内存泄漏有哪些场景

近些年前端业务和功能变得越来越复杂，内存泄露问题也要重点考虑。

无法被垃圾回收机制回收的，占用内存较大且得不到释放的，内存分配和越界的情况都有可能造成内存泄漏。

全局变量、全局函数、自定义事件、定时器，在页面关闭或组件卸载时需要清除。

DOM引用：获取DOM元素的引用后，如果该元素在后续被删除，但引用依然存在，那么这个元素可能无法被垃圾回收。

循环引用：导致被引用的变量/函数/对象，得不到释放。

内存分配和越界：在使用数组或其他数据结构时，如果操作越过了内存的边界，可能会导致内存泄漏。例如，在for循环中，如果循环次数设置错误，可能会导致数组操作越界。

闭包：闭包可以保留函数作用域内的变量，如果不恰当地使用闭包，可能会导致某些变量一直被留在内存中。

全局变量、全局函数直接赋值为 `null` 来清除。 使用 `clearInterval` 和 `clearTimeout` 来清除定时器。 自定义事件，使用 `removeEventListener` 来清除。

如何进行内存分析

大多数现代浏览器（如Chrome、Firefox、Safari等）都内置了开发者工具，其中包含用于内存分析的强大功能。

1. 使用开发者工具

浏览器开发者工具（Chrome DevTools）切换到内存（Memory）选项卡。

- 记录内存使用情况：点击“Record Allocation Timeline”按钮开始记录，执行一些操作，然后停止记录。这将显示一个内存分配的图形表示。
- 执行堆快照：点击“Take Heap Snapshot”按钮，获取当前堆的快照。然后，你可以在“Heap Snapshots”列表中查看和分析快照。

2. 分析堆快照

堆快照是内存分析的关键部分，因为它们显示了当前在内存中的所有对象。

- 查找大对象或对象组：在堆快照中，你可以按大小排序对象，以找出占用大量内存的对象。
- 检查对象保留树：这可以帮助你了解哪些对象保留了其他对象，并因此阻止了它们的垃圾回收。
- 查找泄漏的根源：通过比较多个堆快照，你可以找出随着时间的推移哪些对象持续增长，这可能表明存在内存泄漏。

3. 使用专门的内存分析工具

除了浏览器的开发者工具外，还有一些专门的内存分析工具可以帮助你更深入地分析问题。

Chrome Extension: 例如，Memory Analyzer for Chrome 是一个Chrome扩展程序，用于分析堆快照并查找内存泄漏。 Node.js工具: 对于Node.js项目，你可以使用像heapdump这样的模块来创建堆快照，并使用其他工具（如node-inspector）来分析它们。

4. 代码审查和优化

- 审查代码以查找可能导致内存泄漏的模式，如意外的全局变量、未清除的计时器、DOM引用等。
- 优化数据结构和算法，以减少内存使用。
- 避免在循环中创建大量对象，这可能导致频繁的垃圾回收并降低性能。

5. 监控和警报

- 使用监控工具来持续跟踪内存使用情况，并设置警报以在内存使用量超过预定阈值时通知你。

WeakMap 和 WeakSet

`WeakMap` 和 `WeakSet` 中的对象都是弱引用，垃圾回收机制不考虑 `WeakMap` 和 `WeakSet` 对该对象的引用。

`WeakMap` 的成员 `key` 只能是引用类型。 `WeakSet` 的成员只能是引用类型。

当定义一个全局变量，又不清除时，垃圾回收机制就不会释放这个全局变量占用的内存，会造成内存泄漏。但是如果变量是 `WeakMap` 或 `WeakSet` 时便不会内存泄漏。一旦不再需要，`WeakMap` 里面的键名对象和所对应的键值对会自动消失，不用手动删除引用。