

页面加载

题目

imooc

- ◆ 从输入 url 到渲染出页面的整个过程
- ◆ window.onload 和 DOMContentLoaded 的区别

 慕课网

知识点

- ◆ 加载资源的形式
- ◆ 加载资源的过程
- ◆ 渲染页面的过程

资源的形式

- ◆ html 代码
- ◆ 媒体文件，如图片、视频等
- ◆ javascript css



加载过程

- ◆ DNS 解析：域名 -> IP 地址
- ◆ 浏览器根据 IP 地址向服务器发起 http 请求
- ◆ 服务器处理 http 请求，并返回给浏览器



暂停

07:09 / 09:46

渲染过程 - 1

- ◆ 根据 HTML 代码生成 DOM Tree
- ◆ 根据 CSS 代码生成 CSSOM
- ◆ 将 DOM Tree 和 CSSOM 整合行程 Render Tree



慕课网

渲染过程 - 2

- ◆ 根据 Render Tree 渲染页面
- ◆ 遇到 <script>则暂停渲染，优先加载并执行 JS 代码，完成再继续
- ◆ 直至把 Render Tree 渲染完成



慕课网

性能优化

性能优化原则

- ◆ 多使用内存、缓存或其他方法
- ◆ 减少 CPU 计算量，减少网络加载耗时
- ◆ （适用于所有编程的性能优化 —— 空间换时间）

从何入手

- ◆ 让加载更快
- ◆ 让渲染更快

让加载更快

- ◆ 减少资源体积：压缩代码
- ◆ 减少访问次数：合并代码，SSR 服务器端渲染，缓存
- ◆ 使用更快的网络：CDN



让渲染更快 - 1

- ◆ CSS 放在 head，JS 放在 body 最下面
- ◆ 尽早开始执行 JS，用 DOMContentLoaded 触发
- ◆ 懒加载（图片懒加载，上滑加载更多）



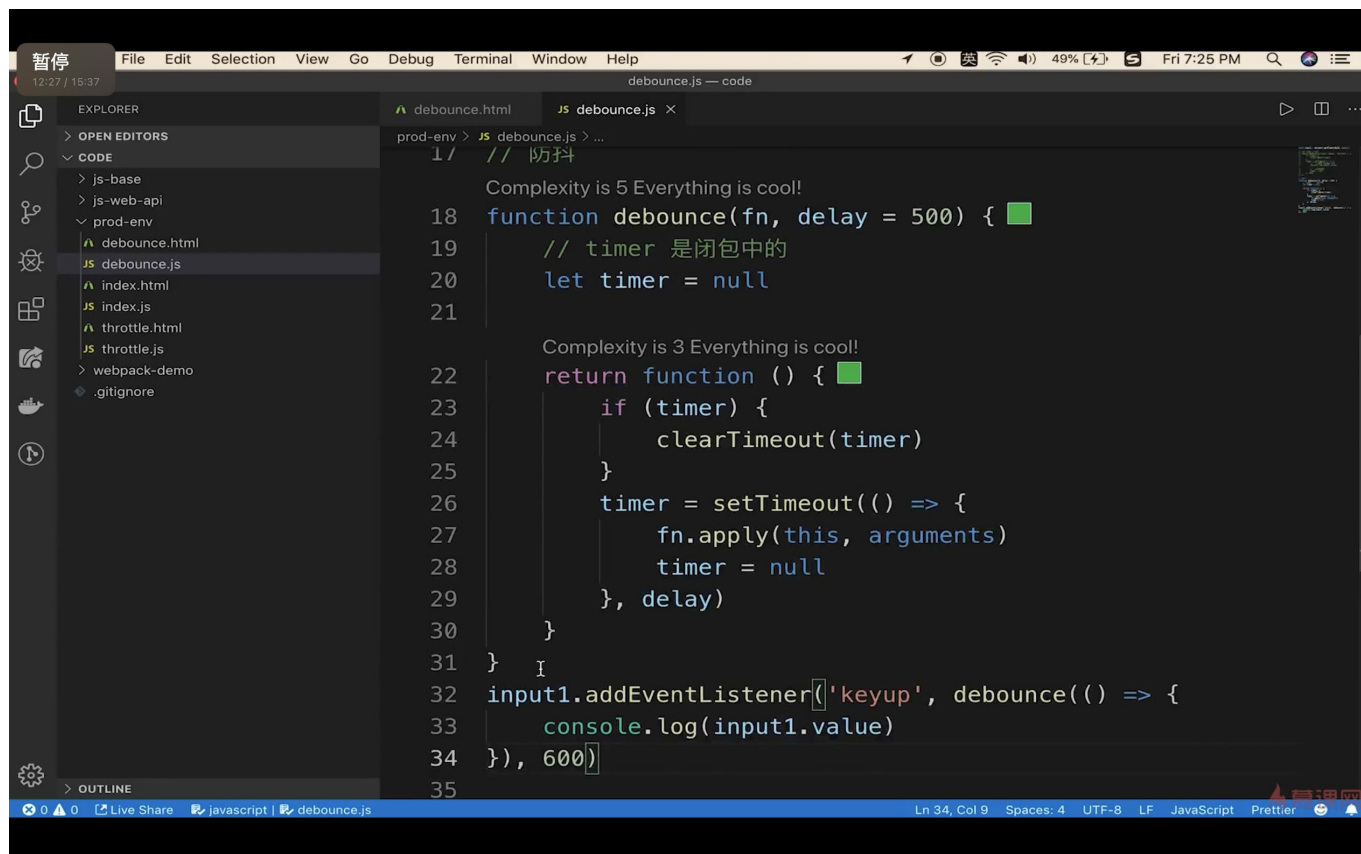
让渲染更快 - 2

- ◆ 对 DOM 查询进行缓存
- ◆ 频繁 DOM 操作，合并到一起插入 DOM 结构
- ◆ 节流 throttle 防抖 debounce



防抖

```
1 const input1 = document.getElementById('input1')
2 let timer = null
3 Complexity is 3 Everything is cool!
4 input1.addEventListener('keyup', function () {
5     if (timer) {
6         clearTimeout(timer)
7     }
8     timer = setTimeout(() => {
9         // 模拟触发 change 事件
10        console.log([input1.value])
11
12        // 清空定时器
13        timer = null
14    }, 500)
15 })
```




```
1 // 防抖
2
3 Complexity is 5 Everything is cool!
4
5 function debounce(fn, delay = 500) {
6   // timer 是闭包中的
7   let timer = null
8
9   Complexity is 3 Everything is cool!
10  return function () {
11    if (timer) {
12      clearTimeout(timer)
13    }
14    timer = setTimeout(() => {
15      fn.apply(this, arguments)
16      timer = null
17    }, delay)
18  }
19 }
20
21 input1.addEventListener('keyup', debounce(() => {
22   console.log(input1.value)
23 }, 600))
```

节流

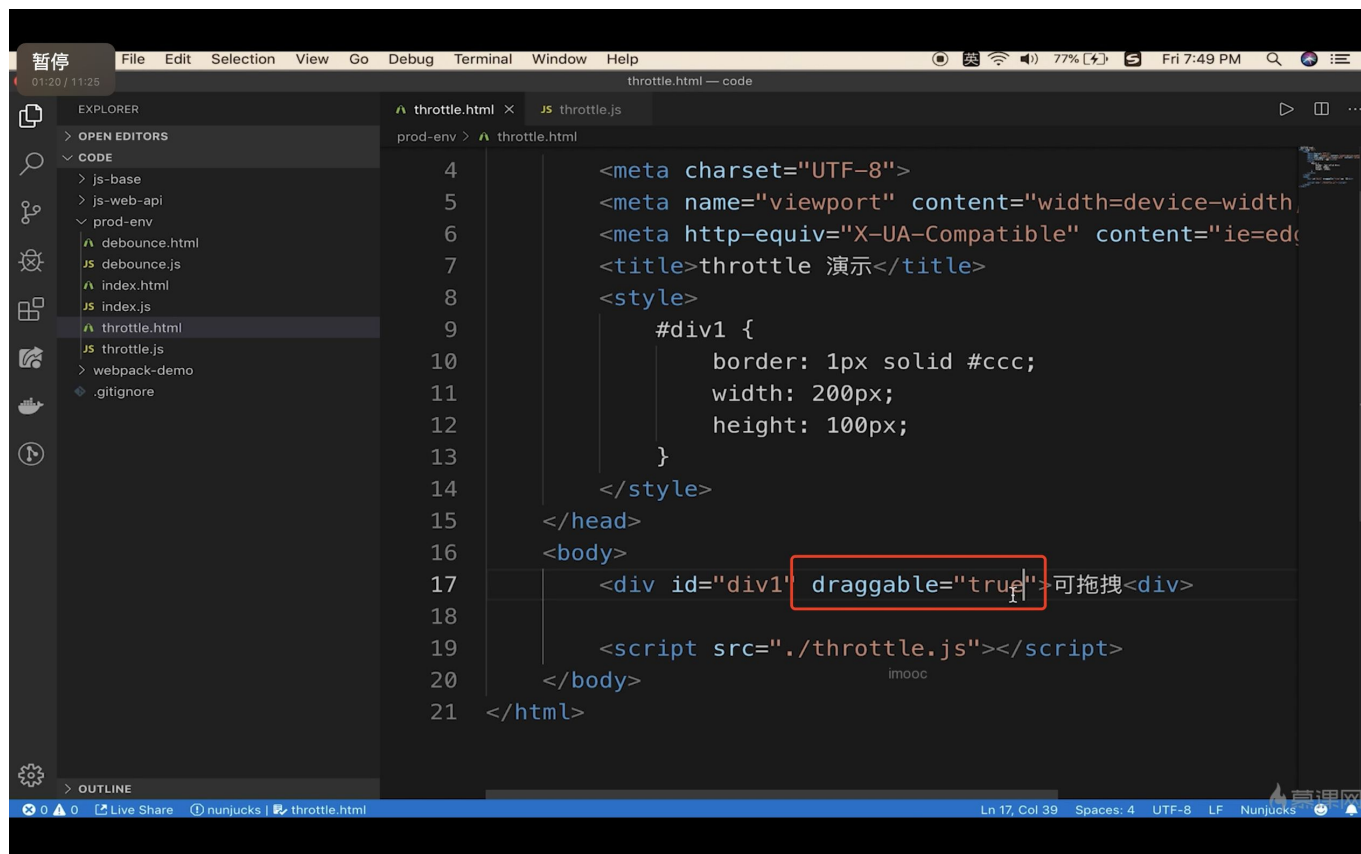
暂停
00:45 / 11:25

节流 throttle

- ◆ 拖拽一个元素时，要随时拿到该元素被拖拽的位置
- ◆ 直接用 drag 事件，则会频发触发，很容易导致卡顿
- ◆ 节流：无论拖拽速度多快，都会每隔 100ms 触发一次

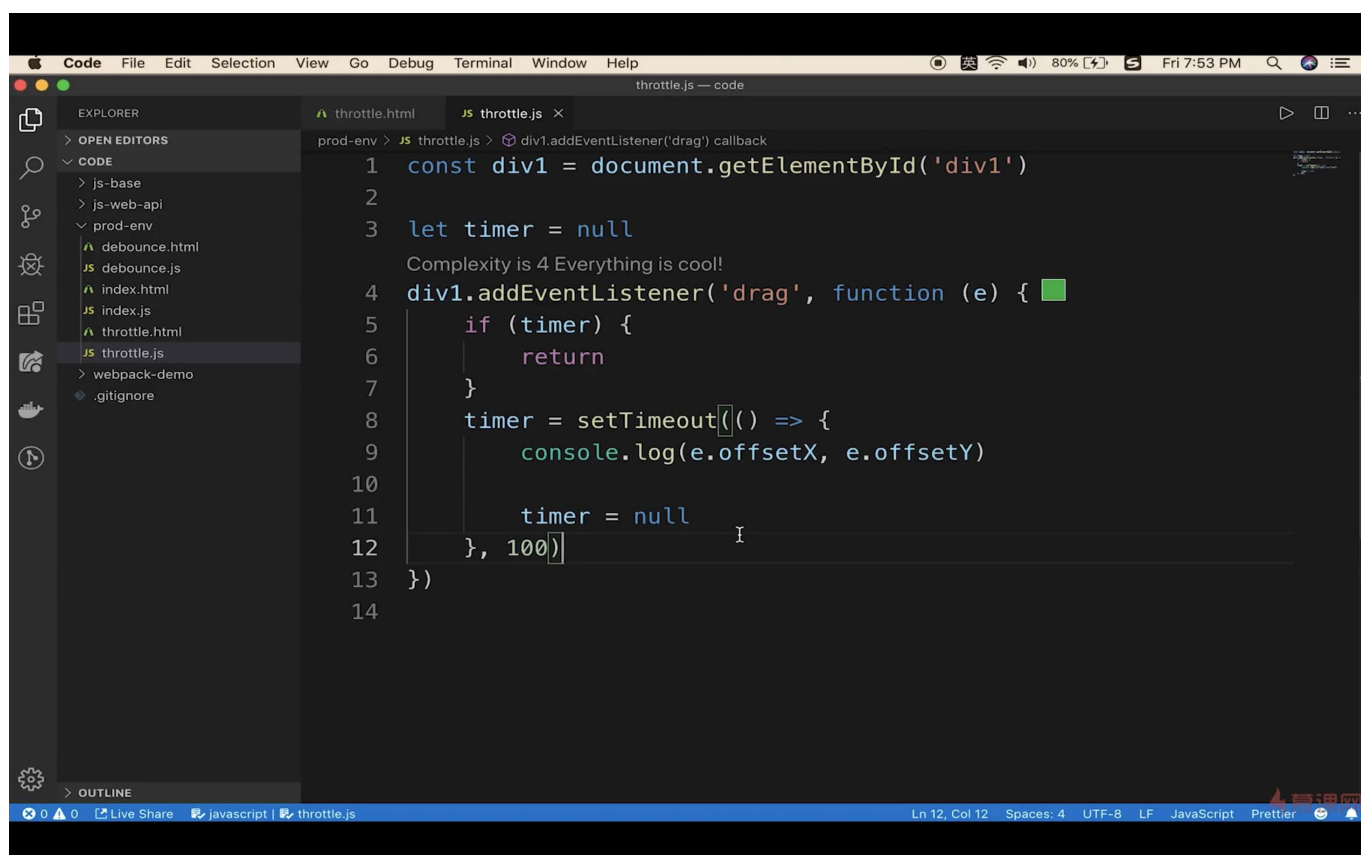


慕课网



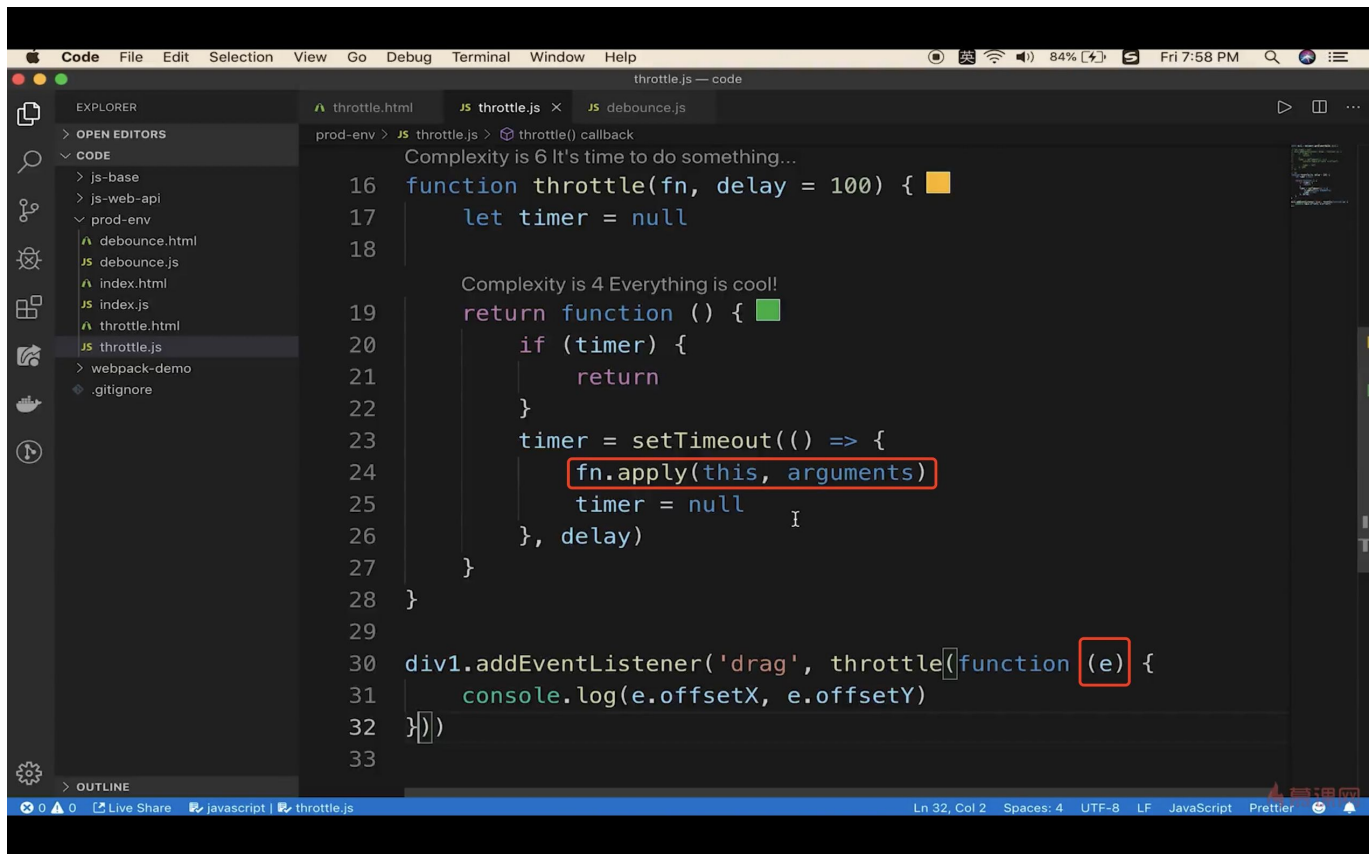
This screenshot shows the VS Code editor with the file `throttle.html` open. The Explorer sidebar on the left shows the project structure, including `prod-env` and `throttle.html`. The main editor area displays the HTML code for `throttle.html`. The code includes a `<div id="div1" draggable="true">` element, which is highlighted with a red box. The status bar at the bottom indicates the current position is Line 17, Column 39.

```
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width,
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>throttle 演示</title>
8      <style>
9          #div1 {
10              border: 1px solid #ccc;
11              width: 200px;
12              height: 100px;
13          }
14      </style>
15  </head>
16  <body>
17      <div id="div1" draggable="true">可拖拽</div>
18
19      <script src="./throttle.js"></script>
20  </body>
21 </html>
```



This screenshot shows the VS Code editor with the file `throttle.js` open. The Explorer sidebar on the left shows the project structure, including `prod-env` and `throttle.js`. The main editor area displays the JavaScript code for `throttle.js`. The code implements a throttle function using `setTimeout` and `clearTimeout` to prevent multiple calls to the callback function within a specified time interval. The status bar at the bottom indicates the current position is Line 12, Column 12.

```
1  const div1 = document.getElementById('div1')
2
3  let timer = null
4  Complexity is 4 Everything is cool!
5  div1.addEventListener('drag', function (e) {
6      if (timer) {
7          return
8      }
9      timer = setTimeout(() => {
10         console.log(e.offsetX, e.offsetY)
11         timer = null
12     }, 100)
13 })
14
```

```
throttle.js — code
prod-env > JS throttle.js > throttle() callback
Complexity is 6 It's time to do something...
16 function throttle(fn, delay = 100) {
17     let timer = null
18
19     Complexity is 4 Everything is cool!
20     return function () {
21         if (timer) {
22             return
23         }
24         timer = setTimeout(() => {
25             fn.apply(this, arguments)
26             timer = null
27         }, delay)
28     }
29
30     div1.addEventListener('drag', throttle(function (e) {
31         console.log(e.offsetX, e.offsetY)
32     }))
33 }
```

安全

imooc

安全

- ◆ XSS 跨站请求攻击
- ◆ XSRF 跨站请求伪造

XSS 攻击

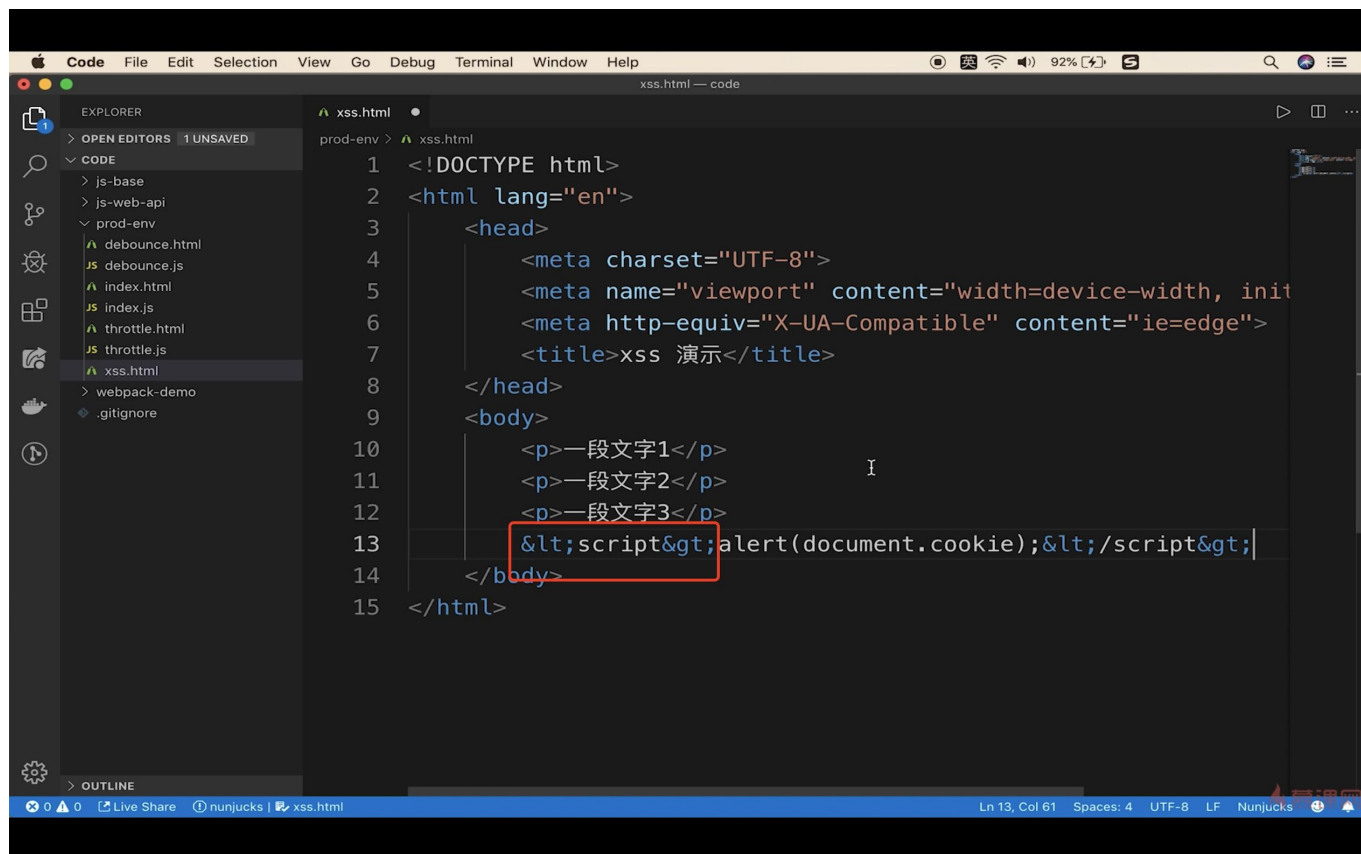
- ◆ 一个博客网站，我发表一篇博客，其中嵌入 `<script>` 脚本
- ◆ 脚本内容：获取 cookie，发送到我的服务器（服务器配合跨域）
- ◆ 发布这篇博客，有人查看它，我轻松收割访问者的 cookie



XSS 预防

- ◆ 替换特殊字符，如 `<` 变为 `<` `>` 变为 `>`
- ◆ `<script>` 变为 `<script>`，直接显示，而不会作为脚本执行
- ◆ 前端要替换，后端也要替换，都做总不会有错





```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>xss 演示</title>
8   </head>
9   <body>
10    <p>一段文字1</p>
11    <p>一段文字2</p>
12    <p>一段文字3</p>
13    <script>alert(document.cookie)</script>
14  </body>
15 </html>
```

XSRF 攻击 - 1

- ◆ 你正在购物，看中了某个商品，商品 id 是 100
- ◆ 付费接口是 xxx.com/pay?id=100，但没有任何验证
- ◆ 我是攻击者，我看中了一个商品，id 是 200

XSRF 攻击 - 2

- ◆ 我向你发送一封电子邮件，邮件标题很吸引人
- ◆ 但邮件正文隐藏着 ``
- ◆ 你一查看邮件，就帮我购买了 id 是 200 的商品



XSRF 预防

- ◆ 使用 post 接口
- ◆ 增加验证，例如密码、短信验证码、指纹等

