React 是一个用于构建用户界面的 JAVASCRIPT 库。React 主要用于构建UI，很多人认为 React 是 MVC 中的 V（视图）。

React的特点：

1. 声明式设计 −React采用声明范式，可以轻松描述应用。
2. 高效 −React通过对DOM的模拟，最大限度地减少与DOM的交互。
3. 灵活 −React可以与已知的库或框架很好地配合。
4. JSX − JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX ，但我们建议使用它。
5. 组件 − 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。
6. 单向响应的数据流 − React 实现了单向响应的数据流，从而减少了重复代码，这也是它为什么比传统数据绑定更简单。

元素是构成React应用的最小单位，它用于描述屏幕上输出的内容

Jsx是一个看起来像XML的JavaScript语法扩展，它的优点：

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

再添加自定义属性时需要用到data-前缀：

例如<p data-myattribute = "somevalue">这是一个很不错的 JavaScript 库!</p>

在jsx中使用表达式时，表达是写在{}中，例如

<p>{1+1}</p>

Jsx中不能使用if else语句，但可以使用conditional（三元运算）来代替，列如

<h1>{i == 1 ? 'True!' : 'False'}</h1>

标签内注释要写在{}中，标签外的不能写在{}， 例如

ReactDOM.render(

/\*注释\*/

<div> <h1>菜鸟教程</h1>

{/\*注释...\*/}

</div>,

document.getElementById('example') );

Jsx允许在模板中插入数组，数组会自动展开所有成员，例如

var arr = [ <h1>菜鸟教程</h1>,

<h2>学的不仅是技术，更是梦想！</h2>, ];

ReactDOM.render(

<div>{arr}</div>,

document.getElementById('example') );

在显示时会自动的展开数组成员

使用内联样式容易出错的地方：

错误的方式：

ReactDOM.render(

<h1 style = {fontSize:12}>菜鸟教程</h1>,

document.getElementById('example'));

正确的方式：

ReactDOM.render(

<h1 style = {{fontSize:12}}>菜鸟教程</h1>,

document.getElementById('example'));

组件：

可以使用函数定义组件，例如

function HelloWorld(props){

return <p>Hello world</p>

}

也可以使用ES6 class定义组件，例如

class HelloWorld extends React.Component{

render(){

return <p>Hello World</p>

}

}

ReactDOM.Render(template,targetDOM)方法接受两个参数

第一个是创建的模板，多个 dom 元素外层需使用一个标签进行包裹，如 <div>；

第二个参数是插入该模板的目标位置。

当使用ES6 class定义的组件的调用过程：

1. 当组件名传递给ReactDOM.Render()函数，调用组件的构造函数设置state
2. React调用render()函数，这是 React 了解屏幕上应该显示什么内容，然后 React 更新 DOM 以匹配 Clock 的渲染输出。
3. 当组件的输出传到DOM中，进行挂载，调用并执行相应的方法（生命周期钩子）。
4. 当组件从DOM中移除时，调用钩子函数将组件卸载。

（不太懂）通过state和props实现父子模块间的通信

React事件处理：

1. React时间绑定属性的命名采用驼峰法
2. 如果采用jsx的语法你需要传入一个函数作为事件处理函数，而不是一个字符串

HTML写法：

<button onclick="activateLasers()">激活按钮</button>

React写法：

<button onclick={activateLasers}>激活按钮</button>

在 React 中另一个不同是你不能使用返回 **false** 的方式阻止默认行为， 你必须明确的使用 preventDefault。

条件渲染：

在 JavaScript 中，true && expression 总是返回 **expression**，而 false && expression 总是返回 **false**。因此，如果条件是 **true**，&& 右侧的元素就会被渲染，如果是 **false**，React 会忽略并跳过它。列如

<div>

<h1>Hello!</h1>

{unreadMessages.length > 2 &&

<h2> 您有 {unreadMessages.length} 条未读信息。 </h2>

}

</div>

其中unreadMessages为一个数组，当数组中的元素个数大于2时，才会显示h2的信息。

组件的生命周期可分成三个状态：

* Mounting：已插入真实 DOM
* Updating：正在被重新渲染
* Unmounting：已移出真实 DOM

生命周期的方法有：

* componentWillMount 在渲染前调用,在客户端也在服务端。
* componentDidMount : 在第一次渲染后调用，只在客户端。之后组件已经生成了对应的DOM结构，可以通过this.getDOMNode()来进行访问。 如果你想和其他JavaScript框架一起使用，可以在这个方法中调用setTimeout, setInterval或者发送AJAX请求等操作(防止异步操作阻塞UI)。
* componentWillReceiveProps 在组件接收到一个新的 prop (更新后)时被调用。这个方法在初始化render时不会被调用。
* shouldComponentUpdate 返回一个布尔值。在组件接收到新的props或者state时被调用。在初始化时或者使用forceUpdate时不被调用。   
  可以在你确认不需要更新组件时使用。
* componentWillUpdate在组件接收到新的props或者state但还没有render时被调用。在初始化时不会被调用。
* componentDidUpdate 在组件完成更新后立即调用。在初始化时不会被调用。
* componentWillUnmount在组件从 DOM 中移除之前立刻被调用。

箭头函数的特点：

* 更简洁的语法
* 没有this
* 不能使用new 构造函数
* 不绑定arguments，用rest参数...解决
* 使用call()和apply()调用
* 捕获其所在上下文的 this 值，作为自己的 this 值
* 箭头函数没有原型属性
* 不能简单返回对象字面量
* 箭头函数不能当做Generator函数,不能使用yield关键字
* 箭头函数不能换行

//1、没有形参的时候

let fun = () => console.log('我是箭头函数');

fun();

//2、只有一个形参的时候()可以省略

let fun2 = a => console.log(a);

fun2('aaa');

//3、俩个及俩个以上的形参的时候

let fun3 = (x,y) =>console.log(x,y); //函数体只包含一个表达式则省略return 默认返回

fun3(24,44);

//4、俩个形参以及函数体多条语句表达式

let fun4 = (x,y) => {

　　console.log(x,y);

　　return x+y; //必须加return才有返回值

}

//5、如果要返回对象时需要用小括号包起来，因为大括号被占用解释为代码块了，正确写法

let fun5 = ()=>({ foo: x }) //如果x => { foo: x } //则语法出错