react 基础教学

jsx 知识介绍

我们可以直接引入 react.js 和 react-dom.js 两个库，使用 react 时，我们可以使用 jsx 语法来书写我们的模版

```
<div className="foo">bar</div>
```

这是一种非常类似 DOM 的 xml 结构的语法，唯一有区别的是，我们可以在 jsx 中使用 `{}` 来使用 js 表达式。

```
<div className="foo">{something ? "something is true" : "something is false"}</div>
```

这里的表达式当然也可以执行函数，数组 map 等等形式，只需要这个式子是一个表达式即可，但这里要注意千万不能直接在 jsx 中写执行诸如 if else 的形式，可以把它改造成一个三目运算符。

```
<div className="foo">{if(xx) {xx} else {xx}}</div> // 错误
```

在 jsx 中我们需要保证渲染的内容必须是合法的 jsx 元素，合法的 jsx 元素有：

- 普通的 DOM 标签，如 div/p/span 等等
- 声明的 react 组件，例如通过 class 或函数创建的 jsx 组件
- null（最终会渲染一个空元素）
- 字符串（最终会渲染一个 text 节点）
- 对于数字类型，最终就会渲染出来，所以有的时候通过布尔表达式判断的时候就会有问题

```
{false && (<p>this is false</p>)}  // 不会渲染内容
{0 && (<p>this is false</p>)}      // 会渲染 0
```

我们可以通过 `React.isValidElement` 来判断一个内容元素是不是一个合法的 react element。

同时需要注意的是，因为 class / for 这类的 html 属性是关键字，所以在 jsx 中我们想要使用，就必须使用 className/htmlFor 的形式来定义。

```
<label className="foo"
htmlFor="name">label</label>
```

事实上我们并不能直接在浏览器中使用 jsx 内容，我们需要搭配一些编译库将 jsx 语法进行编译。比较知名的就是 babel，搭配 babel-plugin-transform-react-jsx 插件，可以将 jsx 编译为 react 的内部方法。

例如这个例子，经过 babel 和配套插件就可以将这种形式进行编译：

```
<div>
  <h3 className="h3">{something ? "something is true" : "something is false"}</h3>
</div>
```

最终结果即为：

```
React.createElement(
  "div",
  null,
  React.createElement(
    "h3",
    {className: 'h3'},
    something ? "something is true" : "something is false"
  )
);
```

React.createElement 主要分为三类参数，第一个是组件的名字，第二个参数是当前组件接受的属性，第三个之后的参数都是当前组件嵌套的子组件。

jsx 总结

- jsx 是一种语法糖，我们需要将他们编译为

`React.createElement` 的形式

- 写 jsx 需要注意类型必须合法，尤其是写布尔表达式的时候需要额外注意，尽量使用三目运算符来书写 jsx
- 需要注意 class 和 for 标签在书写时需要改为 `className` 和 `htmlFor`

create-react-app cli 的使用

create-react-app 是 react 官方维护的一个 cli 工具，里面封装了 webpack babel 等基本的工程化工具，让我们能快速上手和使用。

当然，我们也可以自己封装 webpack 配置来进行使用，封装一个可以编译 react 应用的脚手架非常简单，只需要配置所有 react 文件使用 babel + babel 转译 jsx 插件即可，最终编译的 js 内容即可直接应用于页面中。

有时候我们可以自己封装一些符合公司内部前端架构的 cli 应用。例如我们可以封装一个使用 ts 书写 react 应用的脚手架来为我们自己所用。

函数组件和 class 组件/受控组件和非受控组件

在 react 中，我们可以使用 class 形式或是函数的形式来进行创建一个组件，例如以下两种形式：

```
function Foo(props) { return (<div>{props.text
|| 'Foo'}</div>); }

class Bar extends React.Component {
  render() {
    return (
      <div>{this.props.text || 'Bar'}</div>
    );
  }
}
```

这两种形式有何区别呢？我们简单对比一下：

- 加载的 props 方式不同，函数式定义组件从组件函数的参数加载。class 形式的组件通过 this.props 获取传入的参数
- 函数式组件比较简单，内部无法维护状态。class 形式内部可以通过 this.state 和 this.setState 方法更新内部 state 和更新内部 state，同时更新 render 里面的函数渲染的结果。
- class 组件内部可以定义更多的方法在实例上，但是函数式组件无法定义。

事实上，函数式组件和 class 组件之间的区别也仅停留在部分组件确实不需要维护内部状态。class 组件定义稍微复杂一些，但是内部可以维护更多的方法和状态。

组件生命周期

在 class 形式的组件中，我们可以定义以下声明周期的方法，会在特定的时机执行它。（老版本）

`componentWillMount` -> `render` -> `componentDidMount`

`componentWillReceiveProps` -> `componentWillUpdate` -> `render` -> `componentDidUpdate`

这里需要注意的就是，同 vue 的声明周期一样，我们最好在 componentDidMount 中发送请求，这样整个组件对服务端渲染会比较友好。

常见错误和性能问题

异步过程使用单例的 event 对象

全局单例的 event 对象，所以在异步对象中使用 react 事件时需要额外注意。异步操作最好将对象内部需要的值先进行拷贝赋值。

```
  handleClick(e) {
    setTimeout(function() {
      console.log('button1 click',
e.currentTarget.innerText); // 错误
    }, 1000);
    console.log('button1 click',
e.currentTarget.innerText);
  }
```

组件生命周期，它描述了整个组件在创建、实例化、销毁过程中不同过程中执行的方法。我们需要特别注意，不要在 render 中定义单独引用的内容。也就是不要在 render 中使用箭头函数，否则很容易运行时造成子组件的重新渲染。

为了保证这种引用的相等，我们都会使用 immutable 的不可变数据，来保证组件间传递的数据引用相等。

性能优化方式

- 使用 react dev tools ， 检测组件是否出现不必要的重新渲染。
- why-did-you-render

  why did you render 是一个能检测你的页面中的元素是否出现了不必要的重渲染。https://www.npmjs.com/package/@welldone-software/why-did-you-render

我们可以将它应用于我们的项目中，来检测是否有无意义的渲染的情况。

- class 中提前声明箭头函数，保证 render 执行过程中的函数不会因为引用问题导致重新渲染。

介绍 immutable 库 immutable-js 和 immer

为了配合 shouldComponentUpdate 来进行性能优化，大部分时候我们需要复杂的层级判断，这里我们介绍两个配合 react 最小更新的 immutable 库 immutable-js 和 immer。

immutable-js 是 facebook 的工程师在 2014 年推出的，immer 则是 mobx 作者 2018 年推出的。他们的推出其实是为了实现不可变数据，但实际上这种做法更多的是为了优化我们的 react 应用而做的。

我们可以简单看一下 immutable-js 的用法，着重讲一下 immer 的原理，便于大家更好的理解数据传递过程中的不可变性。