39 | 语法扩展：通过JSX来做语法扩展

2022-12-17 石川 来自北京

《JavaScript进阶实战课》

课程介绍 >

讲述：正霖

时长 00:40　大小 637.33K

你好，我是石川。

在前面一讲中，我们提到了 React 也有一个 JavaScript 的语法扩展，叫做 JSX。它的全称是 JavaScript XML，顾名思义，它是以 XML 为格式设计的。它最主要的目的是与 React 框架结合，用于应用中 Web UI 相关的开发。在 React 中，DOM 树是通过 JSX 来定义的，最终会在浏览器中渲染成 HTML。基于 React 的普及率，即使你没有用过 React，估计对 JSX 也有所耳闻。

今天，我们就来看看 JSX 是如何用在 Web UI 开发中的。即使你不使用 React，这样的模版模式也有很大的借鉴意义。

类 HTML 的元素

首先，我们来看下 JSX 的基本使用，它是如何创建元素、相关的属性和子元素的。

你可以把 JSX 元素当成一种新的 JavaScript 语法表达。我们知道在标准的 JavaScript 中，字符串的字面量是通过双引号来分隔的，正则表达式的字面量是通过斜线号来分隔的。类似的，JSX 的字面量则是通过**角括号**来分隔的。比如下面的 <h1 /> 标签就是一个简单的例子，通过这种方式，我们就创建了一个 React 的标题元素。

复制代码

```
var title = <h1 className="title">页面标题</h1>;
```

但是像上面这样的标签，我们知道 JS 引擎是没办法理解的，那它如何被运行呢？这时，我们可以通过我们在上一讲提到的 Babel 把 JSX 的表达**先转译成标准的 JavaScript，然后才会给到浏览器做编译和执行**。Babel 会把上面变量声明赋值中的 JSX 表达，转换成一个符合 JavaScript 规范的 createElement() 函数，来进行函数的调用。

复制代码

```
var title = React.createElement("h1", {className: 'title'}, "页面标题");
```

因为通过 JSX 来创建的 React 元素和 HTML 的标签元素类似，所以 JSX 生成的 React 元素表达也可以带有属性。当一个元素有一个或多个属性的时候，会作为 createElement() 的第二个参数传入，参数的值是一个带有元素相关属性的对象。

复制代码

```
// 转译前
var image = <img src="logo.png" alt="company logo" />;

// 转译后
var image = React.createElement("img", {
              src: "logo.png",
              alt: "company logo"
            });
```

像 HTML 的元素一样，除了字符串，React 的元素中也可以有子元素。React 元素也可以通过层级的嵌套，来创建 DOM 树。在转译的过程中，上面的这些 JSX 嵌套的 DOM 子元素，将作为 createElement() 调用的第三个参数及以后的参数来传递。

复制代码

```
// 转译前
```

```jsx
var sidebar = (
  <div className="sidebar">
    <h2 className="menu">菜单</h2>
    <p className="text">菜单内容</p>
  </div>
);

// 转译后
var sidebar = React.createElement(
    "div", { className: "sidebar"},
    React.createElement("h1", className: "menu"},
                        "菜单"),
    React.createElement("p", className: "text"},
                        "菜单内容"));
```


React 中的 createElement() 函数所返回的值是一个 JavaScript 对象，React 使用它在浏览器窗口中输出渲染的 UI。在下面的例子中，我们可以看到一个通过 JSX 表达的 React 元素，在转译后，通过 createElement() 生成为对象。之后，再将对象通过 React 中的 render() 渲染到页面上。


```jsx
// 转译前
var element = <h1 class"title">页面标题</h1>;

// 转译后
var element = React.createElement(
  'h1',
  {className: 'title'},
  '页面标题'
);

// createElement 返回的对象
var element = {
  type: 'h1',
  props: {
    className: 'title',
    children: '页面标题'
  }
};

// createElement 返回对象的渲染
var root = ReactDOM.createRoot(
  document.getElementById('root')
);
root.render(element);
```

因为今天我们主要讲的是 JSX 的语法扩展，而不是 React 本身，所以我们就不对返回的 element 对象和渲染过程的细节做赘述了。不过，有一点值得注意的是，你可以用 Babel 配置对 React 中 createElement() 以外的创建元素的函数做转译。这样做，就可以在 React 以外的地方来使用类似的 JSX 表达了。

JS 和 CSS 的表达

上面，我们学习了 JSX 语法扩展的核心概念，下面，我们再来看看它是如何与 JavaScript 以及 CSS 的表达来结合使用的。

React 元素的一个重要特性是可以在 JSX 表达式中嵌入标准的 JavaScript 表达式。你可以使用**大括号**来嵌入标准的 JavaScript 表达式。大括号内的脚本，会在转译的环节被解释为标准的 JavaScript。React 元素中的嵌套表达式可以作为属性值和子元素。下面就是一个例子：


```
function article(className, title, content, linebreak=true) {
  return (
    <div className={className}>
      <h1>{title}</h1>
      { linebreak && <br /> }
      <p>{content}</p>
    </div>
  );
}

function article(className, title, content, subtitle=true) {
  return React.createElement("div", { className: className },
                             React.createElement("h1", null, title),
                             subtitle && React.createElement("br", null),
                             React.createElement("p", null, content));
}
```

在这个例子中，article() 函数的作用是返回一个 JSX 元素，它有四个参数。Babel 会将例子中的代码翻译为以下内容。

这段代码很容易阅读和理解：转译后，大括号消失了，生成的代码将 article() 函数传入的参数传递给 React.createElement()。请注意我们在这里对 linebreak 参数和 && 运算符的使用。在实际的调用中，如果只有三个实参传入 article()，则
 默认为 true，外部 createElement() 调用的第四个参数是
 元素。但是，如果我们将 false 作为第四个参数传递给 article()，那么外部 createElement() 调用的第四个自变量的值将为 false，就不会创建
 元素。使

用 && 运算符是 JSX 中常见的习惯用法，它可以根据其它表达式的值有条件地包含或排除子元素。这个习惯用法适用于 React，因为 React 会忽略 false 或 null 的子级，而不会为它们生成任何输出。

在 JSX 表达式中使用 JavaScript 表达式的时候，不限于前面例子中如字符串和布尔的基础类型值，也可以是对象类型的 JavaScript 值。实际上，使用对象、数组和函数值在 React 中很常见。例如在下面的函数中，JavaScript 和 CSS 的表达如下：

复制代码

```
function list(items, callback) {
  return (
    <ul style={ {padding:10, border:"solid red 4px"} }>
      {items.map((item,index) => {
        <li onClick={() => callback(index)} key={index}>{item}</li>
      })}
    </ul>
  );
}
```

这里，函数使用对象字面量作为 <ul> 元素上 CSS 样式（style）的属性的值。注意，这里需要用到的是双大括号。<ul>元素有一个子元素，但该子元素的值是一个数组。输出的数组是通过在输入的数组上使用 map() 的映射方法来创建 <li> 子元素的。并且在这里，每个嵌套的 <li> 子元素都有一个 onClick 事件处理程序属性，其值是一个箭头函数。

如果我们将上面的 JSX 代码编译为标准的 JavaScript 代码，则会是以下的形式：

复制代码

```
function list(items, callback) {
  return React.createElement(
    "ul",
    { style: { padding: 5, border: "dotted green 3px" } },
    items.map((item, index) =>
      React.createElement(
        "li",
        { onClick: () => callback(index), key: index },
        item
      )
    )
  );
}
```

React 元素类型

另外，JSX 还有一个更重要的特性，就是定义 React 元素的类型。

所有 JSX 元素都以一个**标识符**开头，紧跟在**开口角括号**之后。如果该标识符的第一个字母是小写的，那么该标识符将作为字符串传递给 createElement()。但是，如果标识符的第一个字母是大写的，那么它将被视为实际标识符，并且该标识符的 JavaScript 值将作为第一个参数传递给 createElement()。

这意味着 JSX 表达式 <CustomButton/> 编译为 JavaScript 代码，将全局 CustomButton 对象传递给 React.createElement()。对于 React 来说，这种将**非字符串值**作为第一个参数传递给 createElement() 的能力可以用来创建组件。

在 React 中定义新组件的最简单方法是编写一个函数，该函数将 props 对象作为参数并返回 JSX 表达式。props 对象只是一个表示属性值的 JavaScript 对象，就像作为第二个参数传递给 createElement() 的对象一样。例如，这里是我们的 article() 函数的另外一种写法：

复制代码

```
function Article(props) {
  return (
    <div>
      <h1>{props.title}</h1>
      { props.linebreak && <br /> }
      <p>{props.content}</p>
    </div>
  );
}

function article(className, title, content, linebreak=true) {
  return (
    <div className={className}>
      <h1>{title}</h1>
      { linebreak && <br /> }
      <p>{content}</p>
    </div>
  );
}
```

这个新的 Article() 函数很像以前的 article() 函数。但它的名称以大写字母开头，并且只有一个对象作为参数。这可以使得它成为一个 React 的组件，这就意味着它可以用来代替 JSX 表达式中标准的 HTML 标记：

```
var article = <Article title="文章标题" content="文章内容"/>;
```


这个 ＜Article/＞ 元素在转译后如下：

```
var article = React.createElement(Article, {
  title: "文章标题",
  content: "文章内容"
});
```

这是一个简单的 JSX 表达式，但当 React 渲染它时，它会将第二个参数，也就是 props 对象，传递给第一个参数，也就是 Article() 函数，并将使用该函数返回的 JSX 函数代替 <Article> 表达式。

另外，如果我们的一个模块中有多个组件的话，也可以用 **dot notation** 的方式，来表达模块中的组件。

```
import React from 'react';
var MyComponents = {
  Calendar: function Calendar(props) {
    return <div>一个{props.color}颜色的日历.</div>;
  }
}
function GreenCalendar() {
  return <MyComponents.DatePicker color="绿" />;
}

```

JSX 中对象表达式的另一个用途是使用对象扩展运算符一次指定多个属性。如果你编写了一组带有可能被重用的公共属性的 JSX 表达式，则可以通过将属性抽象定义为一个属性对象，并将其“扩展”到 JSX 元素中，来简化表达式。

```
var greeting = <div firstName="三" lastName="张" />;

var props = {firstName: '三', lastName: '张'};
```

```
4 var greeting = <div className = "greeting" {...props} />;
```

Babel 会将其编译为一个使用 _extends() 的函数，该函数将 className 属性与 props 对象中包含的属性相结合。

复制代码

```
1 var greeting = React.createElement("div",
2                                    _extends({className: "greeting"}, props)
3                                   );
```

总结

今天，我们看到了在 JavaScript 中，如何通过 JavaScript 的扩展 JSX，在 JavaScript 中表达 DOM 元素。这样做有几点好处：

1. **它可以更好地赋能以 UI 和事件驱动的开发**，并且通过我们在上节讲到的 Babel 编译器，将相关的表达转译成 createElement() 的函数表达。再利用 createElement() 的函数调用，创建并返回包含相关元素属性定义的对象。最后再通过 render() 的方式将元素对象在浏览器中做渲染。同时，它可以和 JavaScript 和 CSS 的表达交叉使用，我们可以在 JSX 的表达中参入 JS 和 CSS 的表达。并且，除了基础类型的值，我们也可以使用数组、函数等对象的数据类型。
2. 通过对 JSX 的使用，也**有助于模块化和组件化的开发**。
3. 最后，从安全性上来看，JSX 的好处是所有的内容在渲染前都被转换成了字符串，这样也可以有效地防止我们在安全的一讲中提到的跨站脚本攻击（XSS），让我们的应用更安全。

思考题

最后，留给你一道思考题，其实在 JSX 之前，基于 JavaScript 就有很多的模版引擎，例如 jQuery Template 和 Mustache.js 等。你觉得 JSX 和传统的 JavaScript 模版引擎有什么区别吗？各自的优劣势是怎样的？

欢迎在留言区分享你的经验、交流学习心得或者提出问题，如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。我们下节课再见！

上一篇　38丨编译和打包：通过Webpack、Babel做编译和打包

精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。