50. setTimeout 最小延迟是多少？浏览器如何处理嵌套定时器?

1. 核心概念 (Core Concept)

`setTimeout` 是浏览器和 Node.js 环境中用于在指定的毫秒数后执行一次函数或指定的代码片段的全局方法。

2. 为什么需要它？ (The "Why")

- **异步执行:** 将某些任务推迟到未来执行，避免阻塞主线程，提高页面响应性。
- **时间控制:** 允许开发者控制函数执行的时间点，例如实现动画、延迟加载、或 주기적인 작업（周期性工作 - 但更常用 `setInterval`）。
- **解除阻塞:** 在某些需要等待 DOM 渲染或资源加载后执行的场景中使用，确保操作有效。

3. API 与用法 (API & Usage)

`setTimeout` 的基本语法如下：

```
let timerId = setTimeout(func|code, delay, arg1, arg2, ...);
```

- `func|code`: 要在定时器到期时执行的函数或字符串（不推荐使用字符串）。
- `delay`: 延迟的毫秒数。默认为 0。
- `arg1, arg2, ...`: 传递给函数的回调函数的附加参数（IE9+ 支持）。

返回值 `timerId` 是一个非零数值，可以用来调用 `clearTimeout()` 方法取消定时器的执行。

经典代码示例 (来自 MDN):

```
function greet(param) {
  alert(param);
}

const timerId = setTimeout(greet, 1000, 'Hello!'); // 在 1000 毫秒后执行
greet 函数，并传递参数 'Hello!'

// 如果需要，可以取消定时器
// clearTimeout(timerId);
```

4. 关键注意事项 (Key Considerations)

1. **最小延迟 (Minimum Delay):**
   - HTML Living Standard (WHATWG) 定义了一个最小延迟（minimum delay）的概念。对于嵌套的定时器（timeout after timeout is about fourth level down or more），或在某些特定情况下（如背景标签页、资源限制），延迟会被强制设定为至少 4 毫秒（`4ms`）。这是为了限制 CPU 负载和节省电池。
   - 在旧版本浏览器和特定平台下，最小延迟可能更高（例如，Chrome 的早期版本曾强制最小延迟为 10ms）。
   - 对于非嵌套的、顶层 `setTimeout(fn, 0)`，它会被尽可能快地执行，但仍然是异步的，会在当前同步代码执行完毕后，在下一个事件循环周期中运行。实际上，其延迟通常大于 0ms，取决于事件循环的繁忙程度。
   - 因此，不能假定 `setTimeout` 的延迟参数是绝对精确的，尤其对于小于 10-20ms 的值，实际执行延迟可能会受最小延迟机制和事件循环调度影响。
2. **嵌套定时器处理:**
   - 浏览器根据 HTML 标准对嵌套定时器应用最小延迟规则。当一系列 `setTimeout` 调用（通常连续四层或更多）的延迟小于 4ms 时，后续的调用会被强制至少延迟 4ms。
   - 这个机制旨在防止瞬时创建大量定时器，阻塞事件循环并消耗过多资源。
3. **`this` 值问题:** 在 `setTimeout` 中执行的回调函数，其 `this` 默认指向全局对象（`window` 或 `global`），而不是调用 `setTimeout` 的上下文。这可以通过箭头函数或 `bind` 方法来解决。
4. **定时器 ID:** `setTimeout` 返回一个定时器 ID，必须使用 `clearTimeout(timerId)` 来取消该定时器，防止不必要的执行和潜在的问题（尤其是在组件卸载时）。

5. 参考资料 (References)

- MDN Web Docs - setTimeout
- HTML Living Standard - Timers (WHATWG) (查阅 "minimum timeout delay" 相关章节)