React面试题笔记

React面试题笔记

虚拟 DOM 的原理是什么?

是什么?

虚拟 DOM 就是虚拟节点（这句汉化很重要）。React 用 JS 对象来模拟 DOM 节点， 然后将其渲染成真实的 DOM 节点。 **怎么做** 第一步是模拟，用 JSX 语法写出来的 div 其实是一个虚拟节点：

```html
<div id="x">
<span class="red">hi</span>
</div>
```

这代码会得到这样一个对象：

```js
{
    tag: 'div',
    props: {
    id: 'x'
    },
    children: [
        {
        tag: 'span',
        props: {
        className: 'red'
        },
        children: [
            'hi'
            ]
        }
    ]
}
```

能做到这一点是因为 JSX 语法会被转译为 createElement 函数调用（也叫 h 函数），如下：

```js
React.createElement("div", { id: "x"},
    React.createElement("span", { class: "red" }, "hi")
)
```

第二步是将虚拟节点渲染为真实节点

```js
function render(vdom) {
    // 如果是字符串或者数字，创建一个文本节点
    if (typeof vdom === 'string' || typeof vdom === 'number') {
        return document.createTextNode(vdom)
    }
    const { tag, props, children } = vdom
    // 创建真实DOM
    const element = document.createElement(tag)
    // 设置属性
    setProps(element, props)
    // 遍历子节点，并获取创建真实DOM，插入到当前节点
    children
    .map(render)
    .forEach(element.appendChild.bind(element))
    // 虚拟 DOM 中缓存真实 DOM 节点
    vdom.dom = element
    // 返回 DOM 节点
    return element
}
function setProps // 略
function setProp // 略
```

注意，如果节点发生变化，并不会直接把新虚拟节点渲染到真实节点，而是先经过diff 算法得到一个 patch 再更新到真实节点上。

解决了什么问题

1. DOM 操作性能问题。通过虚拟 DOM 和 diff 算法减少不必要的 DOM 操作，保证性能不太差
2. DOM 操作不方便问题。以前各种 DOM API 要记，现在只有 setState

优点

1. 为 React 带来了跨平台能力，因为虚拟节点除了渲染为真实节点，还可以渲染为其他东西。
2. 让 DOM 操作的整体性能更好，能（通过 diff）减少不必要的 DOM 操作。缺点

缺点

1. 性能要求极高的地方，还是得用真实 DOM 操作（目前没遇到这种需求）
2. React 为虚拟 DOM 创造了**合成事件**，跟原生 DOM 事件不太一样，工作中要额外注意

- 所有 React 事件都绑定到根元素，自动实现事件委托
- 如果混用合成事件和原生 DOM 事件，有可能会出 bug

如何解决缺点

不用 React，用 Vue 3

React 或 Vue 的 DOM diff 算法是怎样的?

是什么 DOM diff 就是对比两棵虚拟 DOM 树的算法（废话很重要）。当组件变化时，会 render 出一个新的虚拟 DOM，diff 算法对比新旧虚拟 DOM 之后，得到一个 patch，然后 React 用 patch 来更新真实 DOM。

怎么做

首先，对比两棵树的**根节点**

1. 如果根节点的类型改变了，比如 div 变成了 p，那么直接认为整棵树都变了，不再对比子节点。此时直接删除对应的真实 DOM 树，创建新的真实 DOM树。
2. 如果根节点的类型没变，就看看属性变了没有
    - 如果没变，就保留对应的真实节点
    - 如果变了，就只更新该节点的属性，不重新创建节点。
        - 更新 style 时，如果多个 css 属性只有一个改变了，那么 React 只更新改、变的。

然后，同时遍历两棵树的**子节点**，每个节点的对比过程同上，不过存在如下两种情况。

情况一

```html 复制代码
<ul>
<li>A</li>
<li>B</li>
</ul>
<ul>
<li>A</li>
<li>B</li>
<li>C</li>
</ul>
```

React 依次对比 A-A、B-B、空-C，发现 C 是新增的，最终会创建真实 C 节点插入页面

情况二

```html 复制代码
<ul>
<li>B</li>
<li>C</li>
</ul>
<ul>
<li>A</li>
<li>B</li>
<li>C</li>
</ul>
```

React 对比 B-A，会删除 B 文本新建 A 文本；对比 C-B，会删除 C 文本，新建 B 文本；（注意，并不是边对比边删除新建，而是把操作汇总到 patch 里再进行 DOM 操作。）对比空-C，会新建 C 文本。

你会发现其实只需要创建 A 文本，保留 B 和 C 即可，为什么 React 做不到呢?

因为 React 需要你加 key 才能做到：

```html 复制代码
<ul>
<li key="b">B</li>
<li key="c">C</li>
</ul>
<ul>
<li key="a">A</li>
<li key="b">B</li>
<li key="c">C</li>
```

```
</ul>
```

React 先对比 key 发现 key 只新增了一个，于是保留 b 和 c，新建 a。 以上是 React 的 diff 算法（源码分析在下一节补充视频中，时长一小时，有能力者选看）。

传送门：DOM Diff算法

传送门：Vue Diff算法

补充：React DOM diff 和 Vue DOM diff 的区别?

React DOM diff 和 Vue DOM diff 的区别：

1. React 是从左向右遍历对比，Vue 是双端交叉对比。
2. React 需要维护三个变量（有点扯），Vue 则需要维护四个变量。
3. Vue 整体效率比 React 更高，举例说明：假设有 N 个子节点，我们只是把最后子 节点移到第一个，那么 a. React 需要进行借助 Map 进行 key 搜索找到匹配项，然后复用节点 b. Vue 会发现移动，直接复用节点 附 React DOM diff 代码查看流程：
4. 运行 git clone ht tps://github.com/facebook/react.git
5. 运行 cd react; git switch 17.0.2
6. 用 VSCode 或 WebStorm 打开 react 目录
7. 打开 packages/react-reconciler/src/ReactChildFiber.old.js 第 1274 行查看旧 版代码，或打开 packages/react-reconciler/src/ReactChildFiber.new.js 第 1267 行查看新代码（实际上一样）
8. 忽略所有警告和报错，因为 React JS 代码中有不是 JS 的代码
9. 折叠所有代码
10. 根据 React 文档中给出的场景反复在大脑中运行代码

- 场景0：单个节点，会运行到 reconcileSingleElement。接下来看多个节点的情况。
- 场 景1：没 key，标签名变了，最终会走createFiberFromElement（存疑）
- 场景2：没 key，标签名没变，但是属性变了，最终走到 updateElement 里的useFiber
- 场景3：有 key，key 的顺序没变，最终走到 updateElement
- 场景4：有 key，key 的顺序变了，updateSlot 返回 null，最终走到 mapRemainingChildren、updateFromMap 和 updateElement(matchedFiber)，整个过程较长，效率较低

8. 代码查看要点：

- 声明不看（用到再看）
- if 先不看（但 if else 要看）
- 函数调用必看

9. 必备快捷键：折叠所有、展开、向前、向后、查看定义

React 有哪些生命周期钩子函数？数据请求放在哪个钩子里？

React 的文档稍微有点乱，需要配合两个地方一起看才能记忆清楚：

传送门

传送门

总得来说：

1. 挂载时调用 constructor，更新时不调用
2. 更新时调用 shouldComponentUpdate 和 getSnapshotBeforeUpdate，挂载时不调用
3. should... 在 render 前调用，getSnapshot... 在 render 后调用
4. 请求放在 componentDidMount 里，最好写博客，容易忘。

React 如何实现组件间通信？

1. 父子组件通信：props + 函数
2. 爷孙组件通信：两层父子通信或者使用 Context.Provider 和 Context.Consumer
3. 任意组件通信：其实就变成了状态管理了

- Redux
- Mobx

- Recoil

你如何理解 Redux?

1. 文档第一句话背下来：Redux 是一个状态管理库/状态容器。
2. 把 Redux 的核心概念说一下：

- State
- Action = type + payload 荷载
- Reducer
- Dispatch 派发
- Middleware

3. 把 ReactRedux 的核心概念说一下：

- connect()(Component)
- mapStateToProps
- mapDispatchToProps

什么是高阶组件 HOC?

参数是组件，返回值也是组件的函数。什么都能做，所以抽象问题就具体回答。 举例说明即可：

1. React.forwardRef
2. ReactRedux 的 connect
3. ReactRouter 的 withRouter

传送门：参考阅读

React Hooks 如何模拟组件生命周期?

1. 模拟 componentDidMount
2. 模拟 componentDidUpdate
3. 模拟 componentWillUnmount

代码示例如下：

```
import { useEffect, useState, useRef } from "react";
import "./styles.css";
export default function App() {
    const [visible, setNextVisible] = useState(true)
    const onClick = () => { setNextVisible(!visible) }
    return (
        <div className="App">
            <h1>Hello CodeSandbox</h1>
            {visible ? <Frank /> : null}
            <div> <button onClick={onClick}>toggle</button> </div>
        </div>
    );
}
function Frank(props) {
    const [n, setNextN] = useState(0)
    const first = useRef(true)
    useEffect(() => {
        if (first.current === true) { return }
        console.log('did update')
    })
    useEffect(() => {
        console.log('did mount')
        first.current = false
        return () => {
            console.log('did unmount')
        }
    }, [])
    const onClick = () => {
        setNextN(n + 1)
    }
    return (
        <div>Frank
            <button onClick={onClick}>+1</button>
        </div>
    )
}
```