JavaScript 节流（throttle）和防抖(debounce)

节流（throttle）

- 定义：如果一个函数持续的，频繁地触发，那么让它在一定的时间间隔后再触发。
- 比喻：机场排队过安检，当人很多的时候（持续地要进门），安保会隔一段时间放进去几个进行安检（一定时间的间隔）。
- 场景：在浏览器 DOM 事件里面，有一些事件会随着用户的操作不间断触发。比如：重新调整浏览器窗口大小（resize），浏览器页面滚（scroll），鼠标移动（mousemove）。

场景演示

```
function scrollFn(){
  console.log(1)
}

window.onscroll = scrollFn
```

改成节流模式，滑动过程中，每隔200ms运行一次

```
var timer = null;

function scroll() {
  console.log(1)
}

function throttle(delay) {
  if (!timer) {
    timer = setTimeout(function () {
       scroll();
       timer = null;
     }, delay);
   }
}

window.onscroll = function () {
   throttle(200)
};
```

使用闭包改写，使用高阶函数封装函数和时间

```
function scroll() {
  console.log(1)
}

var throttle = function (func, delay) {
  var timer = null;

  return function () {
    if (!timer) {
       timer = setTimeout(function () {
         func();
         timer = null;
       }, delay);
     }
```

```
  }
}

var t = throttle(scroll, 200);
window.onscroll = t
```

防抖(debounce)

- 定义： 作用是在一段时间内多次触发同一个函数，只执行最后一次。
- 比喻：像是两个人的对话，A在不停的balabala（持续触发），如果他说话的时候有停顿（一定间隔），但是停顿的时间不够长，就认为
说完， 当停顿时间超过一某个范围就认为A说完了，然后B开始回答（响应）。
- 场景：输入联想，在连续输入完成后请求

场景演示

```
// 处理函数
function handle() {
  console.log(1);
}

function debounce(fn, delay) {
  var timeout = null;

  return function () {
    if (timeout !== null)
      clearTimeout(timeout);
    timeout = setTimeout(fn, delay);
  }
}

var d = debounce(handle, 200);
window.onscroll = d
```

总结：

1. 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
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。

2. 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。

3. 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
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
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。用户连续输入可以用防抖实现