回调函数。该函数会在**replaceProps**设置成功，且组件重新渲染后调用。

**replaceProps()**方法与**setProps**类似，但它会删除原有props。

## 强制更新：forceUpdate

forceUpdate([function callback])

**参数说明**

* **callback**，可选参数，回调函数。该函数会在组件**render()**方法调用后调用。

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**。从DOM 中读取值的时候，该方法很有用，如：获取表单字段的值和做一些 DOM 操作。

## 判断组件挂载状态：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载到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。

*本文参考：http://itbilu.com/javascript/react/EkACBdqKe.html*

# 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

生命周期的方法有：

* **componentWillMount**：在客户端也在服务端渲染前调用。
* **componentDidMount**：在客户端第一次渲染后调用。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)。
* **componentWillReceiveProps**：在组件接收到一个新的prop时调用。在初始化render时这个方法不会被调用。
* **shouldComponentUpdate**：返回一个布尔值。在组件接收到新的props或者state时调用。在初始化时或者使用forceUpdate时不被调用。可以在你确认不需要更新组件时使用。
* **componentWillUpdate**：在组件接收到新的props或者state，但还没有render时调用。在初始化时不会被调用。
* **componentDidUpdate**：在组件完成更新后立即调用。在初始化时不会被调用。
* **componentWillUnmount**：在组件从 DOM 中移除的时候立刻被调用。

这些方法的详细说明，可以参考[官方文档](http://facebook.github.io/react/docs/component-specs.html#lifecycle-methods)。

以下实例在 Hello 组件加载以后，通过 componentDidMount 方法设置一个定时器，每隔100毫秒重新设置组件的透明度，并重新渲染：

**React 实例**

**var** Hello = React.createClass({

getInitialState: **function** () {

**return** {

opacity: 1.0

};

},

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);

},

render: **function** () {

**return** (

<div style={{opacity: **this**.state.opacity}}>

Hello {**this**.props.name}

</div>

);

}

});

ReactDOM.render(

<Hello name="world"/>,

document.body

);

以下实例初始化 **state** ， **setNewnumber** 用于更新 **state**。所有生命周期在 **Content** 组件中。

**React 实例**

**var** Button = React.createClass({

getInitialState: **function**() {

**return** {

data:0

};

},

setNewNumber: **function**() {

**this**.setState({data: **this**.state.data + 1})

},

render: **function** () {

**return** (

<div>

<button onClick = {**this**.setNewNumber}>INCREMENT</button>

<Content myNumber = {**this**.state.data}></Content>

</div>

);

}

})

**var** Content = React.createClass({

componentWillMount:**function**() {

console.log('Component WILL MOUNT!')

},

componentDidMount:**function**() {

console.log('Component DID MOUNT!')

},

componentWillReceiveProps:**function**(newProps) {

console.log('Component WILL RECEIVE PROPS!')

},

shouldComponentUpdate:**function**(newProps, newState) {

**return** **true**;

},

componentWillUpdate:**function**(nextProps, nextState) {

console.log('Component WILL UPDATE!');

},

componentDidUpdate:**function**(prevProps, prevState) {

console.log('Component DID UPDATE!')

},

componentWillUnmount:**function**() {

console.log('Component WILL UNMOUNT!')

},

render: **function** () {

**return** (

<div>

<h3>{**this**.props.myNumber}</h3>

</div>

);

}

});

ReactDOM.render(

<div>

<Button />

</div>,

document.getElementById('example')

);

# 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:

**React 实例**

**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

);

以上代码使用 jQuery 完成 Ajax 请求。

# 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。

**React 实例**

**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')

);

上面的代码将渲染出一个值为 Hello Runoob! 的 input 元素，并通过 onChange 事件响应更新用户输入的值。

## 实例 2

在以下实例中我们将为大家演示如何在子组件上使用表单。 **onChange** 方法将触发 state 的更新并将更新的值传递到子组件的输入框的 **value** 上来重新渲染界面。

你需要在父组件通过创建事件句柄 (**handleChange**) ，并作为 prop (**updateStateProp**) 传递到你的子组件上。

**React 实例**

**var** Content = React.createClass({

render: **function**() {

**return** <div>

<input type="text" value={**this**.props.myDataProp} onChange={**this**.props.updateStateProp} />

<h4>{**this**.props.myDataProp}</h4>

</div>;

}

});

**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>

<Content myDataProp = {value}

updateStateProp = {**this**.handleChange}></Content>

</div>;

}

});

ReactDOM.render(

<HelloMessage />,

document.getElementById('example')

);

## React 事件

以下实例演示通过 onClick 事件来修改数据：

**React 实例**

**var** HelloMessage = React.createClass({

getInitialState: **function**() {

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

},

handleChange: **function**(event) {

**this**.setState({value: '菜鸟教程'})

},

render: **function**() {

**var** value = **this**.state.value;

**return** <div>

<button onClick={**this**.handleChange}>点我</button>

<h4>{value}</h4>

</div>;

}

});

ReactDOM.render(

<HelloMessage />,

document.getElementById('example')

);

当你需要从子组件中更新父组件的 **state** 时，你需要在父组件通过创建事件句柄 (**handleChange**) ，并作为 prop (**updateStateProp**) 传递到你的子组件上。实例如下：

**React 实例**

**var** Content = React.createClass({

render: **function**() {

**return** <div>

<button onClick = {**this**.props.updateStateProp}>点我</button>

<h4>{**this**.props.myDataProp}</h4>

</div>

}

});

**var** HelloMessage = React.createClass({

getInitialState: **function**() {

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

},

handleChange: **function**(event) {

**this**.setState({value: '菜鸟教程'})

},

render: **function**() {

**var** value = **this**.state.value;

**return** <div>

<Content myDataProp = {value}

updateStateProp = {**this**.handleChange}></Content>

</div>;

}

});

ReactDOM.render(

<HelloMessage />,

document.getElementById('example')

);

# 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();

## 完整实例

你可以通过使用 this 来获取当前 React 组件，或使用 ref 来获取组件的引用，实例如下：

**React 实例**

**var** MyComponent = React.createClass({

handleClick: **function**() {

// 使用原生的 DOM API 获取焦点

**this**.refs.myInput.focus();

},

render: **function**() {

// 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}

/>

</div>

);

}

});

ReactDOM.render(

<MyComponent />,

document.getElementById('example')

);

实例中，我们获取了输入框的支撑实例的引用，子点击按钮后输入框获取焦点。

我们也可以使用 getDOMNode()方法获取DOM元素

1. DOM是Document Object Model的英文缩写，翻译过来是文档对象模型，提供给Javascript用来动态修改文档状态，就是将XML（或者HTML）内的节点定义成基本统一的对象数据可以供程序语言（如javaScript）控制的技术规范。 [↑](#footnote-ref-1)
2. 页面渲染都是浏览器的工作。大致分为以下几步：

   * 加载：根据请求的[URL](https://www.baidu.com/s?wd=URL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)进行域名解析，向服务器发起请求，接收文件（[HTML](https://www.baidu.com/s?wd=HTML&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)、[JS](https://www.baidu.com/s?wd=JS&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)、[CSS](https://www.baidu.com/s?wd=CSS&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)、图象等）。
   * 解析：对加载到的资源（[HTML](https://www.baidu.com/s?wd=HTML&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)、[JS](https://www.baidu.com/s?wd=JS&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)、[CSS](https://www.baidu.com/s?wd=CSS&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)等）进行语法解析，建议相应的内部数据结构（比如[HTML](https://www.baidu.com/s?wd=HTML&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)的[DOM](https://www.baidu.com/s?wd=DOM&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)树，[JS](https://www.baidu.com/s?wd=JS&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)的（对象）属性表，[CSS](https://www.baidu.com/s?wd=CSS&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznyFhPAu-m1fkuHRYuADv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWfsPH0LPj63PWc4nWR1PW6Y)的样式规则等等）
   * 渲染：构建渲染树，对各个元素进行位置计算、样式计算等等，然后根据渲染树对页面进行渲染（可以理解为“画”元素）
   * 这几个过程不是完全孤立的，会有交叉，比如HTML加载后就会进行解析，然后拉取HTML中指定的CSS、JS等。

   [↑](#footnote-ref-2)
3. npm 的包安装分为本地安装（local）、全局安装（global）两种，-g为全局安装。

   * 本地安装：

   1. 将安装包放在 ./node\_modules 下（运行 npm 命令时所在的目录），如果没有 node\_modules 目录，会在当前执行 npm 命令的目录下生成 node\_modules 目录。

   2. 可以通过 require() 来引入本地安装的包。

   * 全局安装：

   1. 将安装包放在 /usr/local 下或者你 node 的安装目录。

   2. 可以直接在命令行里使用。 [↑](#footnote-ref-3)
4. 内联式css样式表就是把css代码直接写在现有的HTML标签中。 [↑](#footnote-ref-4)