防抖与节流 & 若每个请求必须发送，如何平滑地获取最后一个接口返回的数据

知识总结 2019-10-08 492 1

优化

14:29:35

日常浏览网页中，在进行窗口的 resize、scroll 或者重复点击某按钮发送请求，此时事件处理函数或者接口调用的频率若无限制，则会加重浏览器的负担，界面可能显示有误，服务端也可能出问题，导致用户体验非常糟糕
此时可以采用 debounce（防抖）和 throttle（节流）的方式来减少事件或接口的调用频率，同时又能实现预期效果

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

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，而函数防抖只是在连续触发的事件后才触发最后一次事件的函数

> 上面的解释，摘抄网上的解答

#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 1000ms 内没有触发 scroll 事件时，才会延时触发 scroll 事件


```
function debounce(fn, wait) {
  let timeout = null
```

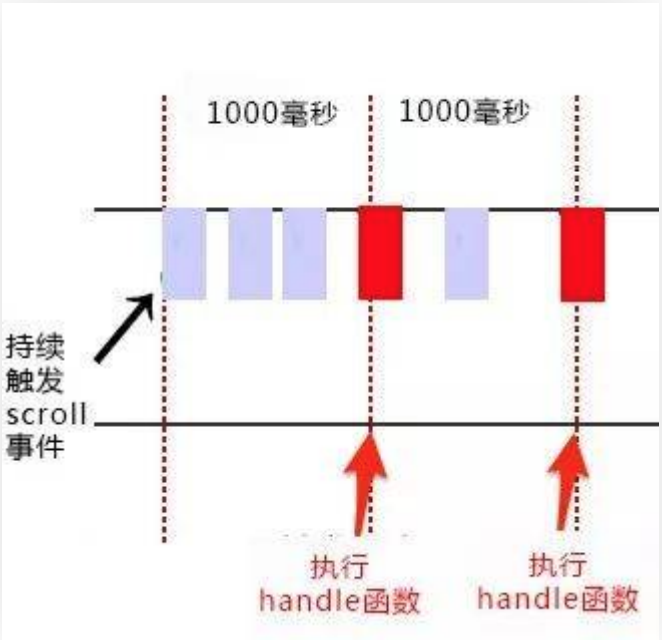
```
    }
    timeout = setTimeout(fn, wait)
  }
}
// 处理函数
function handle() {
  console.log('处理函数', Math.random())
}

// 滚动事件
window.addEventListener('scroll', debounce(handle, 1000))
```

#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
仔细了解了才知道，我以前刚学前端的时候，做 banner 图特效，两边的点击按钮如果一直重复点击就会出问题，后面摸索了此方法，原来这名字叫做节流

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



时间戳方法

```
let throttle = function(func, delay) {
  let prev = Date.now()
  return function() {
    let context = this
```

```
      let args = arguments
      let 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))
```

定时器方法

```
let throttle = function(func, delay) {
  let timer = null
  return function() {
    let context = this
    let 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))
```

时间戳+定时器

```
let throttle = function(func, delay) {
  let timer = null
  let startTime = Date.now()
  return function() {
    let curTime = Date.now()
    let remaining = delay - (curTime - startTime)
    let context = this
```

```
      let 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))
```

#每个请求必须发送的问题

如下图的购买页，操作发现一个购买明细的查价接口的频繁调用问题

购买页改变任何一个选项，都会调用查价接口，然后右边会显示对应的价格。尤其是购买数量，这是一个数字选择器，如果用户频繁点击 + 号，就会连续调用多次查价接口，但 最后一次的查价接口返回的数据才是最后选择的正确的价格

每个查价接口逐个请求完毕的时候， 右边的显示价格也会逐个改变 ，最终变成最后正确的价格，一般来说，这是比较不友好的，用户点了多次后，不想看到价格在变化，尽管最终是正确的价格，但这个变化的过程是不能接受的

也不应该使用上面的防抖解决方式，不能设置过长的定时器，因为查价接口不能等太久，也不能设置过短的定时器，否则会出现上面说的问题（价格在变化）

所以这是一个 每个请求必须发送，但是只显示最后一个接口返回的数据的问题

我这里采用入栈、取栈顶元素比对请求参数的方法解决：

```
// 查价
async getPrice() {
  // 请求参数
  const reqData = this.handleData()
  // push 入栈
  this.priceStack.push(reqData)
  const { result } = await getProductPrice(reqData)
  // 核心代码，取栈顶元素（最后请求的参数）比对
  if(this.$lang.isEqual(this.$array.last(this.priceStack), reqData)) {
    // TODO
    // 展示价格代码...
  }
}
```

注解，上述的 this.lang.isEqual、this.array.last 均是 lodash 插件提供的方法
注册到 Vue 中

```
import array from 'lodash/array'
import Lang from 'lodash/lang'

Vue.prototype.$array = array
Vue.prototype.$lang = Lang
```

本文由 Krry 创作，转载请注明
最后编辑时间：2019-11-07 22:09:11

发表评论

昵称 邮箱 网址(http://)

留下你的足迹...（支持 Markdown）

Emoji | Preview

不愿意透露姓名的小白

2019-11-07