React基础 - JSX语法

王红元 coderwhy

目录 content



- 人 1 认识JSX语法
- 2 JSX的基本使用
- JSX的事件绑定
- 4 JSX的条件渲染
- JSX的列表渲染
- JSX的原理和本质



```
·//-1. 定义根组件
·const·element·=·<div>Hello·World</div>
·//-2. 渲染根组件
·const·root·=·ReactDOM.createRoot(document.querySelector("#root"))
·root.render(element)
```

■ 这段element变量的声明右侧赋值的标签语法是什么呢?

- □ 它不是一段字符串(因为没有使用引号包裹);
- □ 它看起来是一段HTML元素,但是我们能在js中直接给一个变量赋值html吗?
- □ 其实是不可以的,如果我们将 type="text/babel" 去除掉,那么就会出现语法错误;
- □ 它到底是什么呢? 其实它是一段jsx的语法;

■ JSX是什么?

- □ JSX是一种JavaScript的语法扩展 (eXtension) ,也在很多地方称之为JavaScript XML,因为看起就是一段XML语法;
- □ 它用于描述我们的UI界面,并且其完成可以和JavaScript融合在一起使用;
- □ 它不同于Vue中的模块语法,你不需要专门学习模块语法中的一些指令(比如v-for、v-if、v-else、v-bind);



为什么React选择了JSX

- React认为渲染逻辑本质上与其他UI逻辑存在内在耦合
 - □ 比如UI需要绑定事件 (button、a原生等等);
 - □ 比如UI中需要展示数据状态;
 - □ 比如在某些状态发生改变时,又需要改变UI;
- 他们之间是密不可分,所以React没有将标记分离到不同的文件中,而是将它们组合到了一起,这个地方就是组件 (Component);
 - □ 当然, 后面我们还是会继续学习更多组件相关的东西;
- 在这里,我们只需要知道,JSX其实是嵌入到JavaScript中的一种结构语法;

■ JSX的书写规范:

- □ JSX的顶层只能有一个根元素,所以我们很多时候会在外层包裹一个div元素(或者使用后面我们学习的Fragment);
- □ 为了方便阅读,我们通常在jsx的外层包裹一个小括号(),这样可以方便阅读,并且jsx可以进行换行书写;
- □ JSX中的标签可以是单标签,也可以是双标签;
 - ✓ 注意: 如果是单标签, 必须以/>结尾;



■ jsx中的注释

- JSX嵌入变量作为子元素
 - □情况一: 当变量是Number、String、Array类型时,可以直接显示
 - □情况二: 当变量是null、undefined、Boolean类型时,内容为空;
 - ✓ 如果希望可以显示null、undefined、Boolean, 那么需要转成字符串;
 - ✓ 转换的方式有很多,比如toString方法、和空字符串拼接, String(变量)等方式;
 - □情况三: Object对象类型不能作为子元素 (not valid as a React child)
- JSX嵌入表达式
 - □运算表达式
 - □三元运算符
 - □执行一个函数



JSX的使用

■ jsx绑定属性

- □比如元素都会有title属性
- □比如img元素会有src属性
- □比如a元素会有href属性
- □比如元素可能需要绑定class
- □比如原生使用内联样式style



React事件绑定

■ 如果原生DOM原生有一个监听事件,我们可以如何操作呢?

□ 方式一: 获取DOM原生, 添加监听事件;

□ 方式二:在HTML原生中,直接绑定onclick;

- 在React中是如何操作呢? 我们来实现一下React中的事件监听,这里主要有两点不同
 - React 事件的命名采用小驼峰式 (camelCase) ,而不是纯小写;
 - □ 我们需要通过{}传入一个事件处理函数,这个函数会在事件发生时被执行;



this的绑定问题

- 在事件执行后,我们可能需要获取当前类的对象中相关的属性,这个时候需要用到this
 - □ 如果我们这里直接打印this,也会发现它是一个undefined
- 为什么是undefined呢?
 - □ 原因是btnClick函数并不是我们主动调用的,而且当button发生改变时,React内部调用了btnClick函数;
 - □ 而它内部调用时,并不知道要如何绑定正确的this;

■ 如何解决this的问题呢?

□ 方案一: bind给btnClick显示绑定this

□ 方案二: 使用 ES6 class fields 语法

□ 方案三:事件监听时传入箭头函数 (个人推荐)



事件参数传递

- 在执行事件函数时,有可能我们需要获取一些参数信息: 比如event对象、其他参数
- 情况一: 获取event对象
 - □ 很多时候我们需要拿到event对象来做一些事情(比如阻止默认行为)
 - □ 那么默认情况下, event对象有被直接传入, 函数就可以获取到event对象;
- 情况二: 获取更多参数
 - □ 有更多参数时,我们最好的方式就是传入一个箭头函数,主动执行的事件函数,并且传入相关的其他参数;



React条件渲染

- 某些情况下,界面的内容会根据不同的情况显示不同的内容,或者决定是否渲染某部分内容:
 - □ 在vue中, 我们会通过指令来控制: 比如v-if、v-show;
 - □ 在React中,所有的条件判断都和普通的JavaScript代码一致;
- 常见的条件渲染的方式有哪些呢?
- 方式一: 条件判断语句
 - □ 适合逻辑较多的情况
- 方式二: 三元运算符
 - □ 适合逻辑比较简单
- 方式三: 与运算符&&
 - □ 适合如果条件成立, 渲染某一个组件; 如果条件不成立, 什么内容也不渲染;
- v-show的效果
 - □ 主要是控制display属性是否为none



React列表渲染

- 真实开发中我们会从服务器请求到大量的数据,数据会以列表的形式存储:
 - □ 比如歌曲、歌手、排行榜列表的数据;
 - □ 比如商品、购物车、评论列表的数据;
 - □ 比如好友消息、动态、联系人列表的数据;
- 在React中并没有像Vue模块语法中的v-for指令,而且需要我们通过JavaScript代码的方式组织数据,转成JSX:
 - □ 很多从Vue转型到React的同学非常不习惯,认为Vue的方式更加的简洁明了;
 - □ 但是React中的JSX正是因为和JavaScript无缝的衔接,让它可以更加的灵活;
 - □ 另外我经常会提到React是真正可以提高我们编写代码能力的一种方式;
- 如何展示列表呢?
 - □ 在React中,展示列表最多的方式就是使用数组的map高阶函数;
- 很多时候我们在展示一个数组中的数据之前,需要先对它进行一些处理:
 - □ 比如过滤掉一些内容: filter函数
 - □ 比如截取数组中的一部分内容: slice函数



列表中的key

■ 我们会发现在前面的代码中只要展示列表都会报一个警告:

```
Warning: Each child in a list should have a unique "key" prop.
Check the render method of `App`. See <a href="https://fb.me/react-warning-keys">https://fb.me/react-warning-keys</a> for more in li (created by App) in App
```

- 这个警告是告诉我们需要在列表展示的jsx中添加一个key。
 - □ key主要的作用是为了提高diff算法时的效率;
 - □ 这个我们在后续内容中再进行讲解;



JSX的本质

- 实际上, jsx 仅仅只是 React.createElement(component, props, ...children) 函数的语法糖。
 - 所有的jsx最终都会被转换成React.createElement的函数调用。
- createElement需要传递三个参数:
- 参数一: type
 - □ 当前ReactElement的类型;
 - □ 如果是标签元素, 那么就使用字符串表示 "div";
 - □ 如果是组件元素,那么就直接使用组件的名称;
- 参数二: config
 - 所有jsx中的属性都在config中以对象的属性和值的形式存储;
 - □ 比如传入className作为元素的class;
- 参数三: children
 - □ 存放在标签中的内容,以children数组的方式进行存储;
 - □ 当然,如果是多个元素呢? React内部有对它们进行处理,处理的源码在下方



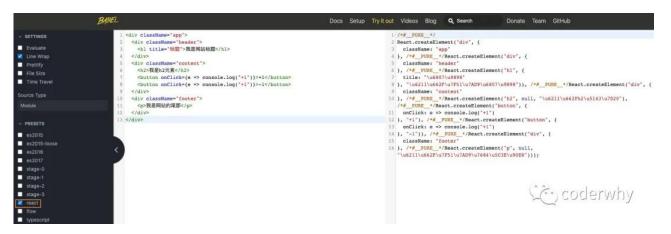
createElement源码

```
ReactElement.js — react-main
 EXPLORER
                                     JS ReactElement.js ×
∨ REACT-MAIN
                                      packages > react > src > JS ReactElement.js > ♥ createElement
         JS React.js
                                              export function createElement(type, config, children) {
         JS ReactAct.js
         JS ReactBaseClasses.js
                                                let propName;
         JS ReactChildren.js
         JS ReactContext.is
         JS ReactCreateRef.js
                                                const props = {};
         JS ReactCurrentActQueue.js
         JS ReactCurrentBatchConfig.js
         JS ReactCurrentDispatcher.is
                                                let key = null;
         JS ReactCurrentOwner.js
                                                let ref = null;
         JS ReactDebugCurrent rame.js
                                                let self = null;
                                       370
          ReactElement.is *
                                                let source = null;
         JS ReactElementValidator.js
         JS ReactForwardRef.js
                                                if (config != null) {
         JS ReactHooks.js
         JS ReactLazy.js
                                                  if (hasValidRef(config)) {
         JS ReactMemo.js
                                                      ref = config.ref;
         JS ReactMutableSource.js
         JS ReactNoopUpdateQueue.js
                                                     if (__DEV__) {
         JS ReactServerContext.js
                                                        warnIfStringRefCannotBeAutoConverted(config);
         JS ReactServerContextRegistry.js
         JS ReactSharedInternals.js
         JS ReactSharedSubset.experim...
         JS ReactSharedSubset.is
                                                   if (hasValidKey(config)) {
         JS ReactStartTransition.js
                                                     ·if (__DEV__) {
       JS index.classic.fb.is
                                                        checkKeyStringCoercion(config.key);
       JS index.experimental.js
       JS index.js
       JS index.modern.fb.is
                                                      key = '' + config.key;
 OUTLINE
> TIMELINE
> NPM SCRIPTS
                                                   self = config. self === undefined ? null : config. self:
                                                                                                                                     Ln 370, Col 19 Spaces: 2 UTF-8 LF {} JavaScript P Go Live ESLint Q
```



Babel官网查看

- 我们知道默认jsx是通过babel帮我们进行语法转换的,所以我们之前写的jsx代码都需要依赖babel。
- 可以在babel的官网中快速查看转换的过程: https://babeljs.io/repl/#?presets=react





直接编写jsx代码

■ 我们自己来编写React.createElement代码:

- □ 我们就没有通过jsx来书写了,界面依然是可以正常的渲染。
- □ 另外,在这样的情况下,你还需要babel相关的内容吗?不需要了
 - ✓ 所以, type="text/babel"可以被我们删除掉了;
 - ✓ 所以, <script src="../react/babel.min.js"></script>可以被我们删除掉了;

```
const result = React.createElement("div", {
 className: "app"
·},·/*#__PURE__*/React.createElement("div",·{
 className: "header"
}, */*#__PURE__*/React.createElement("h1", {
 title: "\u6807\u9898"
`},·"\u6211\u662F\u7F51\u7AD9\u6807\u9898")),·/*#__PURE__*/React.createElement("div",·{
 className: "content"
`}, /*#__PURE__*/React.createElement("h2", null, "\u6211\u662Fh2\u5143\u7D20"), /*#__PURE__*/React.
createElement("button", {
 onClick: e => console.log("+1")
·},·"+1"),·/*#__PURE__*/React.createElement("button", ·{
  onClick: e => console.log("+1")
·},·"-1")),·/*#__PURE__*/React.createElement("div",·{
 className: "footer"
`},<sup>,</sup>/*#__PURE__*/React.createElement("p",-null,<sup>,</sup>"\u6211\u662F\u7F51\u7AD9\u7684\u5C3E\u90E8")));
return result;
```



虚拟DOM的创建过程

■ 我们通过 React.createElement 最终创建出来一个 ReactElement对象:

- 这个ReactElement对象是什么作用呢? React为什么要创建它呢?
 - 原因是React利用ReactElement对象组成了一个JavaScript的对象树;
 - □ JavaScript的对象树就是虚拟DOM (Virtual DOM);

- 如何查看ReactElement的树结构呢?
 - □ 我们可以将之前的jsx返回结果进行打印;
 - □ 注意下面代码中我打jsx的打印;

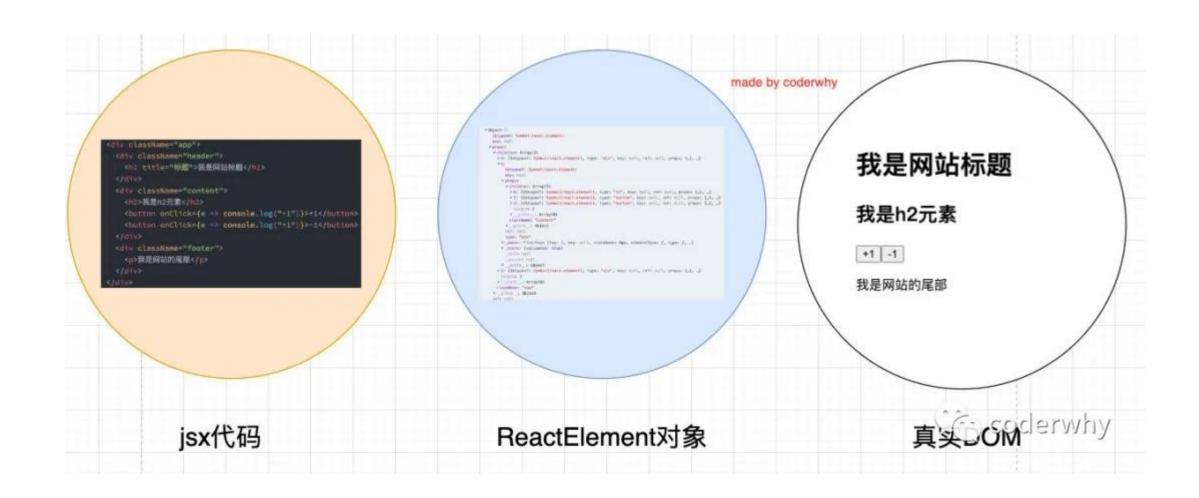
■ 而ReactElement最终形成的树结构就是Virtual DOM;

```
return ReactElement(
   type,
   key,
   ref,
   self,
   source,
   ReactCurrentOwner.current,
   props,
);
```

```
$$typeof: Symbol(react.element)
key: null
▼ children: Array(3)
  ▶ 0: {$$typeof: Symbol(react.element), type: "div", key: null, ref: null, props: {...}, ...}
      $$typeof: Symbol(react.element)
      key: null
    ▼ props:
      ▼ children: Array(3)
       ▶ 0: {$$typeof: Symbol(react.element), type: "h2", key: null, ref: null, props: {...}, ...}
       ▶ 1: {$$typeof: Symbol(react.element), type: "button", key: null, ref: null, props: {...}, ...}
       > 2: {$$typeof: Symbol(react.element), type: "button", key: null, ref: null, props: {...}, ...}
        ▶ __proto__: Array(0)
       className: "content"
      ▶ __proto__: Object
      ref: null
    ▶_owner: FiberNode {tag: 1, key: null, stateNode: App, elementType: f, type: f, ...}
    ▶ _store: {validated: true}
      self: null
      source: null
    ▶ __proto__: Object
  ▶ 2: {$$typeof: Symbol(react.element), type: "div", key: null, ref: null, props: {...}, ...}
                                                                                                 <u>Coc</u>erwhy
    __proto__: Array(0)
  className: "app"
```



jsx - 虚拟DOM - 真实DOM





声明式编程

- 虚拟DOM帮助我们从命令式编程转到了声明式编程的模式
- React官方的说法: Virtual DOM 是一种编程理念。
 - □ 在这个理念中,UI以一种理想化或者说虚拟化的方式保存在内存中,并且它是一个相对简单的JavaScript对象
 - 我们可以通过ReactDOM.render让 虚拟DOM 和 真实DOM同步起来,这个过程中叫做协调(Reconciliation);
- 这种编程的方式赋予了React声明式的API:
 - □ 你只需要告诉React希望让UI是什么状态;
 - □ React来确保DOM和这些状态是匹配的;
 - □ 你不需要直接进行DOM操作,就可以从手动更改DOM、属性操作、事件处理中解放出来;

■ 关于虚拟DOM的一些其他内容,在后续的学习中还会再次讲到;



阶段案例练习

- 1.在界面上以表格的形式,显示一些书籍的数据;
- 2.在底部显示书籍的总价格;
- 3.点击+或者-可以增加或减少书籍数量(如果为1,那么不能继续-);
- 4.点击移除按钮,可以将书籍移除(当所有的书籍移除完毕时,显示:购物车为空~);

	书籍名称	出版日期	价格	购买数量	操作
1	《算法导论》	2006-9	¥85.00	- 1 +	移除
2	《UNIX编程艺术》	2006–2	¥59.00	- 1 +	移除
3	《编程珠玑》	2008–10	¥39.00	- 1 +	移除
4	《代码大全》	2006-3	¥128.00	- 1 +	移除

总价格: ¥311.00

