# JS防抖和节流

在进行窗口的resize、scroll，输入框内容校验等操作时，如果事件处理函数调用的频率无限制，会加重浏览器的负担，导致用户体验非常糟糕。此时我们可以采用debounce（防抖）和throttle（节流）的方式来减少调用频率，同时又不影响实际效果。

**函数防抖**

函数防抖（debounce）：当持续触发事件时，一定时间段内没有再触发事件，事件处理函数才会执行一次，如果设定的时间到来之前，又一次触发了事件，就重新开始延时。如下图，持续触发scroll事件时，并不执行handle函数，当1000毫秒内没有触发scroll事件时，才会延时触发scroll事件。


一起来实现个简单的debounce~

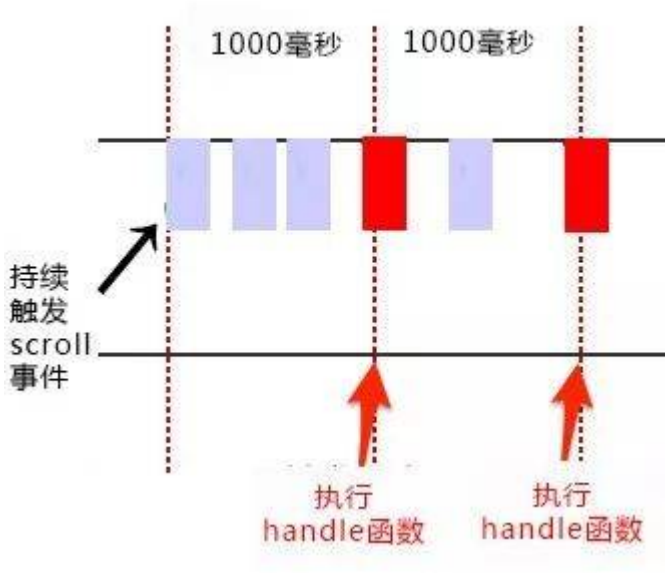
防抖debounce代码：

```
// 防抖
function debounce(fn, wait) {
    var timeout = null;
    return function() {
        if(timeout !== null)   clearTimeout(timeout);
        timeout = setTimeout(fn, wait);
    }
}
// 处理函数
function handle() {
    console.log(Math.random());
}
// 滚动事件
window.addEventListener('scroll', debounce(handle, 1000));
```

当持续触发scroll事件时，事件处理函数handle只在停止滚动1000毫秒之后才会调用一次，也就是说在持续触发scroll事件的过程中，事件处理函数handle一直没有执行。

**函数节流**

函数节流（throttle）：当持续触发事件时，保证一定时间段内只调用一次事件处理函数。节流通俗解释就比如我们水龙头放水，阀门一打开，水哗哗的往下流，秉着勤俭节约的优良传统美德，我们要把水龙头关小点，最好是如我们心意按照一定规律在某个时间间隔内一滴一滴的往下滴。如下图，持续触发scroll事件时，并不立即执行handle函数，每隔1000毫秒才会执行一次handle函数。




函数节流主要有两种实现方法：时间戳和定时器。接下来分别用两种方法实现throttle~

节流throttle代码（时间戳）：

```
var throttle = function(func, delay) {
    var prev = Date.now();
    return function() {
        var context = this;
        var args = arguments;
        var now = Date.now();
        if (now - prev >= delay) {
            func.apply(context, args);
            prev = Date.now();
        }
    }
}
function handle() {
    console.log(Math.random());
}
window.addEventListener('scroll', throttle(handle, 1000));
```

当高频事件触发时，第一次会立即执行（给scroll事件绑定函数与真正触发事件的间隔一般大于delay，如果你非要在网页加载1000毫秒以内就去滚动网页的话，我也没办法o(╥﹏╥)o），而后再怎么频繁地触发事件，也都是每delay时间才执行一次。而当最后一次事件触发完毕后，事件也不会再被执行了 （最后一次触发事件与倒数第二次触发事件的间隔小于delay，为什么小于呢？因为大于就不叫高频了呀( ' ▽ ' )）。

节流throttle代码（定时器）：

```
// 节流throttle代码（定时器）：
var throttle = function(func, delay) {
    var timer = null;
    return function() {
        var context = this;
        var args = arguments;
```

```
        if (!timer) {
            timer = setTimeout(function() {
                func.apply(context, args);
                timer = null;
            }, delay);
        }
    }
}
function handle() {
    console.log(Math.random());
}
window.addEventListener('scroll', throttle(handle, 1000));
```

当触发事件的时候，我们设置一个定时器，再次触发事件的时候，如果定时器存在，就不执行，直到delay时间后，定时器执行执行函数，并且清空定时器，这样就可以设置下个定时器。当第一次触发事件时，不会立即执行函数，而是在delay秒后才执行。而后再怎么频繁触发事件，也都是每delay时间才执行一次。当最后一次停止触发后，由于定时器的delay延迟，可能还会执行一次函数。

节流中用时间戳或定时器都是可以的。更精确地，可以用时间戳+定时器，当第一次触发事件时马上执行事件处理函数，最后一次触发事件后也还会执行一次事件处理函数。

节流throttle代码（时间戳+定时器）：

```
// 节流throttle代码（时间戳+定时器）：
var throttle = function(func, delay) {
    var timer = null;
    var startTime = Date.now();
    return function() {
        var curTime = Date.now();
        var remaining = delay - (curTime - startTime);
        var context = this;
        var args = arguments;
        clearTimeout(timer);
        if (remaining <= 0) {
            func.apply(context, args);
            startTime = Date.now();
        } else {
            timer = setTimeout(func, remaining);
        }
    }
}
function handle() {
    console.log(Math.random());
}
window.addEventListener('scroll', throttle(handle, 1000));
```

在节流函数内部使用开始时间startTime、当前时间curTime与delay来计算剩余时间remaining，当remaining<=0时表示该执行事件处理函数了（保证了第一次触发事件就能立即执行事件处理函数和每隔delay时间执行一次事件处理函数）。如果还没到时间的话就设定在remaining时间后再触发 （保证了最后一次触发事件后还能再执行一次事件处理函数）。当然在remaining这段时间中如果又一次触发事件，那么会取消当前的计时器，并重新计算一个remaining来判断当前状态。

**总结**

函数防抖：将几次操作合并为一此操作进行。原理是维护一个计时器，规定在delay时间后触发函数，但是在delay时间内再次触发的话，就会取消之前的计时器而重新设置。这样一来，只有最后一次操作能被触发。

函数节流：使得一定时间内只触发一次函数。原理是通过判断是否到达一定时间来触发函数。

区别： 函数节流不管事件触发有多频繁，都会保证在规定时间内一定会执行一次真正的事件处理函数，而函数防抖只是在最后一次事件后才触发一次函数。 比如在页面的无限加载场景下，我们需要用户在滚动页面时，每隔一段时间发一次 Ajax 请求，而不是在用户停下滚动页面操作时才去请求数据。这样的场景，就适合用节流技术来实现。