一、什么是防抖和节流

我们所说的防抖和节流，是函数的防抖和节流，我们在进行窗口的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（debounce）：当持续触发事件时，一定时间段内没有再次触发事件，事件处理函数才会执行一次，如果设定的时间到来之前又触发了事件，就重新开始延时。

前端开发过程中常见的事件例如resize、scroll、mousemove、mousehover等，会被频繁地触发，不做限制的话，有可能一秒执行几十次、几百次，如果在这些函数内部执行了其他函数，尤其是执行了操作DOM的函数，我们知道浏览器操作DOM是很耗性能的，如果事件频繁的触发，不仅会造成计算机资源的浪费，还会降低程序运行速度，甚至造成浏览器卡死、崩溃。

这种问题显然是致命的，除此之外，重复的ajax调用不仅会造成数据关系的混乱，还会造成网络阻塞，增加服务器压力，显然这个问题也是需要解决的，所以“函数防抖”的关键在于，在一个动作发生**一定时间**之后，才会去执行特定事件。

防抖函数存在的问题是：假设一个用户一直触发这个事件，且每次触发事件的时间间隔小于delay，但防抖函数在delay时间内只会执行一次，就会导致原本用户需要在这个时间间隔内多次操作，但程序却告诉你对不起，我只能在这个时间内执行一次。

- 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

函数防抖关注的是一定时间内连续触发只在最后一次执行，函数节流侧重于一段时间内执行一次。

二、防抖和节流的实现

简单的防抖函数代码示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
<button>点击</button>
<script>
  let oBtn = document.getElementsByTagName('button')[0];

  let debounce = (fn, delay) => {
  let timer = null;
  return function (x) {
  timer && clearTimeout(timer);
  timer = setTimeout(() => {
  fn(x);
  }, delay)
  }
  }

  let fn = function () {
  console.log('走你');
  }

  oBtn.onclick = debounce(fn, 1000);
 </script>
 </body>
 </html>
```

简单的节流函数代码示例：

```
<!DOCTYPE html>
<html lang="en">
<head>
```

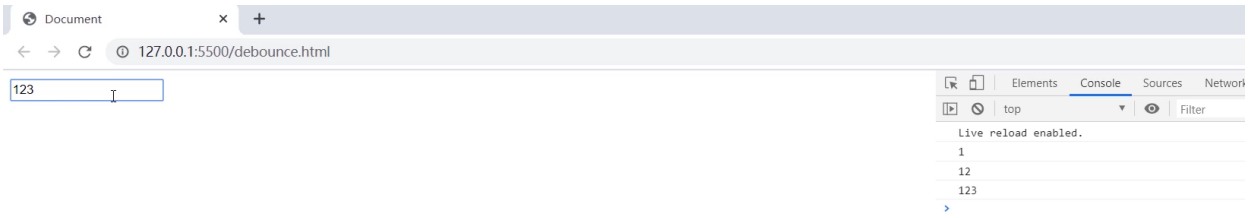
```
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
<button>点击</button>
<script>
  let oBtn = document.getElementsByTagName('button')[0];

  let throttle = (fn, delay) => {
  let flag = true;
  return function () {
  if (!flag) return;
  flag = false;
  setTimeout(() => {
  fn();
  flag = true;
  }, delay);
  }
  }

  let fn = function () {
  console.log('走你');
  }

  oBtn.onclick = throttle(fn, 1000);
</script>
</body>
</html>
```

三、防抖和节流在开发中的使用场景

```html
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>

</head>

<body>
  <input type="text">
  <script>
    let deBounce = (fn, delay) => {
      let timer = null;
      return function (...args) {
        if (timer) {
          clearTimeout(timer);
        }

        timer = setTimeout(() => {
          fn(...args);
        }, delay)
      }
    }
    let oInput = document.getElementsByTagName('input')[0];
    let ajax = (content) => {
      let message = content;
      let json = { message };
      console.log(JSON.stringify(json));
    }
    let doAjax = deBounce(ajax,2000);
    // 不使用防抖
    oInput.addEventListener('keyup',(e)=>{
      console.log(e.target.value);
    })

    // // 使用防抖
    // oInput.addEventListener('keyup',(e)=>{
    //   doAjax(e.target.value);
    // })
```

这是我们在开发中经常使用的一个场景，上面的代码执行结果就是，我们只要按下键盘输入数据，就会触发这次模拟的ajax请求，这样不仅非常浪费资源，而且在实际运用中，用户也是输入完整的字符后才会发起请求。

运行之后可以看到：



我们每次输入值都会触发一次请求，这样非常浪费服务器资源。

如果我们使用上防抖函数

```
<body>
  <input type="text">
  <script>
    let deBounce = (fn, delay) => {
      let timer = null;
      return function (...args) {
        if (timer) {
          clearTimeout(timer);
        }

        timer = setTimeout(() => {
          fn(...args);
        }, delay)
      }
    }
    let oInput = document.getElementsByTagName('input')[0];
    let ajax = (content) => {
      let message = content;
      let json = { message };
      console.log(JSON.stringify(json));
    }
    let doAjax = deBounce(ajax,2000);
    // 不使用防抖
    // oInput.addEventListener('keyup',(e)=>{
    //    console.log(e.target.value);
    // })

    // 使用防抖
    oInput.addEventListener('keyup',(e)=>{
      doAjax(e.target.value);
    })


  </script>
</body>
```

当你在频繁输入的时候它并不会发起请求，只有当你在指定的间隔内没有输入时才会执行函数，如果我们停止输入，但是在指定间隔内又输入会重新触发计时。当用户不断输入时，用

防抖来节约请求资源。（例如有道云笔记这些类似的编辑器工具，使用防抖函数特别好）