day01-jsx-组件-状态-事件

1. 概述·

### React简介

React是Facebook开发的一款JS库, 那么Facebook为什么要造React呢，主要为了解决什么问题，通过这个又是如何解决的？

**Facebook为什么要建造react呢？**

Facebook认为MVC模式会使他们的项目变得更加复杂，致使代码变得脆弱和不可预测，非常难以理解和调试，特别是模型和视图间可能存在的双向数据流动。

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

**主要是为了解决什么问题？**

那么React是解决什么问题的，在官网可以找到这样一句话

We built React to resolve one problem: building large applications with data that changes over time.

构建那些数据会随时间改变的大型应用

### React特点

1. 虚拟DOM

我们以前操作dom的方式是通过document.getElementById()的方式，这样的过程实际上是先去读取html的dom结构，将结构转换成变量，再进行操作。而reactjs定义了一套变量形式的dom模型，一切操作和换算直接在变量中，这样减少了操作真实dom，性能真是相当的高，和主流MVC框架有本质的区别，并不和dom打交道.

1. 组件系统

react最核心的思想是将页面中任何一个区域或者元素都可以看做一个组件 component那么什么是组件呢？组件指的就是同时包含了html、css、js、image元素的聚合体。使用react开发的核心就是将页面拆分成若干个组件，并且react一个组件中同时耦合了css、js、image，这种模式整个颠覆了过去的传统的方式。

3. 单向数据流

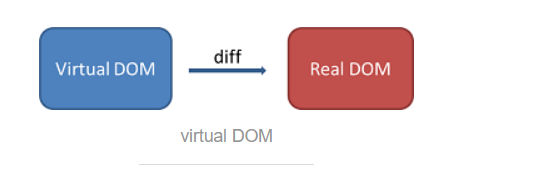
其实reactjs的核心内容就是数据绑定，所谓数据绑定指的是指要将一些服务端的数据和前端页面绑定好，开发者只关注实现业务就行了

4、JSX

在react语法中，我们使用render函数来构建组件的dom结构性能较高，因为省去了查找和编译模板的过程，但是在render中利用createElement创建结构的时候代码可读性较低，较为复杂，此时可以利用jsx语法来在render中创建dom，解决这个问题，但是前提是需要使用工具来编译jsx

**这些问题是如何解决的 - React的主要原理**

传统的web应用，操作DOM一般是直接更新操作的，但是我们知道DOM更新通常是比较昂贵的。而React为了尽可能减少对DOM的操作，提供了一种不同的而又强大的方式来更新DOM，代替直接的DOM操作。就是Virtual DOM,一个轻量级的虚拟的DOM，就是React抽象出来的一个对象，描述dom应该什么样子的，应该如何呈现。通过这个Virtual DOM去更新真实的DOM，由这个Virtual DOM管理真实DOM的更新。



为什么通过这多一层的Virtual DOM操作就能更快呢？ 这是因为React有个diff算法，更新Virtual DOM并不保证马上影响真实的DOM，React会等到事件循环结束，然后利用这个diff算法，通过当前新的dom表述与之前的作比较，计算出用最小的步骤更新真实的DOM。

特点可以概括为: **虚拟DOM**, **diff算法**

在**react 16之后发布的一种react 核心算法，React Fiber**是对核心算法的一次重新实现(官网说法)。之前用的是diff算法。在之前React中，更新过程是同步的，这可能会导致性能问题。当React决定要加载或者更新组件树时，会做很多事，比如调用各个组件的生命周期函数，计算和比对Virtual DOM，最后更新DOM树，这整个过程是同步进行的，也就是说只要一个加载或者更新过程开始，中途不会中断。因为JavaScript单线程的特点，如果组件树很大的时候，每个同步任务耗时太长，就会出现卡顿。

React Fiber的方法其实很简单——分片。把一个耗时长的任务分成很多小片，每一个小片的运行时间很短，虽然总时间依然很长，但是在每个小片执行完之后，都给其他任务一个执行的机会，这样唯一的线程就不会被独占，其他任务依然有运行的机会。

### React与传统MVC的关系

**MVC模式**

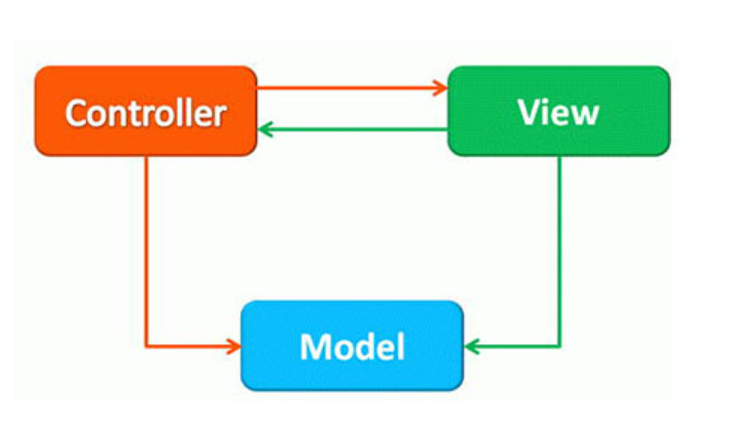
mvc是一种使用mvc（model view controller 模型-视图-控制器）设计创建web应用程序的模式

MVC是包括view视图层、controller控制层、model数据层。各部分之间的通信都是单向的。

model（模型）表示应用程序核心数据（比如数据记录列表）

view（视图）负责显示数据

controller（控制器）处理输入（写入数据库记录）（react主要放在控制器里，适合开发中大型项目）

View 传送指令到 ControllerController 完成业务逻辑后，要求 Model 改变状态Model 将新的数据发送到 View，用户得到反馈

轻量级的视图层库！A JavaScript library for building user interfacesReact

不是一个完整的MVC框架，最多可以认为是MVC中的V（View），甚至React并不非常认可MVC开发模式；React 构建页面 UI 的库。可以简单地理解为，React 将将界面分成了各个独立的小块，每一个块就是组件，这些组件之间可以组合、嵌套，就成了我们的页面。

### 开发工具的安装

Reactjs code snippets

### React初识

5.1 React.createElement

React.createElement( type, [props], [...children] )

第一个参数是必填，传入的是似HTML标签名称，eg: ul, li, div

第二个参数是选填，表示的是属性，eg: className

第三个参数是选填, 子节点，eg: 要显示的文本内容, 其他标签

* 1. ReactDOM.render

ReactDOM.render( template, container,[ callback ] )

第一个参数: 模版的渲染内容(HTML形式)

第二个参数: 模板插入的DOM节点

第三个参数, 选填: 渲染后的回调(一般用不到,可以省略)

5.3 渲染h1节点

const divVD = React.createElement('h1', { title: 'hello react' }, 'Hello React！！！')

ReactDOM.render(divVD, document.getElementById('app'))

1. JSX

### 概述

JSX是React的核心组成部分，它使用XML标记的方式去直接声明界面，界面组件之间可以互相嵌套。可以理解为在JS中编写与XML类似的语言,一种定义带属性树结构（DOM结构）的语法，它的目的不是要在浏览器或者引擎中实现，它的目的是通过各种编译器将这些标记编译成标准的JS语言。

虽然你可以完全不使用JSX语法，只使用JS语法，但还是推荐使用JSX，可以定义包含属性的树状结构的语法，类似HTML标签那样的使用，而且更便于代码的阅读。

这就是 JSX ，他是 JavaScript 的一种扩展语法。我们推荐在 React 中使用这种语法来描述 UI 信息。JSX 可能会让你想起某种模板语言，但是它具有 JavaScript 的全部能力。

我们把这种东西称之为，react元素，它可以很好的描述UI信息（dom节点的组合）

### JSX重构Hello world

ReactDOM.render(

<h1>欢迎进入React的世界</h1>,

document.getElementById('root') // 渲染到哪里

)

参数1这里就比较奇怪了，它并不是一个字符串，看起来像是纯 HTML 代码写在 JavaScript 代码里面。语法错误吗？这并不是合法的 JavaScript 代码, “在 JavaScript 写的标签的”语法叫 JSX- JavaScript XML。

### JSX语法中的js表达式

**3.1、嵌入JS表达式**

js表达式必须通过 {表达式} 的方式才能嵌入到jsx语法中.

<h1>欢迎进入React的世界 { 1+1 } </h1>

**3.2、属性绑定**

<h1 title={ a + 1 }>欢迎进入React的世界</h1>

class的绑定用className

<h1 className={`big ${i==index?'active':''}`}>欢迎进入React的世界</h1>

style的动态绑定

<h1 style={{width:20,fontSize:20}} >user</h1>

label中使用for的时候，要改成htmlfor

<h1 htmlFor="xxx">欢迎进入React的世界</h1>

**3.3、数组渲染**

3.3.1、直接渲染

const *list* = [  
 <li key="1">sunyang</li>,  
 <li key="2">lining</li>,  
 <li key="3">chenhe</li>  
]

数组成员可以是简单数据 number, string , 也可以是 dom 元素

3.3.2、处理并渲染

当数组成员是复合数据时,需要经过forEach 或 map方法处理后再渲染输出;

const *list* = [  
 { name:'sunyang',age:10 },  
 { name:'lining',age:20 },  
 { name:'chenhe',age:30 }  
];

<ul>  
 {  
 *list*.map(function (item) {  
 return <li key={item.age}>{item.name}</li>  
 })  
 }  
</ul>

### JSX语法中的注释

**写法一**（不推荐）：

{  
  // 注释  
  // ...  
}

**写法二**（推荐，把多行写到单行中）：

{ /\* 单行注释 \*/ }

**写法三**（多行）：

{  
  /\*  
  \* 多行注释  
  \*/  
}

注释在babel转换过程中会被忽略掉, 不会出现在最终的js代码中.

### 基本语法规则

1. 必须有且只能有一个根节点
2. 多标签包到一个小括号中，防止 JavaScript 自动分行不往后执行的问题。
3. 遇到以 < 开头的HTML 标签就用 HTML 规则解析
4. 遇到以 { 开头代码块，就用 JavaScript 规则解析
5. 单标签不能省略结束标签 , 必须结束 />
6. JSX 允许直接在模板中插入 JavaScript 变量
7. 如果这个变量是一个数组，则会展开这个数组的所有成员添加到模板中
8. 项目构建

### 初始化项目

1.1 create-react-app

1.2 react-script

### 搭建脚手架环境步骤:

1. 全局安装create-react-app
   1. 安装命令: npm install create-react-app -g
   2. 安装成功后create-react-app和npx命令就都可以使用了;
2. 使用create-react-app或npx命令创建项目
   1. create-react-app 项目名

或者

* 1. npx create-react-app 项目名

1. 进入项目目录
   1. cd 项目名
2. 启动项目
   1. npm start

### 目录结构

项目目录

node\_modules

public

src

1. 组件

### 组件的创建方式

#### 函数式组件

// 特别注意这里的写法，如果要在JSX里写js表达式(只能是表达式，不能流程控制)，就需要加 {}，包括注释也是一样，并且可以多层嵌套

const app = (props) => <h1>欢迎进入{props.name}的世界</h1>

​

ReactDOM.render(

app({

name: 'react'

}),

document.getElementById('root')

)

这里我们定义的方法实际上也是react定义组件的第一种方式-定义函数式组件，这也是无状态组件。但是这种写法不符合react的jsx的风格，更好的方式是使用以下方式进行改造

const App = (props) => <h1>欢迎进入{props.name}的世界</h1>

ReactDOM.render(

<App name="react" />, // React组件的调用方式

document.getElementById('root')

)

它的特点:

组件不会被实例化，整体渲染性能得到提升

组件不能访问this对象

组件无法访问生命周期的方法

无状态组件只能访问外部输入的props，同样的props会得到同样的渲染结果，不会有副作用

#### 类组件

ES6的加入让JavaScript直接支持使用class来定义一个类，react的第二种创建组件的方式就是使用的类的继承，ES6 class是目前官方推荐的使用方式，它使用了ES6标准语法来构建

class App extends React.Component {

render () {

return (

// 注意这里得用this.props.name, 必须用this.props

<h1>欢迎进入{this.props.name}的世界</h1>

)

}

}

ReactDOM.render(

<App name="react" />,

document.getElementById('root')

)

### 组件命名规范

组件名称必须以大写字母开头。

<Welcome name="james">

小驼峰命名来定义属性

<Welcome className="blue" tabIndex="1" >

### 组件传值 props

函数组件通过参数props接收外部组件传入的数据, 在组件内根据props处理UI

**传递值:**

<Header id="1001" url="http://xxx.com"></Header>

**接收值( 函数式组件 ):**

函数式组件用参数 props 接收值

function *Header* ( props ){  
 return (  
 <div className="header">  
 {/\* 组件模板中使用 组件的属性 \*/}  
 <h1>{props.id},{props.url}</h1>  
 </div>  
 )  
}

类组件中处理props

类组件通过this.props接收外部组件传入的数据, 在组件内根据this.props处理UI

**接收值( class组件 ):**

**class**组件内部用 this.props 接收值

class Header extends React.Component{  
 render(){  
 return (  
 <div className="header">  
 {/\* 组件模板中使用 组件的属性 \*/}  
 <h1>{this.props.id},{this.props.url}</h1>

</div>  
 )  
 }  
}

1. State状态

### 基本使用,初始化

组件构造方法中定义组件初始状态;

组件模板中可以用 this.state.xxx 访问状态;

### 修改状态

修改组件状态,必须用 this.setState({ xxx:yyy })

在React中，如果是由React引发的事件处理（比如通过onClick引发的事件处理），调用setState不会同步更新this.state，除此之外的setState调用会同步执行this.state。所谓“除此之外”，指的是绕过React通过addEventListener直接添加的事件处理函数，还有通过setTimeout/setInterval产生的异步调用。

### props与state的区别

props是一个父组件传递给子组件的数据流，这个数据流可以一直传递到子孙组件。而state代表的是一个组件内部自身的状态。

props或者 组件内部state改变都会导致组件重新渲染. 既然两者的变化都有可能导致组件重新渲染，所以只有理解props与state的意义，才能很好地决定到底什么时候用props或state。

对于组件自身来讲,props是只读的, state是可变的.

1. 事件处理

### 事件绑定

示例:

<div>  
 { /\* <button onClick={ handleClick }>clickme</button> \*/}  
 <button onClick={ ()=>this.handleClick() }>clickme</button>  
</div>

需要注意:

1.事件名必须符合驼峰命名法

2.事件值必须绑定一个函数 即 {handleClick}, 不能是字符串形式 “handleClick”

3.可以是匿名函数

### 事件对象

run=(event)=>{

alert( event.target ) //获取dom节点

}

render(){

return(

<div>

<button aid="123" onClick={this.run}>事件对象</button>

</div>

)

}

### 事件函数传参

**在给方法传递新参数时，方法原有的参数会排在新参数之后.**

run=(param, event)=>{

alert( param , event.target ) //获取dom节点

}

render(){

return(

<div>

<button aid="123" onClick={ ()=>this.run(100) }>事件对象 </button>

</div>

)

}

### this指向问题

4.1. constructor里面bind和render里面bind

4.2. 箭头函数

4.3.行内定义事件使用bind绑定this

### 合成事件

如果DOM上绑定了过多的事件处理函数，整个页面响应以及内存占用可能都会受到影响。React为了避免这类DOM事件滥用，同时屏蔽底层不同浏览器之间的事件系统差异，实现了一个中间层——SyntheticEvent。

当用户在为onClick添加函数时，React并没有将Click事件绑定在DOM上面。

而是在document处监听所有支持的事件，当事件发生并冒泡至document处时，React将事件内容封装交给中间层SyntheticEvent（负责所有事件合成）

所以当事件触发的时候，对使用统一的分发函数dispatchEvent将指定函数执行。

注意: 不要将原生事件（addEventListener）和React合成事件一起混合使用，这两个机制是不一样的

### React支持的事件类型

来看一下React都支持哪些事件，以及这些事件的写法；

键盘事件 onKeyDown onKeyPress onKeyUp

焦点事件 onFocus onBlur

表单事件 onChange onInput onSubmit

鼠标事件

onClick onContextMenu onDoubleClick onMouseDown onMouseUp onMouseOver onMouseOut onMouseMove onMouseEnter onMouseLeave onDrag

选择事件 onSelect

触摸事件 onTouchCancel onTouchEnd onTouchMove onTouchStart

UI事件 onScroll

动画事件 onAnimationStart onAnimationEnd onAnimationIteration

图像事件 onLoad onError

媒体事件 onPause onPlay onCanPlay onLoadStart onProgress

剪贴板事件 onCopy onCut onPaste

1. 综合案例

<cdf-会员购> 店铺首页-渲染快捷入口

<cdf-会员购> 店铺首页-渲染超级单品

作业一:

<cdf-会员购> 店铺首页-今日值得买(秒杀)

1.用class组件 + jsx语法 开发组件;

2.在各个组件内部定义组件状态(数据);

3.页面样式要100%还原.

4.封装头部输入组件 MyHeader;

5.封装轮播图组件MySwipe( 不用实现, 先用色块占位, 能渲染即可 );

6.封装快捷入口列表组件MyEntry;

7.封装今日值得买(秒杀)组件MySecKill;

8.封装底部Tabbar组件 MyTabbar;

作业二:

<cdf-会员购> 全部商品页-商品排序(综合排序,销量排序,价格排序)

1.用class组件 + jsx语法 开发组件;

2.在各个组件内部定义组件状态(数据);

3.页面样式要100%还原.

4.封装头部输入组件 MyHeader;

5.封装商品列表组件MyList;

6.封装底部MyTabbar组件;

7.实现点击综合/销量/新品/价格 降序排序;

8.实现点击综合/销量/新品/价格 按钮高亮且只有一个高亮(红色);

作业三:

<cdf-会员购> 分类页-选项卡切换

1.用class组件 + jsx语法 开发组件;

2.在各个组件内部定义组件状态(数据);

3.页面样式要100%还原.

5.封装选项卡组件MyTab;

6.封装底部MyTabbar组件;

7.实现点击左侧一级分类按钮, 动态渲染右侧二级分类;

8.实现点击左侧一级分类按钮, 按钮高亮且只有一个高亮(红色);