深入React技术栈和企业级项目开发

第1天课堂笔记

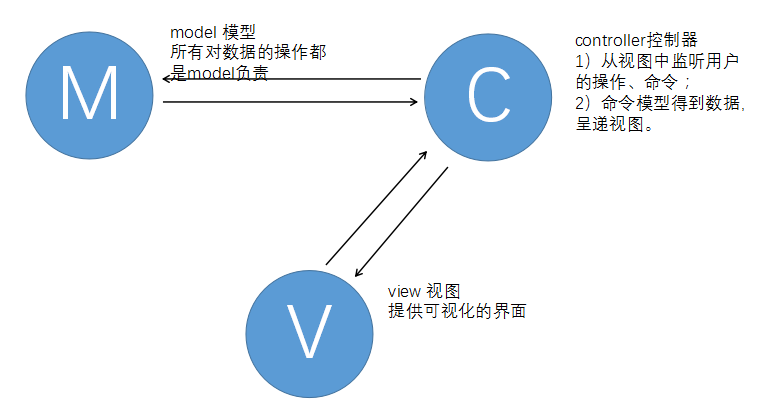
讲师：邵山欢

日期：2017年11月18日

# MVVM框架简介和React简介

## MVVM

从MVC开始：



我们写一段伪代码，来体现MVC的编程模式：

app.get("/" , function(req,res){

User.find({"id" : id},function(err,data1){

Question.find({"email" : data.email} , function(err,data2){

res.json({"result" : data2});

});

});

});

上面的代码是nodejs中的片段，是控制器的代码，可以看出控制器的两个层面的任务：

1. 读取get请求、得到参数；
2. 命令模块读取数据（从数据库中），综合结果，将数据呈递在视图上。

我们再来看前端开发中的Ajax的语句：

<script>

$.get("/api" , function(data){

document.getElementById("title").innerHTML = data.result;

});

</script>

你会发现这里的思路就是发出一个请求，请求数据，这个数据实际上就是M层。此时要调用DOM语句呈递在视图上。如果我们增加两个按钮，一个按钮让数据加1，一个减1：

<script>

//全局变量a

var a;

//拉取默认的a数据

$.get("/api" , function(data){

a = data.result;

**render();**

});

//两个按钮的监听

var btn1 = document.getElementById("btn1");

var btn2 = document.getElementById("btn2");

btn1.onclick = function(){

a++;

**render();**

}

btn2.onclick = function () {

a--;

**render();**

}

//render函数的任务很简单，就是让h1标签中的内容改为a

function render(){

document.getElementById("title").innerHTML = a;

}

</script>

如果页面上对a的操作更加丰富呢？此时代码可读性、清晰度会变差。我们发现每一次对数据a的操作，都要进行一次render()函数的调用。

有的程序员在思考，**能不能让a变化的时候，DOM自动变化，此时就是MVVM架构**。



上图就是MVVM架构的示意图，MVVM实际上是三部分：

**m**odel 模型，对数据的操作；

**v**iew 视图

**v**iew-**m**odel 模型和视图的粘合剂；

Angular、React、Vue都是MVVM框架，都能够实现数据变化视图自动变化。但是他们三个人的原理完全不同：

* Angular：脏检查，词法分析，进行隐形的视图的更新；
* React：setState()等调用视图函数，配合DIFF算法和Virtual DOM让DOM的变化效率更高；
* Vue ： 数据劫持

● React中封装了一些函数，只有通过这些函数改变a属性的值，才能引发视图的变化。比如下面案例中的setstate()函数，通过setState()函数改变a属性值，会引发视图的变化。

var obj = {

a : 100

};

//我们不能随便更改a,你如果要更改a，要通过这个函数：

function setState(k,v){

obj[k] = v;

document.getElementById("title").innerHTML = obj.a;

}

● Vue中采用数据劫持的方法来实现数据变化视图就跟着变化。Vue的作者尤雨溪发现了Object.defineProperty()方法，可以在obj对象的a属性变化的时候，做一些事情：

//IE8开始有这个函数，表示定义某一个对象的属性

**Object.defineProperty(obj, "a", {**

**"set" : function(v){**

document.getElementById("title").innerHTML = v;

this.a = v;

**}**,

"get" : function(){

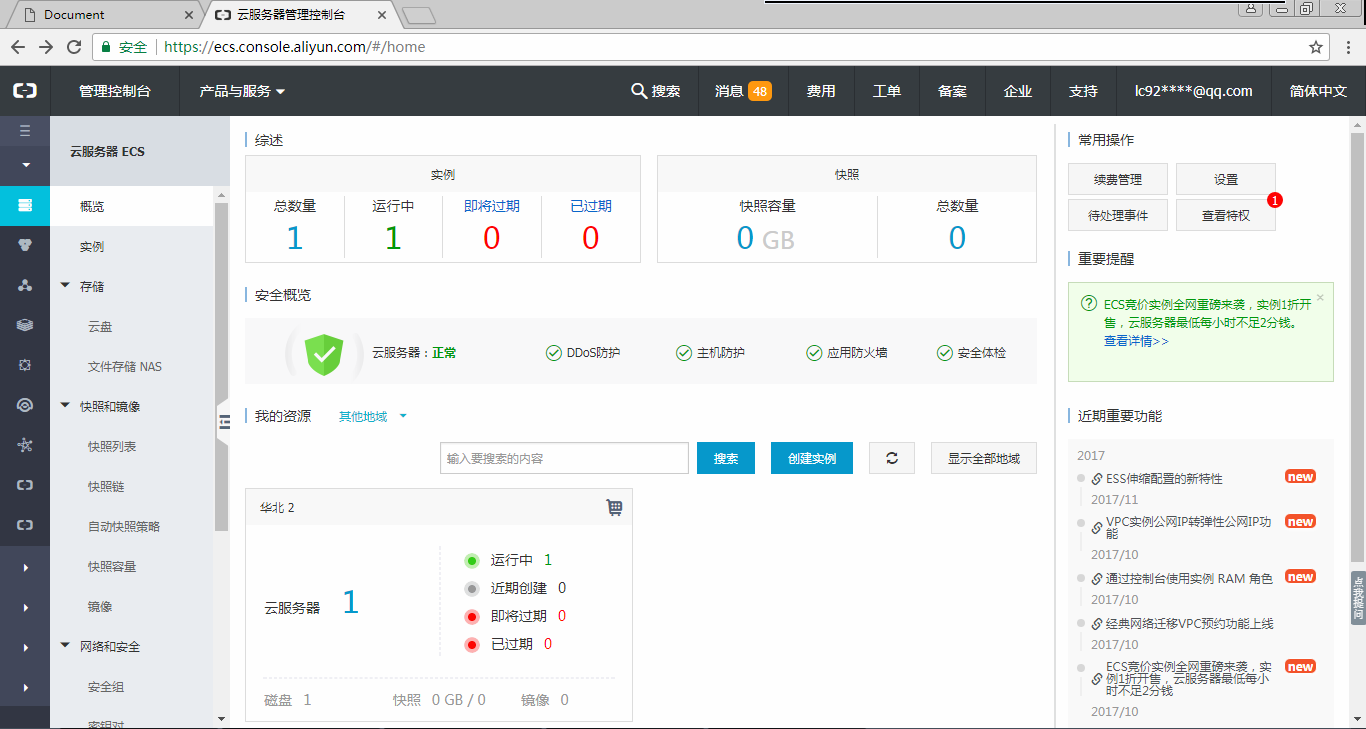
return this.a;

}

**});**

## 1.2 单页面应用

这几年非常流行单页面应用，SPA（single page application）。**单页面应用就是把类似桌面的应用，挪到了网页中，用前端技术去实现**。



上图就是阿里云的管理面板，单页面应用实际上就是将桌面的应用程序搬到了网页上。这个应用程序非常的大。逻辑、流程都很复杂。

**一提到单页面应用，就要知道三个词：组件化、hash路由、封装。**

● 组件化

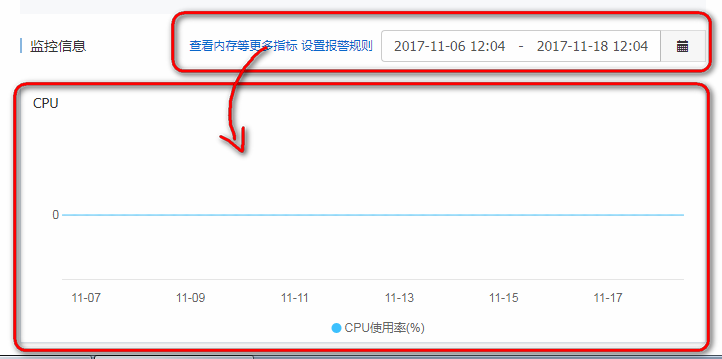
我们看看阿里云的组件化开发的情况，两个不同的页面出现了完全相同的组件。

**组件：一个HTML、CSS、JS到任何地方都能被复用的功能，称为组件**。

|  |  |
| --- | --- |
|  |  |

我们的MVVM框架，比如Angular、React、Vue还要肩负组件的管理、维护能力。

组件之间还有数据传输：



React的组件管理能力是所有框架中最好的！

● hash路由

|  |  |
| --- | --- |
|  |  |

我们的整个项目都是在一个页面中完成的，只有一个html页面，**页面之间的跳转实际上就是组件之间的上场和退场**。我们用hash形式的路由暗示用户页面已经跳转了，并且可以便于分享网址。

MVVM框架还要提供hash路由的功能。

● 封装

我们查看阿里云的源代码，会发现HTML是一个空的标签，在页面加载完毕之后，才上了DOM、CSS和JS功能。实际上它们都被封装到了一个js文件中。



MVVM框架还要提供对HTML、CSS、JS的封装功能。

总结，我们刚才1.1、1.2讲解的就是时下的流行框架，它们都在为我们解决什么问题：

第1：数据变化视图自动变化；

第2：组件的管理能力（比如组件的创建、组件的嵌套、组件之间的数据传输）；

第3：管理hash路由的能力；

第4：封装html、css、js的能力；

## 1.3 React简介



官网：https://reactjs.org/

稍微了解一下，不懂的，随着课程进行就懂了：

* 来自Facebook，2013年诞生
* 和Angular不同，React并不是一个完整的MVC/MVVM框架，它专注于提供清晰、简介的视图层解决方案。它的功能全部以构建组件视图为核心，并提供类似控制器的函数接口和生命周期函数。
* React非常轻，目标是Minimal API interface，只提供组件化相关的非常少量的API。简化了一切可以简化的东西。
* React可以搭配Redux、Flux等架构使用。
* React使用Virtual DOM技术，使得它性能卓越，开销很小。

市场份额：React（60%）、Vue（40%）、Angular（不到1%）。

可以说死：所有新项目，一定不是Angular的，包括1、2、5。

# React项目起步

React要进行一些配置：webpack + babel-loader + react

想一下依赖，分层想：

* webpack层面，我们要-g**安装**webpack，写webpack.config.js文件，指导webpack进行文件的打包；
* babel-loader层面，我们要--save-dev**安装两个：**babel-core和babel-loader，所谓的loader就是webpack在构建文件、打包文件的时候，捎带脚做的事情。还要--save-dev**安装两个**presets：babel-preset-env（负责翻译ES6、7、8的高级语法）、babel-preset-react（负责翻译jsx语法糖）；
* react层面，我们要--save**安装两个**：react、react-dom，他们就是react框架的源码。就和写jQuery必须先引包一样，我们开发react的适合，也要先引包。

cnpm intsall --save-dev babel-core babel-loader babel-preset-env babel-preset-react

cnpm intsall --save react react-dom

webpack.config.js文件：

const path = require('path');

module.exports = {

entry: "./app/main",

output: {

path: path.resolve(\_\_dirname, "dist"),

filename: "bundle.js"

},

module: {

rules: [

{

test: /\.jsx?$/,

include: [

path.resolve(\_\_dirname, "app")

],

exclude: [

path.resolve(\_\_dirname, "node\_modules")

],

loader: "babel-loader",

options: {

presets: ["env" , "react"]

}

}

]

},

watch : true

}

项目结构：

|  |
| --- |
| ┣ index.html  ┣ webpack.config.js  ┣ package.json  ┣ app  ┃ ┣ main.js |

app/main.js主入口文件：

import React from "react";

import ReactDOM from "react-dom";

**ReactDOM.render(**

<div>

<h1>我是React！哈哈</h1>

</div>

,

document.querySelector("#app-container")

**);**

开启webpack，因为在webpack.config.js文件中写了watch:true，所以文件是能改变的时候自动打包新的bundle.js的。

index.html只需要引入一个js文件：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>React学习</title>

</head>

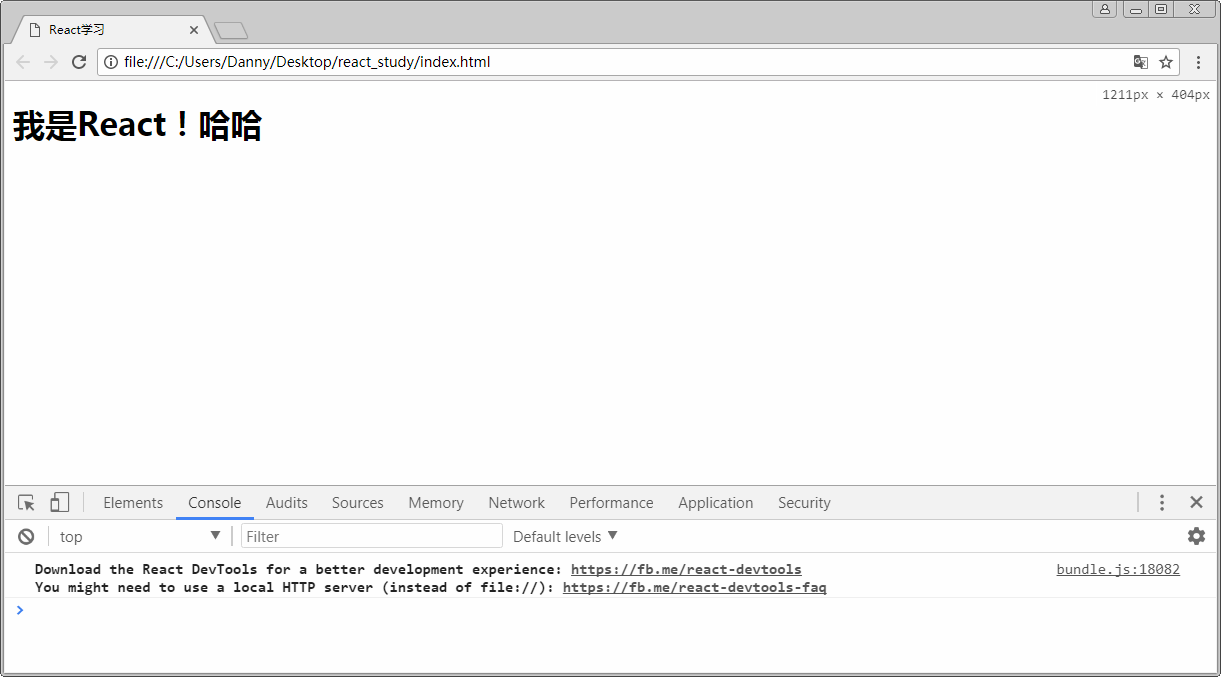
<body>

<div id="app-container"></div>

**<script src="dist/bundle.js"></script>**

</body>

</html>



我们来看：

<div id="app-container"></div>

We call this a “root” DOM node because everything inside it will be managed by React DOM.

我们称这个节点叫做“根”节点，因为里面所有的东西被ReactDOM管理。

Applications built with just React usually have a single root DOM node.

一个React应用程序，通常只有一个根节点。

If you are integrating React into an existing app, you may have as many isolated root DOM nodes as you like.

如果你的React的app要混合进入别人已经存在的app中，你可能要写多个独立的根节点。

To render a React element into a root DOM node, pass both to ReactDOM.render()。

为了呈递React element到根节点中，此时将它俩作为参数传入ReactDOM.render()方法中。

ReactDOM.render(

<div>

<h1>我是React！哈哈</h1>

</div>

,

document.querySelector("#app-container")

);

红色的部分我们叫做React Element，蓝色的部分叫做根节点。

# jsx语法

## 3.1 简介

我们将helloworld进行变化，将React Element提出成为变量：

import React from "react";

import ReactDOM from "react-dom";

const element = <div>

<h1>我是React！哈哈嘿嘿</h1>

</div>;

ReactDOM.render(

element

,

document.querySelector("#app-container")

);

感觉少了引号，但是是合格的语法，这就是jsx。

This funny tag syntax is neither a string nor HTML.

有意思的语法，这里不是HTML也不是字符串。

It is called JSX, and it is a syntax extension to JavaScript.

它被叫做JSX，是一个JavaScript语法的拓展。

We recommend using it with React to describe what the UI should look like.

我们推荐用JSX语法来写React，它用来描述UI长成什么样子。

JSX may remind you of a template language, but it comes with the full power of JavaScript.

JSX可能让你想起模板引擎（比如ejs、php），但是它拥有JavaScript全部的能力。

JSX produces React “elements”.

JSX生产了React的“elements”。

## 3.2 在JSX中使用JS表达式

You can embed any JavaScript expression in JSX by wrapping it in curly braces.

你可以将JavaScript表达式通过大括号融合在JSX中。

//定义一个函数

const power = a => a \* a;

const element = <div>

<h1>我是React！哈哈嘿嘿，爱你**{5000\*2}**年！</h1>

<h1>我是React！哈哈嘿嘿，爱你**{power(100)}**年！</h1>

</div>;

## 3.3 将JSX当做函数返回值

//定义一个函数，这个函数返回jsx

const sayHello = (sex) =>{

if(sex == "男"){

return <div>

<b>欢迎你！先生！请你购买：</b>

<ul>

<li>刮胡刀</li>

<li>香烟</li>

</ul>

</div>;

}else{

return <div>

<b>欢迎你！女士！请你购买：</b>

<ul>

<li>化妆品</li>

<li>香包</li>

</ul>

</div>;

}

}

//调用：

const element = <div>

<h1>我是React！哈哈嘿嘿，爱你{5000\*2}年！</h1>

<h1>我是React！哈哈嘿嘿，爱你{power(100)}年！</h1>

**{sayHello('女')}**

</div>;

## 3.4 在HTML标签的属性上使用JSX语法

const showPic = (picname) => {

if(picname == "孙杨"){

return "images/0.jpg";

} else if (picname == "女排") {

return "images/1.jpg";

}

}

var w = 200;

const element = <div>

<div>

<img src=**{showPic('孙杨')}** width=**{w}**/>

</div>

</div>;

这里有一个问题，错了是灾难性的，就是标签的属性用jsx语法的时候，不要用引号：

<img ~~src="{showPic('孙杨')}" width="{w}"~~/>

## 3.5 JSX标签必须严格封闭

所有的<div>必须有匹配的</div>，单标签必须有末尾的反斜杠：

<img src={showPic('孙杨')} width={w} **/>**

<br **/>**

## 3.6 JSX中如果出现class要用className代替

因为class是关键字，所以要用className来代替；

<div className="spec">我是重要的文字</div>

同样的：

<label for="" />

用

<label htmlFor="" />

## 3.7 JSX中遇见数组，会被自动展开（关键）

我们创建了一个数组，数组里面有三个React Element：

const arr = [<p>哈哈</p>, <p>嘻嘻</p>, <p>么么哒</p>];

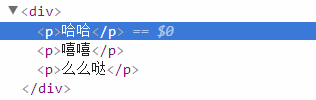
当数组写在其他JSX语法解构中，会被自动展开。

<div>

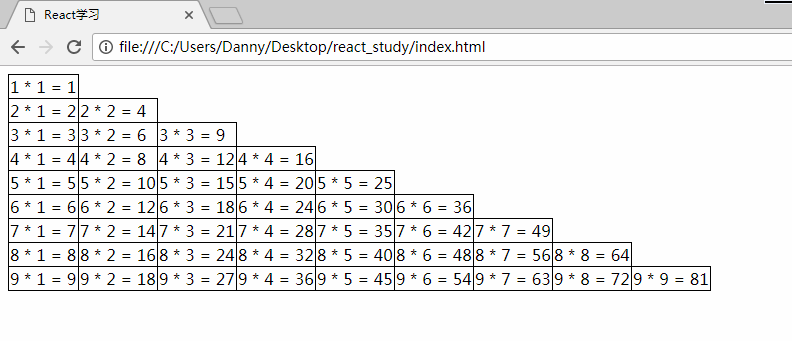
{arr}

</div>

当页面渲染的时候：



## 3.8 小案例：九九乘法表



在JSX语法中，数组是能够自动展开的；反过来说，一些循环的东西，要想到用数组展开。

写一个返回二维数组的函数，数组是能够自动展开的：

const showJiuJiu = () => {

//这是最终要返回的结果，是一个二维数组。

var result = [];

for(var i = 1 ; i <= 9 ; i++){

//某一行的数组

var temp = [];

for(var j = 1 ; j <= 9 ; j++){

if(j <= i){

temp.push(

<td key={j}>

{i} \* {j} = {i \* j}

</td>

);

}else{

temp.push(

<td key={j} className="noborder"></td>

);

}

}

result.push(<tr key={i}>{temp}</tr>);

}

return result;

}

使用的时候调用函数即可：

ReactDOM.render(

<div>

<table>

<tbody>

**{showJiuJiu()}**

</tbody>

</table>

</div>

,

document.querySelector("#app-container")

);

我们来看看VUE中实现九九乘法表，非常的简单：

<body>

<div id="app-container">

<table>

<tr **v-for="i in 9"**>

<td **v-for="j in 9"** **v-show="j <= i"**>{{i}} \* {{j}} = {{i \* j}}</td>

</tr>

</table>

</div>

<script src="vue.min.js"></script>

<script>

new Vue({

el : "#app-container"

});

</script>

</body>

Angular和Vue都有指令系统，angular是ng-前缀，vue中是v-前缀。

**React中没有指令！也就是说，你需要知道好多对DOM元素操作的“套路”**，下午讲解。

# 组件的创建

Components let you split the UI into independent, reusable pieces，and think about each piece in isolation

组件让你将UI拆分为一个个独立的、可以复用的块，并且每一个块都是独立的。

components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

组件很像JavaScript函数。它们接受任意数量的参数（我们称为props），并且返回React Elements，这些React Elements描述了组件长得样子。

## 4.1 函数式组件

Functional Components，函数式组件。

先看下面的案例，我们在3.3小节已经讲过了:

import React from "react";

import ReactDOM from "react-dom";

//这是一个函数，它返回了jsx

function showHello(){

return <h1>你好</h1>

}

ReactDOM.render(

<div>

{showHello()}

</div>

,

document.querySelector("#app-container")

);

将上面的函数做两个变化：① 将函数的名字变为大写； ② 调用的时候采用HTML的标签的形式：

**React中的组件的名字是大写字母开头的**！

import React from "react";

import ReactDOM from "react-dom";

function **ShowHello**(){

return <h1>你好！哈哈！</h1>;

}

ReactDOM.render(

<div>

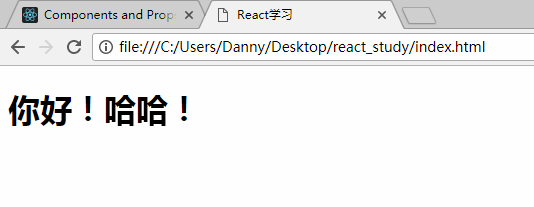
**<ShowHello></ShowHello>**

</div>

,

document.querySelector("#app-container")

);



这就是最简单的组件！

**在React中一个大写字母开头的函数，返回了一个JSX代码段，这个函数就是一个组件！它可以通过自定义HTML标签的方式进行调用！**

**React是angular、react、vue中最简单制作组件的框架。**



三个特点：

● 组件是可复用的：

**//定义一个函数，函数的名字是大写的，暗示这是一个组件**

function ShowHello(){

return <h1>你好！我爱你！</h1>;

}

ReactDOM.render(

<div>

**<ShowHello></ShowHello>**

**<ShowHello></ShowHello>**

**<ShowHello></ShowHello>**

**<ShowHello></ShowHello>**

**<ShowHello></ShowHello>**

</div>

,

document.querySelector("#app-container")

);



● 组件之间是相互独立的

这里有一个重要知识点：**React中自定义组件是通过标签的自定义的属性来往组件中传值的**：

//定义一个函数，函数的名字是大写的，暗示这是一个组件

function ShowHello(**{a=10000}**){

return <h1>你好！我爱你**{a}**年！</h1>;

}

ReactDOM.render(

<div>

<ShowHello **a={34}**></ShowHello>

<ShowHello **a={123}**></ShowHello>

<ShowHello **a={3455}**></ShowHello>

<ShowHello **a={34444}**></ShowHello>

<ShowHello></ShowHello>

</div>

,

document.querySelector("#app-container")

);



● 组件可以任意组合的！

我们现再封装一个大组件，叫做ShowMain，里面可以使用小组件：

function **ShowMain**(){

return <section>

<div>广告</div>

**<ShowHello></ShowHello>**

</section>

}

使用的时候，只要有大组件上场，就会有小组件：

ReactDOM.render(

<div>

<ShowMain></ShowMain>

</div>

,

document.querySelector("#app-container")

);



## 4.2 类组件

import React from "react";

class Compo1 extends React.Component{

**constructor(props){**

**//调用超类的构造函数，语法规定**

**super(props);**

**}**

**render(){**

return <div>

<h1>哈哈，我是类组件1！</h1>

</div>

**}**

}

export default Compo1;

**做一个不一定正确的总结：先学习了es6的class方法，构造函数接受实例化类时传入的参数，以此添加到构造函数的比如this.name中，调用super是为了调用父类的构造函数，如此不需要自己再多些一遍变量的定义。而在react中，在react组件在reactdom.render中实例化时候，会自动根据标签上的属性，组合成一个props对象，传入到具体的react组件对象中，就相当于执行了new Compo1(props)语句。如此，在构造函数中写上super（props）,就不需要在构造函数中，写this.props = props语句了。**

**然而，很特别的是，在render方法中，可以不需要调用父类的构造函数，也可以直接使用this.props,这一点和es6类的继承有点不一样。类中都的构造函数没有this.props = props,然而再render方法中可以直接使用this.props，我想是因为react的源码中肯定写了，相关语句，是的render中返回的jsx中直接使用**

**super关键字，它指代父类的实例（即父类的this对象）。子类必须在constructor方法中调用super方法，否则新建实例时会报错。这是因为子类没有自己的this对象，而是继承父类的this对象，然后对其进行加工。如果不调用super方法，子类就得不到this对象。**

**只有一个理由需要传递props作为super()的参数，那就是你需要在构造函数内使用this.props**

React has a few different kinds of components, but we’ll start with React.Component subclasses。

React有多种形式创建组件，但是最基本的就是写一个类，继承React.Comonent。

Compo1 is a **React component class**, or **React component type**.

Compo1叫做“组件类”或者“组件类型”。

A component takes in parameters, called props, and returns a hierarchy of views to display via the **render** method.

一个组件接受一个叫做“props”的参数，这个组件返回的界面样式是通过一个叫做“render”的方法。

The render method returns a description of what you want to render, and then React takes that description and renders it to the screen.

render方法返回了描述组件长成什么样子，React就会将它呈递到屏幕上。

在使用的时候main.js，进行引包，然后就可以用自定义标签的形式来进行调用。

import React from "react";

import ReactDOM from "react-dom";

**import Compo1 from "./Compo1.js";**

ReactDOM.render(

<div>

**<Compo1></Compo1>**

**<Compo1></Compo1>**

**<Compo1></Compo1>**

**<Compo1></Compo1>**

</div>

,

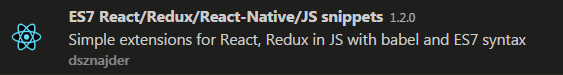
document.querySelector("#app-container")

);

总结类组件的书写：

1. 每一个类组件都是单独的文件，文件的名字必须和类的名字完全相同！类的名字大写字母开头！
2. 类必须继承自React.Component类，所以要引入react包;
3. 类的构造器中，必须调用超类的构造器，super(props);
4. 界面用render()方法返回jsx即可！
5. 记得暴露！！

在visual studio code中，可以安装下面这个插件：



此时可以使用快捷键rcc然后tab键，即可快速展开模板。

# Local State和Virtual DOM

## 5.1 local state是什么

**在任何一个类组件（函数式组件没有local state）的构造函数中，可以添加一个state属性，它的值必须是一个对象，叫做组件内部状态。在视图中，可以通过this.state.\*\*的形式来访问得到这个属性值**。

import React from "react";

export default class App extends React.Component{

constructor(props){

super(props);

//这个就叫做内部state，英语叫做local state

this.state = {

"a" : 100

}

}

render(){

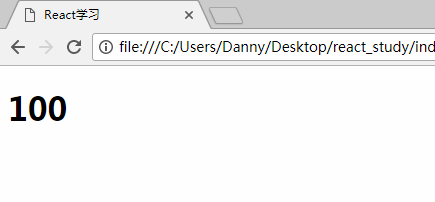
return <div>

<h1>{this.state.a}</h1>

</div>

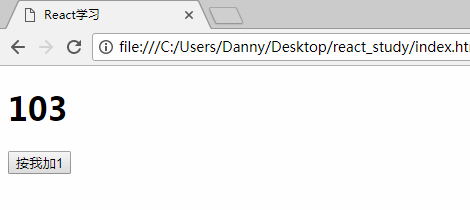
}

}



## 5.2 通过this.ssetState()改变local state的值

我们做一个计数器的小功能，点击按钮加1。



import React from "react";

export default class App extends React.Component{

constructor(props){

super(props);

//这个就叫做内部state，英语叫做local state

**this.state = {**

**"a" : 100**

**}**

}

**add(){**

**this.setState({**

**"a" : this.state.a + 1**

**});**

**}**

render(){

return <div>

<h1>**{this.state.a}**</h1>

<button **onClick={this.add.bind(this)}**>按我加1</button>

</div>

}

}

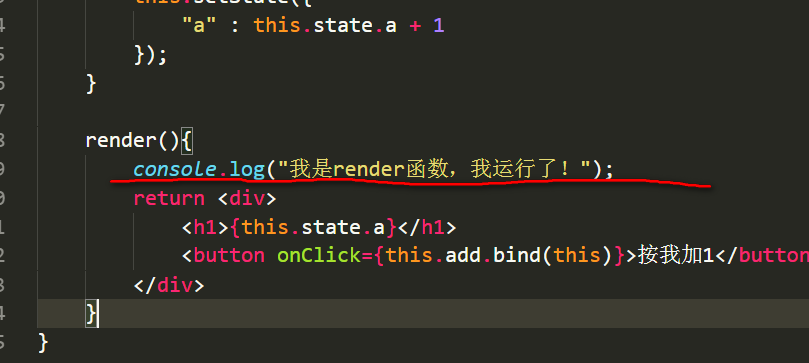
数据变化了！视图就自动发生变化！这是MVVM框架的最基本的诉求。

**注意，改变state的值，必须使用this.setState()方法，如果直接使用this.state.\*\* = \*\*的语法，此时视图不会变化！**

也就是说，setState()暗含有调用render方法的语句！（明天你将知道，实际上setState()将组件的生命周期进行了等级跃迁）。

事件监听一律on后面字母大写！！

我们做一个测试，在render里面加上log语句：



此时点击一下按钮，就会看见log出这行文字。

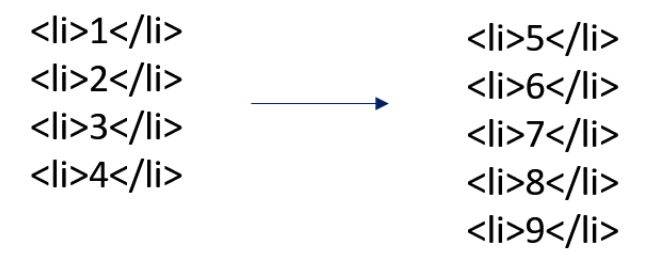


## 5.3 Virtual DOM（面试爱问）

每一次我们调用setState()的时候，此时都要执行render()函数里面的语句，会return一个庞大的jsx语法，此时所有DOM都要跟着重绘么？很显然效率很低。

所以React采用了Virtual DOM虚拟DOM和DIFF算法，**在内存中快速比较setState变化前后的差异，做最小量更新**！

比如下图，我们点击按钮之后，state影响界面从左侧的形式变为了右侧的形式：

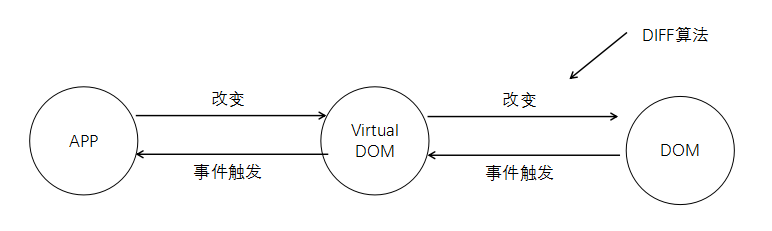


方案1：删除4个li，插入5个li；

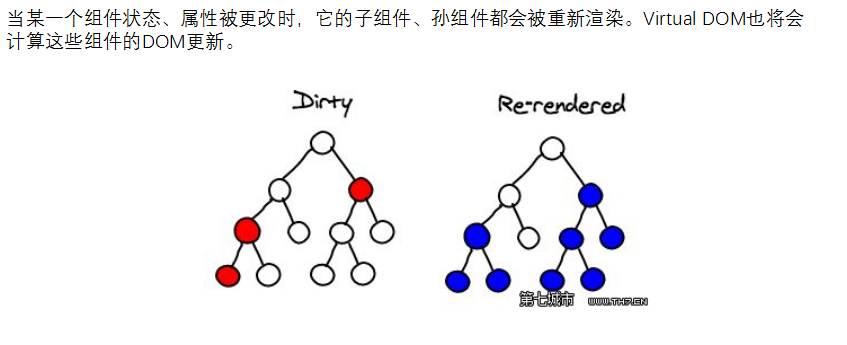
方案2：增加1个li，改变5个li的innerHTML；

很明显，方案2更加节约内存，效率高。

* React中有DIFF算法，就是可以快速比较两个DOM结构的最小化差异。
* React会将DOM结构编码（类似于数据结构中的树的结构），存储在内存中，称为Virtual DOM，虚拟DOM。



当组件有组件套组件的情况下，Virtual DOM还会自动计算子孙组件的最小化差异：



# locate state和界面的操作套路（重点）

## 6.1 上树和下树

现在组件有一个state的属性叫做show，它的真假影响盒子的上树与下树。怎么做？

this.state = {

"show" : true

}

套路是，写一个函数：

render() {

**const showBox = () => {**

**return this.state.show ? <div className="box"></div> : null;**

**}**

return <div>

<button onClick={this.shangshu.bind(this)}>按我</button>

**{showBox()}**

</div>

}

返回null表示没有标签。

引申两个问题：

● 可以不写函数，在jsx中直接写三元运算符：

render() {

return <div>

<button onClick={this.shangshu.bind(this)}>按我</button>

{this.state.show ? <div className="box"></div> : null}

</div>

}

但是if语句绝对不行！jsx中的{}中只能写简单js表达式，不能写if、for、while语句等等。

render() {

return <div>

<button onClick={this.shangshu.bind(this)}>按我</button>

~~{if(this.state.show){<div className="haha"></div>}}~~

</div>

}

● 如何决定按钮的文字也变化呢？

render() {

//下面这个函数，根据state中的show不同返回一个div盒子或者null，从而决定了盒子是否上树

**const showBox = () => {**

**return this.state.show ? <div className="box"></div> : null;**

**}**

**const buttonTxt = () => {**

**return this.state.show ? "按我下树" : "按我上树";**

**}**

return <div>

<button onClick={this.changeshu.bind(this)}> **{buttonTxt()}** </button>

**{showBox()}**

</div>

}

## 6.2 行内样式

比如现在state中，有w、h表示宽度、高度：

this.state = {

"w" : 600,

"h" : 300

}

如何和行内样式对应呢？套路是：

render() {

return <div>

<div

className="box"

**style={{"width" : this.state.w + "px" , "height" : this.state.h + "px"}}**

></div>

</div>

}

style属性写双大括号，里面是标准JSON。

● 显示和隐藏，此时可以在双大括号的JSON中，写三元运算符。

<div

className="box"

style={{

"width" : this.state.w + "px" ,

"height" : this.state.h + "px" ,

"display" : **this.state.show ? "block" : "none"**

}}

></div>

● 提供一个文本框供设置宽度、高度。

想要得到文本框的内容，文本框必须有ref属性，用this.refs得到。

<input type="text" ref="wTxt" />

按钮的监听：

//设置宽度

setW(){

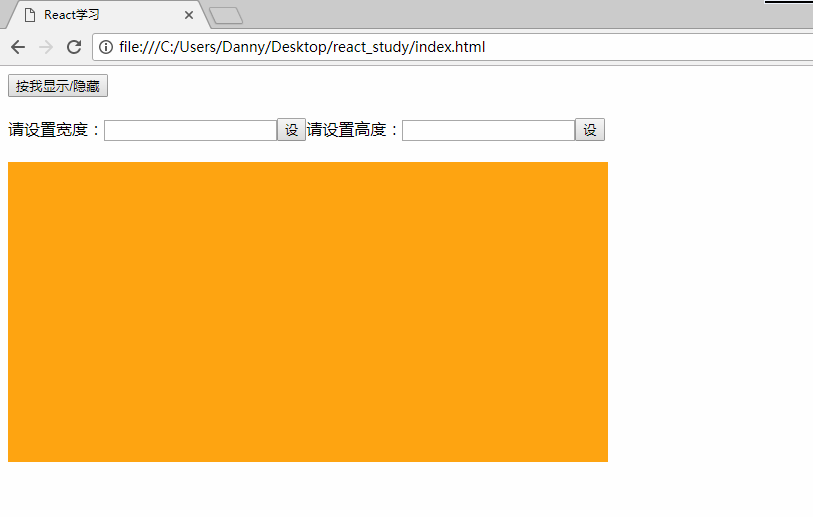
//想要得到文本框的内容，文本框必须有ref属性，这里用this.refs得到

this.setState({

"w" : **this.refs.wTxt.value**

});

}



## 6.3 类名

安装依赖

cnpm install --save classnames

比如现在有一个state里面的true和false决定了盒子的类名：

this.state = {

"xie" : true ,

"cu" : false,

"xian" : true

}

此时这样写：

<p

className={

classnames({

"xie" : this.state.xie ,

"cu" : this.state.cu ,

"xian" : this.state.xian

})

}

>文字</p>

值是true那项就能成为类名。



## 6.4 和数组相关的东西

this.state = {

"students" : [

{"id" : 10001 , "name":"小明"},

{"id" : 10002 , "name":"小花"},

{"id" : 10003 , "name":"小强"},

{"id" : 10004 , "name":"小刚炮"}

]

}

呈递数组，通常用map：

render() {

return <div>

<ul>

**{**

**this.state.students.map((item , index)=>{**

**return <li key={index}>{item.name}</li>**

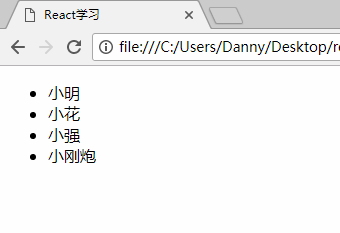
**})**

**}**

</ul>

</div>

}



所有的push、unshift、pop、shift、splice一律不能用了，因为他们不触发视图的更新！

所以要用我们ES6将map、reduce、filter的时候那些训练的东西。

通过参数删除：

render() {

return <div>

<input type="text" ref="nameTxt"/>

<button onClick={this.zeng.bind(this)}>增</button>

<ul>

{

this.state.students.map((item , index)=>{

return <li key={index}>

<b>{item.id}</b> {item.name}

<button **onClick={()=>{this.shan(item.id)}}**>删</button>

</li>

})

}

</ul>

</div>

}