**Debouncing（防抖动）**

有些浏览器事件能在很短的时间内被触发多次，例如调整窗口大小或滚动页面。如果你给窗口滚动事件添加一个事件监听器，然后用户不停地快速向下滚动页面，那你的事件可能在 3 秒之内被触发数千次。这会导致非常严重的性能问题。

如果在面试中讨论到构建应用程序，以及滚动事件，窗口调整事件，或者键盘事件等，请务必提及 debouncing 或者 throttling，作为提高页面速度与性能的方法。来一个 [css-tricks](https://css-tricks.com/debouncing-throttling-explained-examples/) 的实例：

2011 年，Twitter 出了一个问题：当滚动 Twitter 摘要时，页面变的很卡甚至无响应。John Resig 写了[一篇关于这个问题的博客](http://ejohn.org/blog/learning-from-twitter)，解释了直接将耗时的函数绑定在 scroll 事件上是一个多么糟糕的想法。

Debouncing 是解决这个问题的一种方法，它的做法是限制下次函数调用之前必须等待的时间间隔。正确实现 debouncing 的方法是将若干个函数调用 *合成* 一次，并在给定时间过去之后仅被调用一次。下面是一个原生 JavaScript 的实现，用到了[作用域](https://toddmotto.com/everything-you-wanted-to-know-about-javascript-scope/), 闭包, [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this), 和 [计时事件](http://www.w3schools.com/jsref/met_win_settimeout.asp)：

|  |
| --- |
|  |

|  |
| --- |
| // 将会包装事件的 debounce 函数  function debounce(fn, delay) {  // 维护一个 timer  let timer = null;  // 能访问 timer 的闭包  return function() {  // 通过 ‘this’ 和 ‘arguments’ 获取函数的作用域和变量  let context = this;  let args = arguments;  // 如果事件被调用，清除 timer 然后重新设置 timer  clearTimeout(timer);  timer = setTimeout(function() {  fn.apply(context, args);  }, delay);  }  } |

这个函数 — 当传入一个事件（fn）时 — 会在经过给定的时间（delay）后执行。

函数这样用：

|  |
| --- |
|  |

|  |
| --- |
| // 当用户滚动时被调用的函数  function foo() {  console.log('You are scrolling!');  }  // 在 debounce 中包装我们的函数，过 2 秒触发一次  let elem = document.getElementById('container');  elem.addEventListener('scroll', debounce(foo, 2000)); |

Throttling 是与 debouncing 类似的一种技术，但它不是在调用函数之前等待一段时间，throttling 是在较长的时间间隔内调用函数。所以如果一个事件每 100 毫秒被触发 10 次，throttling 会在每隔 2 秒时执行一次这个函数，而不是在 100 毫秒内执行 10 次事件。

Throttling也就是会说预先设定一个执行周期，当调用动作的时刻大于等于执行周期则执行该动作，然后进入下一个新周期。

var throttle = function(delay, action){

var last = 0

return function(){

var curr = +new Date()

if (curr - last > delay){

action.apply(this, arguments)

last = curr

}

}

}