## vue和react的异同

react和vue大体上是相同的，比如都使用[虚拟DOM](https://so.csdn.net/so/search?q=%E8%99%9A%E6%8B%9FDOM&spm=1001.2101.3001.7020" \t "/Users/gdj/Documents\\x/_blank)高效的更新视图，都提倡组件化，都实现了数据驱动视图，都使用diff算法，也都对diff算法进行了优化，都有router库实现url到组件的映射，都有状态管理等等

### 组件化的异同

相同点：react和vue都推崇组件化，通过将页面拆分成一个一个小的可复用单元来提高代码的复用率和开发效率。在开发时react和vue有相同的套路，比如都有父子组件传参，都有数据状态管理，都有前端路由等。

差异：

React推荐的做法是JSX + inline style, 也就是把 HTML 和 CSS 全都写进 JavaScript 中,即 all in js;

Vue 推荐的做法是 template 的单文件组件格式(简单易懂，从传统前端转过来易于理解),即 html,css,JS 写在同一个文件(vue也支持JSX写法)

### 2.虚拟DOM异同

相同点：

Vue与React都使用了 Virtual DOM + Diff算法， 不管是Vue的Template模板+options api 写法， 还是React的Class或者Function写法,最后都是生成render函数，而render函数执行返回VNode(虚拟DOM的数据结构，本质上是棵树)。

当每一次UI更新时，总会根据render重新生成最新的VNode，然后跟以前缓存起来老的VNode进行比对，再使用Diff算法（框架核心）去真正更新真实DOM（虚拟DOM是JS对象结构，同样在JS引擎中，而真实DOM在浏览器渲染引擎中，所以操作虚拟DOM比操作真实DOM开销要小的多）

不同点：

react函数式组件思想，在发生数据（setState）更改后，会重新生成新的虚拟dom树，然后进行新旧虚拟dom树的diff（自顶向下的全量diff）

vue是组件响应思想，采用代理监听（watcher）数据，当一个组件内数据更改，可以明确知道并响应这个组件进行diff比较（局部订阅）

#### diff算法源码异同

在处理老节点部分，都需要把节点处理 key - value 的 Map 数据结构，方便在往后的比对中可以快速通过节点的 key 取到对应的节点。同样在比对两个新老节点是否相同时，key 是否相同也是非常重要的判断标准。所以不同是 React, 还是 Vue，在写动态列表的时候，都需要设置一个唯一值 key，这样在 diff 算法处理的时候性能才最大化。

不同点：react 会自顶向下全diff。vue会跟踪每一个组件的依赖关系,不需要重新渲染整个组件树。

在react中，当状态发生改变时，组件树就会自顶向下的全diff, 重新render页面， 重新生成新的虚拟dom tree, 新旧dom tree进行比较， 进行patch打补丁方式，局部更新dom。所以react为了避免父组件更新而引起不必要的子组件更新， 可以在shouldComponentUpdate做逻辑判断，减少没必要的render， 以及重新生成虚拟dom，做差量对比过程。

在vue中， 通过Object.defineProperty 把 data 属性全部转为 getter/setter。同时watcher实例对象会在组件渲染时，将属性记录为dep, 当dep 项中的 setter被调用时，通知watch重新计算，使得关联组件更新。（vue中的diff是用双端比较的方式实现的，但是vue2与vue3有区别）

### 3.数据驱动视图

#### vue中的数据驱动视图

Vuejs的数据驱动是通过MVVM这种框架来实现的。MVVM框架主要包含3个部分:model、view和 viewModel。

* Model:指的是数据部分，对应到前端就是javascript对象
* View:指的是视图部分，对应前端就是dom
* ViewModel:就是连接视图与数据的中间件

ViewModel是实现数据驱动视图的核心，当数据变化的时候，ViewModel能够监听到这种变化，并及时的通知view做出修改。同样的，当页面有事件触发时，ViewModel也能够监听到事件，并通知model进行响应。ViewModel就相当于一个观察者，监控着双方的动作，并及时通知对方进行相应的操作。

首先，vuejs在实例化的过程中，会对遍历传给实例化对象选项中的data 选项，遍历其所有属性并使用 Object.defineProperty 把这些属性全部转为 getter/setter。

同时每一个实例对象都有一个watcher实例对象，他会在模板编译的过程中,用getter去访问data的属性，watcher此时就会把用到的data属性记为依赖，这样就建立了视图与数据之间的联系。当之后我们渲染视图的数据依赖发生改变（即数据的setter被调用）的时候，watcher会对比前后两个的数值是否发生变化，然后确定是否通知视图进行重新渲染。这样就实现了所谓的数据对于视图的驱动。

#### react的数据驱动视图

首先了解一些列内容：

* pending：当前所有等待更新的state队列。
* isBatchingUpdates：React中用于标识当前是否处理批量更新状态，默认false。
* dirtyComponent：当前所有待更新state的组件队列。

React通过setState实现数据驱动视图，通过setState来引发一次组件的更新过程从而实现页面的重新渲染(除非shouldComponentUpdate返回false)。

* setState()首先将接收的第一个参数state存储在pending队列中；（state）
* 判断当前React是否处于批量更新状态，是的话就将需要更新state的组件添加到dirtyComponents中；（组件）
* 不是的话，它会遍历dirtyComponents的所有组件，调用updateComponent方法更新每个dirty组件（开启批量更新事务）

## AST抽象语法树

概念

AST 全称为 Abstract Syntax Tree，译为抽象语法树。在 JavaScript 中，任何一个对象（变量、函数、表达式等）都可以转化为一个抽象语法树的形式。抽象语法树本质就是一个树形结构的对象。

用途

* 常用各类转义、编译的插件中。比如最典型的 ES6 转换为 ES6 工具 、JSX 语法转换为 JavaScript 语法。即 babel 模块。
* 代码语法的检查，比如代码规范工具 ESLint 模块。
* 各类 JS/CSS/HTML 压缩工具。
* 代码的格式化、高亮。
* 代码错误提示。
* 代码自动补全。

## React的特点是什么？

* JSX
* 组件
* 单向数据绑定
* 虚拟 DOM
* 简单
* 性能

## React最大的限制(缺点)是什么？

* React 只是一个库。它不是一个完整的框架。
* React有一个庞大的库，需要时间来理解。
* 新程序员可能很难理解和编码。
* React 使用内联模板和 JSX，这可能会很困难并充当障碍。它也使编码变得复杂。

## JSX是什么？

JSX 代表 JavaScript XML。它是一个 React 扩展，允许编写看起来类似于 HTML 的 JavaScript 代码。它使 HTML 文件易于理解。JSX 文件使 React 应用程序健壮并提高其性能。JSX 允许在编写 JavaScript 代码的同一文件中编写类似 XML 的语法，然后预处理器(即，像 Babel 这样的转译器)将这些表达式转换为实际的 JavaScript 代码。就像 XML/HTML 一样，JSX 标签也有标签名称、属性和子标签。

### JSX是什么，它和JS有什么区别

JSX是react的语法糖，它允许在html中写JS，它不能被浏览器直接识别，需要通过webpack、babel之类的编译工具转换为JS执行

### JSX与JS的区别：

* JS可以被打包工具直接编译，不需要额外转换，jsx需要通过babel编译，它是React.createElement的语法糖，使用jsx等价于React.createElement。
* jsx是js的语法扩展，允许在html中写JS；JS是原生写法，需要通过script标签引入。

## 浏览器为什么不能读取 JSX？

浏览器不能直接读取 JSX，因为它们只能理解 JavaScript 对象，而 JSX 不是常规的 JavaScript 对象。因此，我们需要使用 Babel 等转译器将 JSX 文件转换为 JavaScript 对象，然后将其传递给浏览器。

## 为什么要使用JSX？

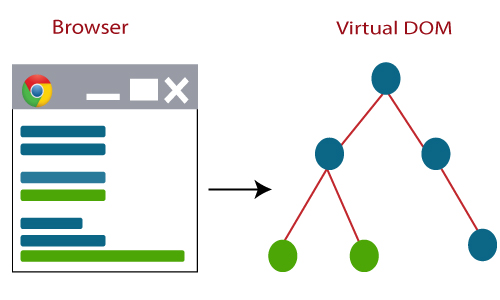
* 它比常规 JavaScript 更快，因为它在将代码转换为 JavaScript 时执行优化。
* React 不是通过将标记和逻辑放在单独的文件中来分离技术，而是使用包含两者的组件。
* JSX是类型安全的，大部分错误都可以在编译时发现。
* JSX使创建模板变得更加容易。

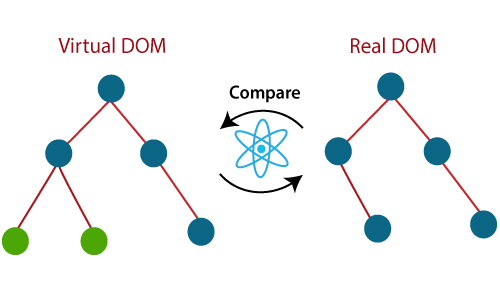
## 虚拟(Virtual)DOM 是什么？

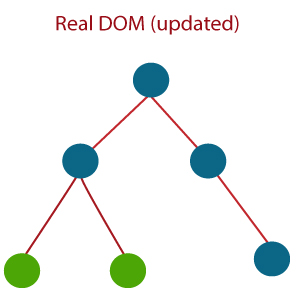
虚拟(Virtual)DOM是一个轻量级的 JavaScript 对象，它是真实 DOM 的内存表示。它是调用渲染函数和在屏幕上显示元素之间的中间步骤。它类似于将元素、它们的属性和内容作为对象及其属性列出的节点树。渲染函数创建 React 组件的节点树，然后更新此节点树以响应由用户或系统执行的各种操作导致的数据模型中的突变。

## 虚拟(Virtual)DOM 的工作原理是什么？

虚拟(Virtual)DOM 分三个步骤工作：

第1步： 每当 React App 中的任何数据发生变化时，整个 UI 都会以 Virtual DOM 表示形式重新渲染。  


第2步： 计算之前的 DOM 表示和新的 DOM 之间的差异。  


第3步： 一旦计算完成，真实的 DOM 只更新那些被改变的东西。  


## 真实 DOM 和虚拟 DOM 有什么区别？

下表指定了 Real DOM 和 Virtual DOM 之间的主要区别：

如果元素更新，真实的 DOM 会创建一个新的 DOM。

|  |  |
| --- | --- |
| 真实 DOM | 虚拟 DOM |
| 真正 DOM 更新较慢。 | 虚拟 DOM 更新更快。 |
| 真正 DOM 可以直接更新 HTML。 | 虚拟 DOM 不能直接更新 HTML。 |
| 如果元素更新 | 虚拟 DOM 会更新 JSX。 |
| 在真实 DOM 中，DOM 操作非常昂贵。 | 在虚拟 DOM 中，DOM 操作非常容易。 |
| 真实 DOM 中存在大量内存浪费。 | 虚拟 DOM 中没有内存浪费。 |

## 从“在 React 中，一切都是组件”中理解了什么?

在 React 中，组件是 React 应用程序的构建块。这些组件将整个 React 应用程序的 UI 划分为小的、独立的、可重用的代码片段。React 独立渲染这些组件中的每一个，而不会影响应用程序 UI 的其余部分。因此，可以说，在 React 中，一切都是组件。

## 高阶组件(HOC)是什么？

在 React 中，高阶组件是一种重用组件逻辑的高级技术。它是一个接受一个组件并返回一个新组件的函数。换句话说，它是一个接受另一个函数作为参数的函数。根据官网的说法，它不是 React API 中的特性(部分)，而是从 React 的组合性质中出现的一种模式。

## 可以用 HOC 做什么？

* 代码可重用性
* Props 操作
* 状态操纵
* 渲染劫持

## 为什么必须以大写字母开头的组件名称？

在 React 中，组件名称必须以大写字母开头。如果以小写开头的组件名称，它将作为无法识别的标签抛出错误。这是因为，在 JSX 中，小写标记名称被视为 HTML 标记

## React 中 keys 的作用是什么？

密钥是唯一标识符。在 React 中，它用于识别哪些项目已从列表中更改、更新或删除。当动态创建组件或用户更改列表时，它很有用。它还有助于确定集合中的哪些组件需要重新渲染，而不是每次都重新渲染整个组件集。它提高了应用程序的性能。

Keys 是 React 用于追踪哪些列表中元素被修改、被添加或者被移除的辅助标识。

在开发过程中，我们需要保证某个元素的 key 在其同级元素中具有唯一性。在 React Diff 算法中 React 会借助元素的 Key 值来判断该元素是新近创建的还是被移动而来的元素，从而减少不必要的元素重渲染。此外，React 还需要借助 Key 值来判断元素与本地状态的关联关系，因此我们绝不可忽视转换函数中 Key 的重要性。

## 调用 setState 之后发生了什么？

在代码中调用 setState 函数之后，React 会将传入的参数对象与组件当前的状态合并，然后触发所谓的调和过程（Reconciliation）。经过调和过程，React 会以相对高效的方式根据新的状态构建 React 元素树并且着手重新渲染整个 UI 界面。在 React 得到元素树之后，React 会自动计算出新的树与老树的节点差异，然后根据差异对界面进行最小化重渲染。在差异计算算法中，React 能够相对精确地知道哪些位置发生了改变以及应该如何改变，这就保证了按需更新，而不是全部重新渲染。

## react 生命周期函数

初始化阶段：

* getDefaultProps:获取实例的默认属性
* getInitialState:获取每个实例的初始化状态
* componentWillMount：组件即将被装载、渲染到页面上
* render:组件在这里生成虚拟的 DOM 节点
* componentDidMount:组件真正在被装载之后

运行中状态：

* componentWillReceiveProps:组件将要接收到属性的时候调用
* shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以返回 false，接收数据后不更新，阻止 render 调用，后面的函数不会被继续执行了）
* componentWillUpdate:组件即将更新不能修改属性和状态
* render:组件重新描绘
* componentDidUpdate:组件已经更新

销毁阶段：

* componentWillUnmount:组件即将销毁

生命周期指的是组件实例从创建到销毁的流程，函数组件没有生命周期，只有类组件才有，因为只有class组件会创建组件实例。  
组件的生命周期可以分为挂载、更新、卸载阶段。  
render是class组件中唯一必须实现的方法

### 挂载

constructor 可以进行state和props的初始化

static getDerivedStateFromProps

render

componentDidMount 第一次渲染后调用，可以访问DOM，进行异步请求和定时器、消息订阅

### 更新（当组件的props或state变化会触发更新）

static getDerivedStateFromProps

shouldComponentUpdate 返回一个布尔值，默认返回true，可以通过这个生命周期钩子进行性能优化，确认不需要更新组件时调用

### render

getSnapShotBeforeUpdate

componentDidUpdate 在组件完成更新后调用

### 卸载

componentWillUnmount 组件从DOM中被移除的时候调用

### 错误捕获

static getDerivedStateFromError 在errorBoundary中使用

componentDidCatch

## shouldComponentUpdate 是做什么的，（react 性能优化是哪个周期函数？）

shouldComponentUpdate 这个方法用来判断是否需要调用 render 方法重新描绘 dom。因为 dom 的描绘非常消耗性能，如果我们能在 shouldComponentUpdate 方法中能够写出更优化的 dom diff 算法，可以极大的提高性能。

## 为什么虚拟 dom 会提高性能?(必考)

虚拟 dom 相当于在 js 和真实 dom 中间加了一个缓存，利用 dom diff 算法避免了没有必要的 dom 操作，从而提高性能。

用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异把 2 所记录的差异应用到步骤 1 所构建的真正的 DOM 树上，视图就更新了。

## react diff 原理（常考，必考）

* 把树形结构按照层级分解，只比较同级元素。
* 给列表结构的每个单元添加唯一的 key 属性，方便比较。
* React 只会匹配相同 class 的 component（这里面的 class 指的是组件的名字）
* 合并操作，调用 component 的 setState 方法的时候, React 将其标记为 dirty.到每一个事件循环结束, React 检查所有标记 dirty 的 component 重新绘制.
* 选择性子树渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能。

## 展示组件(Presentational component)和容器组件(Container component)之间有何不同

* 展示组件关心组件看起来是什么。展示专门通过 props 接受数据和回调，并且几乎不会有自身的状态，但当展示组件拥有自身的状态时，通常也只关心 UI 状态而不是数据的状态。
* 容器组件则更关心组件是如何运作的。容器组件会为展示组件或者其它容器组件提供数据和行为(behavior)，它们会调用 Flux actions，并将其作为回调提供给展示组件。容器组件经常是有状态的，因为它们是(其它组件的)数据源。

## 类组件(Class component)和函数式组件(Functional component)之间有何不同

语法：类组件使用ES6的class语法创建组件，而函数式组件使用函数声明来创建组件。

状态管理：类组件可以使用state来管理组件的内部状态，而函数式组件则通常使用userState Hook来管理状态。

生命周期：类组件可以使用生命周期方法，如componentDidMount、componentDidUpdate等来管理组件的生命周期，而函数式组件则使用useEffect Hook来管理。

调用方式：如果是一个函数组件，调用则是执行函数即可，如果是一个类组件，则需要将组件进行实例化，然后调用实例对象的render方法

性能：函数式组件通常比类组件更轻量级，因为类组件需要实例化，而函数式组件只是普通函数调用。

## (组件的)状态(state)和属性(props)之间有何不同

* State 是一种数据结构，用于组件挂载时所需数据的默认值。State 可能会随着时间的推移而发生突变，但多数时候是作为用户事件行为的结果。
* Props(properties 的简写)则是组件的配置。props 由父组件传递给子组件，并且就子组件而言，props 是不可变的(immutable)。组件不能改变自身的 props，但是可以把其子组件的 props 放在一起(统一管理)。Props 也不仅仅是数据--回调函数也可以通过 props 传递。

## 何为受控组件(controlled component)

在 HTML 中，类似 <input>, <textarea> 和 <select> 这样的表单元素会维护自身的状态，并基于用户的输入来更新。当用户提交表单时，前面提到的元素的值将随表单一起被发送。但在 React 中会有些不同，包含表单元素的组件将会在 state 中追踪输入的值，并且每次调用回调函数时，如 onChange 会更新 state，重新渲染组件。一个输入表单元素，它的值通过 React 的这种方式来控制，这样的元素就被称为"受控元素"。

## 何为高阶组件(higher order component)

高阶组件是一个以组件为参数并返回一个新组件的函数。HOC 运行你重用代码、逻辑和引导抽象。最常见的可能是 Redux 的 connect 函数。除了简单分享工具库和简单的组合，HOC 最好的方式是共享 React 组件之间的行为。如果你发现你在不同的地方写了大量代码来做同一件事时，就应该考虑将代码重构为可重用的 HOC。

## 为什么建议传递给 setState 的参数是一个 callback 而不是一个对象

因为 this.props 和 this.state 的更新可能是异步的，不能依赖它们的值去计算下一个 state。

## 除了在构造函数中绑定 this，还有其它方式吗

你可以使用属性初始值设定项(property initializers)来正确绑定回调，create-react-app 也是默认支持的。在回调中你可以使用箭头函数，但问题是每次组件渲染时都会创建一个新的回调。

## (在构造函数中)调用 super(props) 的目的是什么

在 super() 被调用之前，子类是不能使用 this 的，在 ES2015 中，子类必须在 constructor 中调用 super()。传递 props 给 super() 的原因则是便于(在子类中)能在 constructor 访问 this.props。

## 应该在 React 组件的何处发起 Ajax 请求

在 React 组件中，应该在 componentDidMount 中发起网络请求。这个方法会在组件第一次“挂载”(被添加到 DOM)时执行，在组件的生命周期中仅会执行一次。更重要的是，你不能保证在组件挂载之前 Ajax 请求已经完成，如果是这样，也就意味着你将尝试在一个未挂载的组件上调用 setState，这将不起作用。在 componentDidMount 中发起网络请求将保证这有一个组件可以更新了。

## 描述事件在 React 中的处理方式。

为了解决跨浏览器兼容性问题，您的 React 中的事件处理程序将传递 SyntheticEvent 的实例，它是 React 的浏览器本机事件的跨浏览器包装器。

这些 SyntheticEvent 与您习惯的原生事件具有相同的接口，除了它们在所有浏览器中都兼容。有趣的是，React 实际上并没有将事件附加到子节点本身。React 将使用单个事件监听器监听顶层的所有事件。这对于性能是有好处的，这也意味着在更新 DOM 时，React 不需要担心跟踪事件监听器。

## createElement 和 cloneElement 有什么区别？

React.createElement():JSX 语法就是用 React.createElement()来构建 React 元素的。它接受三个参数，第一个参数可以是一个标签名。如 div、span，或者 React 组件。第二个参数为传入的属性。第三个以及之后的参数，皆作为组件的子组件。

React.createElement(

type,

[props],

[...children]

)

React.cloneElement()与 React.createElement()相似，不同的是它传入的第一个参数是一个 React 元素，而不是标签名或组件。新添加的属性会并入原有的属性，传入到返回的新元素中，而就的子元素奖杯替换。

React.cloneElement(

element,

[props],

[...children]

)

## React 中有三种构建组件的方式

React.createClass()、ES6 class 和无状态函数。

## react 组件的划分业务组件技术组件？

* 根据组件的职责通常把组件分为 UI 组件和容器组件。
* UI 组件负责 UI 的呈现，容器组件负责管理数据和逻辑。
* 两者通过 React-Redux 提供 connect 方法联系起来。

## 了解 redux 么，说一下 redux

* redux 是一个应用数据流框架，主要是解决了组件间状态共享的问题，原理是集中式管理，主要有三个核心方法，action，store，reducer，工作流程是 view 调用 store 的 dispatch 接收 action 传入 store，reducer 进行 state 操作，view 通过 store 提供的 getState 获取最新的数据
* flux 也是用来进行数据操作的，有四个组成部分 action，dispatch，view，store，工作流程是 view 发出一个 action，派发器接收 action，让 store 进行数据更新，更新完成以后 store 发出 change，view 接受 change 更新视图。Redux 和 Flux 很像。
* 主要区别在于 Flux 有多个可以改变应用状态的 store，在 Flux 中 dispatcher 被用来传递数据到注册的回调事件，但是在 redux 中只能定义一个可更新状态的 store，redux 把 store 和 Dispatcher 合并,结构更加简单清晰
* 新增 state,对状态的管理更加明确，通过 redux，流程更加规范了，减少手动编码量，提高了编码效率，同时缺点时当数据更新时有时候组件不需要，但是也要重新绘制，有些影响效率。一般情况下，我们在构建多交互，多数据流的复杂项目应用时才会使用它们

## redux 有什么缺点

* 一个组件所需要的数据，必须由父组件传过来，而不能像 flux 中直接从 store 取。
* 当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新 render，可能会有效率影响，或者需要写复杂的 shouldComponentUpdate 进行判断。

## React 的工作原理

React 会创建一个虚拟 DOM(virtual DOM)。当一个组件中的状态改变时，React 首先会通过diff算法来标记虚拟 DOM 中的改变，第二步是调节(reconciliation)，会用 diff 的结果来更新 DOM。

## 使用 React 有何优点

* 只需查看 render 函数就会很容易知道一个组件是如何被渲染的
* JSX 的引入，使得组件的代码更加可读，也更容易看懂组件的布局，或者组件之间是如何互相引用的
* 支持服务端渲染，这可以改进 SEO 和性能
* 易于测试
* React 只关注 View 层，所以可以和其它任何框架(如Backbone.js, Angular.js)一起使用

## 为什么React自定义组件首字母要大写

jsx通过babel转义时，调用了React.createElement函数，它接收三个参数，分别是type元素类型，props元素属性，children子元素。  
如下所示，从jsx到真实DOM需要经历jsx->虚拟DOM->真实DOM。如果组件首字母为小写，它会被当成字符串进行传递，在创建虚拟DOM的时候，就会把它当成一个html标签，而html没有app这个标签，就会报错。组件首字母为大写，它会当成一个变量进行传递，React知道它是个自定义组件就不会报错了。

## 为什么在文件中没有使用react，也要在文件顶部import React from “react”

只要使用了jsx，就需要引用react，因为jsx本质就是React.createElement

## **React组件为什么不能返回多个元素(React组件为什么只能有一个根元素)**

React组件最后会编译为render函数，函数的返回值只能是1个，如果不用单独的根节点包裹，就会并列返回多个值，这在js中是不允许的。

react的虚拟DOM是一个树状结构，树的根节点只能是1个，如果有多个根节点，无法确认是在哪棵树上进行更新，vue的根节点为什么只有一个也是同样的原因。

## **React组件怎样可以返回多个组件**

* 使用HOC（高阶函数）
* 使用React.Fragment,可以让你将元素列表加到一个分组中，而且不会创建额外的节点（类似vue的template)
* 使用数组返回

## **React中元素和组件的区别**

* react组件有类组件、函数组件
* react元素是通过jsx创建的

## **React性能优化手段**

1. shouldComponentUpdate
2. memo
3. getDerviedStateFromProps
4. 使用Fragment
5. 循环元素使用正确的key
6. 拆分尽可能小的可复用组件，ErrorBoundary
7. 使用React.lazy和React.Suspense延迟加载不需要立马使用的组件

## 如何避免组件的重新渲染？

React 中最常见的问题之一是组件不必要地重新渲染。React 提供了两个方法，在这些情况下非常有用：

* React.memo(): 这可以防止不必要地重新渲染函数组件
* PureComponent: 这可以防止不必要地重新渲染类组件

这两种方法都依赖于对传递给组件的props的浅比较，如果 props 没有改变，那么组件将不会重新渲染。虽然这两种工具都非常有用，但是浅比较会带来额外的性能损失，因此如果使用不当，这两种方法都会对性能产生负面影响。

通过使用 React Profiler，可以在使用这些方法前后对性能进行测量，从而确保通过进行给定的更改来实际改进性能。

## 什么是纯函数？

纯函数是不依赖并且不会在其作用域之外修改变量状态的函数。

本质上，纯函数始终在给定相同参数的情况下返回相同结果。

## 当调用setState时，React render 是如何工作的？

可以将"render"分为两个步骤：

虚拟 DOM 渲染：当render方法被调用时，它返回一个新的组件的虚拟 DOM 结构。当调用setState()时，render会被再次调用，因为默认情况下shouldComponentUpdate总是返回true，所以默认情况下 React 是没有优化的。

原生 DOM 渲染：React 只会在虚拟DOM中修改真实DOM节点，而且修改的次数非常少，它优化了真实DOM的变化，使React变得更快。

## 如何避免在React重新绑定实例？

1.将事件处理程序定义为内联箭头函数

<button onClick={() => { this.setState({ xx: xx }); }}>Submit</button>

2.使用箭头函数来定义方法

handleSubmit = () => { this.setState({ isFormSubmitted: true }); }

3.使用带有 Hooks 的函数组件

## 什么是useCallback和useMemo？

useCallback 和 useMemo 都是react可用于性能优化的内置hooks。

两者的区别在于：useCallback缓存的是一个函数，而useMemo缓存的是计算结果。

### 为什么使用useCallback和useMemo?

在函数式组件中，每次UI的变化，都是通过重新执行整个函数来完成的，这和传统的类组件有很大区别：函数组件中并没有一个直接的方式在多次渲染之间维持一个状态。

在重新执行整个函数组件的过程中，其中的函数和引用类型的变量会创建新的（指向新的引用），导致函数组件在re-render前后，其中函数和引用类型变量是不相等的，这又会导致其他非必要的re-render。

### 什么时候使用useCallback和useMemo?

#### useCallback

当子组件接收一个函数props时，一般会使用useCallback来缓存这个函数，减少不必要的re-render。以下例子：向子组件传递一个函数，在父组件每次re-render的时候，函数会重新创建新的，这会导致使用这个函数props的子组件也re-render，但这是不必要的，可以用useCallback来解决。

#### useMemo

useMemo常用在以下两种场景的优化中：

1. 引用类型的变量
2. 需要大量时间执行的计算函数。

useMemo 和 useCallback 接收的参数都是一样,第一个参数为回调 第二个参数为要依赖的数据

### 异同点

共同作用：

1.仅仅 依赖数据 发生变化, 才会重新计算结果，也就是起到缓存的作用。

两者区别：

1.useMemo 计算结果是 return 回来的值, 主要用于 缓存计算结果的值 ，应用场景如： 需要 计算的状态

2.useCallback 计算结果是 函数, 主要用于 缓存函数，应用场景如: 需要缓存的函数，因为函数式组件每次任何一个 state 的变化 整个组件 都会被重新刷新，一些函数是没有必要被重新刷新的，此时就应该缓存起来，提高性能，和减少资源浪费。

注意： 不要滥用会造成性能浪费，react中减少render就能提高性能，所以这个仅仅只针对缓存能减少重复渲染时使用和缓存计算结果。

### 总结

useMemo返回的是一个变量的值，useCallback返回的是一个函数。对应到function组件最后要return的html代码部分，useMemo就是作为一个值来使用的，而useCallback则是被绑定到onClick上，作为要执行的函数。这就是它俩的本质区别。

useCallback(fn, deps) 等价于 useMemo(() => fn, deps).

## 常用的hooks

useState(): 允许在函数组件中使用状态。使用useState() 声明一个状态变量，并使用它来存储组件的状态。每次更改状态时，组件将重新渲染。

useEffect():用于处理副作用。副作用指在React组件之外进行的操作，例如从服务器获取数据，处理DOM元素等。使用useEffect() hook，您可以执行此类操作，而无需在类组件中编写生命周期方法。

useContext(): 允许您在React中使用上下文。上下文是一种在组件树中传递数据的方法，可以避免通过Props一层层传递数据。使用useContext() hook，您可以访问整个应用程序中定义的上下文对象。

useReducer(): 是useState() hook的替代品，用于管理更复杂的状态。它使用Reducer函数来管理组件状态，Reducer函数接收当前状态和要进行的操作，然后返回新状态。详细使用方式见此文章。

useCallback(): 用于避免在每次渲染时重新创建回调函数。当您需要将回调函数传递给子组件时，这非常有用，因为它可以避免子组件不必要地重新渲染。

useMemo(): 用于缓存计算结果，以避免在每次渲染时重新计算。这非常有用，特别是当计算成本很高时。

useRef(): 用于创建对DOM元素的引用。它还可以用于存储组件之间共享的变量，这些变量不会在组件重新渲染时发生更改。

## setState是同步还是异步？

**v18之前：**

React是希望setState表现为异步的，因为批量更新可以优化性能。因此在React能够管控到的地方，比如生命周期钩子和合成事件回调函数内，表现为异步。

在定时器和原生事件里，因为React管控不到，所以表现为同步。

在某些情况下，我们需要立即获取更新后的状态，这时可以使用第二个可选参数callback，在状态更新后立即执行回调函数来获取更新后的状态。例如：

this.setState({ counter: this.state.counter + 1 }, () => {

console.log(this.state.counter); // 输出更新后的值

});

**v18之后：**

React18之后，默认所有的操作都放到批处理中，因此setState不管在那儿调用都是异步的了。

如果希望同步更新，可以使用flushSync这个API。

## React组件通信方式有哪些？

Props: 父组件可以通过 props 将数据传递给子组件。子组件可以通过 this.props 访问这些数据。

Callback: 父组件可以通过回调函数将函数传递给子组件。子组件可以在适当的时候调用这些回调函数，以便与父组件通信。

Context: 上下文是一种在组件树中共享数据的方法。通过 context，可以在组件树中传递数据，而不需要在每个级别显式地将 props 传递给所有组件。

Redux: 复杂应用全局状态管理可以使用Redux、Mobx等状态管理库，项目里一般使用React Redux或者RTK工具包。

Pub/Sub: 发布/订阅模式是一种通过事件来进行任意组件间通信的方法，和Vue里的事件总线原理一样。

Hooks里也可以通过useReducer和useContext来实现全局组件通信。

## 使用Hooks有踩过哪些坑？

useEffect中没有正确设置依赖数组导致死循环。

useEffect中没有清除副作用导致内存泄漏。

在条件语句和循环中使用Hooks导致报错。

闭包陷阱。

## React逻辑复用方式有哪些？

Mixin：有很多缺点，已被弃用，可以不考虑。

HOC(高阶组件)：高阶组件是一个函数，它接收一个组件作为参数并返回一个新的组件。高阶组件可以将一些通用的逻辑（如：数据获取、权限验证、错误处理等）封装到一个函数中，并将其作为高阶组件的参数传递给其他组件使用，HOC一般以withXxx命名，并可以结合装饰器优雅地使用。

Render Props：通过在组件中传递一个函数作为prop，该函数将用于渲染组件的内容。这个函数可以接收组件需要的数据和方法，并返回React元素。

Hooks：自定义Hooks，将通用逻辑封装到useXxx函数中，可以在多个组件内使用，常见的像数据请求、表单、防抖节流、拖拽等。

## useRef有什么作用？

获取DOM。

存储上一次渲染的值，可以用useRef创建一个对象来存储setState前的旧值。

## React代码层面有哪些性能优化的方式？

React.memo()：可以缓存组件的渲染结果，避免不必要的重渲染。它接受一个函数组件，并返回一个新的组件，新组件将只在props发生变化时才重新渲染。

useMemo和useCallback。

shouldComponentUpdate：在类组件中，可以通过实现shouldComponentUpdate()方法来判断组件是否需要重新渲染。SCU接收两个参数：nextProps和nextState，我们可以在这个方法中比较当前props和state与下一个props和state的变化来决定是否需要重新渲染组件。

使用React.lazy()和Suspense进行组件懒加载。

对于大型应用，可以使用不可变数据的三方库比如Immer.js结合shouldComponentUpdate来做性能优化。

## hook的原理

### 为什么要引入 Hooks？

1. **组件之间一些包含状态的逻辑很难拆分复用**

因为函数式组件不能处理状态，且之前 Mixin 模式已被证伪，目前社区主流的 HoC 和 render props 模式也被诟病导致 wrapper hell

1. **复杂的大型组件越发难以理解和维护**

关联紧密的逻辑被迫分散在各个 lifecycle 方法中，非常容易出现疏漏

1. **组件类的方式对开发者和机器都不友好**

难以理解的 this，回调函数需要手动绑定，还有很多诸如此类的组件类模式的认知成本，导致新手入门难度高。同样地，组件类因为有内部状态、继承逻辑等特征，对编译器的各种优化也非常不友好