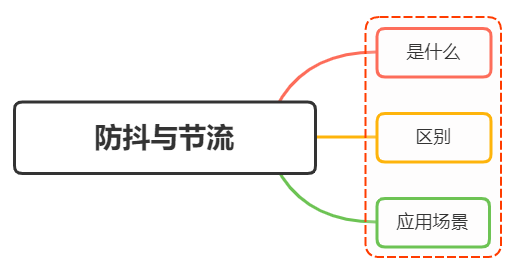
# 面试官：什么是防抖和节流？有什么区别？如何实现？



## 一、是什么

本质上是优化高频率执行代码的一种手段

如：浏览器的 resize、scroll、keypress、mousemove 等事件在触发时，会不断地调用绑定在事件上的回调函数，极大地浪费资源，降低前端性能

为了优化体验，需要对这类事件进行调用次数的限制，对此我们就可以采用 **防抖（debounce）** 和 **节流（throttle）** 的方式来减少调用频率

#### 定义

* 节流: n 秒内只运行一次，若在 n 秒内重复触发，只有一次生效
* 防抖: n 秒后在执行该事件，若在 n 秒内被重复触发，则重新计时

一个经典的比喻:

想象每天上班大厦底下的电梯。把电梯完成一次运送，类比为一次函数的执行和响应

假设电梯有两种运行策略 debounce 和 throttle，超时设定为15秒，不考虑容量限制

电梯第一个人进来后，15秒后准时运送一次，这是节流

电梯第一个人进来后，等待15秒。如果过程中又有人进来，15秒等待重新计时，直到15秒后开始运送，这是防抖

## 代码实现

### 节流

完成节流可以使用时间戳与定时器的写法

使用时间戳写法，事件会立即执行，停止触发后没有办法再次执行

function throttled1(fn, delay = 500) {  
 let oldtime = Date.now()  
 return function (...args) {  
 let newtime = Date.now()  
 if (newtime - oldtime >= delay) {  
 fn.apply(null, args)  
 oldtime = Date.now()  
 }  
 }  
}

使用定时器写法，delay毫秒后第一次执行，第二次事件停止触发后依然会再一次执行

function throttled2(fn, delay = 500) {  
 let timer = null  
 return function (...args) {  
 if (!timer) {  
 timer = setTimeout(() => {  
 fn.apply(this, args)  
 timer = null  
 }, delay);  
 }  
 }  
}

可以将时间戳写法的特性与定时器写法的特性相结合，实现一个更加精确的节流。实现如下

function throttled(fn, delay) {  
 let timer = null  
 let starttime = Date.now()  
 return function () {  
 let curTime = Date.now() // 当前时间  
 let remaining = delay - (curTime - starttime) // 从上一次到现在，还剩下多少多余时间  
 let context = this  
 let args = arguments  
 clearTimeout(timer)  
 if (remaining <= 0) {  
 fn.apply(context, args)  
 starttime = Date.now()  
 } else {  
 timer = setTimeout(fn, remaining);  
 }  
 }  
}

### 防抖

简单版本的实现

function debounce(func, wait) {  
 let timeout;  
  
 return function () {  
 let context = this; // 保存this指向  
 let args = arguments; // 拿到event对象  
  
 clearTimeout(timeout)  
 timeout = setTimeout(function(){  
 func.apply(context, args)  
 }, wait);  
 }  
}

防抖如果需要立即执行，可加入第三个参数用于判断，实现如下：

function debounce(func, wait, immediate) {  
  
 let timeout;  
  
 return function () {  
 let context = this;  
 let args = arguments;  
  
 if (timeout) clearTimeout(timeout); // timeout 不为null  
 if (immediate) {  
 let callNow = !timeout; // 第一次会立即执行，以后只有事件执行后才会再次触发  
 timeout = setTimeout(function () {  
 timeout = null;  
 }, wait)  
 if (callNow) {  
 func.apply(context, args)  
 }  
 }  
 else {  
 timeout = setTimeout(function () {  
 func.apply(context, args)  
 }, wait);  
 }  
 }  
}

## 二、区别

相同点：

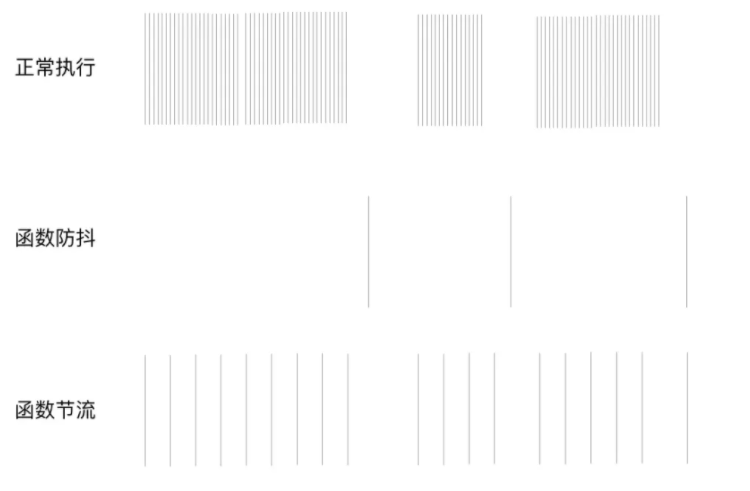
* 都可以通过使用 setTimeout 实现
* 目的都是，降低回调执行频率。节省计算资源

不同点：

* 函数防抖，在一段连续操作结束后，处理回调，利用clearTimeout和 setTimeout实现。函数节流，在一段连续操作中，每一段时间只执行一次，频率较高的事件中使用来提高性能
* 函数防抖关注一定时间连续触发的事件，只在最后执行一次，而函数节流一段时间内只执行一次

例如，都设置时间频率为500ms，在2秒时间内，频繁触发函数，节流，每隔 500ms 就执行一次。防抖，则不管调动多少次方法，在2s后，只会执行一次

如下图所示：



## 三、应用场景

防抖在连续的事件，只需触发一次回调的场景有：

* 搜索框搜索输入。只需用户最后一次输入完，再发送请求
* 手机号、邮箱验证输入检测
* 窗口大小resize。只需窗口调整完成后，计算窗口大小。防止重复渲染。

节流在间隔一段时间执行一次回调的场景有：

* 滚动加载，加载更多或滚到底部监听
* 搜索框，搜索联想功能