DOI:10.19651/j.cnki.emt.2518298

网络样本数据轻量化采集脚本语言设计与实现*

曹炳尧 张秘豪
(上海大学通信与信息工程学院 上海 200444)

摘 要:样本数据的完整度与新鲜度直接决定了机器学习模型的泛化能力与预测精度。网络作为开放环境下的核心数据来源,可为模型训练提供广覆盖、高实时的样本支持。然而,网络数据源的动态性、复杂性及规模性导致传统采集方法面临开发效率低、维护成本高的严峻挑战。通过对目前主流采集框架的分析,得出 Selenium 采集框架相较于其他具有开发效率高、动态支持能力强的特点优势。因此本文创新地提出一种面向网络样本数据采集的轻量化脚本语言设计方法,基于乔姆斯基层次理论构建基于 3 型正则文法的轻量化脚本语法体系,在 Selenium 的基础上进一步优化。为了所提出的脚本语法可被实际使用,本文实现了支持多线程异步执行的分层语法解析器,可将其动态转译为标准化 Selenium 代码。本方案通过抽象原生 Selenium API、绑定动态智能等待等机制,显著降低了开发与维护成本,大大提升了开发人员的效率。同时通过两类 DOM 结构差异场景的采集任务,验证了在卫星轨道参数(TLE)采集任务中,相较于传统 Selenium 方案,本文所设计的脚本语言在代码量方面能够减少 85%以上,页面结构变更后的维护成本降低 70%以上,平均采集延迟的增加可忽略不计。本研究为高动态网络环境下的数据高效捕获提供了轻量化解决方案,简化的脚本语言在未来更有利于大语言模型 LLM 的训练和推理,实现对于样本数据采集任务的自动化生成。
关键词:网络样本数据;轻量化采集脚本;Selenium
中图分类号:TN911.73 **文献标识码**:A **国家标准学科分类代码**:510.5030

Design and implementation of a lightweight script for network sample data collection

Cao Bingyao Zhang Jihao
(School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China)

Abstract: The integrity and freshness of sample data directly determine the generalization ability and prediction accuracy of machine learning model. As the core data source in the open environment, the network can provide wide coverage and high real-time sample support for model training. However, the dynamic, complexity and scale of network data sources cause the traditional acquisition methods to face the severe challenges of low development efficiency and high maintenance cost. Through the analysis of the current mainstream collection framework, it is concluded that selenium collection framework has the advantages of high development efficiency and strong dynamic support ability compared with other collection frameworks. Therefore, this paper innovatively proposes a lightweight script language design method for network sample data collection, and constructs a lightweight script syntax system based on type 3 regular grammar based on Chomsky hierarchy theory, which is further optimized on the basis of selenium. In order to make the script syntax proposed can be used in practice, this paper implements a hierarchical syntax parser that supports multi-threaded asynchronous execution, which can be dynamically translated into standardized selenium code. By abstracting the native selenium API, binding dynamic intelligent waiting and other mechanisms, this scheme significantly reduces the cost of development and maintenance, and greatly improves the efficiency of developers. At the same time, through the acquisition task of two kinds of DOM structure difference scenarios, it is verified that in the satellite orbit parameter (TLE) acquisition task, compared with the traditional selenium scheme, the script language designed in this paper can reduce the amount of code by more than 85%, the maintenance cost after page structure change can be reduced by more than 70%, and the increase of average acquisition delay can be ignored. This research provides a lightweight solution for efficient data acquisition in a highly dynamic network environment. The simplified script language is more conducive to the training and reasoning of large language model LLM in the future, and realizes the automatic generation of sample data acquisition tasks.
Keywords: network sample data; lightweight acquisition script; Selenium

0 引 言

在机器学习、数字孪生技术快速发展的背景下。高质量样本数据成为模型性能提升的关键瓶颈。例如,在自然语言处理领域,预训练语言模型如 BERT、GPT 等的成功离不开大规模的文本数据[1]。在制造行业,通过构建生产

收稿日期:2025-03-10
* 基金项目:国家重点研发计划(2021YFB2900800)、高等学校学科创新引智计划(111 项目)(D20031)资助

线数字孪生模型,能够全面优化智能生产[2]。例如结合传感器采集的实时数据,可以对产品性能进行实时监测和分析[3],及时发现潜在的问题并进行调整,从而降低生产成本、提高产品质量[4]。针对电力系统运维问题,黄文德等[5]结合数字孪生系统,提出构建智能电网运维平台,对电网运维实现了极大的改善。

随着前端技术的不断进步,网页的交互性和动态性得到了极大的提升,传统的数据采集方法在面对网页海量性数据、结构高动态变化的情况下虽能保证一定的效率,但往往会代码无法适配的问题,需要花费大量人力维护或二次开发,从而导致开发效率低、学习成本高、维护困难等问题[6]。因此,亟需一种能够提升开发效率、降低学习成本与维护复杂度的轻量化采集脚本语言。

当前主流的网络数据采集框架根据其原理和应用场景可以分为以下 3 类:

1)静态解析工具:如 Beautiful Soup[7]、Scrapy[8]等通过直接解析 HTML 文档提取数据,其特点是无需加载浏览器,资源占用低,运行速度快,同时大部分提供简洁的 API 接口,但其无法处理 JavaScript 渲染的动态内容。

2)浏览器自动化工具:如 Selenium、Playwright 等,其通过模拟用户操作(如点击、滚动、输入)处理动态渲染页面从而实现数据的完整采集,但普遍存在代码冗余度高、开发维护效率低、学习门槛高等特点[9]。

3)混合方案:混合方案结合静态解析与浏览器自动化,平衡效率与动态内容支持。如 Hybrid Rendering[10]优先使用静态解析,仅在检测到动态内容时调用无头浏览器,降低资源消耗;Scrapy+Playwright[11]框架能够动态加载 Ajax 内容,提升覆盖率。混合方案能够将静态解析与自动化工具的功能互补、灵活优化性能,减少浏览器实例的冗余调用,但其架构复杂,无论学习、开发或是维护成本都十分巨大。

由上述总结,静态采集工具效率最高,但动态支持能力不足,浏览器自动化工具以及混合方案能够很好地解决网页动态渲染问题,但是存在开发维护效率低、学习成本高的劣势。

除了上述通用的开源采集框架以外,一些学者基于已有采集框架进行改良,韩贝等[8]提出基于 Scrapy 的分布式爬虫架构,通过任务队列与管道机制提升采集效率。Meng 等[12]开发了支持关键词筛选与多站点并发的 Python 工具,通过 GUI 界面降低使用门槛,提升复杂网络环境下的抓取效率。杨建等[13]设计的混合方案通过 URL 分类动态切换解析方式,使动态内容捕获率提升至 98.2%。陈泽玉等[14]提出基于节点权重的分层增量比对算法,根据节点更新频率和用户交互热区(如视口内元素),动态分配 Diff 计算资源,实验数据显示首屏渲染速度提升 58%。

通过深入分析并学习相关领域内前人所做的工作和研究后发现,目前主流的采集框架如 Selenium 及一些学者所研究的采集方案已能够有效解决动态页面渲染、提供细粒度交互控制,优化性能等疑难,但普遍存在开发效率低、维护成本高等问题。为应对这些问题,亟需一种创新的轻量化、易维护的采集方案,在保留已有工具强大功能的同时,显著降低开发与维护成本。因此,本文总结一系列采集框架的技术特征,归纳并选择基于 Selenium 框架设计一种轻量化的采集脚本语言并设计实现了一个脚本解析器以提升用户的开发效率同时有效降低学习与维护成本,最后根据设计的脚本解析器进行了两类 DOM 结构差异的采集场景实验,验证了脚本语言在不提升采集延迟的情况下提升开发效率、降低学习维护复杂度的优势,突出了本文工作的必要性和创新性。

1 网络数据采集技术

本节将从元素定位方法、等待机制、页面交互操作和数据采集流程 4 个维度,系统阐述在动态网页数据采集中的核心技术。根据前文的分析可知,在目前主流的数据采集框架中,浏览器自动化工具在处理网页动态化渲染中的展现了强大了应变能力,同时相较于一些混合方案在开发效率上更高,因此本节分析过程参考自动化框架中的 Selenium 采集框架。Selenium 是一个开源的浏览器自动化框架,广泛用于动态网页数据采集和自动化测试。其核心组件 WebDriver 允许开发者通过编程方式控制浏览器行为,如点击、输入、滚动等,支持处理 JavaScript 渲染的页面内容,克服传统静态解析工具的局限性。截至 2024 年,Selenium 兼容 Chrome、Firefox、Edge 等主流浏览器,并提供 Java、Python、C#等多语言 API 接口[15]。

1.1 元素定位方法

动态页面的元素通常通过 JavaScript 异步加载或用户交互后生成,其 DOM 结构较复杂,需根据 DOM 结构灵活选择策略。Selenium 提供多种定位策略,适用于不同场景的动态元素定位。表 1 介绍了常用定位方式:

表 1 Selenium 元素定位方式

Table 1 Selenium element locating methods

定位方式	语法示例(Java)	适用场景
ID 定位	By. id("username")	元素具有唯一静态 ID
ClassName 定位	By. className("submit-btn")	元素类名固定且唯一
XPath 定位	By. xpath("//div[contains(@id, 'content_')]")	复杂层级结构相对路径
CSS 选择器定位	By. cssSelector("div. content > span:first-child")	简单层级结构

1.2　等待机制

由于动态页面元素加载时间会随网络波动等因素影响，通常不固定，因此在进行元素定位时需要通过等待机制确保稳定性，Selenium 提供以下两种等待机制：

1)隐式等待(Implicit Wait)：隐式等待是一种全局等待策略，设置固定的查找元素的超时时间(适用于简单场景)，以下隐式等待代码以及后文部分代码统一使用扩展的巴科斯范式(EBNF)或伪代码介绍：

＜implicit_wait＞　::= "implicitwait" ＜timeout＞

＜timeout＞　::= ＜integer＞ "seconds"

上述代码定义了一个 implicitwait 对象并设置了固定的 10 s 等待时间，无论元素是否加载完毕，都会等待该元素 10 s。

2)显示等待(Explicit Wait)：针对动态元素的特定条件等待：

＜explicit_wait＞　::= "explicitwait" ＜condition＞ ＜timeout＞

＜condition＞　::= "elementtobeclickable" | "elementtobevisible"

上述代码表示可以选择等待元素可点击与等待元素可见两种形式的等待逻辑，并根据需要设置最长等待时间。

1.3　页面交互操作

动态页面采用动态 JavaScript 渲染，常需要通过模拟用户与页面的交互操作触发数据的加载从而完成指定采集页面的渲染，因此页面交互操作是动态数据采集必不可少的环节。表 2 介绍了 Selenium 页面交互操作的常用方式：

表 2　Selenium 常用页面交互操作

Table 2　Common page interaction operations in Selenium

命令名称	语法(EBNF)	说明
点击元素	clickbyxpath ＜xpath＞	通过 XPath 定位并点击元素
输入文本	sendkeysbyname ＜name＞ ＜text＞	通过元素 name 属性输入文本
打开网页	get ＜url＞	加载指定 URL 的页面
选择下拉菜单	selectbyvalue ＜xpath＞ ＜value＞	通过下拉框 XPath 和选项值选择内容

1.4　数据采集流程

数据采集流程的设计直接决定采集效率、稳定性和可维护性。动态页面的不确定性(如元素加载延迟、结构变化)要求流程具备容错能力，通过智能重试、异常捕获等机制确保任务持续执行。此外，减少冗余操作(如重复加载页面)、合理利用缓存机制(如复用浏览器实例)以降低资源消耗等优化操作也是必不可少的环节。本文设计了一个标准的数据采集流程，如图 1 所示。

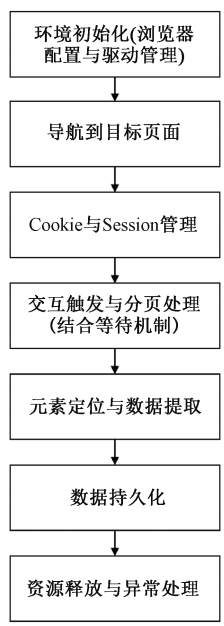

图 1　标准数据采集流程图

Fig. 1　Standard data collection flowchart

2　轻量化脚本语言的设计与实现

2.1　设计理论

在动态网络数据采集场景中，传统 Selenium 开发模式虽然能够模拟用户行为并处理动态页面，但仍面临代码冗余、学习门槛高、维护成本大等问题，限制了其在实际应用中的普及。为解决这些问题，本文依据乔姆斯基层次理论[16]提出一种轻量化脚本语言设计方法，首要的目标是通过自然语言命令降低语言复杂度。

根据乔姆斯基层次理论，形式文法(即定义“语言”的规则集合)G 定义为四元组如式(1)所示。

$$G = (V, T, P, S) \tag{1}$$

式中：V 是有限的非终结符集合，S 为起始符号，可表示为式(2)所示。

$$V = \{A_1, A_2, \cdots, A_m\}, S \in V \tag{2}$$

式中：T 是有限的终结符集合，可表示为式(3)所示。

$$T = \{a_1, a_2, \cdots, a_n\}, V \cap T = \varnothing \tag{3}$$

式中：P 是产生式集合，产生式如式(4)所示。

$$\alpha \to \beta, \alpha \in (V \cup T)^+, \beta \in (V \cup T)^* \tag{4}$$

形式文法共分为 4 个层级，从上到下(0 型～3 型)语法规则越来越简单，对应的语言(即符合规则的句子集合)越来越容易处理。

表 3 介绍了 4 个层次的形式文法含义：

表 3 乔姆斯基层次理论文法规则

Table 3 Grammar rules of the Chomsky hierarchy theory

类型	文法名称	规则形式	数学表示
0 型	无限制文法	无特殊限制	$\alpha\rightarrow\beta,\alpha\in(V\cup T)^{+},\beta\in(V\cup T)^{*}$
1 型	上下文相关文法	$\lvert\alpha\rvert\leqslant\lvert\beta\rvert$(除 S→ε且 S 不在右部)	$\alpha A\beta\rightarrow\alpha\gamma\beta,A\in V,\gamma\in(V\cup T)^{+}$
2 型	上下文无关文法	左部为单个非终结符	$A\rightarrow\gamma,A\in V,\gamma\in(V\cup T)^{*}$
3 型	正则文法	右线性:$A\rightarrow aB$ 或 $A\rightarrow a$;左线性:$A\rightarrow Ba$ 或 $A\rightarrow a$	右线性:$A\in V,a\in T,B\in V$;左线性同理

乔姆斯基层次理论中的 4 层形式文法包含可以用图 2 直观表示。

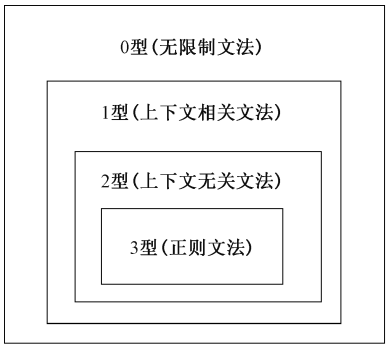

图 2 四层形式文法包含关系

Fig. 2 Inclusion relationship of four-level formal grammars

由图 2 可知 4 个层级之间的包含关系,即 3 型文法是 2 型文法的子集,2 型是 1 型的子集,1 型是 0 型的子集。

此外,根据该理论,2 型上下文无关文法的规则形式为 A→γ,其中 A 是非终结符,γ 是任意符号串(可包含终结符和非终结符),不限制右部结构。其特点是允许递归与嵌套,例如可以定义“句子→主语+谓语+宾语”,而“主语→形容词+名词”,形成层次结构。这种文法复杂度高,需用下推自动机解析,使用 CYK 算法时间分析复杂度为 $O(n^3)$,解析复杂。算法 1 是简化版 CYK 算法的伪代码示例:

根据算法 1,CYK 算法需要填充一个 $n\times n$ 的表格,每个表格元素填充时间为 $O(n)$,所以时间复杂度 $T(n)=O(n^3)$。

例如 Selenium 的显示等待语法是典型的上下文无关文法,存在嵌套结构,等待语句中包含条件,条件中又包含定位策略,需要递归解析,符合 2 型文法的规则,通过此类文法进行开发,开发者需处理多层嵌套,代码冗余度高、学习门槛高。

同时,根据自上而下逐渐简化的规则,针对网络样本数据采集的场景设计一种 3 型正则文法能够有效解决上述困难。3 型正则文法要求 A→aB 或 A→Ba,其中其中 A、B 是非终结符,a 是终结符(如具体的。其特点是简单直观,每个规则最多生成一个终结符,适合描述“线性”结构(如命令→动作+目标+参数),此外其具有高效易解析的

```
算法 1: CYK 解析算法
Input: grammar, inputString
    // 上下文无关文法(CNF 格式),输入字符串
Output: Boolean (是否可被文法生成)
1   n ← length(inputString)
2   Initialize table[n][n] as empty sets   // 创建二维解析表
3   # 填充对角线(终结符处理)
4   for i ← 0 to n-1 do:
5       for each nonTerminal ∈ grammar do:
6           for each production ∈ grammar[nonTerminal] do:
7               if production.length == 1 and production[0] == inputString[i]:
8                   table[i][i].add(nonTerminal)
9   # 处理长度≥2 的子串
10  for length ← 2 to n do:
11      for i ← 0 to n - length do:
12          j ← i + length - 1
13          for k ← i to j-1 do:              // 所有可能分割点
14              for each nonTerminal ∈ grammar do:
15                  for each production ∈ grammar[nonTerminal] do:
16                      if production.length == 2:
17                          left ← production[0]
18                          right ← production[1]
19                          if left ∈ table[i][k] and right ∈ table[k+1][j]:
20                              table[i][j].add(nonTerminal)
21  return startSymbol ∈ table[0][n-1]      // 验证起始符号
```

特征,可被有限状态自动机识别,解析时间复杂度为 $O(n)$(n 是命令长度),即线性时间。同样可以用算法 2 中的伪代码示例证明:

```
算法 2: 3 型正则文法解析算法
Input: inputString, finiteStateMachine
    // 输入字符串,有限状态机(含转移函数)
Output: Boolean (是否被接受)
1   currentState ← finiteStateMachine.startState   // 初始化起始状态
2   for each symbol ∈ inputString do:
3       nextState ← finiteStateMachine.transitionFunction(currentState, symbol)
4       if nextState == undefined then:            // 转移函数未定义
5           return false
6       currentState ← nextState                   // 更新当前状态
7   return currentState ∈ finiteStateMachine.finalStates   // 终态验证
```

算法 2 中,设输入字符串长度为 n,有限状态自动机状态数为 k。每次状态转移时间为 $O(1)$,总共进行 n 次转移,所以时间复杂度 $T(n)=O(n)$。

根据乔姆斯基层次理论，正则文法与上下文无关文法的生成式规则对比，证明前者语法规则数量减少 67%。本文所设计脚本语言的采用 3 型正则文法，每个命令都是“驱动名＋动作＋目标＋参数”的线性组合，避免嵌套或递归，属于原子化命令，简单高效，适合低门槛开发与低成本维护。

2.2　设计目标

除了降低语法规则的复杂程度，轻量化脚本语言的设计目标还包括提升动态页面适配能力、优化并发性能等，表 4 总结了各个设计目标与具体实现方案：

表 4　设计目标与实现方案

Table 4　Design goals and implementation plan

设计目标	技术实现	核心价值
降低复杂度	自然语言命令抽象	降低开发门槛，提升开发效率
动态页面适配	智能等待与参数绑定	提升复杂页面适配性
优化并发性能	多线程优化与资源回收	提升系统效率与稳定性

2.3　脚本语言核心设计

根据 3.1 节所述的乔姆斯基层次理论，轻量化脚本语言的设计应基于 3 型正则文法，采用命令→动作＋目标＋参数的设计方式，避免繁多的代码逻辑嵌套。因此本文所设计的轻量化脚本语言基于 Selenium 操作，将 Selenium 较为复杂的 API 嵌套调用映射为简洁的正则文法与高效的执行机制。本节详细阐述脚本语言的命令结构、常用命令映射逻辑及动态适配策略。

1)命令结构。脚本语言基于 3 型正则文法，同时采用自然语言风格设计，命令结构清晰直观，便于开发者快速上手。基本格式如式(5)所示。

$$[Driver][Action][Target][Params] \tag{5}$$

式中：*Driver*：浏览器实例名称(如 zjhdriver)，用于标识命令的执行主体。*Action*：操作类型(如 sendkeybyname、clickbyxpath)，定义具体的浏览器操作。*Target*：操作目标(如 XPath 表达式、元素名称)，用于定位页面元素。*Params*：附加参数，为可选参数(如输入文本、输出文件名)，支持复杂操作的参数传递，多参数用逗号分隔。

2)常用命令映射逻辑。脚本语言支持多种常用命令，涵盖数据采集、交互操作与流程控制等功能。例如：

(1)数据采集命令 capturelinks：提取页面中符合条件的链接并保存到指定列表。

示例：zjhdriver capturelinks a prefix = https://example.com，out＝links

(2)交互操作命令 clickbyxpath：点击指定 XPath 元素。

示例：zjhdriver clickbyxpath//button[@id＝′submit′]

(3)终止命令 exit：关闭指定浏览器

示例：zjhdriver exit

图 3 展示常用命令与 Selenium 源代码的映射关系：

由图 3 可知，本文设计的轻量化脚本语言将复杂的 API 调用简化为直观指令(如 clickbyxpath、writetofile 等)，极大地减少了代码量，显著降低了开发难度。这种设计使得非专业开发者也能快速上手，提升了开发效率。

3)动态适配策略。现代网页广泛采用 JavaScript 动态渲染技术，页面内容在加载后仍可能发生变化(如 Ajax 请求、延迟加载)。传统静态解析工具无法处理此类动态内容，而 Selenium 虽支持动态交互，但其显式等待机制需手动编写，增加了开发复杂度。本文脚本语言内置智能动态等待机制，通过三层策略实现与 Selenium 原生等待的深度集成，包括基础等待策略、动态参数调节和异常降级机制，其核心参数定义与实现逻辑见表 5。

动态超时公式如式(6)所示。

$$T'_o = T_o \times k \times (1 + 0.1 \times \text{页面 } DOM \text{ 深度}) \tag{6}$$

其中，页面 DOM 深度通过解析页面 HTML 树的最大层级数获取，用于量化页面复杂度。

智能动态等待机制采用混合等待架构，优先使用显式等待(explicit wait)，失败后自动降级为隐式等待(implicit wait)，并通过动态参数调节优化等待效率，算法伪代码如算法 3 所示。

图 4 展示了混合等待架构算法的实现流程图。

为了避免固定轮询间隔导致的过度请求或等待浪费，采用基于指数平滑法的轮询间隔自适应算法根据实时负载动态调整如式(7)所示。

$$T_p^{(t)} = \alpha \cdot T_p^{(t-1)} + (1-\alpha) \cdot T'_p \tag{7}$$

式中：$T_p^{(t)}$ 表示第 t 次调节后的轮询间隔，$T_p^{(t-1)}$ 表示第 $t-1$ 次调节后的轮询间隔即历史值，T'_p 表示第 t 次等待过程中实际使用的轮询间隔即当前观测值，$\alpha \in [0,1]$，代表平滑系数。

此外，通过实时网络质量动态调整基础超时阈值，避免网络延迟导致的等待不足。使用周期性 ping 检测每 30 s 向目标网站发送 HTTP HEAD 请求，记录往返时间。保存最近 10 次 RTT 数据，计算平均值 μ_{RTT}，进而更新波动系数 k，如式(8)所示。

$$k = 1 + \frac{\text{当前 RTT} - \mu_{RTT}}{\mu_{RTT}} \tag{8}$$

式中：若当前 RTT＝历史平均，则 $k=1$ 无波动，若当前 RTT 比平均慢 50%，$k=1.5$(超时阈值同步增加 50%)。经 Box 团队[17]论证，指数平滑法的均方误差优化原理，比简单移动平均法降低 40%以上。

综上，本文所设计的脚本语言内置的智能动态等待机制并采用基于指数平滑法的轮询间隔自适应算法使得动态适配策略能够更有效地贴合实际采集场景，提升了脚本

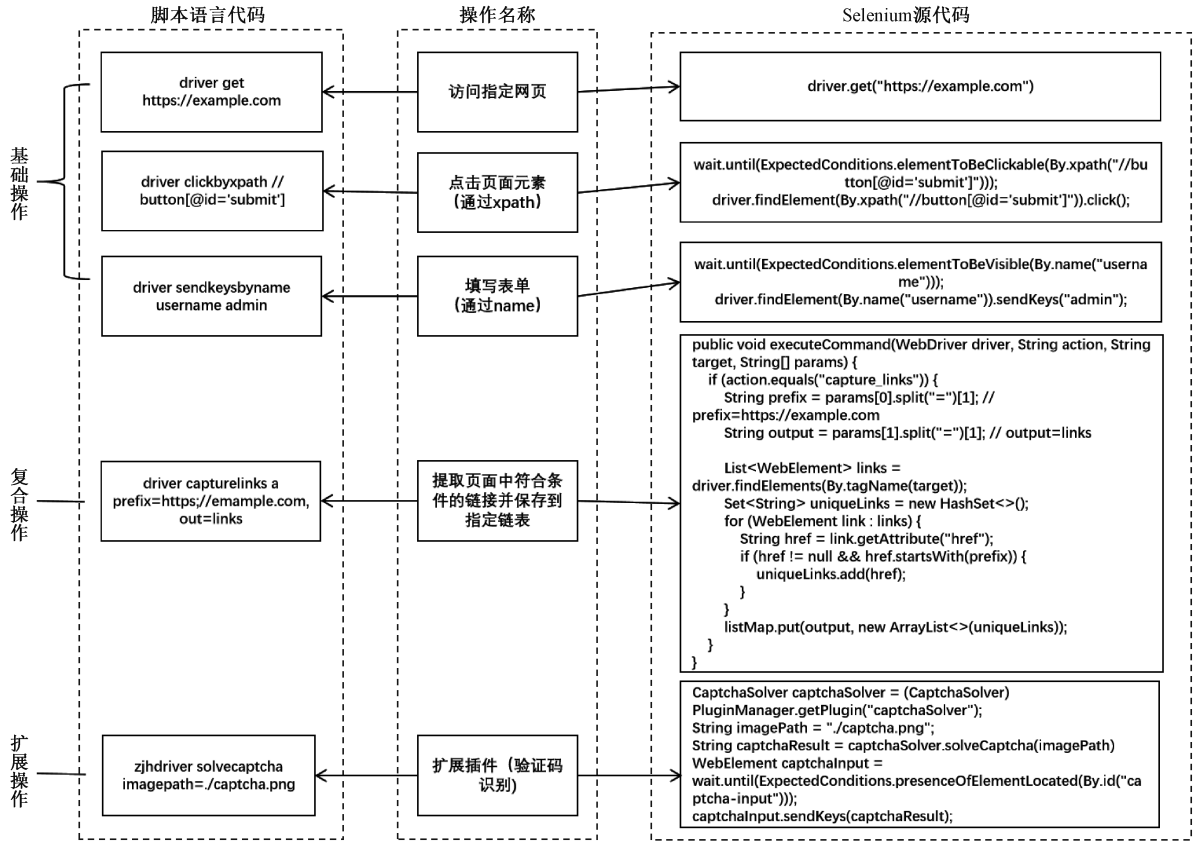

图 3 常用命令映射逻辑表

Fig. 3 Common command mapping logic table

表 5 基础等待参数定义

Table 5 Definition of basic waiting parameters

参数名称	符号	定义	默认值	调节范围
初始轮询间隔	T_p	每次检查元素状态的时间间隔(避免服务器过载)	200 ms	100～500 ms
基础超时阈值	T_o	单个元素操作的最大等待时间(根据页面复杂度动态调整)	10 s	5～30 s
重试次数	R	元素定位失败后的自动重试次数	3 次	1～5 次
网络波动系数	k	基于实时网络质量的动态调节因子(通过 ping 响应时间计算)	1.2	1.0～1.5

语言的可靠性。

2.4 解析器实现

解析器是脚本语言的核心组件,负责将用户输入的脚本命令转换为 Selenium 操作。其设计基于分层解析机制与多线程任务调度框架,确保高效、灵活地处理各种命令,同时通过错误处理与多线程管理提升系统的可靠性与并发性能。解析器按照模块划分为解析模块、资源调度与多线程处理模块、执行模块三部分。整体工作流程如图 5 所示。

1)解析模块。解析模块负责对用户输入的命令进行语义解析,将用户输入的命令按字符串分割为浏览器名称(webDriverName)、操作符(Action)、目标(Target)、参数(Params)。

示例:zjhdriver capturelinks a https://example.com, links 解析为:

(1)*Driver*:zjhdriver

(2)*Action*:capturelinks

(3)*Target*:a

(4)*Params*:prefix=https://example.com, out=links

解析模块包含错误处理,解析同时检查命令格式是否符合规范,若不符合则提示错误信息。

示例:zjhdriver capturelinks

错误提示:Error: Missing target or parameters.

2)资源调度与多线程处理模块。资源调度模块通过全局 Map 存储表管理浏览器实例与线程的绑定关系,并将解析结果传递给执行模块(executeCommand),确保多任务并发执行。为了提高资源的利用率和脚本执行的效率,解析器使用线程池来管理线程。线程池可以预先创建一定

算法 3：动态混合等待策略

```
Input: driver, locator, condition, retry=R   // Web 驱动，定位器，等待条件，最大重试次数
Output: Element object or TimeoutException   //元素对象或异常
for i = 1 to retry do:                       // 重试控制循环
    # Phase 1: Explicit Waiting (显示等待，智能条件监控)
    start_time ← current_timestamp()
    while current_timestamp() - start_time < T_o do:
        elements ← driver.find_elements(locator)
        if condition(elements) then:         // 条件检测（如可点击性）
            return elements[0]               // 成功返回目标元素
        sleep(T_p)                           // 基础轮询间隔
    # Phase 2: Implicit Waiting (全局动态适配)
    driver.implicitly_wait(T_o / 2)          // 设置半超时隐式等待
    try:
        element ← driver.find_element(locator)
        return element                       // 成功捕获元素
    except NoSuchElementException:
        if i < retry then:
            adjust_parameters()              // 动态调节 T_o/T_p 参数
        else:
            raise TimeoutException("元素未找到")
```

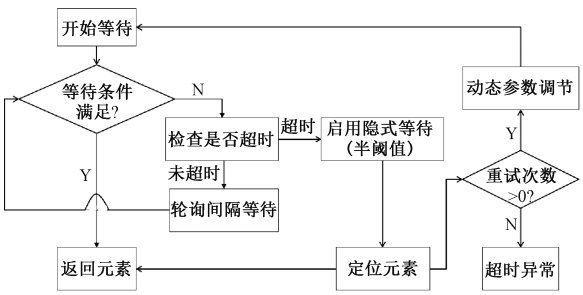

图 4　混合等待架构算法流程图

Fig. 4　Flowchart of hybrid waiting architecture algorithm

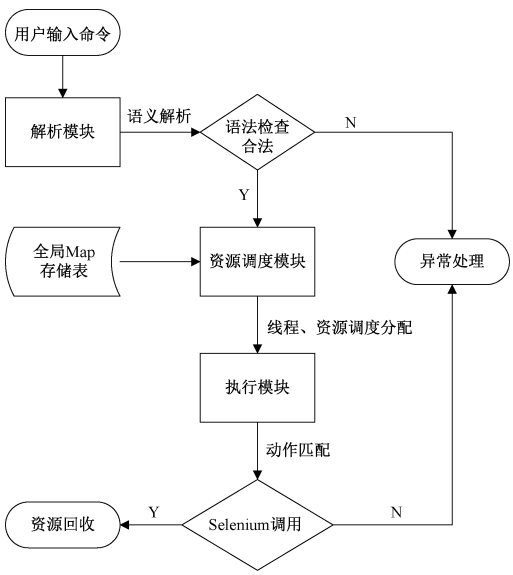

图 5　解析器工作流程图

Fig. 5　Parser workflow diagram

数量的线程，当有新的任务到来时，从线程池中获取空闲的线程来执行任务，避免了频繁创建和销毁线程带来的开销。

(1)全局资源表：使用 HashMap 存储浏览器实例(WebDriver)与线程(Thread)的映射关系，确保多实例独立运行。

(2)线程与资源调度：资源调度的核心任务是将解析结果与全局资源绑定后封装为任务，传递给执行模块，并确保任务在独立线程中执行。在执行新命令前，检查当前线程是否已完成上一任务，避免资源竞争。若线程未完成上一任务，等待其执行完毕并创建新线程执行当前命令，同时更新资源表。

图 6 为资源调度流程图：

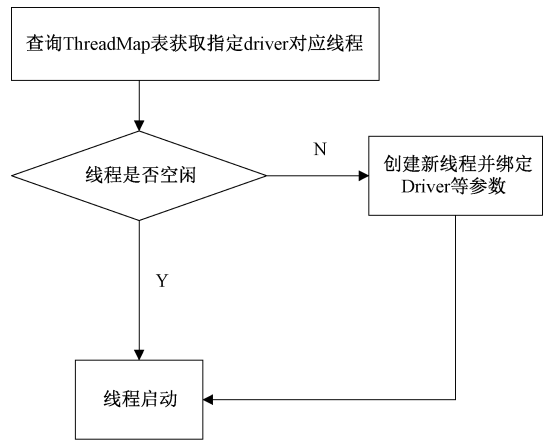

图 6　资源调度流程图

Fig. 6　Resource scheduling flowchart

3)执行模块。执行模块根据传递的参数调用对应的 Selenium API，并处理资源回收与异常。通过 switch-case 结构将 *Action* 映射为具体的 Selenium 操作。

(1)动作匹配：根据动作类型(Action)执行对应的 Selenium API 调用。

示例：

```
case "clickbyxpath":
WebElement clickElement1＝driver. findElement(By. xpath(targetOrValue));
clickElement1. click();
System. out. println("点击元素:"＋targetOrValue);
break;
```

(2)资源回收：当浏览器实例不再使用时，自动释放资源并移除绑定关系。

示例：

```
case "exit":
driver. quit();
  webDriverMap. remove(driverName);
  threadMap. remove(driverName);
```

```
break;
```

(3)错误处理:在执行过程中捕获异常(如元素定位失败),并自动重试或提示错误信息。

示例:

```
try {
    WebElement element = driver.findElement(By.xpath(target));
    element.click();
} catch (NoSuchElementException e) {
    System.out.println("Error: Element not found.");
}
```

3 实 验

3.1 实验目的

本实验针对卫星数字孪生数据采集的多源需求,选取 Celestrak 与 SpaceTrack 作为两类 DOM 结构差异化的典型网站,验证脚本语言在异构页面结构(HTML 表格 vs. JSON 数据)中的开发效率、维护成本与执行性能。

TLE 数据是卫星轨道预测的关键输入,通常发布于专业网站,并通过动态表格与分页加载展示。如图 7 所示:

图 7 CelesTrak 网页卫星数据

Fig. 7 CelesTrak web satellite data

3.2 实验场景设计

针对两个不同场景的两个网页,分别使用本文所设计的轻量化脚本语言与原生 Selenium 代码进行 TLE 数据采集任务,场景涵盖了动态页面适配、交互操作模拟、数据提取与存储等核心功能,验证脚本语言在动态表格与分页加载页面中的实际采集效果,全面评估脚本语言的性能与实用性。实验环境硬件配置为 64 bit Windows 10,Intel Core i7-8750H, 16 GB RAM;软件环境为 JDK11, Selenium4.1.2,Google Chrome 114,ChromeDriver 114。

任务流程:首先导航至目标网页,接着进行一系列页面交互直至到达数据页,最后解析文本内容,提取 TLE 数据并进行结构化存储。通过直接的代码行数比较验证脚本语言的开发成本,通过采集时间比较验证脚本语言的采集延迟。为了减小实验误差,本次实验在相邻时段进行 100 次数据采集任务。

同时,通过模拟变更表格的 DOM 节点、JSON 字段路径变更以及 Xpath 路径增加,验证当页面发生变化时,脚本语言相较于传统 Selenium 代码在维护成本上的优势。

表 6 展示了两个网页的特征对比。

表 6 网页特征对比

Table 6 Comparison of web page features

网页名称	Celetrak	Spacetrack
页面技术	动态表格	静态数据接口
数据组织形式	HTML 表格	JSON 对象数组
DOM 结构特征	Xpath 平均路径深度 12 层	JSON 字段平均嵌套深度 5 层
典型操作	分页按钮点击、表格行数据提取	API 参数拼接、JSON 字段解析
数据解析逻辑	基于 class 属性的奇偶行识别	基于路径表达式的字段提取

3.3 实验数据

本实验各项指标实验数据如表 7 所示。

表 7 实验数据

Table 7 Experimental data

指标	场景	脚本语言	Selenium 原生	t 值	p 值	差异幅度
代码行数(行)	Celestrak	9	59	$t=-32.7$	<0.001	减少 84.7%
	Spacetrack	7	62	$t=-28.7$	<0.001	减少 85.5%
平均维护时间(分钟)	表格变更	1.52±0.18	4.87±0.45	$t=-21.3$	<0.001	减少 68.8%
	JSON 变更	1.83±0.21	6.25±0.58	$t=-19.7$	<0.001	减少 70.7%
平均采集延迟(ms)	Celestrak	390 770±8 215	389 717±7 489	$t=1.32$	0.19	无显著差异
	Spacetrack	350 212±6 987	348 895±6 792	$t=1.05$	0.30	无显著差异

其中 t 检验[18]是为了表示两组数据均值差异的标准化值，绝对值越大，差异越显著。公式如式(9)、(10)所示。

$$t = \frac{\overline{X_1} - \overline{X_2}}{S_P \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{9}$$

$$S_p = \sqrt{\frac{(n_1 - 1)S_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}} \tag{10}$$

式中：$\overline{X}$ 为均值，S_P 为合并标准差，n 为样本量，在代码行数对比中，两组 t 值分别为 -32.7 以及 -28.7，绝对值极大，说明脚本语言与 Selenium 原生代码的代码行数差异极显著。在平均维护时间对比中，两组 t 值分别为 -21.3 以及 -19.7，同样说明差异显著，而平均采集延迟的 t 值绝对值小，证明差异小。

p 值表示假设两组无差异(零假设)时，观察到当前差异的概率。其可用卡方检验来计算，如式(11)所示。

$$X^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \tag{11}$$

式中：O_i 为实际频数，E_i 为理论频数，k 为分类的类别数，在统计学中当 $p<0.05$(此时卡方>3.841)时表示差异由随机误差引起的概率$<5\%$，拒绝零假设，差异显著，通过查询卡方分布表可知 p 值检验同样验证了脚本语言在代码行数及平均维护时间上相较于原生 Selenium 代码差异显著，而在平均采集延迟角度差异不大。

根据上表数据，脚本语言在两种场景下的代码行数平均减少了 85.1%，这是由于脚本语言通过领域特定命令将复杂解析逻辑封装为高层操作，使开发者无需关注底层实现，如原生 Selenium 需编写 59 行代码实现“分页按钮点击+表格行提取”，涉及显式等待(WebDriverWait)、元素定位(find_element)、循环遍历行元素并筛选奇偶行(class="odd"/"even")，而脚本语言通过 capturetable 命令直接指定行 class 属性(tr class='odd,even')，自动处理分页逻辑(loop 5 times: clickbyid PageNext)，代码量减少 84.7%。实现“一行代码对应一个复杂操作”，从根本上减少重复性编码。

当网页 DOM 结构变更时，脚本语言将维护操作从“代码级修改”转化为“参数级调整”，显著降低修改成本，当表格行 class 更名时，原生方案需修改所有行元素的定位表达式(如 tr[class="odd"]→tr[class="rowOdd"])，涉及多处代码改动，平均耗时 4.87 分钟。而脚本语言仅需修改 1 处参数：在 capturetable 命令中更新 class 参数(class='rowOdd,even')，平均耗时 1.52 分钟，效率提升 68.8%。

尽管脚本语言在代码层面大幅简化，但底层仍基于 Selenium 驱动浏览器，因此根据实验结果，采集延迟与原生方案无显著差异。在平均采集延迟的测定中，每个场景独立采集 100 次，计算均值±标准差，降低随机误差影响，同时根据 3σ 原则，在正态分布中，数据几乎全部(约 99.73%)落在均值(μ)±3 倍标准差(σ)的范围内，超出该范围的数据可视为“小概率事件”，通常被判定为异常值。剔除采集延迟的异常值(如网络中断导致的超时数据)后，确保统计结果稳健。此外，网络环境的不确定性是实验数据波动的重要因素，本实验测量的采集延迟数据通过针对 RTT 波动对采集延迟的影响，采用线性修正公式降低误差，如式(12)所示。

$$\text{修正延迟} = \text{观测延迟} \times \left(1 - 0.8 \times \left| \frac{\text{当前 } RTT}{\mu_{RTT}} \right| - 1\right) \tag{12}$$

式中：当 RTT 超过历史均值 μ_{RTT} 时，按比例修正观测延迟(系数 0.8 为经验值，通过最小二乘法拟合得到)，使延迟数据更接近网络稳定时的真实耗时。

3.4　实验结果与分析

本实验通过两类 DOM 结构差异化采集场景验证，表明脚本语言在卫星数字孪生数据采集中显著优于原生 Selenium 方案，其中：

1)代码行数减少 85%以上，开发效率突破提升，核心得益于解析逻辑的高层封装；

2)维护时间缩短 70%以上，通过参数化设计将 DOM 变更影响控制在最小范围；

3)采集延迟无显著差异，实现“效率优先、性能等效”的设计目标。

这些结果证明，脚本语言通过语法抽象与容错机制，有效解决了网络样本数据采集中的开发维护复杂度问题，为数字孪生系统的数据源扩展提供了高效解决方案。未来可进一步扩展更多的功能指令以及队其他数据格式的支持，提升工业级场景的适用性。

4　结　　论

本文提出了一种轻量化的开放脚本数据采集技术，通过设计脚本解析语言，将用户的简单命令转换为 Selenium 的自动化操作指令，显著降低了开发者在网络数据采集中对 Selenium API 和复杂脚本编写的依赖。同时基于该脚本语言设计了脚本解析器，内置多浏览器并行操作、动态智能等待等机制，能够高效处理动态内容加载、复杂交互和大规模数据采集任务。实验结果表明，在不同网页结构的采集场景中，本文设计的轻量化脚本语言相较于原生 Selenium 采集方案减少了 85%以上的代码量以及 70%以上的维护时间，体现了在开发效率和可维护性方面的优势。此外在运行性能上几乎保持与原生方案相同。

随着大语言模型(large language models，LLM)技术的快速发展，将脚本解析方法与 LLM(如 DeepSeek、GPT-4)相结合，成为进一步提升网络数据采集效率与智能化水平的重要方向。未来的研究可以聚焦于自然语言到脚本命令的自动生成，用户只需提供自然语言描述(如“采集某网站 Starlink 卫星的 TLE 数据”)，LLM 即可自动生成对应的脚本命令。这种模式显著降低了开发门槛，使非技术

用户也能高效完成数据采集任务,也使得未来的网络数据采集工具能够更加智能化与用户友好。

参考文献

[1] DEVLIN J, CHANG M W, LEE K, et al. BERT: Pre-training of deep bidirectional transformers for language understanding[C]. 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies(NAACL-HLT), 2019: 4171-4186.

[2] 程昊,王红军,杨俊峰,等.面向生产线全生命周期的数字孪生模型应用[J].电子测量与仪器学报,2023,37(4):115-121.
CHENG H, WANG H J, YANG J F, et al. Application of digital twin model for full lifecycle of production line[J]. Journal of Electronic Measurement and Instrumentation, 2023, 37(4): 115-121.

[3] 冯昊天,王红军,常城,等.基于数字孪生的柔性生产线状态感知[J].电子测量与仪器学报,2021,35(2):17-24.
FENG H T, WANG H J, CHANG CH, et al. State perception of flexible production line based on digital twin[J]. Journal of Electronic Measurement and Instrumentation, 2021, 35(2): 17-24.

[4] TAO F, ZHANG M. Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing[J]. IEEE Access, 2017, 5: 20418-20426.

[5] 黄文德,张晓飞,庞湘萍,等.基于北斗与数字孪生技术的智能电网运维平台研究[J].电子测量技术,2021,44(21):31-35.
HUANG W D, ZHANG X F, PANG X P, et al. Research on smart grid operation and maintenance platform based on beidou and digital twin technologies[J]. Electronic Measurement Technology, 2021, 44(21): 31-35.

[6] ZHANG X, WANG Y, LI Z. Research on web data crawling technology based on dynamic web pages[J]. Journal of Internet Technology, 2020, 21(4): 877-886.

[7] LIU J, LIN L, CAI Z, et al. Deep web data extraction based on visual information processing[J]. Journal of Ambient Intelligence and Humanized Computing, 2024,15: 1481-1491.

[8] 韩贝,马明栋,王得玉. 基于 Scrapy 框架的爬虫和反爬虫研究[J]. 计算机应用研究,2021, 38(5): 1-14.
HAN B, MA M D, WANG D Y. Research on crawler and anti-crawler based on Scrapy framework[J]. Journal of Computer Applications Research, 2021, 38(5): 1-14.

[9] GARCÍAB, del ALAMO J M, LEOTTA M, et al. Exploring browser automation: A comparative study of selenium, cypress, puppeteer, and playwright[C]. International Conference on the Quality of Information and Communications Technology, 2024: 142-149.

[10] LIU Q, YAHYAPOOR R, LIU H, et al. A novel combining method of dynamic and static web crawler with parallel computing[J]. Multimedia Tools and Applications, 2024, 83(21): 60343-60364.

[11] KHAN A. Design and implementation of an automated web scraping system: enhancing accuracy and efficiency for nettileasing finland Oy[D]. Lappeenranta, Lahti: Lappeenranta-Lahti University of Technology(LUT University), 2023.

[12] MENG X, ZHANG Y. Development and evaluation of web crawler data collection tool based on Python technology[C]. 2024 International Conference on Data Science and Web Engineering, 2024: 215-230.

[13] 杨健,陈伟. 基于 Python 的三种网络爬虫技术研究[J]. 计算机应用研究, 2023, 40(5): 88-95.
YANG J, CHEN W. Study on three network crawler technologies based on Python[J]. Journal of Computer Applications Research, 2023, 40(5): 88-95.

[14] 陈泽玉.大规模虚拟 DOM 渲染优化技术研究与应用[D]. 绵阳:西南科技大学,2024.
CHEN Z Y. Research and application of large-scale virtual DOM rendering optimization technology[D]. Mianyang: Southwest University of Science and Technology, 2024.

[15] NAING I, FUNABIKI N, WAI K H, et al. A design of automatic reference paper collection system using selenium and bert model[C]. 2023 IEEE 12th Global Conference on Consumer Electronics(GCCE). Nara, Japan: IEEE, 2023: 267-268.

[16] HUNTER T. The Chomsky hierarchy[C]. A Companion to Chomsky, 2021: 74-95.

[17] BOX G E P, JENKINS G M, REINSEL G C, et al. Time series analysis: forecasting and control[M]. New York: John Wiley & Sons, 2015.

[18] 夏光源.关于假设检验的误解:客观概率与统计检验力[J]. 心理学进展, 2024, 14: 303.
XIA G Y. Misunderstandings about hypothesis testing: Objective probability and statistical power[J]. Advances in Psychology, 2024, 14: 303.

作者简介

曹炳尧,高级实验师,博士,主要研究方向为网络数据处理和模拟技术。

E-mail:caobingyao@shu.edu.cn

张嵇豪(通信作者),硕士研究生,主要研究方向为 Web 数据捕获及脚本语言设计。

E-mail:305955468@qq.com