react面试题

state 和 props 之间的区别是什么?

- props 是传递给组件的（类似于函数的形参），而 state 是在组件内被组件自己管理的（类似于在一个函数内声明的变量）。
- props 是不可修改的，所有 React 组件都必须像纯函数一样保护它们的 props 不被更改。由于 props 是传入的，并且它们不能更改，因此我们可以将任何仅使用 props 的 React 组件视为 pureComponent，也就是说，在相同的输入下，它将始终呈现相同的输出。
- state 是在组件中创建的，一般在 constructor中初始化 state
- state 是多变的、可以修改，每次setState都异步更新的。

什么是 Props?

Props 是 React 中属性的简写。它们是只读组件，必须保持纯，即不可变。它们总是在整个应用中从父组件传递到子组件。子组件永远不能将 prop 送回父组件。这有助于维护单向数据流，通常用于呈现动态生成的数据。

react和react native的区别是什么

react和react native的区别是：1、框架作用的平台不同；2、工作原理有差别；3、渲染周期不同；4、react native中所有元素都会被平台指定的react组件替换；5、宿主平台的API不同。

说说对React的理解？有哪些特性?

React，用于构建用户界面的 JavaScript 库，提供了 UI 层面的解决方案，遵循组件设计模式、声明式编程范式和函数式编程概念，以使前端应用程序更高效，使用虚拟DOM来有效地操作DOM，遵循从高阶组件到低阶组件的单向数据流,帮助我们将界面成了各个独立的小块，每一个块就是组件，这些组件之间可以组合、嵌套，构成整体页面.

特性

- JSX语法
- 单向数据绑定
- 虚拟DOM
- 声明式编程
- Component(组件化)

优势

- 高效灵活
- 声明式的设计，简单使用
- 组件式开发，提高代码复用率
- 单向响应的数据流会比双向绑定的更安全，速度更快

类组件和函数组件之间有什么区别

类组件：

无论是使用函数或是类来声明一个组件，它决不能修改它自己的 props。 所有 React 组件都必须是纯函数，并禁止修改其自身 props。 React是单项数据流，父组件改变了属性，那么子组件视图会更新。 属性 props是外界传递过来的，状态 state是组件本身的，状态可以在组件中任意修改 组件的属性和状态改变都会更新视图。 函数组件：

函数组件接收一个单一的 props 对象并返回了一个React元素 函数组件的性能比类组件的性能要高，因为类组件使用的时候要实例化，而函数组件直接执行函数取返回结果即可。为了提高性能，尽量使用函数组件。

react的diff算法是怎么完成的

-把树形结构按照层级分解，只比较同级元素 通过给列表结构的每个单元添加的唯一 key值进行区分同层次的子节点的比较。 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。 选择性渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能

1.什么是JSX?

JSX是一种Javascript的语法扩展，可以很好的描述UI的架构。是React.createElement的语法糖。浏览器不能直接解析JSX文件，需要通过Bable进行转译成js。

2.讲一下虚拟Dom?

虚拟Dom：虚拟Dom是描述真实Dom的js对象。 特点：

（1）处理了浏览器兼容性问题，避免用户操作真实DOM，不容易出错。

（2）内容经过了XSS处理，可以防范XSS攻击。

（3）可以实现跨平台开发。

（4）在更新的时候，比较两棵虚拟DOM树的差异，差异化更新。

延伸题： 什么是diff算法?

diff算法，就是用来找出两段文本之间的差异的一种算法。

vdom为什么用diff算法?

由于DOM操作是非常昂贵的，就可以通过diff算法来减少DOM操作。

vdom比真实dom快?

回答：不一定。

在比较性能的时候，要分清楚初始渲染、小量数据更新、大量数据更新这些不同的场合。

无效、无意义的diff是需要浪费性能的，因此有些场景不如直接操作原生DOM性能好。

3.类组件和函数组件之间的区别是什么?

相同点：它们都可以接收属性并且返回React元素。

不同点：

（1）类组件需要创建实例，是基于面向对象的方式编程，函数组件不需要创建实例，接收输入，返回输出，是基于函数编程的思想。

（2）类组件需要创建并且保持实例，会占用一定的内存，函数组件不需要创建实例，可以节约内存占用。

（3）类组件有完整的生命周期，函数组件没有生命周期（现在通过useEffect实现类似生命周期的功能）

（4）类组件通过shouldComponent和pureComponent跳过更新，而函数组件可以通过React.memo跳过更新。

（5）类组件服用逻辑一般用HOC，函数组件可以自定义Hook。

4.hooks出现的意义?

Hook 是 React 16.8 的新增特性。它可以让你在不编写 class 的情况下使用 state 以及其他的 React 特性。
Hooks 优势：

（1）函数组件无 this 问题

（2）自定义 Hook 方便复用状态逻辑

（3）副作用的关注点分离

5.了解React Fiber吗?

React Fiber初衷是改变js在浏览器的主线程上长时间执行，会阻塞其他操作，影响用户体验。

Fiber的关键特性：

（1）增量渲染（把渲染任务拆分成块，匀到多帧）

（2）更新时能够暂停，终止，复用渲染任务

（3）给不同类型的更新赋予优先级

（4）并发方面新的基础能力

协调器reconciler： Fiber前的 Reconciler 被命名为Stack Reconciler 运作的过程无法中断（持续占用主线程），这样主线程上的布局、动画等周期性任务以及交互响应就无法立即得到处理，影响体验。

而新的Reconciler 命名为Fiber Reconciler 每执行一段时间，都会将控制权交回给浏览器，可以分段执行。

Fiber Reconciler 在执行过程中，会分为 2 个阶段：

（1）render阶段，生成 Fiber 树，得出需要更新的节点信息。这一步是一个渐进的过程，可以被打断。

（2）commit阶段，将需要更新的节点一次过批量更新，这个过程不能被打断。

6.setState 同步还是异步？（比较常问）

setState 本身代码的执行肯定是同步的，这里的异步是指是多个 state 会合成到一起进行批量更新。 同步还是异步取决于它被调用的环境。

如果 setState 在 React 能够控制的范围被调用，它就是**异步**的。比如**合成事件处理函数**，**生命周期函数**， 此时会进行批量更新，也就是将状态合并后再进行 DOM 更新

如果 setState 在原生 JavaScript 控制的范围被调用，它就是**同步**的。比如原生事件处理函数，定时器回调函数，Ajax 回调函数中，此时 setState 被调用后会立即更新 DOM 。

1）React生命周期中以及事件处理中，为异步。

（2）原生方法（setTimeout，setInnerval，addEventListener ）中是同步 。

原理：setState本身并不是异步，只是因为react的性能优化机制体现为异步。在react的生命周期函数或者作用域下为异步，在原生的环境下为同步。 因为每次调用setState都会触发更新，异步操作是为了提高性能，将多个状态合并一起更新，减少re-render调用。

7.React-router路由模式?

hash模式（HashRouter）：通过监听 hashchange 事件， 在回调里拿到 window.location.hash 的值。 hash 就是指 url 尾巴后的 # 号以及后面的字符。

hash模式原理：

使用window.location.hash属性及窗口的onhashchange事件，可以实现监听浏览器地址hash值变化，执行相应的js切换网页。 hash指的是地址中#号以及后面的字符，也称为散列值。

history模式（BrowserRouter）： 利用history API实现url地址改变，网页内容改变。

history模式原理：

window.history 属性指向 History 对象，它表示当前窗口的浏览历史。 History 对象保存了当前窗口访问过的所有页面网址。

8.React的生命周期?

1. ***componentWillMount***()** – 在渲染之前执行，在客户端和服务器端都会执行。
2. ***componentDidMount***()** – 仅在第一次渲染后在客户端执行。
3. ***componentWillReceiveProps***()** – 当从父类接收到 props 并且在调用另一个渲染器之前调用。
4. ***shouldComponentUpdate***()** – 根据特定条件返回 true 或 false。如果你希望更新组件，请返回**true** 否则返回 **false**。默认情况下，它返回 false。
5. ***componentWillUpdate***()** – 在 DOM 中进行渲染之前调用。
6. ***componentDidUpdate***()** – 在渲染发生后立即调用。
7. ***componentWillUnmount***()** – 从 DOM 卸载组件后调用。用于清理内存空间。

react hooks，它带来了那些便利

在没有 hooks 之前，我们使用函数定义的组件中，不能使用 React 的 state、各种生命周期钩子类组件的特性。在 React 16.8 之后，推出了新功能： Hooks，通过 hooks 我们可以再函数定义的组件中使用类组件的特性。 **好处:**

1. 跨组件复用: 其实 render props / HOC 也是为了复用，相比于它们，Hooks 作为官方的底层 API，最为轻量，而且改造成本小，不会影响原来的组件层次结构和传说中的嵌套地狱；
2. 类定义更为复杂

- 不同的生命周期会使逻辑变得分散且混乱，不易维护和管理；

- 时刻需要关注this的指向问题；
- 代码复用代价高，高阶组件的使用经常会使整个组件树变得臃肿；

1. 状态与 UI 隔离: 正是由于 Hooks 的特性，状态逻辑会变成更小的粒度，并且极容易被抽象成一个自定义 Hooks，组件中的状态和 UI 变得更为清晰和隔离。

- 避免在 循环/条件判断/嵌套函数 中调用 hooks，保证调用顺序的稳定；
- 不能在useEffect中使用useState，React 会报错提示；
- 类组件不会被替换或废弃，不需要强制改造类组件，两种方式能并存

9.你知道那些hook?

useState()`，状态管理钩子。通过在函数组件中调用useState，就会创建一个单独的状态。

useEffect()，副作用钩子。它接收两个参数， 第一个是进行的异步操作， 第二个是数组，用来给出Effect的依赖项

useContext()，共享钩子。该钩子的作用是，在组件之间共享状态。

useReducer()，Action 钩子。useReducer() 提供了状态管理，其基本原理是通过用户在页面中发起action, 从而通过reducer方法来改变state, 从而实现页面和状态的通信。

useRef()，获取组件的实例；渲染周期之间共享数据的存储(state不能存储跨渲染周期的数据，因为state的保存会触发组件重渲染)

useMemo和useCallback：可缓存函数的引用或值，useMemo用在计算值的缓存，注意不用滥用。经常用在下面两种场景（要保持引用相等；对于组件内部用到的 object、array、函数等，如果用在了其他 Hook 的依赖数组中，或者作为 props 传递给了下游组件，应该使用 useMemo/useCallback）

10. 对 React 和 Vue 的理解，它们的异同

相似之处：

- 都将注意力集中保持在核心库，而将其他功能如路由和全局状态管理交给相关的库
- 都有自己的构建工具，能让你得到一个根据最佳实践设置的项目模板。
- 都使用了Virtual DOM（虚拟DOM）提高重绘性能
- 都有props的概念，允许组件间的数据传递

- 都鼓励组件化应用，将应用分拆成一个个功能明确的模块，提高复用

不同之处： Vue默认支持数据双向绑定，而React一直提倡单向数据流

11,**对React SSR的理解**

服务端渲染是数据与模版组成的html，即 HTML = 数据 + 模版。将组件或页面通过服务器生成html字符串，再发送到浏览器，最后将静态标记"混合"为客户端上完全交互的应用程序。页面没使用服务渲染，当请求页面时，返回的body里为空，之后执行js将html结构注入到body里，结合css显示出来;

SSR的优势：

- 对SEO友好
- 所有的模版、图片等资源都存在服务器端
- 一个html返回所有数据
- 减少HTTP请求
- 响应快、用户体验好、首屏渲染快

12,**你了解 Virtual DOM 吗？解释一下它的工作原理。**

Virtual DOM 是一个轻量级的 JavaScript 对象，它最初只是 real DOM 的副本。它是一个节点树，它将元素、它们的属性和内容作为对象及其属性。

Virtual DOM 工作过程有三个简单的步骤。

1，每当底层数据发生改变时，整个 UI 都将在 Virtual DOM 描述中重新渲染。

2，然后计算之前 DOM 表示与新表示的之间的差异。

3，完成计算后，将只用实际更改的内容更新 real DOM。

13,**React Hooks 的一些优势和特性 状态管理更加清晰**

- 将“状态”与“变更状态的逻辑”两两配对，拥有比原来更好的代码结构。
- 将状态进行更细粒度的拆分，将没有联动关系的状态放到不同的组件中单独管理。
- 将状态的管理与视图的渲染进行隔离，把一个带有复杂的render实现的类组件拆分为一个“单纯管理状态的类组件”和一个“实现渲染逻辑的纯函数组件”。

生命周期的淡化

- 代码可读性更强，原本同一块功能的代码逻辑被拆分在了不同的生命周期函数中，容易使开发者不利于维护和迭代，通过 React Hooks 可以将功能代码聚合，方便阅读维护。
- 例如，每个生命周期中常常会包含一些不相关的逻辑。一般我们都会在 componentDidMount 和 componentDidUpdate 中获取数据。但是，同一个 componentDidMount 中可能也包含很多其它的逻辑，如设置事件监听，而之后需在 componentWillUnmount 中清除。相互关联且需要对照修改的代码被进行了拆分，而完全不相关的代码却在同一个方法中组合在一起。如此很容易产生 bug，并且导致逻辑不一致。

组件树层级变浅

- 在原本的代码中，我们经常使用 HOC/render props 等方式来复用组件的状态，增强功能等，无疑增加了组件树层数及渲染，在 React DevTools 中观察过 React 应用，你会发现由 providers，consumers，高阶组件，render props 等其他抽象层组成的组件会形成“嵌套地狱”。而在 React Hooks 中，这些功能都可以通过强大的自定义的 Hooks 来实现。

不用再去考虑 *this* 的指向问题

- 在类组件中，你必须去理解 JavaScript 中 this 的工作方式。

状态不同步

- useEffect不是很好用，依赖参数不好写，而且很容易出错，有时会发生比预想更多的调用次数。这绝对可以成为摒弃react hooks的理由。函数的运行是独立的，每个函数都有一份父级作用域。当我们处理复杂逻辑的时候，经常会碰到状态不同步的问题。

14**React 性能优化手段**

使用 React.memo 来缓存组件。

使用 React.useMemo 缓存大量的计算

避免使用匿名函数。

利用 React.lazy 和 React.Suspense 延迟加载不是立即需要的组件

尽量使用 CSS 而不是强制加载和卸载组件。

使用 React.Fragment 避免添加额外的 DOM。

为什么要用 Virtual DOM：

1）保证性能下限，在不进行手动优化的情况下，提供过得去的性能、、

2 ）跨平台 Virtual DOM本质上是JavaScript的对象，它可以很方便的跨平台操作，比如服务端渲染、uniapp等。

15. React diff 算法的原理是什么?

diff 算法探讨的就是虚拟 DOM 树发生变化后，生成 DOM 树更新补丁的方式。它通过对比新旧两株虚拟 DOM 树的变更差异，将更新补丁作用于真实 DOM，以最小成本完成视图更新。

- 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。
-

16,React key 是干嘛用的 ****为什么要加？ key 主要是解决哪一类问题的

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。

17,React 与 Vue 的 diff 算法有何不同?

React 的 diff 算法，触发更新的时机主要在 state 变化与 hooks 调用之后。

Vue 的整体 diff 策略与 React 对齐，虽然缺乏时间切片能力，但这并不意味着 Vue 的性能更差，因为在 Vue 3 初期引入过，后期因为收益不高移除掉了

18,对函数式编程的理解

数据不可变（无副作用）： 它要求你所有的数据都是不可变的，这意味着如果你想修改一个对象，那你应该创建一个新的对象用来修改，而不是修改已有的对象。 **无状态：** 主要是强调对于一个函数，不管你何时运行，它都应该像第一次运行一样，给定相同的输入，给出相同的输出，完全不依赖外部状态的变化。

19,为什么不能在条件语句中写 hook hook 在每次渲染时的查找是根据一个“全局”的下标对链表进行查找的，如果放在条件语句中使用，有一定几率会造成拿到的状态出现错乱

20,useEffect 和 useLayoutEffect 区别

因为某个事件 state 发生变化。

React 内部更新 state 变量。

React 处理更新组件中 return 出来的 DOM 节点（进行一系列 dom diff 、调度等流程）。

将更新过后的 DOM 数据绘制到浏览器中

useEffect 在第 4 步之后执行，且是异步的，保证了不会阻塞浏览器进程。 useLayoutEffect 在第 3 步至第 4 步之间执行，且是同步代码，所以会阻塞后面代码的执行

21,useEffect 依赖为空数组与 componentDidMount 区别

在 render 执行之后，componentDidMount 会执行，如果在这个生命周期中再一次 setState ，会导致再次 render ，返回了新的值，浏览器只会渲染第二次 render 返回的值，这样可以避免闪屏

但是 useEffect 是在真实的 DOM 渲染之后才会去执行，这会造成两次 render ，有可能会闪屏。

实际上 useLayoutEffect 会更接近 componentDidMount 的表现，它们都同步执行且会阻碍真实的 DOM 渲染的。

22,React.memo() 和 React.useMemo() 的区别

memo 是一个高阶组件，默认情况下会对 props 进行浅比较，如果相等不会重新渲染。多数情况下我们比较的都是引用类型，浅比较就会失效，所以我们可以传入第二个参数手动控制。

useMemo 返回的是一个缓存值，只有依赖发生变化时才会去重新执行作为第一个参数的函数，需要记住的是，useMemo 是在 render 阶段执行的，所以不要在这个函数内部执行与渲染无关的操作，诸如副作用这类的操作属于 useEffect 的适用范畴

23,React.useCallback() 和 React.useMemo() 的区别

- useCallback 可缓存函数，其实就是避免每次重新渲染后都去重新执行一个新的函数。

- useMemo 可缓存值。

24,React.forwardRef 是什么及其作用

一般在父组件要拿到子组件的某个实际的 DOM 元素时会用到。

25,react中fibar**是用来干什么的**

在 React 中，**Fiber 就是 React 16 实现的一套新的更新机制，让 React 的更新过程变得可控，避免了之前采用递归需要一气呵成影响性能的做法。**

把一个**耗时长的任务分成很多小片**，每一个小片的运行时间很短，虽然总时间依然很长，但是在每个小片执行完之后，都**给其他任务一个执行的机会**，这样唯一的线程就不会被独占，其他任务依然有运行的机会。

26,为什么浏览器无法读取JSX?

浏览器只能处理 JavaScript 对象，而不能读取常规 JavaScript 对象中的 JSX。所以为了使浏览器能够读取 JSX，首先，需要用像 Babel 这样的 JSX 转换器将 JSX 文件转换为 JavaScript 对象，然后再将其传给浏览器。

27,你怎样理解“在React中，一切都是组件”这句话。

组件是 React 应用 UI 的构建块。这些组件将整个 UI 分成小的独立并可重用的部分。每个组件彼此独立，而不会影响 UI 的其余部分。

28,React 中的箭头函数是什么？怎么用?

箭头函数（=>）是用于编写函数表达式的简短语法。这些函数允许正确绑定组件的上下文，因为在 ES6 中默认下不能使用自动绑定。使用高阶函数时，箭头函数非常有用。

29. 区分有状态和无状态组件。

有状态组件	无状态组件
1. 在内存中存储有关组件状态变化的信息	1. 计算组件的内部的状态
2. 有权改变状态	2. 无权改变状态
3. 包含过去、现在和未来可能的状态变化情况	3. 不包含过去，现在和未来可能发生的状态变化情况
4. 接受无状态组件状态变化要求的通知，然后将 props 发送给他们。	4.从有状态组件接收 props 并将其视为回调函数。

30,列出一些应该使用 Refs 的情况。

需要管理焦点、选择文本或媒体播放时
触发式动画
与第三方 DOM 库集成

31,如何模块化 React 中的代码?

可以使用 export 和 import 属性来模块化代码。它们有助于在不同的文件中单独编写组件。

32,你对受控组件和非受控组件了解多少?

受控组件	非受控组件
1. 没有维持自己的状态	1. 保持着自己的状态
2.数据由父组件控制	2.数据由 DOM 控制
3. 通过 props 获取当前值，然后通过回调通知更改	3. Refs 用于获取其当前值

33,什么是高阶组件（HOC）

高阶组件是重用组件逻辑的高级方法，是一种源于 React 的组件模式。 HOC 是自定义组件，在它之内包含另一个组件。它们可以接受子组件提供的任何动态，但不会修改或复制其输入组件中的任何行为。你可以认为 HOC 是"纯（Pure）"组件。

34,你能用HOC做什么?

代码重用，逻辑和引导抽象
渲染劫持
状态抽象和控制
Props 控制

35,什么是纯组件?

组件是可以编写的最简单、最快的组件。它们可以替换任何只有 render() 的组件。这些组件增强了代码的简单性和应用的性能

36,React 中 key 的重要性是什么?

key 用于识别唯一的 Virtual DOM 元素及其驱动 UI 的相应数据。它们通过回收 DOM 中当前所有的元素来帮助 React 优化渲染。这些 key 必须是唯一的数字或字符串，React 只是重新排序元素而不是重新渲染它们。这可以提高应用程序的性能

38,Redux遵循的三个原则是什么?

单一事实来源：整个应用的状态存储在单个 store 中的对象/状态树里。单一状态树可以更容易地跟踪随时间的变化，并调试或检查应用程序。

状态是只读的：改变状态的唯一方法是去触发一个动作。动作是描述变化的普通 JS 对象。就像 state 是数据的最小表示一样，该操作是对数据更改的最小表示。

使用纯函数进行更改：为了指定状态树如何通过操作进行转换，你需要纯函数。纯函数是那些返回值仅取决于其参数值的函数。

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

- 类组件不仅允许你使用更多额外的功能，如组件自身的状态和生命周期钩子，也能使组件直接访问 store 并维持状态
- 当组件仅是接收 props，并将组件自身渲染到页面时，该组件就是一个 '无状态组件(stateless component)'，可以使用一个纯函数来创建这样的组件。这种组件也被称为哑组件(dumb components)或展示组件

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。

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

当调用 setState的时候，发生了什么操作?

当调用 setState时， React做的第一件事是将传递给setState的对象合并到组件的当前状态，这将启动一个称为和解（ reconciliation）的过程。

和解的最终目标是，根据这个新的状态以最有效的方式更新DOM。

为此， React将构建一个新的 React虚拟DOM树（可以将其视为页面DOM元素的对象表示方式）。

一旦有了这个DOM树，为了弄清DOM是如何响应新的状态而改变的， React会将这个新树与上一个虚拟DOM树比较。

这样做， React会知道发生的确切变化，并且通过了解发生的变化后，在绝对必要的情况下进行更新DOM，即可将因操作DOM而占用的空间最小化

setState方法的第二个参数有什么用？使用它的目的是什么？

它是一个回调函数，当 setState方法执行结束并重新渲染该组件时调用它。在工作中，更好的方式是使用 React组件生命周期之——"存在期"的生命周期方法，而不是依赖这个回调函数

传入 setstate函数的第二个参数的作用是什么？

第二个参数是一个函数，该函数会在 setState函数调用完成并且组件开始重渲染时调用，可以用该函数来监听渲染是否完成

react有哪几种创建组件的方式？ 有什么区别

React 有三种构建组件的方式

- React.createClass
- ES6 class
- 无状态函数

React.createClass 是 React 最传统、兼容性最好的方法。该方法构建一个组件对象, 当组件被调用时，就会创建几个组件实例

ES6 class 方式和 createClass 类似，只是从调用内部方法变成了用类来实现。

无状态组件创建时始终保持一个实例，避免了不必要的检查和内存分配。

Dom真实被添加在HTML是在哪个生命周期

componentDidMount、componentDidUpdate

state 更新之后发生了什么

state 更新之后，会依次执行 shouldComponentUpdate、componentWillUpdate、render 和componentDidMount。 shouldComponentUpdate 会接收需要更新的 props 和 state，让开发者增加必要的判断条件，在其需要的时候更新，不需要的时候不更新。如果返回的是 false，那么组件就不再向下执行生命周期方法。

setState 之后发生了什么

简单版本： React 利用状态队列机制实现了 setState 的"异步"更新，避免频繁的重复更新 state。

首先将新的 state 合并到状态更新队列中，然后根据更新队列和 shouldComponentUpdate 的状态来判断是否需要更新组件

为什么不能直接使用 this.state 改变数据

setState通过一个队列机制来实现 state 更新。当执行 setState 的时候，会将需要更新的 state 合并后放入状态队列，而不会立刻更新 this.state。队列机制可以高效的批量更新 state，如果不通过 setState 而直接修改 this.state，那么该 state 将不会被放入状态队列中，当下次调用 setState 并对状态队列进行合并时，将会忽略之前被直接修改的 state，而造成无法预知的错误。

React 中 ref 是干嘛的?

使用 refs 获取。组件被调用时会新建一个改组件的实例。refs 会指向这个实例，可以是一个回调函数，回调函数会在组件被挂载后立即执行。

react 的虚拟dom是怎么实现的

React 是把真实的 DOM 树转换为 JS 对象树，也就是 Virtual DOM。每次数据更新后，重新计算 VM，并和上一次生成的 VM 树进行对比，对发生变化的部分进行批量更新。除了性能之外，VM 的实现最大的好处在于和其他平台的集成

简述下 React 的事件代理机制?

React 并不会把所有的处理函数直接绑定在真实的节点上。而是把所有的事件绑定到结构的最外层，使用一个统一的事件监听器，这个事件监听器上维持了一个映射来保存所有组件内部的

事件监听和处理函数。

React 的事件代理机制和原生事件绑定有什么区别?

1. 事件传播与阻止事件的传播： React 的合成事件并没有实现事件捕获 只支持了事件冒泡。阻止事件传播 React 做了兼容性处理，只需要 e.preventDefault() 即可，原生存在兼容性问题。
2. 事件类型：React 是 原生事件类型 的一个子集（React 只是实现了 DOM level3 的事件接口，有些事件 React 并没有实现，比如 window 的 resize 事件。）阻止 React 事件冒泡的行为只能用于 React 合成事件系统，但是 在原生事件中的阻止冒泡行为，却可以阻止 React 合成事件的传播。
3. 事件的绑定方式：原生事件系统中支持多种不同的绑定事件的方式，React 中只有一种
4. 事件对象：原生中存在 IE 的兼容性问题，React 做了兼容处理。

React 组件间的通信

- 父组件向子组件通信：props
- 子组件向父组件通信：回调函数
- 跨级组件通信：context
- 没有嵌套关系的组件通信：eventEmitter，利用全局对象来保存事件，用广播的方式去处理事件。