

# React 入门实例教程

---

作者： 阮一峰

日期： 2015年3月31日

现在最热门的前端框架，毫无疑问是 [React](#) 。

上周，基于 [React](#) 的 [React Native](#) 发布，结果一天之内，就获得了 5000 颗星，受瞩目程度可见一斑。

[React](#) 起源于 Facebook 的内部项目，因为该公司对市场上所有 [JavaScript MVC 框架](#)，都不满意，就决定自己写一套，用来架设 [Instagram](#) 的网站。做出来以后，发现这套东西很好用，就在2013年5月[开源](#)了。



由于 [React](#) 的设计思想极其独特，属于革命性创新，性能出众，代码逻辑却非常简单。所以，越来越多的人开始关注和使用，认为它可能是将来 [Web](#) 开发的主流工具。

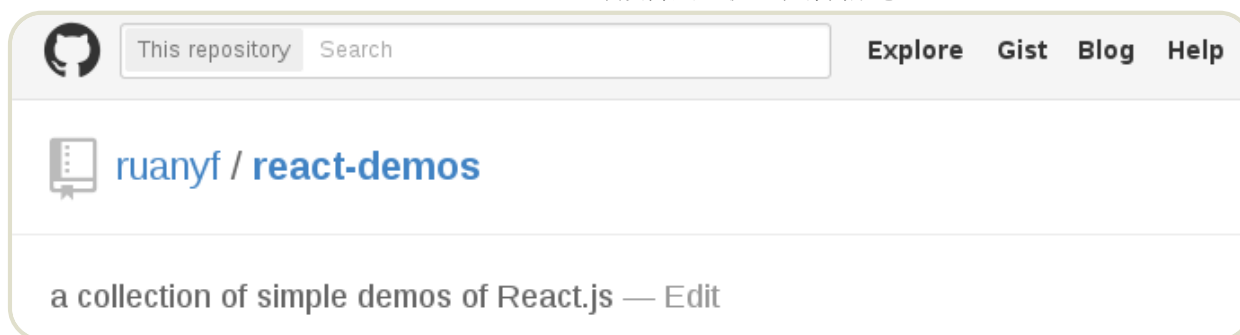
这个项目本身也越滚越大，从最早的UI引擎变成了一整套前后端通吃的 [Web App](#) 解决方案。衍生的 [React Native](#) 项目，目标更是宏伟，希望用写 [Web App](#) 的方式去写 [Native App](#)。如果能够实现，整个互联网行业都会被颠覆，因为同一组人只需要写一次 [UI](#)，就能同时运行在服务器、浏览器和手机（参见[《也许，DOM 不是答案》](#)）。



既然 **React** 这么热门，看上去充满希望，当然应该好好学一下。从技术角度，可以满足好奇心，提高技术水平；从职业角度，有利于求职和晋升，有利于参与潜力大的项目。但是，好的 **React** 教程却不容易找到，这一方面因为这项技术太新，刚刚开始走红，大家都没有经验，还在摸索之中；另一方面因为 **React** 本身还在不断变动，**API** 一直在调整，至今没发布**1.0**版。



我学习 **React** 时，就很苦恼。有的教程讨论一些细节问题，对入门没帮助；有的教程写得不错，但比较短，无助于看清全貌。我断断续续学了几个月，看过二十几篇教程，在这个过程中，将对自己有帮助的 **Demo** 都收集下来，做成了一个库 [React Demos](#)。



下面，我就根据[这个库](#)，写一篇全面又易懂的 **React** 入门教程。你只需要跟着每一个 **Demo** 做一遍，就能初步掌握 **React**。当然，前提是你必须拥有基本 **JavaScript** 和 **DOM** 知识，但是你读完就会发现，**React** 所要求的预备知识真的很少。

## 零、安装

**React** 的安装包，可以到[官网](#)下载。不过，[React Demos](#) 已经自带 **React** 源码，不用另外安装，只需把这个库拷贝到你的硬盘就行了。

```
$ git clone git@github.com:ruanyf/react-demos.git
```

如果你没安装 **git**，那就直接下载 [zip 压缩包](#)。

A screenshot of the commit history page for the 'ruanyf / react-demos' repository. The page shows a table of commits with columns for the commit message, the author, and the time since the commit. The table lists 12 commits, each corresponding to a demo or a change to the README or .gitignore file. The most recent commit is 'adding new demos' by 'ruanyf' 7 days ago.

edit README		
ruanyf	authored 2 days ago	latest commit b3c29f0807
build	adding new demos	7 days ago
demo01	rename directories	7 days ago
demo02	rename directories	7 days ago
demo03	rename directories	7 days ago
demo04	rename directories	7 days ago
demo05	rename directories	7 days ago
demo06	rename directories	7 days ago
demo07	rename directories	7 days ago
demo08	rename directories	7 days ago
demo09	rename directories	7 days ago
demo10	add new demos	7 days ago
demo11	edit README	6 days ago
.gitignore	first commit	8 days ago
README.md	edit README	2 days ago

下面要讲解的12个例子在各个 **Demo** 子目录，每个目录都有一个 **index.html** 文

件，在浏览器打开这个文件（大多数情况下双击即可），就能立刻看到效果。

需要说明的是，**React** 可以在浏览器运行，也可以在服务器运行，但是本教程只涉及浏览器。一方面是为了尽量保持简单，另一方面 **React** 的语法是一致的，服务器的用法与浏览器差别不大。**Demo13** 是服务器首屏渲染的例子，有兴趣的朋友可以自己去看源码。

## 一、HTML 模板

使用 **React** 的网页源码，结构大致如下。

```
<!DOCTYPE html>
<html>
  <head>
    <script src="../build/react.js"></script>
    <script src="../build/react-dom.js"></script>
    <script src="../build/browser.min.js"></script>
  </head>
  <body>
    <div id="example"></div>
    <script type="text/babel">
      // ** Our code goes here! **
    </script>
  </body>
</html>
```

上面代码有两个地方需要注意。首先，最后一个 `<script>` 标签的 `type` 属性为 `text/babel`。这是因为 **React** 独有的 **JSX** 语法，跟 **JavaScript** 不兼容。凡是使用 **JSX** 的地方，都要加上 `type="text/babel"`。

其次，上面代码一共用了三个库：`react.js`、`react-dom.js` 和 `Browser.js`，它们必须首先加载。其中，`react.js` 是 **React** 的核心库，`react-dom.js` 是提供与 **DOM** 相关的功能，`Browser.js` 的作用是将 **JSX** 语法转为 **JavaScript** 语法，这一步很消耗时间，实际上线的时候，应该将它放到服务器完成。

```
$ babel src --out-dir build
```

上面命令可以将 `src` 子目录的 `js` 文件进行语法转换，转码后的文件全部放在 `build` 子目录。

## 二、ReactDOM.render()

`ReactDOM.render` 是 `React` 的最基本方法，用于将模板转为 `HTML` 语言，并插入指定的 `DOM` 节点。

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

上面代码将一个 `h1` 标题，插入 `example` 节点（查看 [demo01](#)），运行结果如下。

Hello, world!

## 三、JSX 语法

上一节的代码，`HTML` 语言直接写在 `JavaScript` 语言之中，不加任何引号，这就是 `JSX` 的语法，它允许 `HTML` 与 `JavaScript` 的混写（查看 [Demo02](#)）。

```
var names = ['Alice', 'Emily', 'Kate'];  
  
ReactDOM.render(  
  <div>  
    {  
      names.map(function (name) {  
        return <div>Hello, {name}!</div>  
      })  
    }  
  </div>  
)
```

```
</div>,  
document.getElementById('example')  
);
```

上面代码体现了 **JSX** 的基本语法规则：遇到 **HTML** 标签（以 `<` 开头），就用 **HTML** 规则解析；遇到代码块（以 `{` 开头），就用 **JavaScript** 规则解析。上面代码的运行结果如下。

Hello, Alice!  
Hello, Emily!  
Hello, Kate!

**JSX** 允许直接在模板插入 **JavaScript** 变量。如果这个变量是一个数组，则会展开这个数组的所有成员（查看 [demo03](#)）。

```
var arr = [  
  <h1>Hello world!</h1>,  
  <h2>React is awesome</h2>,  
];  
ReactDOM.render(  
  <div>{arr}</div>,  
  document.getElementById('example')  
);
```

上面代码的 `arr` 变量是一个数组，结果 **JSX** 会把它的所有成员，添加到模板，运行结果如下。

# Hello world!

## React is awesome

## 四、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。React.createClass 方法就用于生成一个组件类（查看 [demo04](#)）。

```
var HelloMessage = React.createClass({
  render: function() {
    return <h1>Hello {this.props.name}</h1>;
  }
});

ReactDOM.render(
  <HelloMessage name="John" />,
  document.getElementById('example')
);
```

上面代码中，变量 `HelloMessage` 就是一个组件类。模板插入 `<HelloMessage />` 时，会自动生成 `HelloMessage` 的一个实例（下文的"组件"都指组件类的实例）。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。

```
var HelloMessage = React.createClass({
  render: function() {
    return <h1>
      Hello {this.props.name}
    </h1><p>
```

```
    some text
  </p>;
}
});
```

上面代码会报错，因为 `HelloMessage` 组件包含了两个顶层标签：`h1` 和 `p`。

组件的用法与原生的 `HTML` 标签完全一致，可以任意加入属性，比如

`<HelloMessage name="John">`，就是 `HelloMessage` 组件加入一个 `name` 属性，值为 `John`。组件的属性可以在组件类的 `this.props` 对象上获取，比如 `name` 属性就可以通过 `this.props.name` 读取。上面代码的运行结果如下。



添加组件属性，有一个地方需要注意，就是 `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。

## 五、this.props.children

`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（查看 [demo05](#)）。

```
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。

```
1. hello
2. world
```

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` 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](#)。

## 六、PropTypes

组件的属性可以接受任意值，字符串、对象、函数等等都可以。有时，我们需要一种机制，验证别人使用组件时，提供的参数是否符合要求。

组件类的 `PropTypes` 属性，就是用来验证组件实例的属性是否符合要求（查看[demo06](#)）。

```
var MyTitle = React.createClass({
  propTypes: {
    title: React.PropTypes.string.isRequired,
  },

  render: function() {
    return <h1> {this.props.title} </h1>;
  }
});
```

上面的 `Mytitle` 组件有一个 `title` 属性。`PropTypes` 告诉 `React`，这个 `title` 属性是必须的，而且它的值必须是字符串。现在，我们设置 `title` 属性的值是一个数值。

```
var data = 123;

ReactDOM.render(
  <MyTitle title={data} />,
  document.body
);
```

这样一来，`title` 属性就通不过验证了。控制台会显示一行错误信息。

```
Warning: Failed propType: Invalid prop `title` of type `number` sup
```

更多的 `PropTypes` 设置，可以查看[官方文档](#)。

此外，`getDefaultProps` 方法可以用来设置组件属性的默认值。

```
var MyTitle = React.createClass({
  getDefaultProps : function () {
    return {
      title : 'Hello World'
    };
  },

  render: function() {
    return <h1> {this.props.title} </h1>;
  }
});
```

```
});  
  
ReactDOM.render(  
  <MyTitle />,  
  document.body  
);
```

上面代码会输出"Hello World"。

## 七、获取真实的DOM节点

组件并不是真实的 **DOM** 节点，而是存在于内存之中的一种数据结构，叫做虚拟 **DOM**（**virtual DOM**）。只有当它插入文档以后，才会变成真实的 **DOM**。根据 **React** 的设计，所有的 **DOM** 变动，都先在虚拟 **DOM** 上发生，然后再将实际发生变动的部分，反映在真实 **DOM** 上，这种算法叫做 **DOM diff**，它可以极大提高网页的性能表现。

但是，有时需要从组件获取真实 **DOM** 的节点，这时就要用到 **ref** 属性（查看 [demo07](#)）。

```
var MyComponent = React.createClass({  
  handleClick: function() {  
    this.refs.myTextInput.focus();  
  },  
  render: function() {  
    return (  
      <div>  
        <input type="text" ref="myTextInput" />  
        <input type="button" value="Focus the text input" onClick={this.handleClick} />  
      </div>  
    );  
  }  
});  
  
ReactDOM.render(  
  <MyComponent />,  
  document.getElementById('example')  
);
```

上面代码中，组件 **MyComponent** 的子节点有一个文本输入框，用于获取用户的输入。这时就必须获取真实的 **DOM** 节点，虚拟 **DOM** 是拿不到用户输入的。为了做到这

一点，文本输入框必须有一个 `ref` 属性，然后 `this.refs.[refName]` 就会返回这个真实的 DOM 节点。

需要注意的是，由于 `this.refs.[refName]` 属性获取的是真实 DOM，所以必须等到虚拟 DOM 插入文档以后，才能使用这个属性，否则会报错。上面代码中，通过为组件指定 `click` 事件的回调函数，确保了只有等到真实 DOM 发生 `click` 事件之后，才会读取 `this.refs.[refName]` 属性。

React 组件支持很多事件，除了 `click` 事件以外，还有 `keyDown`、`copy`、`scroll` 等，完整的事件清单请查看[官方文档](#)。

## 八、this.state

组件免不了要与用户互动，React 的一大创新，就是将组件看成是一个状态机，一开始有一个初始状态，然后用户互动，导致状态变化，从而触发重新渲染 UI（查看 [demo08](#)）。

```
var LikeButton = React.createClass({
  getInitialState: function() {
    return {liked: false};
  },
  handleClick: function(event) {
    this.setState({liked: !this.state.liked});
  },
  render: function() {
    var text = this.state.liked ? 'like' : 'haven\'t liked';
    return (
      <p onClick={this.handleClick}>
        You {text} this. Click to toggle.
      </p>
    );
  }
});

ReactDOM.render(
  <LikeButton />,
  document.getElementById('example')
);
```

上面代码是一个 `LikeButton` 组件，它的 `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。

由于 `this.props` 和 `this.state` 都用于描述组件的特性，可能会产生混淆。一个简单的区分方法是，`this.props` 表示那些一旦定义，就不再改变的特性，而 `this.state` 是会随着用户互动而产生变化的特性。

## 九、表单

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](#)）。

```
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，更多介绍请参考[官方文档](#)。

## 十、组件的生命周期

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

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

这些方法的详细说明，可以参考[官方文档](#)。下面是一个例子（查看 [demo10](#) ）。

```
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
);
```

上面代码在 `hello` 组件加载以后，通过 `componentDidMount` 方法设置一个定时器，每隔100毫秒，就重新设置组件的透明度，从而引发重新渲染。

另外，组件的 `style` 属性的设置方式也值得注意，不能写成

```
style="opacity:{this.state.opacity};"
```

而要写成

```
style={{opacity: this.state.opacity}}
```

这是因为 React 组件样式 是一个对象，所以第一重大括号表示这是 JavaScript 语法，第二重大括号表示样式对象。

## 十一、Ajax

组件的数据来源，通常是通过 Ajax 请求从服务器获取，可以使用 `componentDidMount` 方法设置 Ajax 请求，等到请求成功，再用

`this.setState` 方法重新渲染 UI（查看 [demo11](#)）。

```
var UserGist = React.createClass({
  getInitialState: function() {
    return {
      username: '',
      lastGistUrl: ''
    };
  },

  componentDidMount: function() {
    $.get(this.props.source, function(result) {
      var lastGist = result[0];
      if (this.isMounted()) {
        this.setState({
          username: lastGist.owner.login,
          lastGistUrl: lastGist.html_url
        });
      }
    }.bind(this));
  },

  render: function() {
    return (
      <div>
        {this.state.username}'s last gist is
        <a href={this.state.lastGistUrl}>here</a>.
      </div>
    );
  }
});

ReactDOM.render(
  <UserGist source="https://api.github.com/users/octocat/gists" />,
  document.body
);
```

上面代码使用 jQuery 完成 Ajax 请求，这是为了便于说明。React 本身没有任何依赖，完全可以不用 jQuery，而使用其他库。

我们甚至可以把一个 Promise 对象传入组件，请看 [Demo12](#)。

```
ReactDOM.render(
```



```

<RepoList
  promise={$.getJSON('https://api.github.com/search/repositories?')}
/>,
document.body
);

```

上面代码从Github的API抓取数据，然后将Promise对象作为属性，传给 RepoList 组件。

如果Promise对象正在抓取数据（pending状态），组件显示"正在加载"；如果Promise对象报错（rejected状态），组件显示报错信息；如果Promise对象抓取数据成功（fulfilled状态），组件显示获取的数据。

```

var RepoList = React.createClass({
  getInitialState: function() {
    return { loading: true, error: null, data: null };
  },

  componentDidMount() {
    this.props.promise.then(
      value => this.setState({loading: false, data: value}),
      error => this.setState({loading: false, error: error})
    );
  },

  render: function() {
    if (this.state.loading) {
      return <span>Loading...</span>;
    }
    else if (this.state.error !== null) {
      return <span>Error: {this.state.error.message}</span>;
    }
    else {
      var repos = this.state.data.items;
      var repoList = repos.map(function (repo) {
        return (
          <li>
            <a href={repo.html_url}>{repo.name}</a> ({repo.stargazer
          </li>
        );
      });
      return (
        <main>
          <h1>Most Popular JavaScript Projects in Github</h1>

```





```
        <ol>{repoList}</ol>
      </main>
    );
  }
}
});
```

## 十二、参考链接

1. [React's official site](#)
2. [React's official examples](#)
3. [React \(Virtual\) DOM Terminology](#), by Sebastian Markbåge
4. [The React Quick Start Guide](#), by Jack Callister
5. [Learning React.js: Getting Started and Concepts](#), by Ken Wheeler
6. [Getting started with React](#), by Ryan Clark
7. [React JS Tutorial and Guide to the Gotchas](#), by Justin Deal
8. [React Primer](#), by Binary Muse
9. [jQuery versus React.js thinking](#), by zigomir

(完)

### 文档信息

- 版权声明： 自由转载-非商用-非衍生-保持署名（[创意共享3.0许可证](#)）
- 发表日期： 2015年3月31日
- 更多内容： [档案](#) » [JavaScript](#)
- 购买文集：  《如何变得有思想》
- 社交媒体：  twitter,  weibo
- Feed订阅： 

## 相关文章

▪ **2016.04.12:** [跨域资源共享 CORS 详解](#)

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing）。

▪ **2016.04.08:** [浏览器同源政策及其规避方法](#)

浏览器安全的基石是"同源政策"（same-origin policy）。很多开发者都知道这一点，但了解得不全面。

▪ **2016.03.12:** [Node 应用的 Systemd 启动](#)

前面的文章介绍了 Systemd 的操作命令和基本用法，今天给出一个实例，如何使用 Systemd 启动一个 Node 应用。

▪ **2016.02.13:** [React 测试入门教程](#)

越来越多的人，使用React开发Web应用。它的测试就成了一个大问题。

---

联系方式 | [ruanyifeng.com](http://ruanyifeng.com) 2003 - 2016