REACT

- REACT简介
- REACT特点
- JSX简介
- REACT Hello World
- REACT元素渲染
- REACT事件处理
- REACT条件渲染
- 学习网址

REACT简介

React 是一个声明式，高效且灵活的用于构建用户界面的 JavaScript 库。使用 React 可以将一些简短、独立的代码片段组合成复杂的 UI 界面，这些代码片段被称作“组件”。

REACT起源于 Facebook 的内部项目，用来架设Instagram 的网站，并在2013年5月开源。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。你可以在React里传递多种类型的参数，如声明代码，帮助你渲染出UI、也可以是静态的HTML DOM元素、也可以传递动态变量、甚至是可交互的应用组件

REACT特点

- 声明式设计 -React采用声明范式，可以轻松描述应用。
- 高效 -React通过对DOM的模拟，最大限度地减少与DOM的交互。
- 灵活 -React可以与已知的库或框架很好地配合。
- 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。
- 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。
- 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。

JSX简介

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

- 这个标签语法既不是字符串也不是 HTML。
- 它被称为 JSX，是一个 JavaScript 的语法扩展。建议在 React 中配合使用 JSX，JSX 可以很好地描述 UI 应该呈现出它应有交互的本质形式。JSX 可能会使人联想到模版语言，但它具有 JavaScript 的全部功能。JSX 可以生成 React “元素”。

JSX简介

在下面的例子中，声明了一个名为 name 的变量，然后在 JSX 中使用它，并将它包裹在大括号中：

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

在 JSX 语法中，你可以在大括号内放置任何有效的 JavaScript 表达式。例如 2 + 2，user.firstName 或 formatName(user) 都是有效的 JavaScript 表达式。

JSX简介

在下面的示例中，调用 了JavaScript 函数 formatName(user) 的结果，并将结果嵌入到 <h1> 元素中。

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

REACT Hello World

最简易的REACT示例如下：

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

它将在页面上展示一个“Hello,world!”的标题

REACT元素渲染

- 元素：
  - 元素是构成 React 应用的最小砖块。元素描述了你在屏幕上想看到的内容。
  - 与浏览器的 DOM 元素不同，React 元素是创建开销极小的普通对象。React DOM 会负责更新 DOM 来与 React 元素保持一致。

```
const element = <h1>Hello, world</h1>;
```

REACT元素渲染

- 将一个元素渲染为DOM：
  - 假设你的 HTML 文件某处有一个 ：

```
<div id="root"></div>
```

  - 我们将其称为“根” DOM 节点，因为该节点内的所有内容都将由 React DOM 管理
  - 仅使用 React 构建的应用通常只有单一的根 DOM 节点。如果你在将 React 集成进一个已有应用，那么你可以在应用中包含任意多的独立根 DOM 节点。
  - 想要将一个 React 元素渲染到根 DOM 节点中，只需把它们一起传入 ReactDOM.render()：

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

REACT元素渲染

- 更新已渲染的元素：
  - React 元素是不可变对象。一旦被创建，你就无法更改它的子元素或者属性。一个元素就像电影的单帧：它代表了某个特定时刻的 UI。更新 UI 唯一的方式是创建一个全新的元素，并将其传入 ReactDOM.render()。
  - 这个计时器例子会在 setInterval() 回调函数，每秒都调用 ReactDOM.render()

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));
}

setInterval(tick, 1000);
```

REACT组件

- 组件：组件允许你将 UI 拆分为独立可复用的代码片段，并对每个片段进行独立构思。组件，从概念上类似于 JavaScript 函数。它接受任意的入参（即 “props”），并返回用于描述页面展示内容的 React 元素。

- 注意： 组件名称必须以大写字母开头。
- React 会将以小写字母开头的组件视为原生 DOM 标签。
- 例如， 代表 HTML 的 div 标签，而 <Welcome /> 则代表一个组件，并且需在作用域内使用 Welcome。

REACT组件

- 函数组件：定义组件最简单的方式就是编写 JavaScript 函数：

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- class组件：还可以使用 ES6 的 class 来定义组件：

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

- 上述两个组件在REACT中等效。

REACT组件

- 将函数组件转换成class组件：
  1. 创建一个同名的 ES6 class，并且继承于 React.Component。
  2. 添加一个空的 render() 方法。
  3. 将函数体移动到 render() 方法之中。
  4. 在 render() 方法中使用 this.props 替换 props。
  5. 删除剩余的空函数声明。

```
class Clock extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.props.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

REACT组件

- 渲染组件：
- React 元素也可以是用户自定义的组件，当 React 元素为用户自定义组件时，它会将 JSX 所接收的属性（attributes）以及子组件（children）转换为单个对象传递给组件，这个对象被称之为 “props”。
- 例如，这段代码会在页面上渲染 “Hello, Sara”：

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

REACT事件处理

- React 元素的事件处理和 DOM 元素的很相似，但是有一点语法上的不同：
  - React 事件的命名采用小驼峰式（camelCase），而不是纯小写。
  - 使用 JSX 语法时你需要传入一个函数作为事件处理函数，而不是一个字符串。

- 例如，传统的 HTML：

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

- 在REACT中略微不同：

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

REACT事件处理

- 在 React 中另一个不同点是你不能通过返回 false 的方式阻止默认行为。你必须显式的使用 preventDefault 。

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

- 在这里，e 是一个合成事件。React 根据 W3C 规范来定义这些合成事件，所以你不需要担心跨浏览器的兼容性问题。

REACT事件处理

- 向事件处理程序传递参数：
- 在循环中，通常我们会为事件处理函数传递额外的参数。例如，若 id 是你要删除那一行的 ID，以下两种方式都可以向事件处理函数传递参数：

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

- 上述两种方式是等价的，分别通过箭头函数和 Function.prototype.bind 来实现
- 在这两种情况下，React 的事件对象 e 会被作为第二个参数传递。如果通过箭头函数的方式，事件对象必须显式的进行传递，而通过 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。

REACT条件渲染

- React 中的条件渲染和 JavaScript 中的一样，使用 JavaScript 运算符 if 或者条件运算符去创建元素来表现当前的状态，然后让 React 根据它们来更新 UI。

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

REACT列表

- 渲染多个组件：可以通过使用 {} 在 JSX 内构建一个元素集合。
- 下面，我们使用 Javascript 中的 map() 方法来遍历 numbers 数组。将数组中的每个元素变成 <li> 标签，最后我们将得到的数组赋值给 listItems：

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

- 我们把整个 listItems 插入到 <ul> 元素中，然后渲染进 DOM：

```
ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

REACT列表

- 基础列表组件：通常需要在一个组件中渲染列表。
- 我们可以把前面的例子重构成一个组件，这个组件接收 numbers 数组作为参数并输出一个元素列表。

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

REACT列表

- key：key 帮助 React 识别哪些元素改变了，比如被添加或删除。因此我们应当给数组中的每一个元素赋予一个确定的标识。
- 一个元素的 key 最好是这个元素在列表中拥有的一个独一无二的字符串。通常，我们使用数据中的 id 来作为元素的 key。当元素没有确定 id 的时候，万不得已可以使用元素索引 index 作为 key。
- 如果列表项目的顺序可能会变化，我们不建议使用索引来用作 key 值，因为这样做会导致性能变差，还可能引起组件状态的问题。如果你选择不指定显式的 key 值，那么 React 将默认使用索引用作为列表项目的 key 值。

REACT列表

- 用key提取组件：

```
function ListItem(props) {
  // 正确！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) =>
    // 正确！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('root')
);
```

学习网址

- https://reactjs.org/
- https://react.docschina.org/
- http://caibaojian.com/react/
- http://react-china.org/
- https://www.runoob.com/react/react-tutorial.html