**React 核心概念：JSX，虚拟 DOM，Diff 算法，setState，state 和 props 梳理**

原创 现代前端教程 [技术漫谈](javascript:void(0);) 6月28日

来自专辑

深入浅出 React 技术栈

本文主要梳理一下我对 React 框架基础内容的认识，之后也会总结一些深度内容的认识。当然，笔者水平也有限，如果你发现不妥之处，望斧正！

**为什么要用 React 等前端框架**

因为可以进行组件化开发？社区强大？使用的人多？有很多好用的第三方库和插件？适用于单页面应用开发？这些都不是最本质的原因。

**最本质的原因是保持 UI 和状态同步并不容易。**

使用原生 JavaScript 编写代码的时候，一般静态内容是在 HTML 中创建，动态内容使用 JavaScript 创建。但是使用 JavaScript 构建 UI 代码量巨大，并且可读性不高，代码可复用性也差。

更重要的是，如果我们想要保持 UI 和状态的同步应该怎么做？例如，我们有一个列表，我们需要实现将列表与服务器同步的功能。我们需要将本地数据和服务器发来的数据进行比较。并且需要点对点的将每个变更同步到 DOM 节点中。如果这个过程中有每一步出现差错都直接导致 UI 同步失败。因此，维护 UI 与数据同步需要编写大量繁琐，脆弱和脆弱的代码。

https://zhuanlan.zhihu.com/p/39852035

为了解决以上问题，降低开发成本，前端社区出现了 React 等框架。到目前为止，这些框架提供的最大的改进是实现应用状态和 UI 同步。我们只需要定义一次 UI，不必为每一次动作都编写 UI。相同的状态总能得到相同的 UI 输出，即状态和 UI 同步，状态变更后会自动更新 UI。

**React VS Vue**

很多人都说 React 上手要难一些，Vue 上手要简单一些。也有很多人在问 React 和 Vue 的区别是什么。我想说一下我的理解。

首先，在框架设计层面，React 是 MVC 软件架构中的 View，它只负责视图层的东西，而对于数据和路由等，则由 Redux 和 Router 等来完成。这也就是为什么 React 官方文档说 “React 是一个用于构建用户界面的 JavaScript 库”。而 Vue 则不同，Vue 是基于 MVVM 架构设计的一个框架，所以在架构层面，Vue 和 React 的思想就是不同的。

React 的设计哲学很简单，不可变（Immutable）思想贯穿了整个框架的设计。React 可以说没引进什么新的概念，让开发者能够以类似写原生 JavaScript 代码的方式来使用 React 进行开发。这也就是很多人说 React 难的原因，即 JavaScript 基础知识。

而很多人说 Vue 容易上手，可能有些同学就迷惑了，我觉得 Vue 不简单啊。笔者认为，说 Vue 简单是因为 Vue 将很多底层逻辑封装好了，直接给你对应的 API，你调用对应 API 就可以完成对应的工作。所以，你不需要有很扎实的 JavaScript 基础，你只要记住或者能够查阅到对应 API 就可以用 Vue 完成一些基础性的开发。所以有些后端的同学可以看看 Vue 官方文档就能上手基础的前端开发。

React 之我见，那一定要说我的看法，我喜欢 React。我喜欢其简洁的设计理念和类原生的开发体验，让我能够感觉到自己是在写代码。那么 Vue 我没有深入学习，但这是一个非常优秀的前端框架，目前也已经安排进了下一阶段的学习任务中。

**React JSX，虚拟 DOM 和 ReactDOM.render**

**JSX 和虚拟 DOM**

JSX 和虚拟 DOM 是必说内容。JSX 也就是 JavaScript 的一个语法扩展，可以通过 Babel 对 JSX 进行转译。

const title = <h1 className="title">Hello, world!</h1>;

例如这样一段 JSX 代码会被 Babel 转译成：

const title = React.createElement(  
    'h1',  
    { className: 'title' },  
    'Hello, world!'  
);

从这里我们可以看出，React.createElement 方法的参数是这样的：

createElement( tag, attrs, child1, child2, child3 );

那么我们可以实现一下 createElement：

function createElement( tag, attrs, ...children ) {  
    return {  
        tag,  
        attrs,  
        children  
    }  
}

然后这样就可以对其进行调用：

*// 将上文定义的 createElement 方法放到对象 React 中*  
const React = {  
    createElement  
}  
const element = (  
    <div>  
        hello<span>world!</span>  
    </div>  
);  
console.log( element );

此处引用 hujiulong 的 从零开始实现一个 React（一）：JSX 和虚拟 DOM，详细内容推荐看其原文。

也就是说，createElement 方法返回的东西就是所谓的虚拟 DOM。说到虚拟 DOM 就必说 diff 算法，这个我们之后再讲。

**ReactDOM.render**

render 其实没有太多好说的，其作用就是将虚拟 DOM 渲染成真实的 DOM，只是在实现的过程中有非常多的细节情况要处理。

**React 组件和生命周期**

在 React 中，我们可以通过 Class 和 Function 两种方式来写组件。以 Class 的方式写组件的时候，我们需要继承 React.Component。这里主要涉及 React.Component 的模拟实现和 React 的生命周期。模拟实现推荐看 从零开始实现一个 React（二）：组件和生命周期。至于 React 的生命周期，之后我会总结一篇文章，推荐大家去看 React 官方文档。

**Diff 算法**

为什么要 diff？因为 DOM 操作十分昂贵，也就是非常耗时耗性能。但是 DOM 操作就一定很慢很耗性能吗？其实不一定，只是普遍来讲是这样的，详细内容推荐看 JJC 的知乎回答：**前端为什么操作 DOM 是最耗性能的呢[1]**。简单讲，浏览器中 DOM 对象是使用 C++ 开发的，性能肯定不慢，有些情况下性能甚至高于操作 JavaScript 对象。而大多数时候操作 DOM 是很耗性能的，这是因为 JavaScript 对象的属性（properties）映射到的是 DOM 对象的特性（attributes）上。

我们操作 JavaScript 对象只是修改一个对象，而修改 DOM 对象时还会修改对象的 attributes，那么性能就消耗在 JavaScript 对象和 DOM 对象的转换和同步上。并且，操作 DOM 还会影响页面的渲染，进而会再一次降低了性能。虽然 DOM 的性能有可能比操作普通对象还要快，但 99% 的场景下，操作 DOM 对象还是昂贵的。所以，高效的 diff 势在必行。

我们可以通过实现高效的 diff 算法，对比出操作前后有差异的 DOM 节点，然后只对更改了的 DOM 进行更新。这样可以大大减少没有必要的 DOM 操作，进而大幅提升性能。

React 的 diff 算法其实很简单：对比当前真实 DOM 和虚拟 DOM，在对比过程中直接更新真实 DOM。并且这个比对是同级比对，当发现这一级的 DOM 不同时，会直接更新该节点及其所有子节点。这种 diff 策略实际上是出于性能上的取舍。首先，DOM 操作很少出现跨层级的情况，所以只需要同级比对就可以满足大多数情况，也就没有必要对比所有的 DOM 节点，因为那样需要 O(n^3) 的时间复杂度，代价太高。

React 虚拟 DOM 的 Diff 原理的思维导图如下所示：

React Virtual Dom Diff Algorithm

React diff 算法的模拟实现推荐参见：从零开始实现一个 React（三）：Diff 算法

**所谓异步的 setState**

**基本知识**

首先明确，所谓异步的 setState 并不是真的异步，只是其行为类似于异步操作的行为。下面我们来详细说说。

首先，将 setState 的行为设置为所谓异步的形式的出发点依旧是性能优化，因为如果每次通过 setState 更改了状态都对组件进行一次更新，会很浪费性能。而通过将多次 setState 合并在一起，进行一次更新渲染，则会大大提升性能。

将多个 setState 的调用合并成一个来执行，也就意味着 state 并不会被立即更新。例如下面这例子：

class App extends Component {  
    constructor() {  
        super();  
        this.state = {  
            num: 0  
        }  
    }  
    componentDidMount() {  
        for ( let i = 0; i < 100; i++ ) {  
            this.setState( { num: this.state.num + 1 } );  
            console.log( this.state.num );    *// 会输出什么？*  
        }  
    }  
    render() {  
        return (  
            <div className="App">  
                <h1>{ this.state.num }</h1>  
            </div>  
        );  
    }  
}

这里定义了一个组件 APP，在组件挂载后，会循环 100 次。但当我们真的对这个组件进行渲染会发现，渲染结果为 1，并且控制台中输出了 100 次 0。也就是说，每次循环中拿到的 state 都还是更新之前的。也就是说 React 对 setState 进行了优化，但如果我们就是想要立刻获得更新后的 state 怎么办呢？React 提供了一种解决方案，setState 接收的参数还可以是一个函数，通过这个函数我们可以拿到更新之前的状态，然后通过这个函数的返回值得到下一个状态：

componentDidMount() {  
    for ( let i = 0; i < 100; i++ ) {  
        this.setState( prevState => {  
            console.log( prevState.num );  
            return {  
                num: prevState.num + 1  
            }  
        } );  
    }  
}

**合并 setState**

对于这个地方，有两个要点：一是如何对多个 setState 调用进行合并。二是我怎么判定一次合并哪些 setState 呢？接下来我们一一解答。

setState 的合并是通过队列实现的。通过创建一个队列来保存每次 setState 的数据，然后每隔一段时间，清空和这个队列并渲染组件。此处的模拟实现推荐阅读：从零开始实现一个 React（四）：异步的 setState

接下来就是一次到底合并哪些 setState。换句话说，我们会将一段时间内的 setState 调用合并成在一起执行，那么这段时间的长短取决于什么？其实此处使用的是 JavaScript 的事件队列机制，也就是事件循环（Event Loop）。关于事件循环的详细内容可参见阮一峰的 **JavaScript 运行机制详解：再谈 Event Loop[2]**。事件循环的核心概念就是同步任务，异步任务，微任务以及宏任务。

在 React 的 setState 中，利用 JavaScript 的事件循环机制对多个 setState 调用进行合并。首先创建一个队列保存每次 setState 的数据，在一次事件循环的所有同步任务之后，清空着队列，将队列中的所有 setState 进行合并，并进行一次性更新渲染。这样在一次事件循环的，最多只会执行一次合并操作，并且只会渲染一次组件。

所以呢，这也就是为什么我说，所谓的 setState 的异步行为并不是真正的异步，只是不会对每一个 setState 操作进行实时更新，而是通过队列的方式对一次事件循环中的所有同步任务的 setState 调用进行合并，合并成一个之后进行一次更新和渲染，所以效果上看上去是异步的，但并不是 setTimeout 或 setInterval 这种真正的异步操作。

**关于 setState 同步异步的总结**

1. setState 只在合成事件和钩子函数中是“异步”的，在原生事件和 setTimeout 中都是同步的。
2. setState 的“异步”并不是说内部由异步代码实现，其实本身执行的过程和代码都是同步的，只是合成事件和钩子函数的调用顺序在更新之前，导致在合成事件和钩子函数中没法立马拿到更新后的值，形式了所谓的“异步”，当然可以通过第二个参数 setState(partialState, callback) 中的 callback 拿到更新后的结果。
3. setState 的批量更新优化也是建立在“异步”（合成事件、钩子函数）之上的，在原生事件和 setTimeout 中不会批量更新，在“异步”中如果对同一个值进行多次 setState ，setState 的批量更新策略会对其进行覆盖，取最后一次的执行，如果是同时 setState 多个不同的值，在更新时会对其进行合并批量更新。

参考文章：https://juejin.im/post/5b45c57c51882519790c7441

**React 中 props 和 state 的关系**

React 中的 props（“properties” 的缩写）和 state 都是普通的 JavaScript 对象。它们都是用来保存信息的，这些信息可以控制组件的渲染输出，而它们的几个重要的不同点就是：

props 是传递给组件的（类似于函数的形参），而 state 是在组件内被组件自己管理的（类似于在一个函数内声明的变量）。

props 是不可修改的，所有 React 组件都必须像纯函数一样保护它们的 props 不被更改。由于 props 是传入的，并且它们不能更改，因此我们可以将任何仅使用 props 的 React 组件视为 pureComponent，也就是说，在相同的输入下，它将始终呈现相同的输出。

state 是在组件中创建的，一般在 constructor 中初始化 state。state 是多变的、可以修改，一般通过 setState 被修改，并且一般是异步更新的。

至此，React 的基本概念已经梳理完毕，感谢阅读。笔者水平有限，如果你发现任何问题，欢迎评论指正！

**参考资料**

* **React Dosc[3]**
* **React 官方文档[4]**
* **从零开始实现一个 React（一）：JSX和虚拟DOM[5]**
* **从零开始实现一个 React（二）：组件和生命周期[6]**
* **从零开始实现一个 React（三）：Diff 算法[7]**
* **从零开始实现一个 React（四）：异步的 setState[8]**
* **前端为什么操作 DOM 是最耗性能的呢 —— JJC 的知乎回答[9]**

**参考资料**

[1]

前端为什么操作 DOM 是最耗性能的呢: https://www.zhihu.com/question/324992717/answer/707044362

[2]

JavaScript 运行机制详解：再谈 Event Loop: http://www.ruanyifeng.com/blog/2014/10/event-loop.html

[3]

React Dosc: https://reactjs.org/docs/getting-started.html

[4]

React 官方文档: https://zh-hans.reactjs.org/docs/getting-started.html

[5]

从零开始实现一个 React（一）：JSX和虚拟DOM: https://github.com/hujiulong/blog/issues/4

[6]

从零开始实现一个 React（二）：组件和生命周期: https://github.com/hujiulong/blog/issues/5

[7]

从零开始实现一个 React（三）：Diff 算法: https://github.com/hujiulong/blog/issues/6

[8]

从零开始实现一个 React（四）：异步的 setState: https://github.com/hujiulong/blog/issues/7

[9]

前端为什么操作 DOM 是最耗性能的呢 —— JJC 的知乎回答: https://www.zhihu.com/question/324992717/answer/707044362

**看完三件事**

如果你觉得本文对你有帮助，我想请你帮个忙：

1. 转发本文或者点个「**在看**」，让更多的人也能看到这篇内容；
2. 关注公众号「**技术漫谈**」，订阅更多精彩内容，获取更多学习资料；
3. 可以在本公众号后台回复「**加群**」，加入技术群，与更多读者交流。

**长按上方二维码关注公众号「技术漫谈」，订阅更多精彩内容。**

**你“在看”我吗？**

#React##前端##源码解读#

阅读 279

赞1在看8