

PYTHON 2025

**专业：数据科学与大数据**

**学号： 2452086**

**学生： 郎子萱**

1. **开发环境简介**

**操作系统：**Windows11

**Python版本：**3.13.5

**IDE版本：**Spyder 6.0.7

1. **爬虫技术介绍**

**（一）本程序整体介绍**

写第一版的时候，我的思路是：先抓取193个网页，提取每个网页的链接，再直接爬这些链接，将内容发放到excel中。

这样是可行的，但当代码报错的时候，又要重新抓193个网页的所有链接，再爬取，整个过程十分的慢。所以我将整件事分为两步，程序由两个核心函数组成。

第一个函数get\_news\_links，用于获取新闻链接，并保存到csv中。这个“保存csv”操作可以让我们只运行一次这个函数就ok了。

第二个函数crawl\_news\_page用于爬取新闻详情页，即每一个链接中的正文内容。

**（二）爬虫相关库介绍与心得**

在最开始的时候，我用到了两个库，requests和BeatifulSoup。

**1 requests**

**版本：2.32.2**

requests 是一个用于发送 HTTP 请求的 Python 库，主要负责从网页获取原始 HTML 内容。

在我的程序中，它用于抓取新闻列表页的静态内容，例如 get\_news\_links 函数（如图1所示）中通过 requests.get(list\_url, headers=headers, timeout=15) 获取页面源代码等。

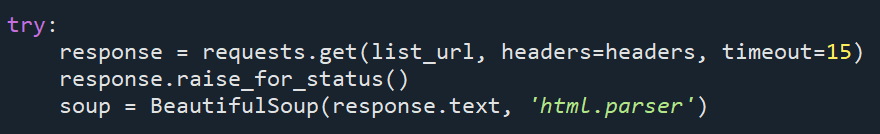


图1

它有很多优势，比如提供了简洁的API，不需要启动浏览器，直接通过 HTTP 请求获取静态 HTML，速度快，资源占用低，能帮助我们快速开发；并且，它有很强的灵活性，支持多种HTTP方法和参数配置，比如，在我的程序中，用到“headers”设置User-Agent，这样可以有效避免一些网站的反爬机制。

同时也存在一些劣势。比如无法处理动态内容、依赖网络响应、无法模拟浏览器的完整行为等。

**2 BeautifulSoup4**

**版本：4.12.3**

BeautifulSoup 是一个 HTML 和 XML 解析库，用于从获取的网页源代码中提取结构化数据

在我的程序中，它用于解析 requests 或 Selenium 获取的 HTML，提取新闻标题、编辑、发布时间、浏览量和正文等信息（图2展示了利用BeautifulSoup中find操作提取“标题”的代码）

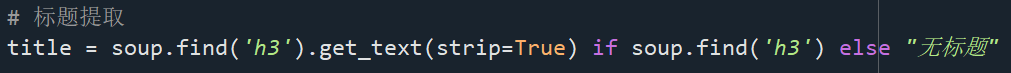


图2

BeautifulSoup 是一个简单易用的 Python HTML/XML 解析库，最大的优势在于其极其友好的语法和强大的容错性。即使面对结构混乱或不规范的 HTML 代码，它也能稳定解析而不会报错中断。它提供了多种灵活的元素查找方式，包括 find()、find\_all() 和 CSS 选择器等，让开发者能够用直观的代码快速定位和提取所需数据。同时，BeautifulSoup 自动处理编码问题，支持丰富的文本清理和 DOM 操作功能，大大降低了网页数据提取的开发门槛。

其主要劣势在于性能相对较慢，特别是在处理大量文档时。BeautifulSoup只能解析静态 HTML 内容，无法执行 JavaScript。此外，它的内存占用较大，因为它需要在内存中构建完整的文档树结构，对于超大型文件处理不够理想。

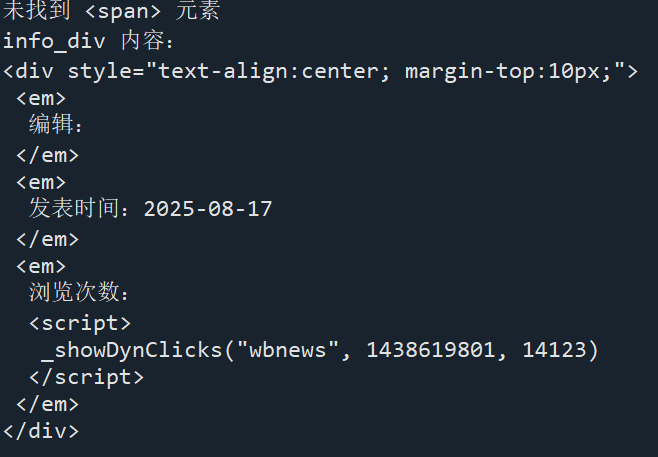
这两个库能提取绝大部分的数据，除了“浏览量”。

由于“浏览量”是由JavaScript 动态生成的，所以只使用上面两个库，无法直接读取<span>中的内容来读出数字。

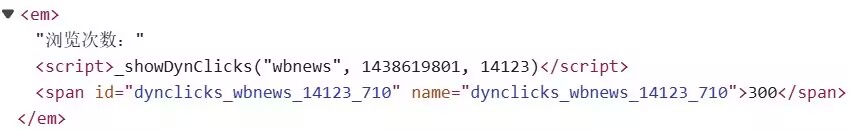
在仅使用上面两个库的时候，我们用以下程序进行分析：



得到了以下结果：



但HTML的结构如下图：



显然，没有提取到红色方框中<span><span>之间的内容！也就是说，无法正确读出这个“300”

这个时候，**BeatifulSoup**就没那么好用了，因为它只能静态解析HTML，我们转向了**Selenium**寻求帮助，因为**Selenium** 可以模拟浏览器环境，执行 JavaScript 并获取动态内容。

**3** **Selenium**

**版本：4.35.0**

Selenium 是一个自动化测试工具，可模拟浏览器行为，执行 JavaScript 并获取动态渲染的网页内容。

在我的程序中，它主要用于 crawl\_news\_page 函数，解决 requests 无法获取动态浏览量的问题。

**Selenium** 最大的优势在于能够完全模拟真实用户操作，处理各种动态加载内容。它可以执行 JavaScript，等待 Ajax 请求完成，自动填充表单，点击按钮，甚至处理复杂的用户交互流程。这使得 **Selenium** 能够爬取那些依赖前端渲染的现代网页应用，解决了许多传统爬虫无法处理的动态内容问题。同时，**Selenium** 支持多种浏览器，提供丰富的等待机制和元素定位策略，能够应对各种复杂的网页场景。

**Selenium** 的主要劣势在于性能开销巨大和资源消耗高。启动浏览器实例需要大量内存，运行速度远慢于轻量级的请求库，不适合大规模数据采集任务。它配置复杂，需要安装浏览器驱动，环境部署相对麻烦。此外，**Selenium** 容易被网站检测到自动化特征，可能触发反爬虫机制。维护成本也较高，网页结构的微小变化就可能导致脚本失效，需要频繁调整定位策略。

**（三）HTML,CSS和正则表达式的学习**

在构建整体程序之前，我们首先做的事，应该都是按下“F12”，查看网页的结构。一个**HTML**由许多部分组成，它通过标签构建内容结构，具有清晰的层级关系和语义化特征，每个标签都有其特定用途，如<div>用于布局，<p>用于段落等。

**CSS**起到一个“美化布局”的作用，通过选择器精准定位元素，利用盒模型控制布局，支持层叠样式和响应式设计，能够实现从简单配色到复杂动画的视觉效果。

**正则表达式**如同文本处理的“智能扫描仪”。通过特定模式匹配字符串，具有强大的搜索、替换和验证能力。初看其实不太好适应，因为它有一套自己的语法，但是其功能性非常强大！

1. **本课程学习体会**

Python小学期很快就结束了。一周三节课的课时安排还算轻松，这给我们了很多时间去消化一些课堂和作业上的内容。

对于课后练习，我一般是自己写一遍，然后再去问问AI有什么优化的建议（除了不会写的题）。这个时候，就体现出来python语言相较于c++的强大的、简洁的“魔力”。我仍然停留在c语言传统的范式中，但每每这个时候，AI就会给我一个更加简洁的写法，比如，用列表推导式代替更大代码量的for循环。

每节课前，曹建平老师都会分享一下别的同学的代码，这是一个很有意思的环节。有些代码非常简洁，一行顶我十行的效果。把这些好的代码拍下来学习，让我对代码的理解能力提升很快。

最终的考核形式是小测＋大作业。小测的题目难度不大，大作业也很“实用”，给了我们自学的机会，让我们去探索“爬虫”这个技术，以及背后相关的库函数和各种网页相关知识。整个学习过程让我感到有一些挑战，也收获颇丰。

最后，想向您提出一点建议：您说后交作业的同学，会扣除一些分数。我理解您的考量，但是，对于我来说，要搞清楚完成作业所需的知识，是需要一定时间的，但我又想迅速交上作业保证不被扣分，就会导致有想“全部交给AI”的冲动（虽然最后还是没这么做，觉得搞懂更重要一点），而不求甚解了。所以我认为，是否能不扣除后交作业的同学的分数？或许他们只是需要一些时间来消化和弄懂作业所需的知识。