React架构设计：项目实战经验分享

前言

这是第一次在GitChat上发布自己的文章，近期我在负责开发一个大型电商平台项目，所采用的前端架构是React.js。设计到的技术栈有Reactjs、 React-Router、 Redux、 Nodejs、 Webpack、 Sass等等。所以思来想去，还是就现在最火的框架React.js来跟小伙伴们聊聊前端。大家应该都知道，以往的前端工作内容，只是一些切图写一些样式。甚至有的公司这些活都是交给设计来完成，当时根本就没有什么前端相关的话题。但是随着业务需求越来越复杂，传统的“前端”已经无法满足实现这些复杂的功能点。HTML5跟CSS3的问世，彻彻底底的推翻了整个“前端”。

我一开始是写Java的。13年从Java转成前端，整整四年时间，见证了前端这三年来天翻地覆的更新。就去年聊的水生火热的三大框架(React、vue、angular)来聊聊,很多干前端的，干后端的都问过我一个问题：“这么多前端框架，哪个好？”。我一般都笑笑跟他们说我也不知道。一个框架的问世到拥有这么多的使用者，你能说它不好吗？那么问题来了，我们要去学哪个才好呢？在这里我给的建议是：“精通其一，熟悉其二，最终还是原生Javascript”。在这里我主要介绍的是Reactjs。分享给大家我近期开发的电商项目涉及到的知识点。如果你是准备近期着手学习Reactjs的，我可以推荐给你一些学习方法。如果你已经玩React易如反掌了，我们可以一起聊聊React，聊聊JavaScript。如果你是React专家级别的，老司机请带带我。废话就不多说了。总之一句话，希望小伙伴能共同学习，共同进步。下面进入正题；

原理

在Web开发中，我们总需要将变化的数据实时反应到UI上，这时就需要对DOM进行操作。而复杂或频繁的DOM操作通常是性能瓶颈产生的原因（如何进行高性能的复杂DOM操作通常是衡量一个前端开发人员技能的重要指标）。React为此引入了虚拟DOM（Virtual DOM）的机制：在浏览器端用Javascript实现了一套DOM API。基于React进行开发时所有的DOM构造都是通过虚拟DOM进行，每当数据变化时，React都会重新构建整个DOM树，然后React将当前整个DOM树和上一次的DOM树进行对比，得到DOM结构的区别，然后仅仅将需要变化的部分进行实际的浏览器DOM更新。

而且React能够批处理虚拟DOM的刷新，在一个事件循环（Event Loop）内的两次数据变化会被合并，例如你连续的先将节点内容从A变成B，然后又从B变成A，React会认为UI不发生任何变化，而如果通过手动控制，这种逻辑通常是极其复杂的。尽管每一次都需要构造完整的虚拟DOM树，但是因为虚拟DOM是内存数据，性能是极高的，而对实际DOM进行操作的仅仅是Diff部分，因而能达到提高性能的目的。这样，在保证性能的同时，开

发者将不再需要关注某个数据的变化如何更新到一个或多个具体的DOM元素，而只需要关心在任意一个数据状态下，整个界面是如何Render的。

如果你像在90年代那样写过服务器端Render的纯Web页面那么应该知道，服务器端所要做的就是根据数据Render出HTML送到浏览器端。如果这时因为用户的一个点击需要改变某个状态文字，那么也是通过刷新整个页面来完成的。服务器端并不需要知道是哪一小段HTML发生了变化，而只需要根据数据刷新整个页面。换句话说，任何UI的变化都是通过整体刷新来完成的。而React将这种开发模式以高性能的方式带到了前端，每做一点界面的更新，你都可以认为刷新了整个页面。至于如何进行局部更新以保证性能，则是React框架要完成的事情。

借用Facebook介绍React的视频中聊天应用的例子，当一条新的消息过来时，传统开发的思路如上图，你的开发过程需要知道哪条数据过来了，如何将新的DOM结点添加到当前DOM树上；而基于React的开发思路如下图，你永远只需要关心数据整体，两次数据之间的UI如何变化，则完全交给框架去做。可以看到，使用React大大降低了逻辑复杂性，意味着开发难度降低，可能产生Bug的机会也更少。

React组件

组件属性

ReactJS是基于组件化的开发，React 允许将代码封装成组件（component），然后像插入普通 HTML 标签一样，在网页中插入这个组件。

组件1

```
    import React from 'react';

    import CheckoutPurchasedHeader from
'../../module/CheckoutPurchasedHeader'

    class OrderSkuList extends React.Component {
        constructor(props) {
            super(props);
        }

        render(){
            let Header = ["商品信息","单价(元)","数量","小计(元)","操
作"]
            return(
                <div className="order-detail-my-order">
                    <CheckoutPurchasedHeader _Header={ Header }
/>
                </div>
            )
        }
```

```
}

export default OrderSkuList
```

组件2

```
import React, { Component, PropTypes } from 'react';
    class CheckoutPurchasedHeader extends Component {
        constructor(props){
            super( props);
        }
        render() {
            let { _Header } = this.props;
            let nodes = [];
            if( _Header ){
                nodes = _Header.map( (data,index)=>{
                    return <li key={ index }>{data}</li>
            })
        }

           return (
                <ul className="checkout-purchased-header">
                    { nodes }
                </ul>
            );
        }
    }

    export default CheckoutPurchasedHeader;
```


看到这段代码，有几点需要注意：

- 获取属性的值用的是this.props.属性名。
- 创建的组件名称首字母必须大写。
- 为元素添加css的class时，要用className。
- 组件的style属性的设置方式也值得注意，要写成style={{width: this.state.witdh}}。

组件状态

组件免不了要与用户互动，React 的一大创新，就是将组件看成是一个状态机，一开始有一个初始状态，然后用户互动，导致状态变化，从而触发重新渲染 UI .

```
import React, { Component, PropTypes } from 'react';
    class CheckoutPurchasedHeader extends Component {
```

```
            constructor(props){
                super( props);
                this.state ={
                    _Header: ["商品信息","单价(元)","数量","小计
(元)","操作"]
                }
            }
            render() {
                let nodes = [];
                if( this.state._Header){
                    nodes = this.state._Header.map(
(data,index)=>{
                        return <li key={ index }>{data}</li>
                })
            }

               return (
                    <ul className="checkout-purchased-header">
                        { nodes }
                    </ul>
                );
            }
        }

        export default CheckoutPurchasedHeader;
```


这里值得注意的几点如下：
- this.state初始化该组件所要用到的对象。

- 访问state的方法是this.state.属性名。
- 变量用{}包裹，不需要再加双引号。
- 可以通过this.setState方法来修改当前组件的state。当检测到state有变化的时候，组件会重新render.

组件的生命周期

组件的生命周期分成三个状态：

Mounting：已插入真实 DOM.

Updating：正在被重新渲染。

Unmounting：已移出真实 DOM.

React 为每个状态都提供了两种处理函数，will 函数在进入状态之前调用，did 函数在进入状态之后调用，三种状态共计五种处理函数。

componentWillMount()

componentDidMount()

componentWillUpdate(object nextProps, object nextState)

componentDidUpdate(object prevProps, object prevState)

componentWillUnmount()

此外，React 还提供两种特殊状态的处理函数。

componentWillReceiveProps(object nextProps)：已加载组件收到新的参数时调用。

shouldComponentUpdate(object nextProps, object nextState)：组件判断是否重新渲染时调用。

组件的嵌套

React是基于组件化的开发，那么组件化开发最大的优点是什么？毫无疑问，当然是复用.仔细阅读上面第1点的代码，你会发现CheckoutPurchasedHeader组件可以在任何地方复用。

为什么选择React？

React最初开发这个特别的框架目标是“搭建数据需要频繁更改的大型应用”。这也就可以说是React项目搭建算是蛮复杂的。比如说写一个简单的helloworld。其他的框架几行代码就解决，但是React写起来就比较繁琐。 但这也可以说是React的优点，代码的组件化，会大大的提升后期开发跟维护的效率。说到这里，我就要开始说说为什么我们要选择React.下面是我总结六点内容来阐述React的优势所在。

> 1. React效率极高
> 2. 同构，纯粹的JavaScript
> 3. Javascript库（JSX）
> 4. 一切都是component
> 5. 单向数据流
> 6. Facebook

React的效率高只要体现在他的虚拟dom上

```
React.render(
<div className="commentBox">
```

```
        Hello, world! I am a CommentBox.
      </div>  );
```

这里的div其实和dom里面的div完全是两码事儿，只不过React提供了和DOM类似的Tag和API，事实上React会通过他自己的逻辑去转化为真正的DOM。所以，把这种叫做虚拟DOM.

那么这样做有什么好处呢？最明显的一点好处就是React所谓的 dom diff ，能够实现delta级别的dom更新。当有数据变动导致DOM变动时，React不是全局刷新，而是通过它内部的 dom diff 算法计算出不同点，然后以最小粒度进行更新。这也是React号称性能好的原因。

其次，还有一点非常好的地方就在于，有了虚拟DOM就可以让UI脱离设备，换句话说，只要有对应的转化关系，如：虚拟DOM -> 浏览器DOM，就能进行渲染。

React 使用 JSX 来替代常规的 JavaScript

JSX 是一个看起来很像 XML 的 JavaScript 语法扩展。它有以下优点：

> JSX 执行更快，因为它在编译为 JavaScript 代码后进行了优化。
>
> 它是类型安全的，在编译过程中就能发现错误。
>
> 使用 JSX 编写模板更加简单快速。

组件化

虚拟DOM(virtual-dom)不仅带来了简单的UI开发逻辑，同时也带来了组件化开发的思想，所谓组件，即封装起来的具有独立功能的UI部件。React推荐以组件的方式去重新思考UI构成，将UI上每一个功能相对独立的模块定义成组件，然后将小的组件通过组合或者嵌套的方式构成大的组件，最终完成整体UI的构建。例如，Facebook的instagram.com整站都采用了React来开发，整个页面就是一个大的组件，其中包含了嵌套的大量其它组件，大家有兴趣可以看下它背后的代码。如果说MVC的思想让你做到视图-数据-控制器的分离，那么组件化的思考方式则是带来了UI功能模块之间的分离。

React本身其实只是一个V，并不是一个完整的框架

所以如果是大型应用想要一套完整的框架的话，react-router和redux是必不可少的。react的最大的问题就在于它组件跟组件之间的数据传递很麻烦，比如说父组件跟子组件公用同一个数据，子组件这时要修改数据的话，需要通过重重回调到父组件上，才能修改该数据，但是有了redux之后，所有的数据存放在顶层state树上，每个组件都可以公用顶层数据源的数据。组件通过触发action来修改数据。

在活跃度跟质量上自然是可以有保障的。上述这些优点也无法断定react就是一个很完美的框架，但最起码，它提出了一个全新的概念，新鲜的东西总是能够吸引大家的眼球。但是在架构选型上，得要根据实际应用的特点来选择合适的框架。

谈谈心得

最后一段我想阐述一些我实际开发中的心得。我做前端开发说长不长，说短也有四年了。所说的有不对的地方望指出。在过去的两年里前端这个市场突然火起来了，导致一下子出来了很多的前端开发人员。需求多了，竞争自然而然的也就多了。那么问题来了，我们要怎样去突出自己？我发表几点个人意见。

1. 不怕技术不成熟，关键是要**用心**。
2. **基础，基础，基础。**重要的事情说三遍。
3. 不要让自己闲下来。建议去一些需要加班的公司（个人偏好加班）。
4. 不懂就问，要多沟通交流。不要让技术有局限性。
5. 时常关注技术论坛。关注其他人写代码的方式，从中领悟优缺点。
6. 时常要关注新的技术。

总之一句话“**态度决定一切**”！