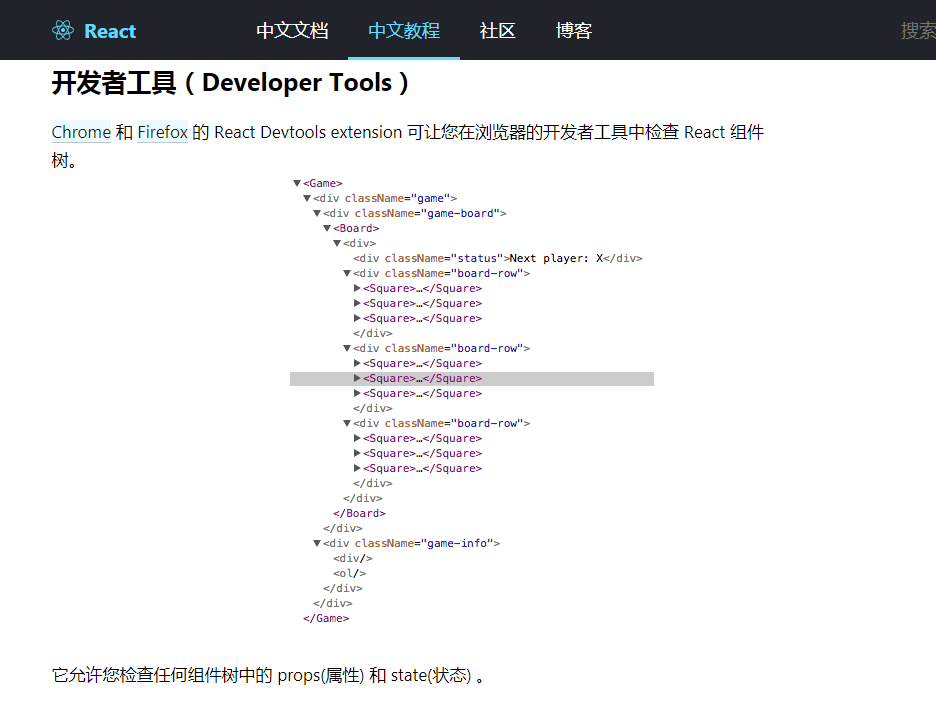
React语法

1. 对棋盘游戏的理解过程
2. 这里使用新的 JavaScript [箭头函数](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Arrow_functions) 语法。 请注意，我们传递一个函数作为 onClick prop(属性)。 如果我们把 onClick={() => alert('click')} 改成 onClick={alert('click')} ，那么不用点击按钮就会立即 alert 。
3. ，构造函数应该被认为是组件的私有。
4. 未完成



1. **当您要聚合来自多个子节点的数据 或 使两个子组件之间相互通信时，提升 state(状态) ，使其存储在父组件中。父组件可以通过 props(属性) 把 state(状态) 传递回子组件，以使子组件始终与父组件同步。**
2. 我们将返回的元素拆分成多行以提高可读性，并在其两边加上括号，这样 JavaScript 就不会在 return 后插入分号并且破坏我们的代码了。

--------------这就是为啥要加()的原因啊

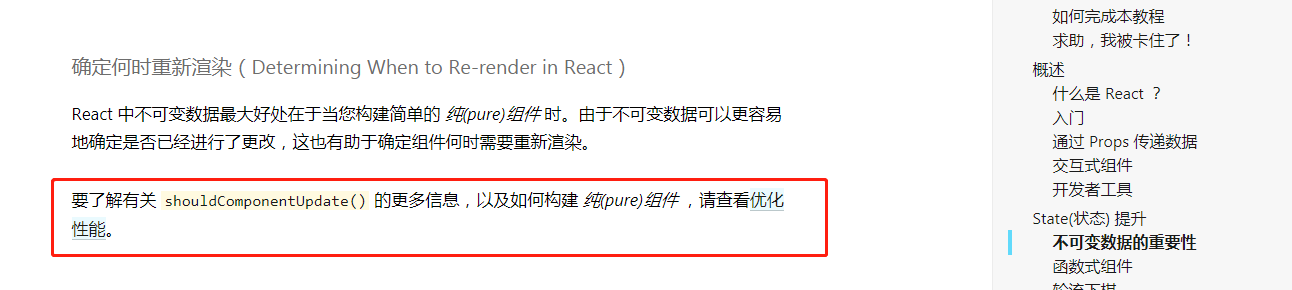
1. **状态提升**的通常的模式是将一个函数从 Board(棋盘) 组件 传递到 Square(方格) 组件上，该函数在方格被点击时调用。 再次修改 Board(棋盘) 组件中 renderSquare ，修改为：

* 7. 从 Square(方格) 组件删除 constructor(构造函数) 定义，因为它不再需要 state(状态) 。

8. 确定可变对象是否已更改是复杂的，因为直接对对象进行更改。这样就需要将当前对象与先前的副本进行比较，遍历整个对象树，并比较每个变量和值。这个过程可能变得越来越复杂。

确定不可变对象如何改变是非常容易的。如果被引用的对象与之前不同，那么对象已经改变了。仅此而已。

9.没有看



10.函数式组件和类组件的区别：

我们已经删除了构造函数，实际上，React 支持一种简单的语法，称为 **函数式组件** ，这是从组件类型方面归类的，例如我们示例中的 Square(方格) 组件，它只包含一个 render 方法。而不是通过扩展 React.Component 类定义的，**函数式组件**只需编写一个函数，传入 props(属性) 并返回应该渲染的内容就可以了。

所以这时候就不需要this.props.value了，因为都是由父组件穿过来的，而不是自己的，所以没有this一说

好好理解：



**2018.6.20**

1.关于ref的理解

ReactDOM.render()渲染组件时返回的是组件实例；而渲染dom元素时，返回是具体的dom节点。

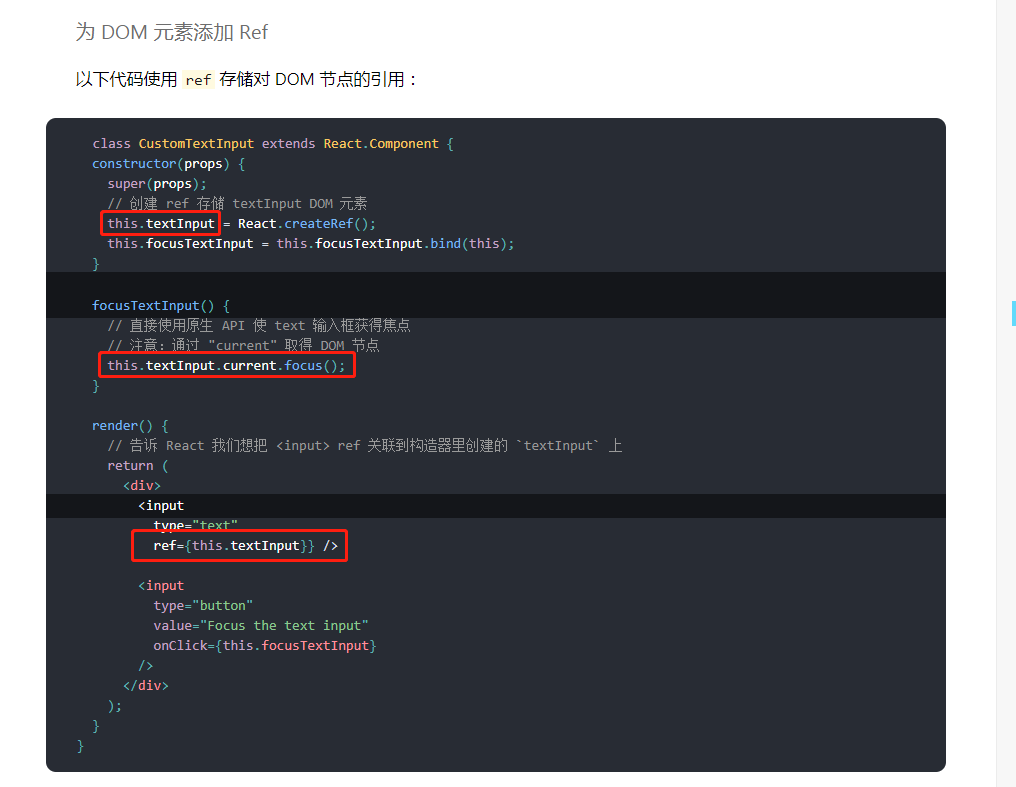
ref可以挂到任何组件上，可以挂到组件上也可以是dom元素上；二者不同是与上图答案一样：

挂到组件（这里组件指的是有状态组件）上的ref表示对组件实例的引用，而挂载到dom元素上时表示具体的dom元素节点。

2. ref的作用就是：

（1）在组件自己内部的任何函数内 访问自己的某个属性。

使用createRef() 创建ref 属性，再使用 ref 中的 current 属性对节点的引用进行访问



（2）访问子组件（图中的内容接上图）

这里就通过红框中的语句访问了子组件的函数focusTextInput（这里父和子都用createRef来创建了ref属性）



注意：这种向子组件添加 ref，来访问子组件的方法虽然可以运行，但这不是一个理想的解决方案，因为你只能获取组件实例而不是 DOM 节点。并且，它还在函数式组件上无效。

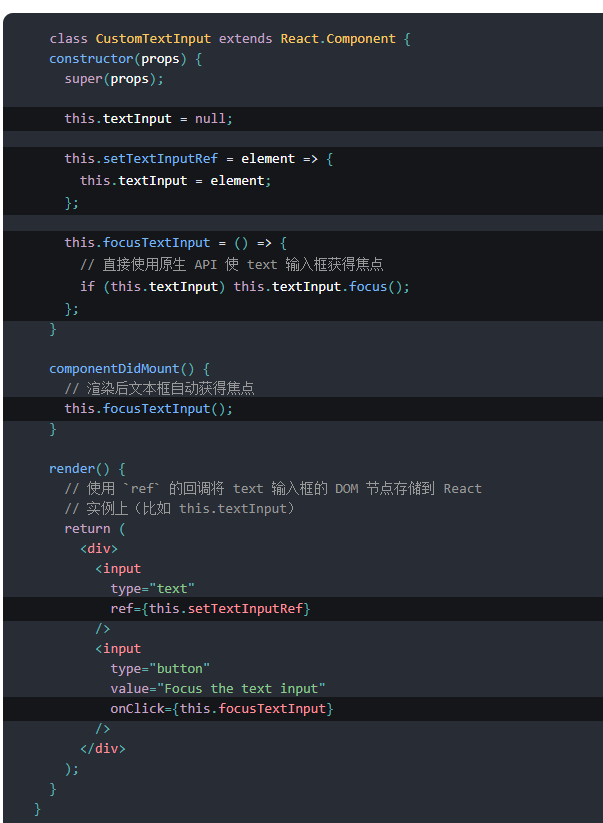
因此，改进方式如下：

如果你使用 React 16.3 或更高, 这种情况下我们推荐使用 ref 转发。

如果你使用 React 16.2 或更低，或者你需要比 ref 转发更高的灵活性，你可以使用 这个替代方案 将 ref 作为特殊名字的 prop 直接传递。

可能的话，我们不建议暴露 DOM 节点，但有时候它会成为救命稻草。注意以上这些方案需要你在子组件中增加一些代码。如果你对子组件的实现没有控制权（无法编辑组件的代码）的话，你剩下的选择是使用 findDOMNode()，但是不推荐。

（3）后面这两种（3）和（4）都是使用回调的形式来达到目的，就不使用createRef()来创建ref了。这个回调函数接受 React 组件的实例或 HTML DOM 元素作为参数，以存储它们并使它们能被其他地方访问



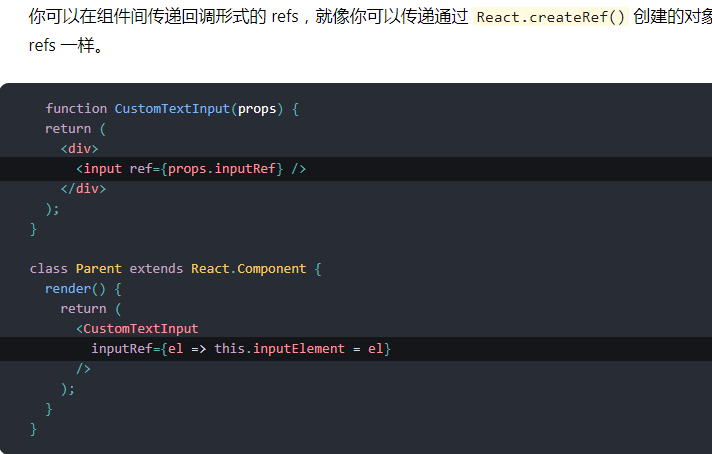
//这个表示能通过回调函数的方式访问自己本组件，让其聚焦。这个的方法和（1）差不多，但这个不是通过React.createRef()创建的，是将自身的引用作为参数传入回调函数

//ref 回调函数会在 componentDidMout 和 componentDidUpdate 生命周期函数前被调用

//记得要声明一句：this.textInput = null; 这样ref回调才可以引用它

（4）在父子组件间传递回调形式的 refs，就像你可以传递通过 React.createRef() 创建的对象 refs 一样，达到父组件可以访问子组件，根据子组件的变化更新自己的目的

方法是将父的回调函数作为参数传递给子组件。（这个就和父子组件通信的子向父传递数据的那个方法一样的）



在上面的例子中，Parent 将它的 ref 回调函数作为 inputRef 传递给 CustomTextInput，然后 CustomTextInput 通过 ref属性将其传递给 <input>。最终，Parent 中的 this.inputElement 将被设置为与 CustomTextIput 中的 <input> 元素相对应的 DOM 节点

（即子组件向父组件通信了）

//记得要声明一句：this.textInput = null; 这样ref回调才可以引用它

综上所述：用回调函数的方式比较好，即（3）（4）

例子：<https://blog.csdn.net/Jingle_cjy/article/details/78804139>

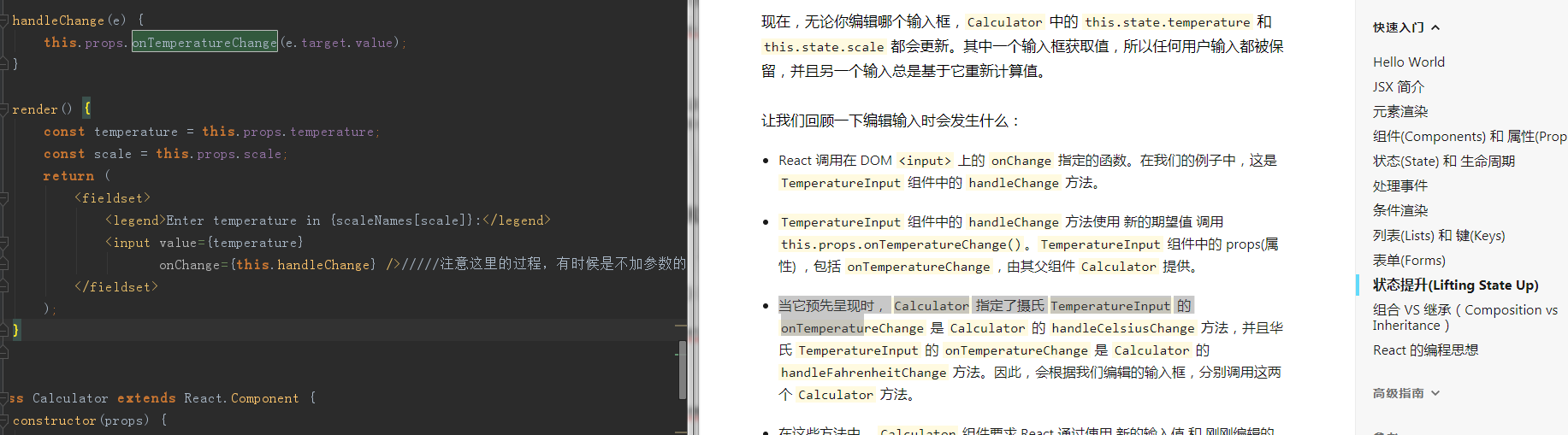
2.自己的理解：如果是在该render函数里定义的函数或变量，则不用this. 而只要超过这个render函数的范围则要加this.

//状态提升

3. 我们知道 [props(属性) 是只读的](http://www.css88.com/react/docs/components-and-props.html#props-are-read-only)

当 temperature 是 本地 state(状态)时， TemperatureInput 可以调用 this.setState() 来更改它。 然而，现在 temperature 来自父级作为 prop(属性) ，TemperatureInput 就无法控制它。当我们想要更改时， 不是调用 this.setState() ，而是调用 this.props.onTemperatureChange()， 这将由 Calculator 提供：

要进一步理解：



4.关于状态提升：

在一个 React 应用中，对于任何可变的数据都应该循序“单一数据源”原则。通常情况下，state 首先被添加到需要它进行渲染的组件。然后，如果其它的组件也需要它，你可以提升状态到它们最近的祖先组件。你应该依赖 [从上到下的数据流向](http://www.css88.com/react/docs/state-and-lifecycle.html#the-data-flows-down) ，而不是试图在不同的组件中同步状态。

5.看react代码时首先看一下是类还是函数定义的

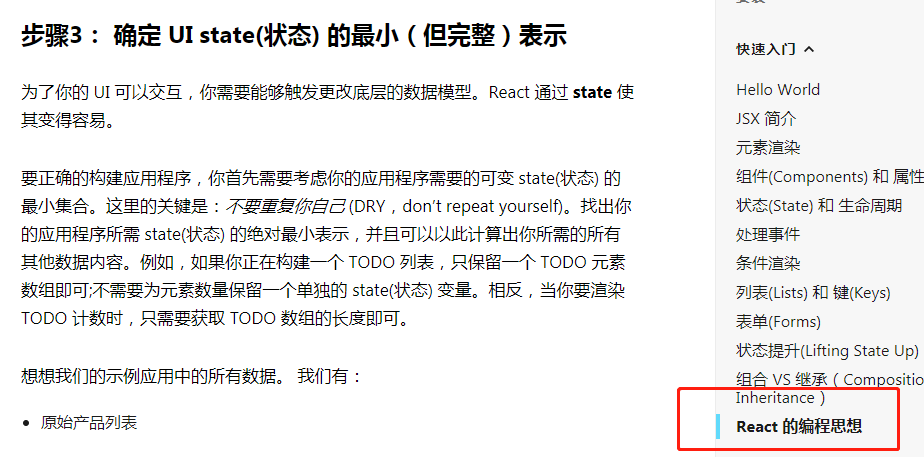
1. 如果要在组件之间重用非 U I功能，我们建议将其提取到单独的 JavaScript 模块中。组件可以导入它并使用该函数，对象或类，而不扩展它。
2. props 是将数据从 父级组件 传递到 子级 的一种方式。

8、在这一步结束时，你已经有了一个可重用的组件库，用于渲染你的数据模型。组件将只有 render() 方法，因为这是你应用程序的静态版本。层次结构顶部的组件（ FilterableProductTable ）应该接收你的数据模型作为 prop 。如果您对基础数据模型进行更改，并再次调用 ReactDOM.render()，UI 将同步更新。这有利于观测UI的更新以及相关的数据变化，因为这中间没有做什么复杂的事情。React 的 **单向数据流**（也称为 单向绑定 ）使所有模块化和高性能。

1. 待理解 ：



1. 记住这里的内容：很重要



10.

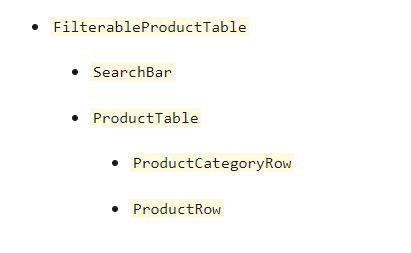
现在，已经确定了应用所需 state(状态) 的最小集合。接下来，需要确定是哪个组件可变，或者说哪个组件*拥有*这些 state(状态) 。

记住：React 单向数据流在层级中自上而下进行。这样有可能不能立即判断出状态属于哪个组件。**这常常是新手最难理解的一部分，**试着按下面的步骤分析操作：

对于你应用中的每一个 state(状态) ：

* 确定每个基于这个 state(状态) 渲染的组件。
* 找出公共父级组件（一个单独的组件，在组件层级中位于所有需要这个 state(状态) 的组件的上面。愚人码头注：父级组件）。
* 公共父级组件 或者 另一个更高级组件拥有这个 state(状态) 。
* 如果找不出一个拥有该 state(状态) 的合适组件，可以创建一个简单的新组件来保留这个 state(状态) ，并将其添加到公共父级组件的上层即可。

11.



12.react是单向的数据流

**13.什么时候需要constructor(props) ?**

我的理解是：当这个类里有自己定义的函数时就需要，若没有的话，直接使用父组件传过来的props就可以了，不需要constructor(props);

https://blog.csdn.net/xiaozidewawa/article/details/77892762

<https://www.jianshu.com/p/26c63a17e362>

https://blog.csdn.net/gx15366039985/article/details/53447098

https://blog.csdn.net/chjj0904/article/details/79291117

**继承：**

super()是继承了整个类的一个引用，super关键字；它在这里表示父类的构造函数，用来新建父类的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；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。

ES6；实质是先创造父类的实例对象this;然后再用子类的构造函数修改this;

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

在es6中实现继承，直接调用super(name)，就可以直接继承父类的属性和方法，所以super作用就相当于上述的实现继承的步骤，不过es6提供了super语法糖，简单化了继承的实现。

如果你用到了constructor就必须写super(),是用来初始化this的

如果你在constructor中要使用this.props,就必须给super加参数：super(props)

无论有没有constructor，在render中this.props都是可以使用的，这是React自动附带的，如果没用到constructor,是可以不写的,不过这种只是用render的情况，使用一般的ES6函数写会更简便

简单解释，就是在ES5的继承中，先创建子类的实例对象this，然后再将父类的方法添加到this上（ Parent.apply(this) ）。而ES6采用的是先创建父类的实例this（故要先调用 super( )方法），完后再用子类的构造函数修改this。

**14.高级指引**

如果不存在子节点，你可以使用自闭合（self-closing）格式的标签

15. 一个 JSX 标签的开始部分决定了 React 元素的类型。

首字母大写的标签指示 JSX 标签是一个 React 组件。

**用户定义组件必须以大写字母开头**

当一个元素类型以小写字母开头，它表示引用一个类似于 <div> 或者 <span> 的内置组件，会给 React.createElement 方法传递 'div' 或者 'span' 字符串。以大写字母开头的类型，类似于 <Foo />，会被编译成 React.createElement(Foo) ，对应于自定义组件 或者在 JavaScript 文件中导入的组件。

16.好好理解：这样也是可以用的



17. 不能使用一个普通的表达式作为 React 元素类型。如果你想使用普通表达式来表示元素类型，首先你需要将其赋值给大写的变量。这通常会出现在根据不同的 props 渲染不同的组件时：

在 JavaScript 中，if 语句和 for 循环不是表达式，因此不能在 JSX 中直接使用。但你可以将他们放在附近的代码块中，例如:

18．

**props(属性) 默认为 “true”**

如果你没给 prop(属性) 传值，那么他默认为 true 。下面两个 JSX 表达式是等价的：

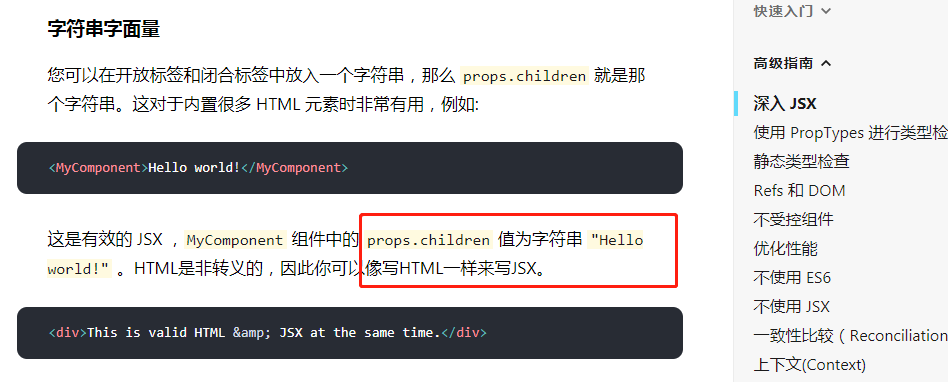
19.

**属性扩展**

如果你已经有一个 object 类型的 props，并且希望在 JSX 中传入，你可以使用扩展操作符 ... 传入整个 props 对象。这两个组件是等效的：

当你构建一个一般容器时，属性扩展非常有用。然而，这可能会使得你的代码非常混乱，因为这非常容易使一些不相关的 props(属性) 传递给组件，而组件并不需要这些 props(属性) 。因此我们建议谨慎使用该语法。

20.

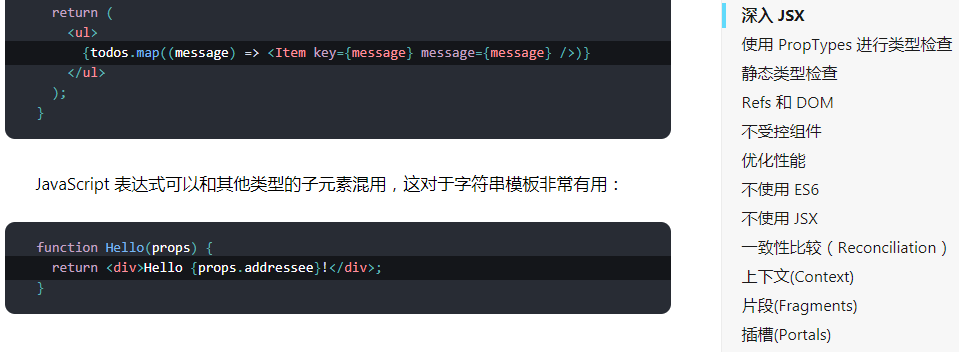


21. JSX会删除每行开头和结尾的空格，并且也会删除空行。邻接标签的空行也会被移除，字符串之间的空格会被压缩成一个空格，因此下面的渲染效果都是相同的：

22.

React组件不能返回多个React元素，但是单个JSX表达式可以有多个子元素，因此如果你想要渲染多个元素，你可以像上面一样，将其包裹在 div 中。

23.



24.

**Functions(函数) 作为 Children(子元素)**

通常情况下，嵌入到 JSX 中的 JavaScript 表达式会被认为是一个字符串、React元素 或者是这些内容的一个列表。然而， props.children 类似于其他的 props(属性) ，可以被传入任何数据，而不是仅仅只是 React 可以渲染的数据。例如，如果有自定义组件，其 props.children 的值可以是回调函数：

25.



26.

**Booleans, Null, 和 Undefined 被忽略**

false，null，undefined，和 true 都是有效的的 children(子元素) 。但是并不会被渲染，下面的JSX表达式渲染效果是相同的：

使用 propTypes进行类型检测

要在组件中进行类型检测，你可以赋值 propTypes 属性。

注意：使用propTypes时记得添加引入：

import PropTypes from 'prop-types';

27.

这个的作用就是定义props的默认值

你可以通过赋值特定的 defaultProps 属性为 props 定义默认值：

对于更大的代码库我们建议使用 Flow 或者 TypeScript 来替代 PropTypes。

**静态类型检查：**

28.

默认情况下， Flow 仅检查包含此批注的文件：

所以，react项目里就算是注释也不能乱删除！

29.

TypeScript

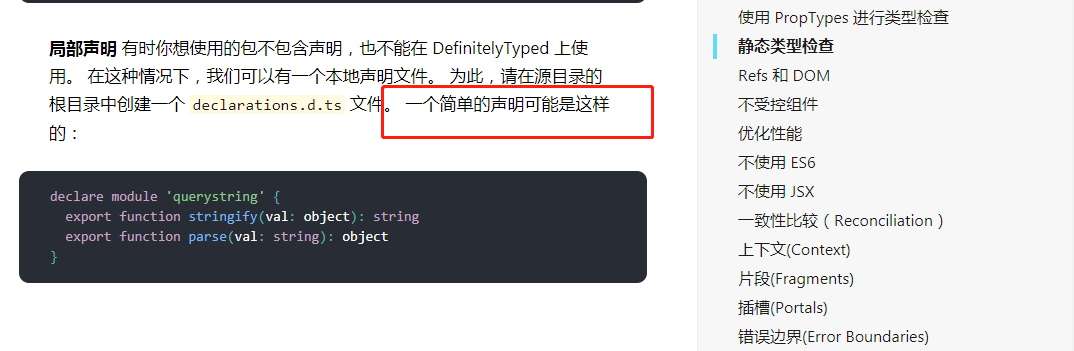
[TypeScript](https://www.typescriptlang.org/) 是一门由微软开发的编程语言。 它是 JavaScript 的一个类型超集，包含它自己的编译器。 作为一种类型化语言，Typescript 可以早在您的应用程序上线之前在构建时发现错误和错误。 你可以在[这里](https://github.com/Microsoft/TypeScript-React-Starter#typescript-react-starter)了解更多关于在 React 中使用 TypeScript 的知识。

编译器将以真实的情况接收 typescript 文件然后生成 javascript 文件。

30.

为了能够显示来自其他包的错误和提示，编译器依赖于声明文件。 声明文件提供了关于库的所有类型信息。 这使我们能够在我们的项目中使用像那些在 npm 中的 JavaScript 库。

31.声明文件看不懂：



Refs和DOM

1. 在常规的 React 数据流中，[props](http://www.css88.com/react/docs/components-and-props.html) 是父组件与子组件交互的唯一方式。要修改子元素，你需要用新的 props 去重新渲染子元素。然而，在少数情况下，你需要在常规数据流外强制修改子元素。被修改的子元素可以是 React 组件实例，或者是一个 DOM 元素。在这种情况下，React 提供了解决办法。

### 2. 不要过度使用 Refs

你可能首先会想到在你的应用程序中使用 refs 来更新组件。如果是这种情况，请花一点时间，更多的关注在组件层中使用 state。在组件层中，通常较高级别的 state 更为清晰。有关示例，请参考[状态提升](http://www.css88.com/react/docs/lifting-state-up.html).

### 3.何时使用 Refs

下面有一些正好使用 refs 的场景:

* 处理focus、文本选择或者媒体播放
* 触发强制动画
* 集成第三方DOM库

如果可以通过声明式实现，就尽量避免使用 refs 。

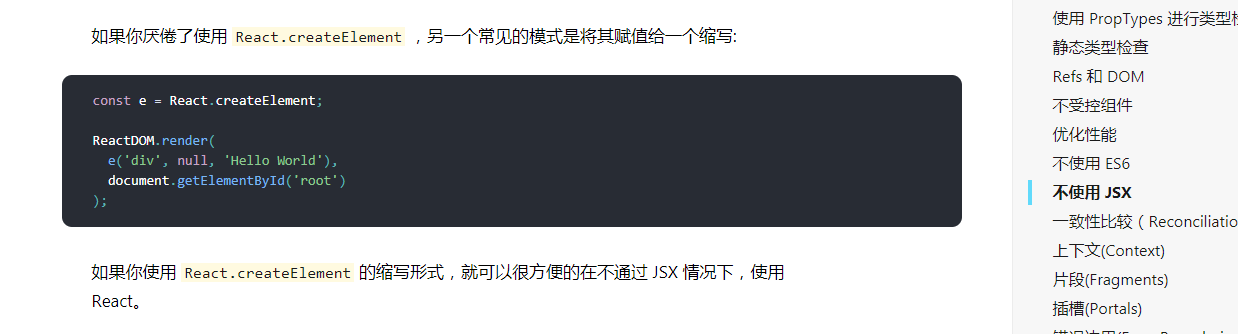
例如，相比于在 Dialog 组件中暴露 open() 和 close() 方法，最好传递 isOpen 属性。

3.

React 支持给任何组件添加特殊属性。ref 属性接受回调函数，并且当组件 装载(mounted) 或者 卸载(unmounted) 之后，回调函数会立即执行。

4. JSX 对使用React 不是必须的

每一个 JSX 元素都是调用 React.createElement(component, props, ...children) 的语法糖，因此，任何你使用 JSX 来做事都可以通过纯 JavaScript 实现。



**5这个没有理解：**



第二遍理解

1. this.props 由 React 本身设定, 而 this.state 具有特殊的含义，但如果需要存储一些不用于视觉输出的内容，则可以手动向类中添加额外的字段。

如果在 render() 方法中没有被引用, 它不应该出现在 state 中。

### 2. state(状态) 更新可能是异步的

React 为了优化性能，有可能会将多个 setState() 调用合并为一次更新。

因为 this.props 和 this.state 可能是异步更新的，你不能依赖他们的值计算下一个state(状态)。



1. this的问题



1. 箭头函数这个语法在 创建 React App中是默认开启的，但不知道es6是不是默认开启的
2. render里的方法就不用this是吗？
3. 



* 6. 事件处理
* React 事件使用驼峰命名，而不是全部小写。
* 通过 JSX , 你传递一个函数作为事件处理程序，而不是一个字符串。
* 另一个区别是，在 React 中你不能通过返回 false（即 return false语句） 来阻止默认行为。必须明确调用 preventDefault

1. react是什么模式
2. 应该都不算，只是view,是单向数据流的

https://segmentfault.com/q/1010000008335416

1. 我们都不会直接使用React 元素，React 内部会自动根据React 元素，渲染出最终的页面DOM。更确切地说，React元素描述的是React虚拟DOM的结构，React会根据虚拟DOM渲染出页面的真实DOM
2. **实例 (Instance）**

这里的实例特指React组件的实例。React 组件是一个函数或类，实际工作时，发挥作用的是React 组件的实例对象。只有组件实例化后，每一个组件实例才有了自己的props和state，才持有对它的DOM节点和子组件实例的引用。在传统的面向对象的开发方式中，实例化的工作是由开发者自己手动完成的，但在React中，组件的实例化工作是由React自动完成的，组件实例也是直接由React管理的。换句话说，开发者完全不必关心组件实例的创建、更新和销毁。

1. super()在子类constructor构造方法中是为了获取this上下文环境,所以如果在constructor中使用到this,必须在使用this之前调用super(),反之不在constructor中使用this则不必调用super()，这样子类才继承了父类的方法，在虚拟子类DOM中调用 this+ 父类方法。

11. 在class方法中，继承是使用 extends 关键字来实现的。子类 必须 在 constructor( )调用 super( )方法，否则新建实例时会报错。

     报错的原因是：子类是没有自己的 this 对象的，它只能继承自父类的 this 对象，然后对其进行加工，而super( )就是将父类中的this对象继承给子类的。没有 super，子类就得不到 this 对象

12.

先看ref，第一点就有问题，官方文档说的很清楚，ref添加到Compoennt上获取的是Compoennt实例，添加到原生HTML上获取的是DOM

13. React提供的这个ref属性，**表示为对组件真正实例的引用，其实就是ReactDOM.render()返回的组件实例**；需要区分一下，ReactDOM.render()渲染组件时返回的是组件实例；而渲染dom元素时，返回是具体的dom节点。

挂到组件（这里组件指的是有状态组件）上的ref表示对组件实例的引用，而挂载到dom元素上时表示具体的dom元素节点。

在react典型的数据流中，props传递是父子组件交互的唯一方式；通过传递一个新的props值来使子组件重新re-render,从而达到父子组件通信。当然，就像react官网所描述的一样，在react典型的数据量之外，某些情况下（例如和第三方的dom库整合，或者某个dom元素focus等）为了修改子组件我们可能需要另一种方式，这就是ref方式。

### ref 简介

React提供的这个ref属性，**表示为对组件真正实例的引用，其实就是ReactDOM.render()返回的组件实例**；需要区分一下，ReactDOM.render()渲染组件时返回的是组件实例；而渲染dom元素时，返回是具体的dom节点。

<https://segmentfault.com/a/1190000008665915>

使用 ref 回调只是为了在类上设置一个属性，是访问 DOM 元素的常见模式。首选的方法是在 ref 回调中设置属性 就像上面的例子一样

16.组件实例，虚拟DOM,真实DOM

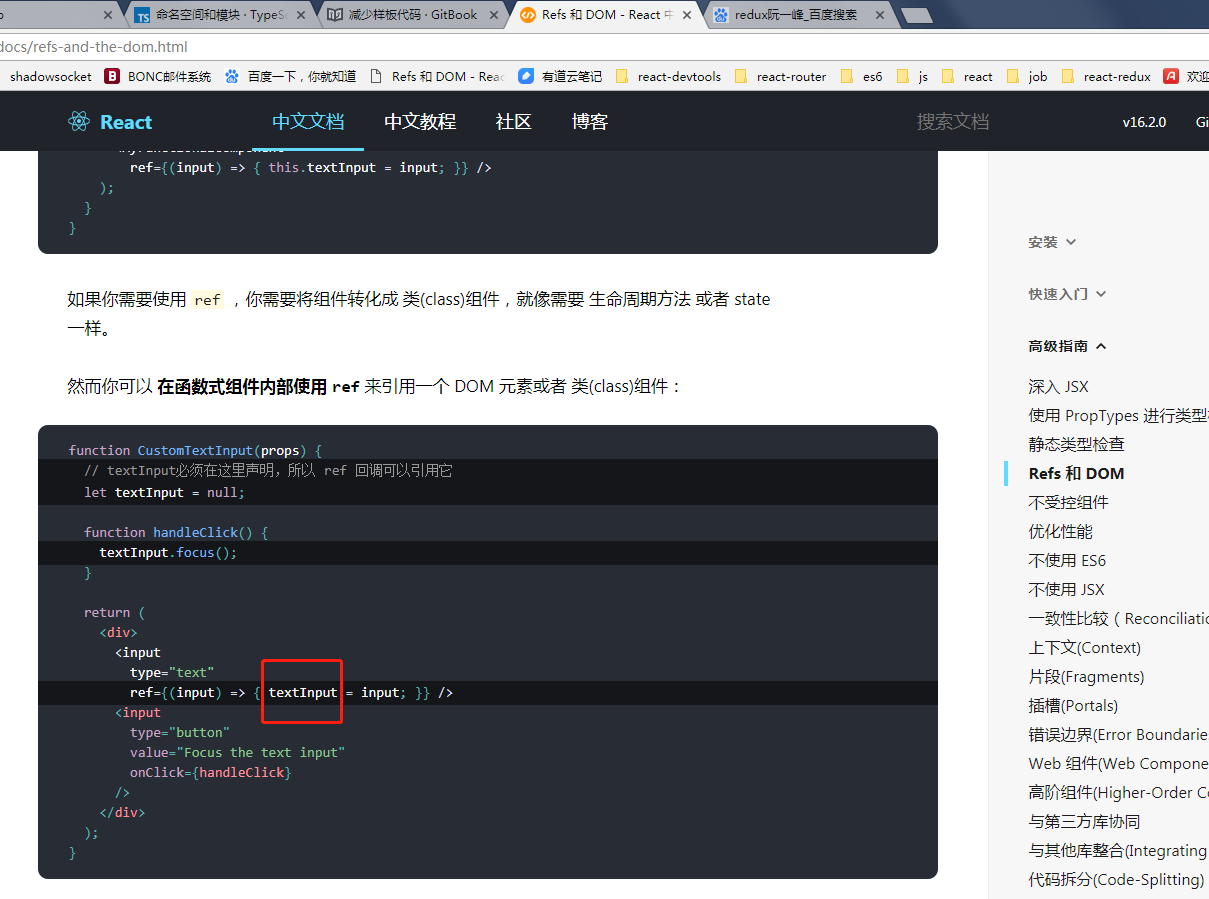
1. React里的虚拟元素什么意思

https://blog.csdn.net/zhouziyu2011/article/details/71171567

12.许多例子的讲解：//一定记得看

<https://blog.csdn.net/u014388408/article/details/50595047>

13.为啥有的有this，有的没有？而且在函数式组件内部用ref时要声明，为啥在类组件里就不用声明？//对比14





14. react中若是function组件，则里面没有this,若是class组件，则使用this

参考这个示例代码：//这个的原因不知是不是和上面13的一样的

http://www.css88.com/react/tutorial/tutorial.html

2018.06.27

Ref

1. ref 属性接受回调函数，并且当组件 装载(mounted) 或者 卸载(unmounted) 之后，回调函数会立即执行
2. 在 DOM 元素上添加 Ref

React 组件在加载时将 DOM 元素传入 ref 的回调函数，在卸载时则会传入 null

当给 HTML 元素添加 ref 属性时， ref 回调接受底层的 DOM 元素作为参数。例如，下面的代码使用ref 回调来存储 DOM 节点的引用

1. 为 类(Class) 组件添加 Ref

当 ref 属性用于类(class)声明的自定义组件时，ref 回调函数收到的参数是装载(mounted)的组件实例 ！

1. Refs 与 函数式组件

你不能在函数式组件上使用 ref 属性，因为它们没有实例，但你可以 在函数式组件内部使用 ref 来引用一个 DOM 元素或者 类(class)组件，这时和类里面的就有区别了，需要声明变量了，在类里面是不需要的//why ?

2.父组件访问子节点

方法是：在子节点上暴露一个特殊的属性（inputRef）。子节点将会获得一个函数属性，并将其作为 ref 属性附加到 DOM 节点。这允许父代通过中间件将 ref 回调给子代的 DOM 节点。适用于类组件和函数式组件//但是注意：ref和inputRef属性都是写在子节点上的，但是位置不一样，特殊属性是在父组件里的子组件上写的，而ref是在子组件的定义里写的

//在上面的例子中，Parent 将它的 ref 回调作为一个特殊的 inputRef 传递给 CustomTextInput，然后 CustomTextInput 通过 ref 属性将其传递给 <input>。最终，Parent 中的 this.inputElement (这里是在父组件的子组件里添加的属性，但是里面的this.inputElement是父组件的，就像给props传递的数据一样)将被设置为与 CustomTextInput 中的 <input> 元素相对应的 DOM 节点（这大概就是目的了，达到了）

（2）祖父节点要访问孙子节点

这种模式的另一个好处是它能作用很深。假如有个 Parent 组件不需要 DOM 节点 A，但是某个渲染 Parent 的组件（我们称之为 Grandparent）需要通过它访问。这时我们可以让 Grandparent 传递 inputRef 给 Parent 组件，然后让 Parent 组件将其转发给 CustomTextInput:

ref其实就2种用法(但字符串的形式不推荐，要被废除了)、回调函数形式的三种触发情况

https://www.jianshu.com/p/b4b301562020

ref的作用就是父组件能或修改子组件访问子组件

3. 向子组件添加 ref, 只能获取组件实例而不是 真实的DOM 节点

4.总结：

1. 受控组件：props state

(2) 不受控组件：ref

2. findDOMNode()



1. 如果 ref 回调以内联函数的方式定义，在更新期间会被调用两次，第一次参数是 null ，之后参数是 DOM 元素。这是因为在每次渲染中都会创建一个新的函数实例。因此，React 需要清理旧的 ref 并且设置新的。通过将 ref 的回调函数定义成类的绑定函数的方式可以避免上述问题，但是在大多数例子中这都不是很重要
2. 未控制组件，你可以使用一个 ref 来从 DOM 获得 表单值，而不是为每个状态更新编写一个事件处理程序。，即使用ref就是不受控组件，因为不由父组件传props下去

不受控组件的数据来源是 DOM 元素

3. react里的类 实例 this 和function里为啥不加this,必须声明变量，二class里直接用this就可以了

1. 内联函数是指用inline关键字修饰的函数。在类内定义的函数被默认成内联函数
2. [类的绑定方法与](https://www.cnblogs.com/huang-yc/p/8992504.html)

优化性能

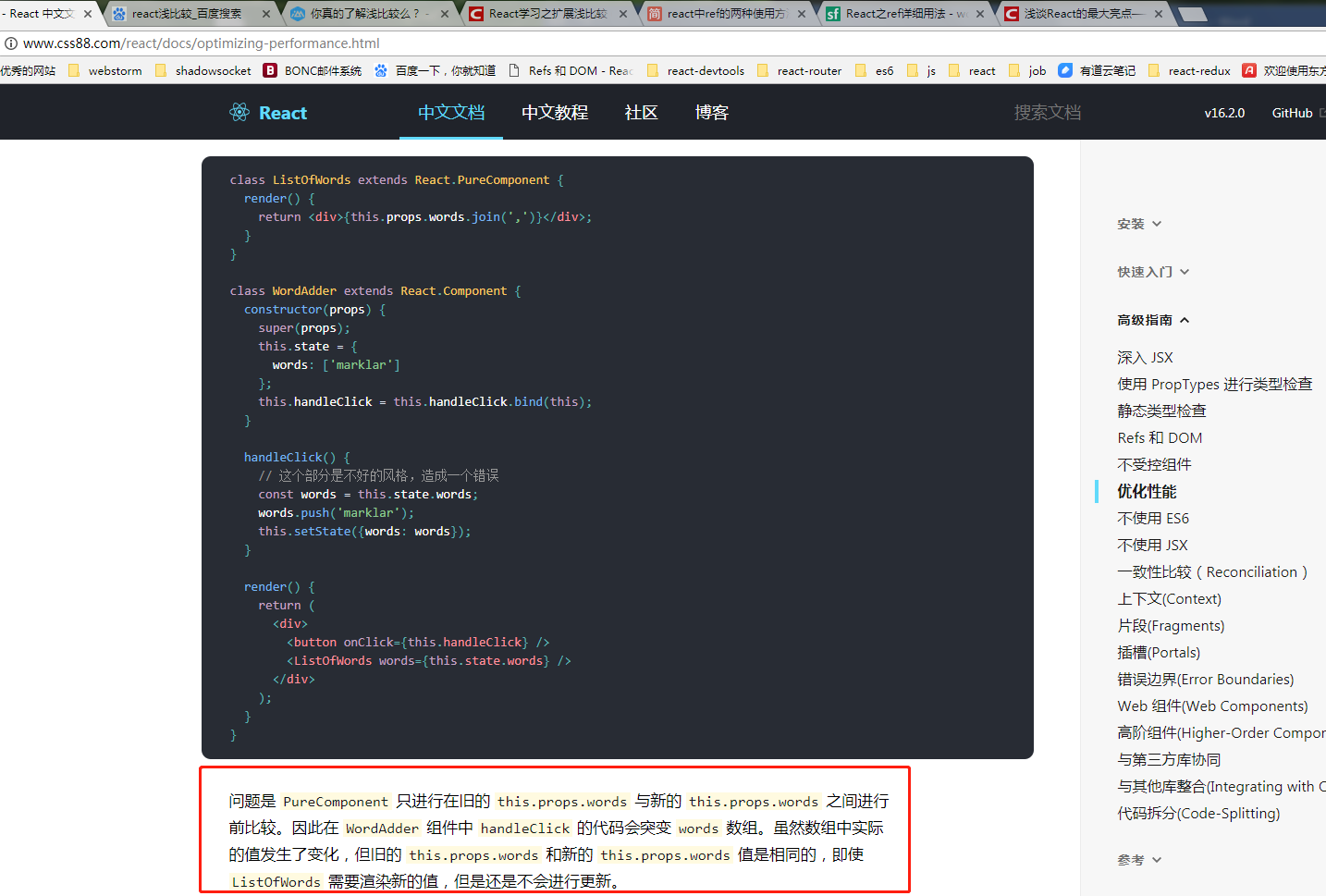
1. 因为对 JavaScript 对象的操作是比 DOM 操作更快。这被称为”虚拟DOM”，React Native 也是基于上述原理
2. 如果你想达到跳过整个渲染过程的目的，就要重写 渲染前的那个生命周期函数shouldComponentUpdate(), 不过在大多数情况下，您可以不用手写 shouldComponentUpdate() ，而是从 React.PureComponent 继承。 这相当于用当前和以前 props(属性) 和 state(状态) 的浅层比较来实现shouldComponentUpdate()
3. shouldComponentUpdate : 返回true或false, 表明是否要重新渲染

通过比较元素的state和props： 判断是否需要更新

1. 浅比较

<http://www.imweb.io/topic/598973c2c72aa8db35d2e291>

5．这段没看懂



还有这段， 这不是和有问题的那一部分一样的吗？换个写法就不是突变了？不直接赋值就不是突变？



还有这个：



1. 展开语法

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Spread_syntax>

1. 注意用法



1. 要注意



不使用ES6

1. 有的视频里的定义一个类和官网不同的，就是用的这种不使用es6，用js来写的

一致性比较

1. React 基于以下两个假设实现了时间复杂度为 O(n) 的算法:

不同类型的两个元素将会产生不同的树。

开发人员可以使用一个 key prop 来指示在不同的渲染中那个那些元素可以保持稳定。

2. 相同类型的组件

当一个组件更新的时候，组件实例保持不变，以便在渲染中保持state。React会更新组件实例的属性来匹配新的元素，并在元素实例上调用 componentWillReceiveProps() 和 componentWillUpdate()。

1. 子元素递归

添加元素时最好在末尾添加，这样性能更高

当给子元素末尾添加一个元素，在两棵树之间转化中性能就不错；如果在开始处插入一个节点也是这样简单地实现，那么性能将会很差

所以为了解决这个问题，添加一个key是好的做法

1. React.Fragment这个元素是用在tr td这种中间不能有其他组件，但使用中一个在父组件，一个在子组件的情况下。这时就要用到React.Fragment元素了

key 是唯一可以传递给 Fragment 的属性

插槽

1. 对于 portal 的一个典型用例是当父组件有 overflow: hidden 或 z-index 样式，但你需要子组件能够在视觉上 “跳出(break out)” 其容器。例如，对话框、hovercards以及提示框
2. 没看
3. componentWillMount: 在组件 render 之前执行且永远只执行一次。

componentDidMount: 组件加载完毕之后立即执行，并且此时才在 DOM 树中生成了对应的节点，因此我们通过 this.getDOMNode() 来获取到对应的节点。

4.

