react 常见面试题集合，面试官常问的

以下是一份精心整理的React高频面试题集合，涵盖面试官最常考察的核心知识点，按重要性和考察频率排序：

★ 核心基础（必考）

1. **React组件通信方式有哪些?**
   - 父传子：`props`
   - 子传父：回调函数（父组件通过props传递函数给子组件）
   - 兄弟组件：状态提升/Lifting State Up
   - 跨层级：Context API
   - 全局状态：Redux/MobX
   - 事件总线：第三方库（如events）
2. **虚拟DOM vs 真实DOM**

MarkDown

特性	虚拟DOM	真实DOM
更新成本	低（JS对象操作）	高（触发重排重绘）
更新机制	Diff算法局部更新	直接全量更新
内存占用	相对较小	较大
跨平台能力	支持（React Native）	仅浏览器环境

3. **useEffect完整生命周期**

Jsx

```
useEffect(() => {
  // 组件挂载和依赖更新时执行
  console.log('Effect run');

  return () => {
    // 清理函数（组件卸载或依赖更新前执行）
    console.log('Cleanup');
      };
}, [dependencies]); // 依赖数组
```

Hooks高频问题

4. useMemo vs useCallback

Jsx

```
// 缓存计算结果（避免重复计算）
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

// 缓存函数引用（避免子组件无效重渲染）
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

5. useEffect依赖数组的三种情况

- `[]`：仅在组件挂载/卸载时执行（模拟componentDidMount）
- 无依赖：每次渲染后都执行
- `[dep1, dep2]`：依赖变化时执行

6. 自定义Hook实现原理

Jsx

```
function useWindowWidth() {
  const [width, setWidth] = useState(window.innerWidth);

  useEffect(() => {
    const handleResize = () => setWidth(window.innerWidth);
    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize);
  }, []);

      return width;
    }
// 使用：const width = useWindowWidth();
```

性能优化重点

7. React.memo优化策略

```jsx
const MyComponent = React.memo(({ data }) => {
  return <div>{data}</div>;
}, (prevProps, nextProps) => {
  // 自定义props比较函数（默认浅比较）
  return prevProps.data === nextProps.data;
    });
```

8. 避免useEffect无限循环

- 问题场景：在useEffect中修改依赖项导致循环更新
- 解决方案：

```jsx
// ✖ 错误示例（导致循环）
const [count, setCount] = useState(0);
    useEffect(() => {
  setCount(count + 1);
}, [count]);

    // ✅ 正确解法
    useEffect(() => {
  setCount(prev => prev + 1); // 使用函数式更新
}, []); // 空依赖
```

进阶原理

9. Fiber架构核心机制

- 解决的问题：同步渲染阻塞问题
- 核心特性：
  - 增量渲染（将任务拆分成小任务块）
  - 可中断/恢复的渲染过程
  - 优先级调度（高优先级任务可插队）

10. Diff算法优化策略

- 树对比：同级比较（O(n)复杂度）
- 组件类型不同直接替换

- Key策略：稳定key值减少节点移动

高频场景题

11. 如何处理表单？

```jsx
// 受控组件方案
const [value, setValue] = useState('');
  <input
 value={value}
 onChange={(e) => setValue(e.target.value)}
  />

  // 非受控组件方案
const inputRef = useRef();
<input ref={inputRef} />
  // 通过inputRef.current.value获取值
```

12. SSR实现原理

- 服务端：`ReactDOMServer.renderToString()`
- 客户端：`ReactDOM.hydrate()`
- 关键挑战：数据同步、状态管理、CSS处理

🆕 React 18特性

13. 并发模式（Concurrent Mode）

- 特性：可中断渲染、自动批处理、Suspense改进
- 新API：
  - `startTransition`：标记非紧急更新
  - `useDeferredValue`：延迟更新非关键内容
  - `useId`：生成唯一ID（SSR友好）

调试与错误处理

14. 错误边界（Error Boundary）实现

```jsx
```

```
class ErrorBoundary extends React.Component {
  state = { hasError: false };

    static getDerivedStateFromError() {
  return { hasError: true };
    }

    componentDidCatch(error, info) {
      logErrorToService(error, info);
    }

    render() {
  return this.state.hasError
   ? <FallbackUI />
   : this.props.children;
    }
  }
```

高频陷阱题

15. setState异步问题

Jsx

```
handleClick = () => {
  // ✖ 连续调用不会累加
  this.setState({ count: this.state.count + 1 });
  this.setState({ count: this.state.count + 1 });

  // ✔ 使用函数式更新
  this.setState(prev => ({ count: prev.count + 1 }));
  this.setState(prev => ({ count: prev.count + 1 }));
    }
```

这些题目覆盖了React面试中90%以上的高频考点，建议重点掌握：

1. 深入理解Hooks原理和优化策略
2. 熟悉组件通信和状态管理方案
3. 掌握性能优化核心技巧
4. 了解Fiber架构和并发模式
5. 准备2-3个复杂组件的实现思路（如表单/数据看板）

最后提醒：面试时结合项目经验回答（如”我在XX项目中用Context解决了跨层级通信问题”）会让答案更有说服力！

(注:文档部分内容可能由AI生成)