7.useEffect 的执行时机是什么？和 useLayoutEffect 有何区别?

主题：`useEffect` 与 `useLayoutEffect` 的执行时机与区别

面试题

Q1: 请解释一下 `useEffect` 的主要作用及其执行时机。
Q2: `useLayoutEffect` 又有何不同？它的执行时机是怎样的?
Q3: 请总结一下 `useEffect` 和 `useLayoutEffect` 最核心的区别是什么?
Q4: `useEffect` 的异步执行和 `useLayoutEffect` 的同步执行分别对浏览器的渲染过程有什么影响?
Q5: 在实际开发中，你会优先选择使用 `useEffect` 还是 `useLayoutEffect`？为什么?
Q6: 请举一个你认为必须使用 `useLayoutEffect` 的场景，并解释为什么在这种情况下 `useEffect` 可能不适用。
Q7: 哪些常见的副作用操作适合放在 `useEffect` 中处理？请举例说明。
Q8: 如果你需要在组件加载后，根据窗口大小动态设置一个图表库的尺寸，你会考虑使用哪个 Hook？请说明你的选择理由。
Q9: 假设你需要实现一个功能：当用户滚动页面到某个元素附近时，给这个元素添加一个高亮类名。你会选择 `useEffect` 还是 `useLayoutEffect`？为什么?
Q10: 如果在 `useLayoutEffect` 中执行了一个非常耗时的操作，可能会导致什么问题?
Q11: 在选择使用 `useEffect` 还是 `useLayoutEffect` 时，你遵循的一般性经验法则或决策策略是什么?

参考答案

A1:
`useEffect` 的主要作用是处理 React 组件中的副作用，例如数据获取、设置订阅、手动更改 DOM 等。
它的执行时机是：在浏览器完成布局和绘制之后，异步执行。这意味着 `useEffect` 内部的代码不会阻塞浏览器的渲染过程，用户可以先看到更新后的 UI。

A2:
`useLayoutEffect` 同样用于处理副作用，但其执行时机与 `useEffect` 不同。
它的执行时机是：在所有 DOM 变更之后，浏览器进行绘制之前，同步执行。这意味着

`useLayoutEffect` 内部的代码会阻塞浏览器的绘制，直到其执行完毕，浏览器才会将更改绘制到屏幕上。

A3:
`useEffect` 和 `useLayoutEffect` 最核心的区别主要有两点：

1. **执行时机**：
   - `useEffect`：浏览器完成布局和绘制**之后**，异步执行。
   - `useLayoutEffect`：所有 DOM 变更**之后**，浏览器进行绘制**之前**，同步执行。
2. **阻塞行为**：
   - `useEffect`：**不阻塞**浏览器绘制。
   - `useLayoutEffect`：**会阻塞**浏览器绘制。

A4:

- **`useEffect` (异步执行)**：由于它在浏览器完成渲染和绘制之后才执行，并且是异步的，所以它不会阻塞浏览器的绘制过程。这使得页面响应更快，用户可以更快地看到更新，应用给人的感觉会更流畅。
- **`useLayoutEffect` (同步执行)**：它在浏览器绘制前同步执行，因此会阻塞浏览器的绘制过程。如果在 `useLayoutEffect` 中执行了耗时操作，可能会导致页面卡顿，用户需要等待其内部代码执行完毕才能看到最终的界面。

A5:
在实际开发中，我会**优先选择使用 `useEffect`**。
原因是：

1. **性能考虑**：`useEffect` 是异步执行的，不会阻塞浏览器的渲染，能提供更好的用户体验。
2. **通用性**：大部分副作用操作，如数据请求、事件监听等，并不需要在浏览器绘制前同步完成，使用 `useEffect` 完全可以满足需求。
   只有当特定场景下（如避免视觉闪烁）需要同步操作 DOM 并读取布局信息时，才会考虑使用 `useLayoutEffect`。

A6:
一个必须使用 `useLayoutEffect` 的典型场景是：当某个状态改变后，需要读取一个 DOM 元素的尺寸或位置，并根据这些信息立即同步更新另一个 DOM 元素的样式或触发重新渲染，以避免用户看到视图的“闪烁”。

例如，动态计算一个 tooltip 的位置：

1. 状态改变导致 tooltip 内容或其锚点元素变化。
2. React 更新 DOM。
3. **如果使用 `useEffect`**：
   - 浏览器先绘制第一次更新后的 DOM（tooltip 可能位置不正确）。

- `useEffect` 异步执行，读取锚点元素位置，计算 tooltip 正确位置，更新 tooltip 样式。
- React 再次更新 DOM，浏览器再次绘制。
- 用户可能会看到 tooltip 从一个不正确的位置“闪”到正确位置。

4. **如果使用 `useLayoutEffect`**：
   - React 更新 DOM。
   - `useLayoutEffect` 同步执行，立即读取锚点元素位置，计算并更新 tooltip 样式。
   - 浏览器根据所有这些更改（包括 `useLayoutEffect` 中的更新）一起进行绘制。
   - 用户直接看到的就是最终调整好的界面，没有闪烁。

在这种情况下，`useEffect` 不适用，因为它在绘制后执行，无法阻止中间状态的呈现，可能导致视觉上的不一致或闪烁。

A7:
以下常见的副作用操作适合放在 `useEffect` 中处理：

1. **数据获取（Data Fetching）**：组件挂载后从服务器请求数据。例如：

```
useEffect(() => {
  fetch('/api/data')
    .then(res => res.json())
    .then(setData);
}, []);
```

2. **设置和清除订阅（Subscriptions）**：例如监听 Redux store 的变化、设置 `setInterval` 或 `setTimeout`，并在组件卸载时清除它们。例如：

```
useEffect(() => {
  const timerId = setInterval(() => {
    console.log('Tick');
  }, 1000);
  return () => clearInterval(timerId); // 清除函数
}, []);
```

3. **手动更改 DOM（非布局关键型）**：例如手动聚焦一个输入框，或者集成一些需要直接操作 DOM 但不要求在绘制前严格完成的第三方库。例如：

```
const inputRef = useRef(null);
useEffect(() => {
  inputRef.current.focus();
}, []);
```

A8:
如果需要在组件加载后，根据窗口大小动态设置一个图表库的尺寸，我会根据具体情况选择：

- **优先考虑 `useLayoutEffect`**：如果图表库的渲染对初始尺寸非常敏感，并且不希望用户在初次渲染时看到图表尺寸从一个默认值调整到正确值的过程（即避免闪烁或布局错位），那么 `useLayoutEffect` 更合适。因为它会在浏览器绘制前同步执行，可以读取窗口大小并立即设置图表尺寸，浏览器会直接绘制最终正确的状态。
- **`useEffect` 也可能适用**：如果图表库自身能够很好地处理异步的尺寸更新（例如，它内部有自己的重绘逻辑，或者轻微的尺寸调整延迟是可以接受的，不会造成明显的视觉问题），那么使用 `useEffect` 也是可以的。这样可以避免阻塞渲染。

在面试中，我会向面试官阐明我的考量：主要是为了避免视觉上的闪烁或布局不一致。如果同步更新对于用户体验至关重要，则选择 `useLayoutEffect`；如果可以接受异步更新，则 `useEffect` 更佳以保证性能。

A9:
我会选择使用 **`useEffect`**。
原因：
这个场景通常不需要严格的同步操作。用户滚动页面是一个持续的异步事件，当滚动到元素附近时，轻微的延迟（异步执行 `useEffect` 带来的延迟）添加一个高亮类名通常是可以接受的，用户体验影响不大。使用 `useEffect` 可以避免在滚动这样的高频事件处理中阻塞浏览器的渲染，从而保证滚动体验的流畅性。如果使用 `useLayoutEffect`，并且其回调中有复杂逻辑，反而可能导致滚动卡顿。

A10:
如果在 `useLayoutEffect` 中执行了一个非常耗时的操作，可能会导致以下问题：

1. **页面卡顿/冻结**：由于 `useLayoutEffect` 是同步执行的，并且会阻塞浏览器的绘制过程，耗时的操作会直接阻塞主线程。浏览器无法进行绘制，也无法响应用户交互，导致页面看起来卡顿甚至完全冻结，直到 `useLayoutEffect` 中的操作执行完毕。
2. **糟糕的用户体验**：用户会明显感觉到应用的响应变慢，界面更新延迟，这会严重影响用户体验。

因此，`useLayoutEffect` 中的代码应该尽可能地轻量和快速，避免进行复杂的计算或长时间的同步操作。

A11:
在选择使用 `useEffect` 还是 `useLayoutEffect` 时，我遵循的一般性经验法则或决策策略是：

1. **始终优先考虑 `useEffect`**：这是默认和最常见的选择。因为它是异步的，不会阻塞浏览器渲染，对性能更友好，能提供更好的用户体验。
2. **仅在必要时使用 `useLayoutEffect`**：当且仅当你发现使用 `useEffect` 导致了视觉上的闪烁（flicker）或不一致（例如，在DOM更新后需要立即读取布局信息并用该信息同步更

新DOM以避免用户看到中间状态），并且这个操作确实需要在浏览器下一次绘制前完成时，才考虑切换到 `useLayoutEffect`。

3. **警惕 `useLayoutEffect` 的性能影响**：时刻记住 `useLayoutEffect` 是同步且阻塞渲染的。如果决定使用它，务必确保其回调函数中的代码执行效率高，避免耗时操作。

简单总结就是：**性能优先，默认用 `useEffect`；视觉一致性关键且操作依赖 DOM 布局时，才考虑 `useLayoutEffect`。**