**YG课堂批量评分浏览器脚本实践**

**网页自动化与DOM操作：教育评分系统自动化实践**

**1. 项目背景与基础概念**

**1.1 什么是用户脚本？**

想象一下，浏览器就像是一个能播放各种节目的电视机。用户脚本就像是一个遥控器，让你能够控制这个电视做一些它原本不支持的事情——比如自动跳过广告、自动填写表单，或者在我们的案例中，自动点击评分按钮。

|  |
| --- |
| JavaScript // ==UserScript== // @name 批量评分小助手 // @description 帮老师自动完成评分工作 // @match https://evaluation.yungu.org/\* // ==/UserScript==  (function() {  // 这里是脚本代码 })(); |

**2. DOM：网页的"骨架"**

**2.1 网页的骨架：DOM是什么？**

DOM（文档对象模型）可以理解为网页的"骨架"。

想象你正在看一本立体书，书里有各种可以翻动、拉动的小机关。DOM就像是这本书的设计图纸，它告诉浏览器："这里有一个按钮"，"那里有一个文本框"，"这个按钮被点击后应该跳到第几页"。

在我们的评分系统中：

* 每个学生的评分项是DOM中的一个元素
* "精熟"、"基本符合"按钮也是DOM中的元素
* 我们的脚本需要找到这些元素并模拟点击

**2.2 DOM树的实际例子**

评分系统中的一小部分DOM树可能是这样的：

|  |
| --- |
| Plaintext - 学生列表区域  - 学生1行  - 姓名单元格  - 评分状态单元格  - "精熟"标签  - 操作单元格  - 评分按钮(class="radioCheckLevel\_\_\_31V4G")  - 学生2行  - ... |

当我们执行document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')时，就是在这个"骨架"中查找所有评分按钮。

**3. 第一次成功，后面全失败？DOM引用的挑战**

**3.1 消失的按钮：DOM引用问题通俗解释**

想象你在一个魔术表演中：

1. 魔术师给你看了10张牌，编了号
2. 你记住了每张牌的位置
3. 魔术师打了个响指（页面刷新）
4. 虽然看起来牌还在原位，但实际上已经被替换了

这就是我们的脚本遇到的问题：

|  |
| --- |
| JavaScript // 👎 问题写法："记住"所有按钮的位置 const buttons = document.querySelectorAll('.radioCheckLevel\_\_\_31V4G'); // 之后使用这些"记忆"中的位置 buttons[5].click(); // 可能失败  // 👍 正确写法：每次需要时重新"看"一遍 document.querySelectorAll('.radioCheckLevel\_\_\_31V4G')[5].click(); |

**3.2 "幽灵按钮"：为什么按钮看起来在但点不了？**

在我们的评分系统中，当你点击第一个评分按钮并完成操作后，页面会进行局部更新。虽然在屏幕上看起来没什么变化，但在"幕后"，整个学生列表可能已经被重新创建了！

这就像是：

* 你指着餐厅窗口的一个座位说"那是我的座位"
* 但餐厅刚刚装修，换了新桌椅
* 虽然新座位看起来和原来一模一样，位置也相同
* 但你之前指的那个具体的座位已经被搬走了

这就是为什么第一次评分成功，但后续操作失败——脚本仍在试图点击已经不存在的"幽灵按钮"。

**4. 现代网页如何工作：React和虚拟DOM**

**4.1 React如何更新页面：通俗解释**

云谷评分系统可能使用了React等现代框架。这些框架的工作方式可以类比为：

想象你是一个画家，需要修改一幅墙上的壁画：

* **传统方式**（直接DOM操作）：直接在墙上画，每改一处就要动一次笔
* **现代方式**（React虚拟DOM）：

1. 先在草稿纸上画出整幅画的新版本
2. 仔细比较草稿和墙上的画有什么不同
3. 只修改不同的部分

**4.2 为什么评分系统会"重建"整个列表？**

在评分系统中，即使你只评分了一个学生，React可能会重新渲染整个学生列表。这不是bug，而是设计如此：

* 简化开发：让开发者可以把页面视为状态的函数，而不用记住每个DOM操作
* 优化性能：React会计算最小必要的DOM更新，虽然看起来"重建"了，但实际的浏览器工作可能很少

对我们的脚本来说，这意味着存储的DOM引用会失效，我们需要每次操作前重新获取最新的DOM元素。

**5. 解决方案：评分系统自动化实现**

**5.1 成功的批量评分脚本**

下面是一个能够成功对多个学生进行评分的脚本：

|  |
| --- |
| JavaScript // ==UserScript== // @name 批量评分小助手 // @namespace http://tampermonkey.net/ // @version 1.0 // @description 帮老师自动完成评分工作 // @match https://evaluation.yungu.org/\* // @grant none // ==/UserScript==  (function() {  'use strict';    // 创建一个漂亮的绿色按钮  const button = document.createElement('button');  button.textContent = '批量初始化评分';    // 设置按钮样式...使它在右下角显示  button.style.position = 'fixed';  button.style.bottom = '20px';  button.style.right = '20px';  button.style.zIndex = '9999';  button.style.padding = '10px 15px';  button.style.backgroundColor = '#4CAF50';  button.style.color = 'white';  button.style.border = 'none';  button.style.borderRadius = '5px';  button.style.cursor = 'pointer';  button.style.boxShadow = '0 2px 5px rgba(0,0,0,0.3)';  button.style.fontSize = '14px';    // 添加鼠标悬停效果  button.addEventListener('mouseover', function() {  this.style.backgroundColor = '#45a049';  });    button.addEventListener('mouseout', function() {  this.style.backgroundColor = '#4CAF50';  });    // 点击按钮时的操作  button.addEventListener('click', async function() {  // 每次重新获取所有评分按钮 - 这是关键！  const radioCheckLevels = document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]');  console.log(`找到 ${radioCheckLevels.length} 个待评分项`);    // 显示进度条  const progressIndicator = document.createElement('div');  progressIndicator.textContent = '准备中...';  progressIndicator.style.position = 'fixed';  progressIndicator.style.top = '20px';  progressIndicator.style.right = '20px';  progressIndicator.style.backgroundColor = '#333';  progressIndicator.style.color = '#fff';  progressIndicator.style.padding = '10px';  progressIndicator.style.borderRadius = '5px';  progressIndicator.style.zIndex = '10000';  document.body.appendChild(progressIndicator);    // 记录成功和失败的数量  let successCount = 0;  let failCount = 0;    // 逐个处理每个评分项  for (let i = 0; i < radioCheckLevels.length; i++) {  // 更新进度条  progressIndicator.textContent = `正在处理: ${i+1}/${radioCheckLevels.length}`;    try {  // 重点1：每次都重新获取最新的DOM元素  console.log(`点击第 ${i+1} 个评分按钮`);  document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[i].click();    // 重点2：检查是否有图片，不同类型的作业需要不同评分  let hasImg = document.querySelectorAll('div[class="evidence\_\_\_1oYV-"]')[i].querySelector("img");    // 等待弹窗显示  await new Promise(resolve => setTimeout(resolve, 2000));    // 找到弹出的选项按钮  const radioChecks = document.querySelectorAll('div[class="radioCheck\_\_\_JcdAv"]');  console.log(`找到 ${radioChecks.length} 个评分选项`);    if (radioChecks.length > 1) {  // 根据作业类型选择不同的评分  if(hasImg) {  // 有图片的作业选"精熟"  radioChecks[1].click();  console.log("选择了精熟");  } else {  // 无图片的作业选择其他评级  radioChecks[3].click();  console.log("选择了基本符合");  }  successCount++;    // 等待评分完成  await new Promise(resolve => setTimeout(resolve, 2000));  } else {  console.log(`警告: 第 ${i+1} 个评分项未找到足够的选项`);  failCount++;  }  } catch (error) {  console.error(`处理第 ${i+1} 个评分项时出错:`, error);  failCount++;  }  }    // 移除进度条  document.body.removeChild(progressIndicator);    // 显示操作结果  alert(`评分完成!\n成功: ${successCount} 个\n失败: ${failCount} 个`);  });    // 将按钮添加到页面  document.body.appendChild(button); })(); |

**5.2 核心要点解析**

脚本成功的关键在于：

1. **每次重新获取元素**：不依赖存储的引用，而是每次操作前重新查询DOM

|  |
| --- |
| JavaScript document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[i].click(); |

1. **区分不同类型的作业**：

|  |
| --- |
| JavaScript // 方法一：检查当前作业是否包含图片 let hasImg = document.querySelectorAll('div[class="evidence\_\_\_1oYV-"]')[i].querySelector("img");  if(hasImg) {  radioChecks[1].click(); // 选择"精熟" } else {  radioChecks[3].click(); // 选择"基本符合" } |

1. **适当的等待时间**：给页面留出足够时间进行DOM更新

|  |
| --- |
| JavaScript await new Promise(resolve => setTimeout(resolve, 2000)); |

**6. 进阶：直接调用评分系统API**

**6.1 为什么考虑API调用？**

想象一下，如果你可以不通过点击按钮，而是直接给老师发个短信说"请把小明的作业评为精熟"，这就是API调用的概念。

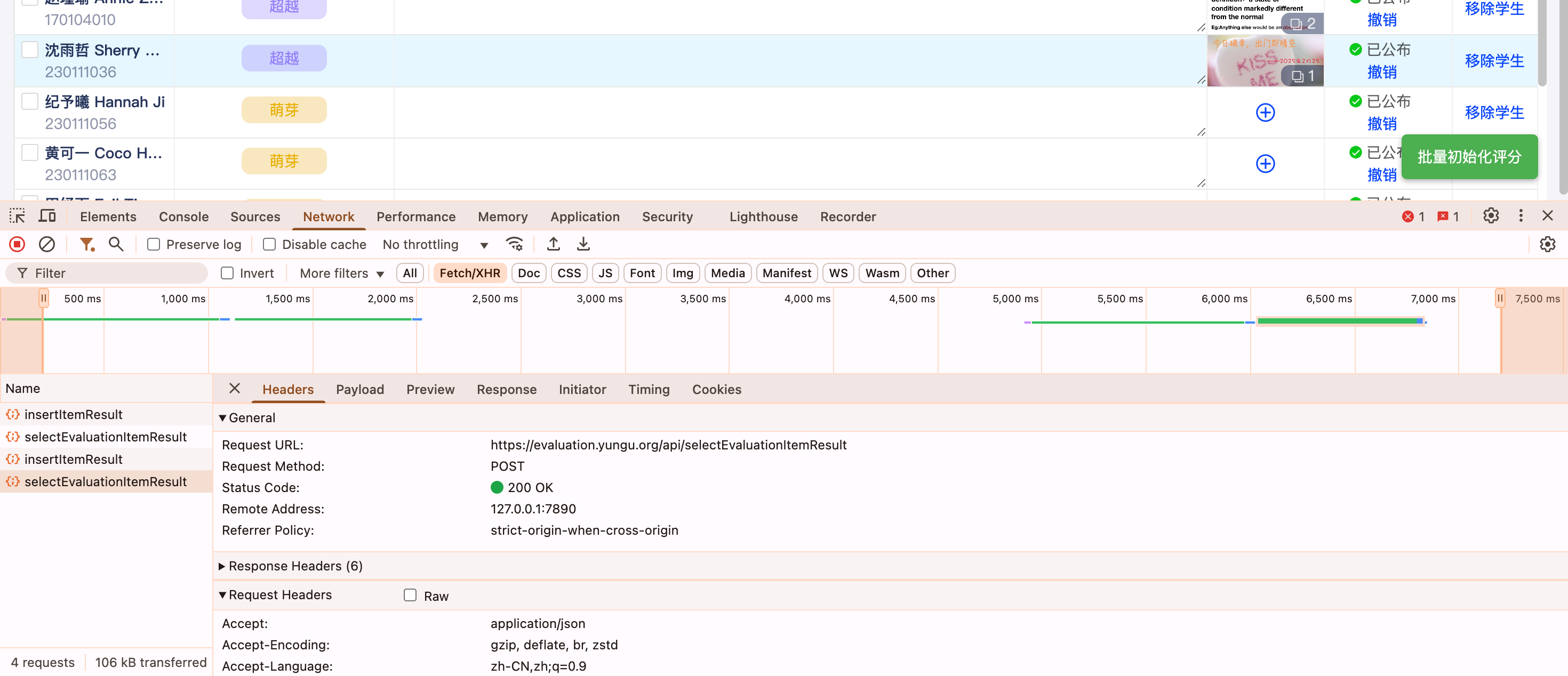
优点：

* 更快：不需要等待页面加载和按钮点击
* 更可靠：不受页面变化影响
* 更高效：可以一次性评分多个学生

**6.2 如何找到评分系统的API？**

就像侦探工作，我们需要观察系统是如何与服务器通信的：

1. 打开评分系统页面
2. 按F12打开开发者工具，切换到"网络"标签
3. 手动评分一个学生作业，观察出现的网络请求
4. 寻找类似"setRating"、"updateScore"之类的请求
5. 查看请求的URL、参数和响应



**6.3 API调用示例**

找到API后，我们可以这样调用它：

|  |
| --- |
| JavaScript async function setRatingViaAPI(studentId, taskId, rating) {  try {  // 这里的URL和参数需要根据实际观察到的请求进行调整  const response = await fetch('https://evaluation.yungu.org/api/setStudentRating', {  method: 'POST',  headers: {  'Content-Type': 'application/json',  // 可能需要认证信息  'Authorization': 'Bearer ' + localStorage.getItem('token')  },  body: JSON.stringify({  studentId: studentId,  taskId: taskId,  rating: rating, // 例如 "mastery", "basic", 等  })  });    const result = await response.json();  if (result.success) {  console.log(`学生${studentId}的评分设置成功`);  return true;  } else {  console.error(`评分失败: ${result.message}`);  return false;  }  } catch (error) {  console.error('API调用出错:', error);  return false;  } }  // 使用示例 setRatingViaAPI('230111067', 'task001', 'mastery'); |

***小贴士****：使用API方法前，一定要先观察真实请求的格式和参数。错误的参数可能导致评分失败甚至错误的评分！*

**7. 实用调试技巧与问题解决**

**7.1 在浏览器控制台中测试命令**

一个非常实用的调试方法是先在浏览器控制台中手动测试你的命令：

1. 打开浏览器开发者工具（F12）
2. 切换到"控制台"（Console）标签
3. 输入并执行你计划在脚本中使用的命令

例如，在我们的评分系统中，可以这样测试：

|  |
| --- |
| JavaScript // 点击第一个评分按钮 document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[0].click() // 点击"精熟"选项 document.querySelectorAll('div[class="radioCheck\_\_\_JcdAv"]')[1].click()  // 点击第二个评分按钮 document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[1].click() // 点击"精熟"选项 document.querySelectorAll('div[class="radioCheck\_\_\_JcdAv"]')[1].click() |

这种方法的优势：

* 立即看到每个命令的效果
* 确认选择器是否正确
* 在开发脚本前验证思路

**7.2 使用debugger进行脚本调试**

当脚本行为不符合预期时，使用debugger语句可以帮助你找出问题：

|  |
| --- |
| JavaScript button.addEventListener('click', async function() {  // 获取所有评分按钮  const radioCheckLevels = document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]');    for (let i = 0; i < radioCheckLevels.length; i++) {  debugger; // 脚本将在此处暂停执行  document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[i].click();    // 更多代码...  } }); |

使用debugger的好处：

* 脚本会在debugger语句处暂停
* 可以检查当前的变量值
* 可以在控制台执行命令测试
* 可以单步执行查看每一步的结果

**7.3 脚本循环失败但手动执行成功的情况分析**

一个常见且令人困惑的情况是：在控制台中逐个执行命令都能成功，但放入脚本的循环中就只有第一项成功。例如：

|  |
| --- |
| JavaScript // 控制台中这些命令逐个执行都能成功 document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[0].click() document.querySelectorAll('div[class="radioCheck\_\_\_JcdAv"]')[1].click() document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[1].click() document.querySelectorAll('div[class="radioCheck\_\_\_JcdAv"]')[1].click() document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[2].click() document.querySelectorAll('div[class="radioCheck\_\_\_JcdAv"]')[1].click()  // 但在脚本循环中却只有第一个成功 for (let i = 0; i < 3; i++) {  document.querySelectorAll('div[class="radioCheckLevel\_\_\_31V4G"]')[i].click();  setTimeout(() => {  document.querySelectorAll('div[class="radioCheck\_\_\_JcdAv"]')[1].click();  }, 2000); } |

**为什么会这样？**

这通常是由于：

1. **事件循环与异步操作**：循环会立即执行所有点击，而不等待每次点击的结果
2. **弹窗处理机制**：第一个弹窗显示后，系统可能阻止了后续弹窗
3. **事件冲突**：快速连续的点击事件可能导致系统忽略部分事件

**解决方案：**

1. 使用async/await与Promise进行真正的顺序执行
2. 每次操作后添加充足的等待时间
3. 确保每次操作前获取最新的DOM状态【这次主要是这个问题。】
4. 考虑使用API调用替代UI操作

***实际案例****：在我们的评分系统中，即使增加了setTimeout延时，循环中仍然只有第一次评分成功，评分小弹窗（超越、精熟、生长、萌芽）后续不再显示。这可能是因为评分系统DOM实时更新的原因。*

**7.4 新版本评分系统适配**

教育系统经常更新，如果脚本突然不工作了：

1. **检查选择器**：类名可能已更改（如radioCheckLevel\_\_\_31V4G）
2. **观察新流程**：评分步骤可能有变化
3. **寻找新API**：系统更新可能改变了API接口
4. **调整等待时间**：新系统可能需要更长/更短的响应时间

**总结：从失败到成功的关键**

在这个教育评分系统自动化项目中，我们学到了：

1. 现代网页框架（如React）会重建DOM元素，导致存储的元素引用失效
2. 成功的自动化脚本必须每次操作前重新获取最新DOM元素
3. 不同类型的学生作业可能需要不同的评分策略
4. 适当的等待时间对于确保操作成功至关重要
5. 直接API调用可能是更高效的替代方案

通过理解这些概念，我们成功将评分工作从繁琐的手动点击转变为一键自动处理，节省了大量宝贵的时间。

**完整的脚本文件**

**[YG课堂批量登记评分.js]**

**250307 更新**

优化功能：

1. Settimeout 时间更新，能够
2. 支持多评价项场景（1个任务多个评价维度）

* 智能处理不同评估项与证据项的比例关系

1. 新增智能初始化功能，可跳过已有评价的项目

* 双按钮界面设计，用不同颜色区分功能

1. 提高响应速度

使用MutationObserver替代固定等待时间，提高响应速度和稳定性

添加三个专用DOM观察函数：waitForElement、waitForElements、waitForElementToDisappear

添加可视化进度条，直观显示处理进度

添加实时日志面板，显示详细操作过程和状态

日志分类显示（普通、警告、错误）

改进错误处理和恢复机制

添加超时设置，防止无限等待

1. 完全重构为模块化架构（Config、UI、Util、DOMObserver、Evaluator、Controller）
2. 添加时间统计功能，显示总用时和每项平均处理时间
3. 集中所有配置参数到Config模块，便于统一管理

**[updated-script.js]**

**250307 更新**

1. 调整按钮位置，增加悬停功能

**[modular-script (1).js]**

**参考资料**

* [DOM MDN文档](https://developer.mozilla.org/zh-CN/docs/Web/API/Document_Object_Model)
* [Tampermonkey文档](https://www.tampermonkey.net/documentation.php)
* [React虚拟DOM概念](https://reactjs.org/docs/faq-internals.html)