《前端高手进阶》

朱德龙 前中兴软创主任工程师

— 拉勾教育出品 —

第03讲：3 个使用场景助你用好 DOM 事件

DOM 事件

DOM 事件数量非常多，即使分类也有十多种

比如键盘事件、鼠标事件、表单事件等

而且不同事件对象属性也有**差异**

DOM 事件是前端工程师必须掌握的重要内容

同时也是 DOM 的重要组成部分

防抖

绑定 input 元素的键盘事件，然后在监听函数中发送 AJAX 请求

```
const ipt = document.querySelector('input')
ipt.addEventListener('input', e => {
  search(e.target.value).then(resp => {
    // ...
  }, e => {
    // ...
  })
})
```

防抖

1.搜索“l”

2.搜索“la”

3.搜索“lag”

4.搜索“lago”

5.搜索“lagou”

“防抖”功能

为函数的执行设置一个合理的时间间隔

避免事件在时间间隔内**频繁触发**

同时又保证用户输入后能即时看到搜索结果

防抖

setTimeout() 函数来让函数延迟执行

```
//代码1
const ipt = document.querySelector('input')
let timeout = null
ipt.addEventListener('input', e => {
 if(timeout) {
  clearTimeout(timeout)
  timeout = null
 }
 timeout = setTimeout(() => {
  search(e.target.value).then(resp => {
   // ...
  }, e => {
   // ...
  })
 }, 500)
})
```

防抖

拉勾教育
互联网人实战大学

这个操作是完全可以抽取成**公共函数**的

在抽取成公共函数的同时，还需要考虑更复杂的情况

- **参数和返回值如何传递?**
- **防抖化之后的函数是否可以立即执行?**
- **防抖化的函数是否可以手动取消?**

防抖


```
// 代码2
const debounce = (func, wait = 0) => {
  let timeout = null
  let args
  function debounced(...arg) {
    args = arg
    if(timeout) {
      clearTimeout(timeout)
      timeout = null
    }
```

防抖


```
//以Promise的形式返回函数执行结果
return new Promise((res, rej) => {
 timeout = setTimeout(async () => {
  try{
   const result = await func.apply(this, args)
   res(result)
  } catch(e) {
   rej(e)
  }
 }, wait)
```

防抖


```
    }, wait)
  })
}
//允许取消
function cancel() {
  clearTimeout(timeout)
  timeout = null
}
//允许立即执行
function flush() {
```

防抖


```
}
//允许立即执行
function flush() {
  cancel()
  return func.apply(this, args)
}
debounced.cancel = cancel
debounced.flush = flush
return debounced
}
```

节流

一个左右两列布局的查看文章页面

左侧为文章大纲结构，右侧为文章内容

添加功能：

当用户**滚动**阅读右侧文章内容时

左侧大纲相对应部分**高亮**显示，提示用户当前阅读位置

节流


```
//监听scroll事件
wrap.addEventListener('scroll', e => {
 let highlightId = ''
 //遍历大纲章节位置，与滚动距离比较，得到当前高亮章节id
 for (let id in offsetMap) {
  if (e.target.scrollTop <= offsetMap[id].offsetTop) {
   highlightId = id
   break
  }
 }
```

节流


```
const lastDom = document.querySelector('.highlight')
const currentElem =
document.querySelector(`a[href="#${highlightId}"]`)
//修改高亮样式
if (lastDom && lastDom.id !== highlightId) {
  lastDom.classList.remove('highlight')
  currentElem.classList.add('highlight')
} else {
  currentElem.classList.add('highlight')
}
```

节流


```
  const currentElem =
document.querySelector(`a[href="#${highlightId}"]`)
  //修改高亮样式
  if (lastDom && lastDom.id !== highlightId) {
    lastDom.classList.remove('highlight')
    currentElem.classList.add('highlight')
  } else {
    currentElem.classList.add('highlight')
  }
})
```

“节流”功能

设置在指定一段时间内只调用一次函数

从而降低函数调用频率

节流


```
const throttle = (func, wait = 0, execFirstCall) => {
  let timeout = null
  let args
  let firstCallTimestamp


  function throttled(...arg) {
    if (!firstCallTimestamp) firstCallTimestamp = new
Date().getTime()
    if (!execFirstCall || !args) {
      console.log('set args:', arg)
```

节流


```
const throttle = (func, wait = 0, execFirstCall) => {
  let timeout = null
  let args
  let firstCallTimestamp


  function throttled(...arg) {
    if (!firstCallTimestamp) firstCallTimestamp = new
Date().getTime()
    if (!execFirstCall || !args) {
      console.log('set args:', arg)
```

节流


```
    console.log('set args:', arg)
    args = arg
  }
  if (timeout) {
    clearTimeout(timeout)
    timeout = null
  }
  //以Promise的形式返回函数执行结果
  return new Promise(async(res, rej) => {
    if (new Date().getTime() - firstCallTimestamp >= wait) {
      try {
```

节流


```
    try {
      const result = await func.apply(this, args)
      res(result)
    } catch (e) {
      rej(e)
    } finally {
      cancel()
    }
  } else {
    timeout = setTimeout(async () => {
      try {
```

节流


```
    try {
      const result = await func.apply(this, args)
      res(result)
    } catch (e) {
      rej(e)
    } finally {
      cancel()
    }
  }, firstCallTimestamp + wait - new Date().getTime())
}
})
```

节流


```
}
//允许取消
function cancel() {
  clearTimeout(timeout)
  args = null
  timeout = null
  firstCallTimestamp = null
}
//允许立即执行
function flush() {
  cancel()
```

节流


```
  firstCallTimestamp = null
}
// 允许立即执行
function flush() {
  cancel()
  return func.apply(this, args)
}
throttled.cancel = cancel
throttled.flush = flush
return throttled
}
```

代理


```
<ul class="list">
 <li class="item" id="item1">项目1<span class="edit">编辑</span><span class="delete">删除</span></li>
 <li class="item" id="item2">项目2<span class="edit">编辑</span><span class="delete" >删除</span></li>
 <li class="item" id="item3">项目3<span class="edit">编辑</span><span class="delete">删除</span></li>
 ...
</ul>
```

代理

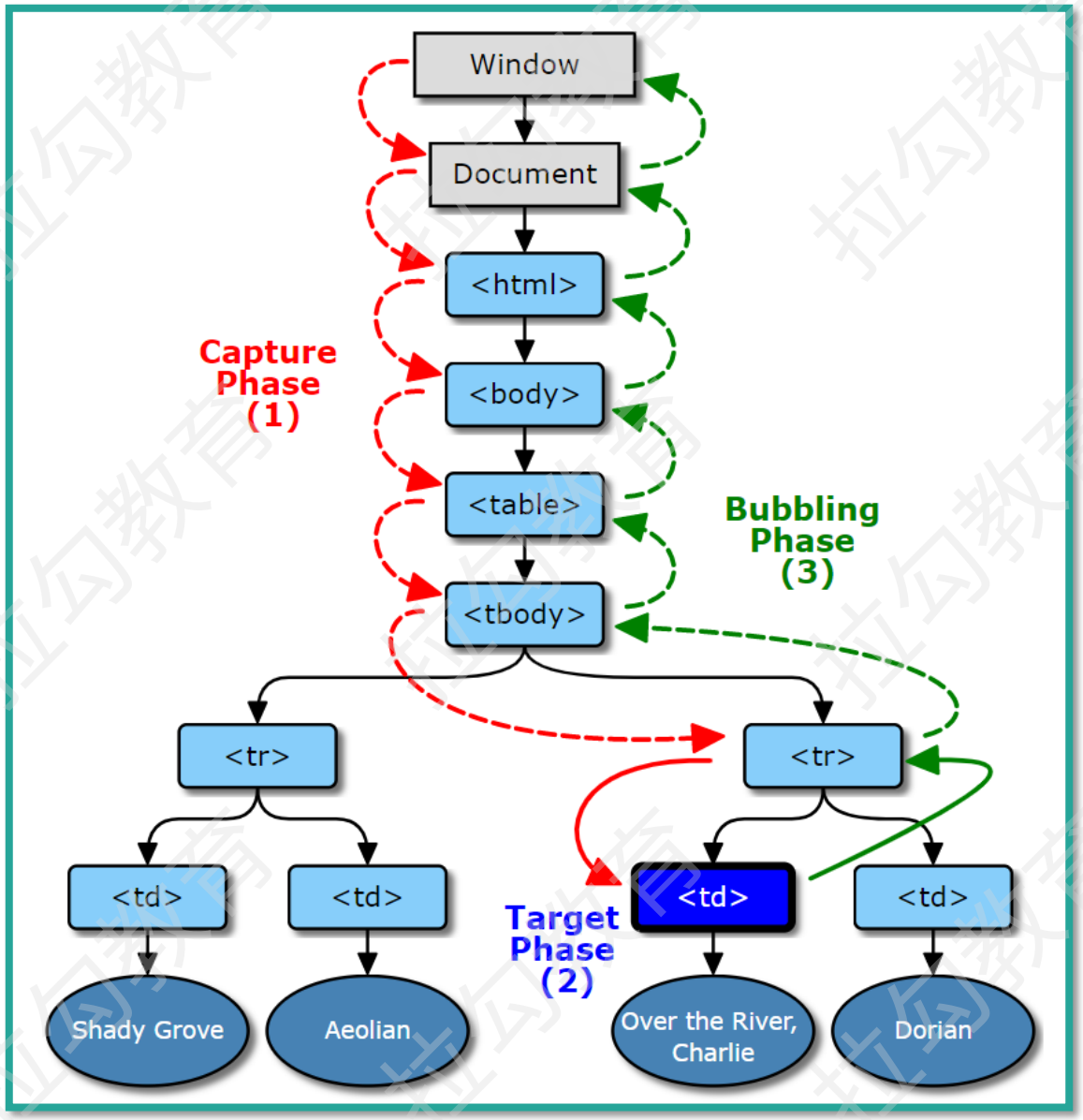
如果数据量一旦增大

事件绑定占用的内存以及执行时间将会成线性增加

其实这些事件监听函数逻辑一致，只是参数不同而已

我们可以以**亍仄以琉**或**亍仄矩扚**来进行优化

代理



代理


```
<body>
 <button>click</button>
</body>
<script>
document.querySelector('button').addEventListener('click', function () {
 console.log('bubble')
})
document.querySelector('button').addEventListener('click', function () {
 console.log('capture')
}, true)
// 执行结果
// buble
// capture
</script>
```

代理

```
const ul = document.querySelector('.list')
ul.addEventListener('click', e => {
  const t = e.target || e.srcElement
  if (t.classList.contains('item')) {
    getInfo(t.id)
  } else {
    id = t.parentElement.id
    if (t.classList.contains('edit')) {
      edit(id)
    } else if (t.classList.contains('delete')) {
      del(id)
    }
  }
})
```

补充：关于 DOM 事件标准


```
// 方式1
<input type="text" onclick="click()"/>
// 方式2
document.querySelector('input').onClick = function(e) {
  // ...
}
// 方式3
document.querySelector('input').addEventListener('click', function(e) {
  //...
})
```

总结

你还能举出关于事件代理在开源项目中使用的例子吗？

点击这里下载示例代码

Next：第04讲《掌握 CSS 精髓：布局》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U