

函数防抖和节流

 淘淘笙悦

关注

14

2018.08.04 11:31:10 字数 1,122 阅读 48,889

在前端开发的过程中，我们经常会需要绑定一些持续触发的事件，如 `resize`、`scroll`、`mousemove` 等等，但有些时候我们并不希望在事件持续触发的过程中那么频繁地去执行函数。

通常这种情况下我们怎么去解决的呢？一般来讲，防抖和节流是比较好的解决方案。

让我们先来看看在事件持续触发的过程中频繁执行函数是怎样的一种情况。

html 文件中代码如下

```
1 <div id="content" style="height:150px;line-height:150px;text-align:center; color: #fff;background-color: #ccc;">
2 <script>
3   let num = 1;
4   let content = document.getElementById('content');
5
6   function count() {
7     content.innerHTML = num++;
8   };
9   content.onmousemove = count;
10 </script>
```

在上述代码中，`div` 元素绑定了 `mousemove` 事件，当鼠标在 `div`（灰色）区域中移动的时候会持续地去触发该事件导致频繁执行函数。效果如下



可以看到，在没有通过其它操作的情况下，函数被频繁地执行导致页面上数据变化特别快。所以，接下来让我们来看看防抖和节流是如何去解决这个问题的。

防抖 (debounce)

所谓防抖，就是指触发事件后在 `n` 秒内函数只能执行一次，如果在 `n` 秒内又触发了事件，则会重新计算函数执行时间。

防抖函数分为非立即执行版和立即执行版。

非立即执行版：


```
1 function debounce(func, wait) {
2   let timeout;
3   return function () {
4     let context = this;
5     let args = arguments;
6
7     if (timeout) clearTimeout(timeout);
8
9     timeout = setTimeout(() => {
10       func.apply(context, args);
11     }, wait);
12   };
13 }
```

Standard Chartered 渣打银行



没车没房做抵押？
说贷就贷，月利率低至1%
资料齐全快至1天放款
贷款期限最长5年
注：贷款年利率=贷款月利率*12。

广告

 淘淘笙悦

关注

总资产21 (约2.09元)

Webpack 模块打包机制浅析
阅读 339

实现 `call()`、`apply()` 和 `bind()` 方法
阅读 107

推荐阅读

知乎热门一句话打脸父母，揭开多少
人不敢说的隐痛
阅读 28,056

韩信为什么活埋了母亲？
阅读 19,471

《中餐厅》黄晓明被骂的背后，藏着
一个杠精诞生的底层逻辑
阅读 21,161

杨紫，你真的该火了！
阅读 22,370

成年人社交中，没人告诉你的3个潜
规则
阅读 11,906

DELL 戴尔官网 12.12大促

18999元起拍，最低可降至1元



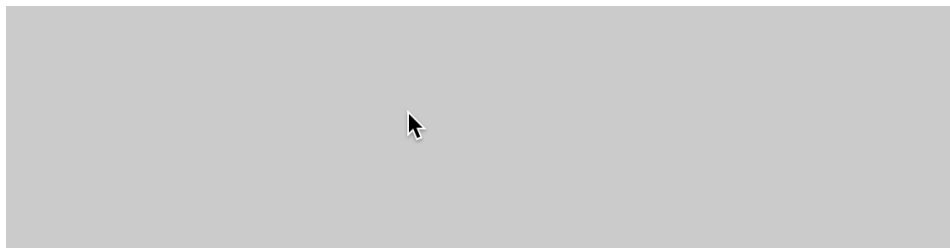
广告

非立即执行版的意思是触发事件后函数不会立即执行，而是在 n 秒后执行，如果在 n 秒内又触发了事件，则会重新计算函数执行时间。

我们依旧使用上述绑定 `mousemove` 事件的例子，通过上面的防抖函数，我们可以这么使用

```
1 | content.onmousemove = debounce(count,1000);
```

效果如下



可以看到，在触发事件后函数 1 秒后才执行，而如果我在触发事件后的 1 秒内又触发了事件，则会重新计算函数执行时间。

上述防抖函数的代码还需要注意的是 `this` 和 参数的传递

```
1 | let context = this;  
2 | let args = arguments;
```

防抖函数的代码使用这两行代码来获取 `this` 和 参数，是为了让 `debounce` 函数最终返回的函数 `this` 指向不变以及依旧能接受到 `e` 参数。

立即执行版：

```
1 | function debounce(func,wait) {  
2 |   let timeout;  
3 |   return function () {  
4 |     let context = this;  
5 |     let args = arguments;  
6 |  
7 |     if (timeout) clearTimeout(timeout);  
8 |  
9 |     let callNow = !timeout;  
10 |    timeout = setTimeout(() => {  
11 |      timeout = null;  
12 |    }, wait)  
13 |  
14 |    if (callNow) func.apply(context, args)  
15 |  }  
16 | }
```

立即执行版的意思是触发事件后函数会立即执行，然后 n 秒内不触发事件才能继续执行函数的效果。

使用方法同上，效果如下

在开发过程中，我们需要根据不同的场景来决定我们需要使用哪一个版本的防抖函数，一般来讲上述的防抖函数都能满足大部分的场景需求。但我们也可以将非立即执行版和立即执行版的防抖函数结合起来，实现最终的双剑合璧版的防抖函数。

双剑合璧版：

```
1  /**
2   * @desc 函数防抖
3   * @param func 函数
4   * @param wait 延迟执行毫秒数
5   * @param immediate true 表立即执行, false 表非立即执行
6   */
7  function debounce(func, wait, immediate) {
8      let timeout;
9
10     return function () {
11         let context = this;
12         let args = arguments;
13
14         if (timeout) clearTimeout(timeout);
15         if (immediate) {
16             var callNow = !timeout;
17             timeout = setTimeout(() => {
18                 timeout = null;
19             }, wait);
20             if (callNow) func.apply(context, args)
21         } else {
22             timeout = setTimeout(function(){
23                 func.apply(context, args)
24             }, wait);
25         }
26     }
27 }
28 }
```

节流 (throttle)

所谓节流，就是指连续触发事件但是在 n 秒中只执行一次函数。节流会稀释函数的执行频率。

对于节流，一般有两种方式可以实现，分别是时间戳版和定时器版。

时间戳版：

```
1  function throttle(func, wait) {
2      let previous = 0;
3      return function() {
4          let now = Date.now();
5          let context = this;
6          let args = arguments;
7          if (now - previous > wait) {
8              func.apply(context, args);
9              previous = now;
10         }
11     }
12 }
```

使用方式如下

写下你的评论...

评论27

赞124

...

1

可以看到，在持续触发事件的过程中，函数会立即执行，并且每 1s 执行一次。

定时器版:

```
1 function throttle(func, wait) {
2   let timeout;
3   return function() {
4     let context = this;
5     let args = arguments;
6     if (!timeout) {
7       timeout = setTimeout(() => {
8         timeout = null;
9         func.apply(context, args)
10      }, wait)
11    }
12  }
13 }
14 }
```

使用方式同上，效果如下



可以看到，在持续触发事件的过程中，函数不会立即执行，并且每 1s 执行一次，在停止触发事件后，函数还会再执行一次。

我们应该可以很容易的发现，其实时间戳版和定时器版的节流函数的区别就是，时间戳版的函数触发是在时间段内开始的时候，而定时器版的函数触发是在时间段内结束的时候。

同样地，我们也可以将时间戳版和定时器版的节流函数结合起来，实现双剑合璧版的节流函数。

双剑合璧版:

```
1 /**
2  * @desc 函数节流
3  * @param func 函数
4  * @param wait 延迟执行毫秒数
5  * @param type 1 表时间戳版, 2 表定时器版
6  */
7 function throttle(func, wait ,type) {
8   if(type===1){
9     let previous = 0;
10   }else if(type===2){
11     let timeout;
12   }
13 }
```

写下你的评论...

评论27

赞124

...