密级：_____

浙江大学

硕 士 学 位 论 文

论文题目 面向 Web 应用的智能化服务封装系统设计与实现

作者姓名 王乃博

指导教师 尹建伟教授

学科(专业) 计算机科学与技术

所在学院 计算机科学与技术学院

提交日期 2020 年 3 月 30 日

A Dissertation Submitted to Zhejiang University for the Degree of Master of Engineering

TITLE: Design and Implementation of Intelligent Service Wrapper System for Web Application

Author: Naibo Wang

Supervisor: Prof. Jianwei Yin

Subject: Computer Science and Technology

College: Computer Science and Technology

Submitted Date: 30 March, 2020

摘要

互联网与物联网技术的发展改变了链接方式，服务化成为各行业的发展趋势，也成为企业业务开放的重要模式，IBM 更是提出了 API 经济的战略，这一战略模式要求把硬件、软件、数据、人力等资源封装成 Web 服务，以 API 的方式供第三方使用，从而支撑生态体系的孵化。Web 应用是一种重要的软件与数据资源，随着互联网的快速发展和普遍应用，Web 应用数据和资源的数量出现了爆发式的增长。面向 Web 应用资源，研究智能化服务封装方法，实现以服务请求的方式执行 Web 数据的采集和处理任务，将会简化用户的工作量、促进产业生态的孵化。

针对 Web 应用封装 Web 服务时生成效率低、技术要求高的挑战，本文设计了一个面向 Web 应用的智能化服务封装系统（OKAPI），从而实现可视化、去编程化的 Web 应用数据采集，有效的服务于数据分析、金融财政、IT 等行业用户。

本文结合 Web 数据提取、Web 页面分块等相关研究工作，基于面向服务的架构和 Chrome 扩展开发技术，对系统中服务提供者、服务请求者和服务注册中心三类用户的 Web 应用数据采集需求进行了分析，并提出了系统的总体设计架构。

针对 Web 页面结构清晰，用户提交表单后可得到 Web 数据的简单数据采集场景，本文研究了 Web 页面表单检测、Web 页面分块和块排序等算法，以实现对 Web 应用数据采集服务的生成和调用。同时，针对用户自行配置操作规则以进行多样化 Web 数据采集的复杂数据采集场景，本文研究了同类型元素匹配、服务流程定义和执行等算法，从而满足对 Web 页面进行自定义的数据采集的需求。

对简单数据采集和复杂数据采集分系统中生成的定制化的 Web 应用数据采集服务的测试结果表明，由智能化服务封装系统生成的 Web 应用数据采集服务的生成成功率和可达性均在 99%以上，服务响应时间均低于 500ms，相比于 XWRAP 等 Web 数据提取系统，该系统生成 Web 数据采集任务的效率提升了 10 倍以上；同时，Web 页面分块算法的执行准确率在 96%以上，数据采集速率超过 600 条/分钟，系统执行大规模数据采集任务的稳定性超过 95%，验证了系统中各服务运行的鲁棒性及服务执行的准确性、高效性。

最后，系统在高分服务网格平台国家重大工程上封装的 128 个高分相关服务的总调用次数已超过 1 万次，总检索次数超过 1000 次，体现了系统对国家级重大专项的支撑作用，验证了系统的实用性。

关键词：WEB 应用，服务封装，Web 数据提取，Web 页面分块，流程管理

Abstract

The development of the Internet and the Internet of Things technology has changed the way of connection, service-oriented business has become the development trend of various industries, and it has also become an important model for enterprise business opening. IBM has also proposed a strategy for the API economy. Data, human resources and other resources are wrapped into web services for use by third parties with APIs to support the incubation of the ecosystem. Web applications are important software and data resources. With the rapid development and widespread application of the Internet, the amount of data and resources of web application has exploded. Doing research for web application resources on intelligent service wrapper methods and implementation of Web data collection and processing tasks in the form of service requests will simplify the workload of users and promote the incubation of industrial ecology.

In response to the challenges of low generation efficiency and high technical requirements of web services wrapping process from web applications, this thesis designs an intelligent service wrapper system (OKAPI) for web applications to achieve visual and de-programmed web application data collection, which can serve various industries users such as data analysis, finance, and IT.

This thesis combines the related research work of web data extraction, web page segmentation to analyze the requirements of service provider, service requester, and service registry as well as the functions implemented by the system, which are based on the web service architecture and Chrome extension development technology. Then, this thesis proposes the overall design architecture of the system.

In view of the simple data collection scenario where users can obtain Web data after submitting the form from web pages whose structures are clear, this thesis studies the web page form detection, web page segmentation and block sorting algorithms to achieve the generation and call of the web application data collection service. Meanwhile, for the complex data collection scenarios where users configure their own operation rules for diversified web data collection tasks, this thesis studies the same type element matching, service process definition and execution algorithms to meet the needs of custom data collection for web pages.

The test results of the customized web application data collection service generated in the simple data collection and complex data collection subsystems show that the generation success rate and reachability of the web application data collection service

generated by the intelligent service wrapper system are both more than 99%, the service response time is less than 500ms, compared with XWRAP and other Web data extraction systems, the efficiency of the system to generate web data collection tasks has increased by more than 10 times; at the same time, the execution accuracy of the web page segmentation algorithm is 96% as well as the data collection rate exceeds 600 per minute. Also, the stability of the system in performing large-scale data collection tasks exceeds 95%, which verifies the robustness of each service operation in the system and the accuracy and efficiency of service execution.

Finally, the total invocation number of 128 high-resolution related services wrapped by the system on the national major project of the high- resolution service grid platform has exceeded 10,000 times, and the total retrieval number has exceeded 1,000, which reflects the support role of the system for major national projects, and verifies the practicability of the system.

Keywords： web application, service wrapper, web data extraction, web page segmentation, flow management

目录

图目录

表目录

第1章 绪论

1.1 课题背景与意义

云时代的到来使得万物皆服务（XaaS）的概念深入到各行各业。很多企业（如阿里巴巴）将自己的业务封装成服务对外开放，各大云平台和大型互联网公司都在以 Web 服务的方式开展业务。IBM 提出了 API 经济的战略，从而将硬件、软件、数据和人力等资源统一的封装成 Web 服务，并以 API 的方式提供给第三方使用。如图 1.1 所示，美国 IT 企业的 API 管理费用在逐年增长，说明 API 业务模式正被各行各业广泛使用。服务时代的到来大大提高了社会的运作效率，为多个组织之间业务流程的集成提供了一个通用的机制。在 IT 行业，使用 Web 服务提供的应用程序接口（API）来开发自己的应用程序（APP）已经成为一种通用的做法。如许多医院的化验单识别应用都使用了科大讯飞公司提供的图像转文字（OCR）服务来将化验单上的打印和手写字体转换为可编辑字符；研究自然语言处理（NLP）的研究人员也可以通过购买各服务集市中的新浪微博评论提取服务来获得最新的微博评论数据，从而进行下一步的操作，如模型预测等。面向服务的架构（SOA）大大的提高了程序开发的效率，而通过 API 来进行服务访问的方式也大大的降低了程序开发的门槛。因此，现在的业务提供方和使用者都在以 Web 服务的方式进行交互。

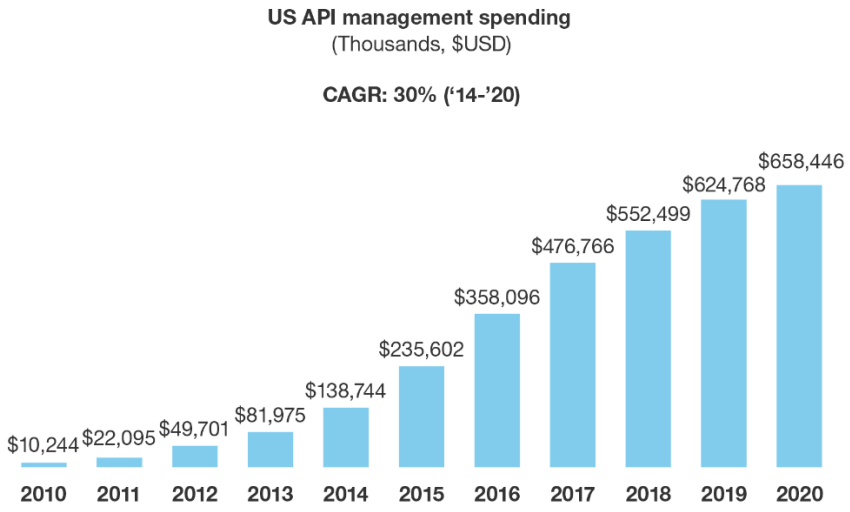


图 1.1 美国 IT 企业 API 管理费用图

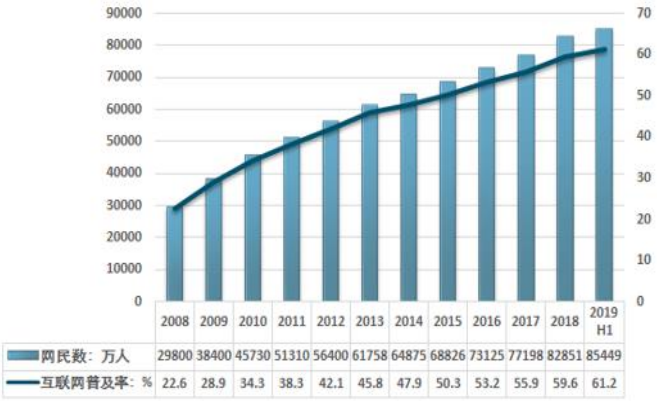


图 1.2 2008-2019 年 6 月我国网民规模及互联网普及率走势

如图 1.2 所示，随着互联网的普及和大数据时代的到来，越来越多的数据和信息在互联网上不断的堆积，Web 应用包含的数据量发生了爆炸式的增长。2019 年 7 月 11 日中国互联网协会发布的《中国互联网发展报告（2019）》[1] 显示，我国的网民总数已达到了 8.29 亿，网页数量更是达到了 2816 亿之多，4G，5G 以及移动互联网技术的迅速普及更使得互联网的规模迅速扩大，使用互联网获取信息几乎成了每个人每天必须要做的事情。同时，获得 Web 应用中的数据并对其进行处理和分析已经成为各行各业不可或缺的需求，如企业需要数据来分析竞争对手的信息，以及分析自己用户的行为以提升自己的产品；金融公司可以通过股票数据来进行股票行情预测；许多人工智能和机器学习应用需要构建大规模的数据集来进行模型预测等。对 Web 应用中的数据进行采集通常是很多工作的首要任务。

为了快速简单的在互联网上找到自己想要的信息，人们发明了搜索引擎技术，这大大提高了人们查找想要信息的效率，降低了人们获取信息的门槛。然而，搜索引擎设计的初衷并不是为了进行大规模的数据采集，而是为了帮助用户对互联网上的信息进行快速的检索。用户使用搜索引擎得到的是网页的原始页面，而没有对页面结构进行任何格式化的处理，不适宜直接对页面中的数据进行分析和处理。因此，为了定向抓取相关的 Web 数据，聚焦爬虫技术应运而生。

聚焦爬虫技术既可以有选择的访问互联网上的页面和相关的图片、链接等资源，又可以定向的抓取只和自己需求有关的信息。它为面向主题的用户提供数据资源，自动化的完成 Web 数据采集的任务。然而，聚焦爬虫通常需要专业的技术人员来为客户编写，因此提高了人们爬取 Web 数据的门槛。同时，当网页结构发生变化导致无法采集数据时，客户只能求助于技术人员来对程序进行修改和维护，无法自行调整；而 Web 应用数据采集任务通常只是许多业务的第一步，对任何业务都需要编写爬虫代码来采集特定 Web 网站中的数据显然增加了不必要的工作

量。如何既简单又定制化的爬取 Web 应用中的数据逐渐成为了各行业的一个重要需求和挑战。

因此，如果可以研究面向 Web 应用资源的智能化服务的封装方法，实现以服务请求的方式执行 Web 数据的采集和处理任务，将会简化用户的工作量、促进产业模式的升级，给使用者带来极大的便利。

1.2 本文研究内容

高分辨率对地观测系统是我国正在建设的基于卫星、平流层飞艇和飞机的对地观测系统，该系统与其它观测手段结合，将形成全天候、全天时、全球覆盖的对地观测能力，从而建成先进的陆地、大气、海洋对地观测系统，为现代农业、减灾、资源环境、公共安全等重大领域提供服务和决策支撑。高分网格是基于网格、集群计算、大数据等互联网新技术构建的数据、服务、管理网格系统，从而实现高分专项卫星等海量数据资源、应用成果的有效集成与共享，建设成高分数据、应用和服务类产品的开放平台、管理平台和高分+众创平台。《高分辨率对地观测系统重大专项（民用部分）高分网格平台（二期）》（简称：高分二期）项目是面向高分重大专项的需求，围绕高分专项需求电子化、统一编目的流程化、共享资源及统一监管自动化、服务平台开放化的目标，为实现高分卫星海量数据资源、应用成果的有效集成与共享而研制和部署的高分网格平台系统。

由于高分 Web 应用相关信息通常发布在各部门（如国家林业局）的 Web 网站上，为了将这些分散的各高分应用信息整合在高分服务网格平台，并以 Web 服务的方式提供给各行业用户，从而方便用户对高分应用数据进行检索，本文开发了一个面向 Web 应用的智能化服务封装系统（OKAPI），旨在让使用者可以在脱离代码编写和程序开发的场景下，以 Web 服务的方式对 Web 应用中的数据进行采集。同时，系统还可以对互联网上 Web 应用中任意类型的信息，如火车时刻列表信息进行服务整合，从而满足各类不同用户的 Web 数据采集需求。用户可以使用本系统提供的图形化用户界面（GUI）来定义自己想要采集的流程，通用化的对 Web 页面数据进行采集。系统根据用户使用场景的不同，提出了针对简单数据采集场景下基于 Web 页面分块算法的数据采集方案，和针对复杂数据采集场景下用户自定义操作流程的数据采集方案，从而满足 90%以上的 Web 应用的数据采集需求。系统将生成的 Web 数据采集任务封装成 Web 服务，从而让使用者可以通过调用服务 API 的方式来执行数据采集任务，从而得到格式化处理后的 Web 应用中的有效数据。

本文的主要工作如下：

1、针对结构清晰简单的 Web 页面，提出了简单数据采集场景下的数据采集方案，详细阐述了系统中各个模块的作用以及相关算法的实现细节。

2、针对更加通用化的复杂 Web 数据采集场景，详细介绍了系统从服务生成到服务调用阶段的各个功能点，并对系统中各个模块中使用的算法、数据结构和其他相关的技术细节进行了说明。

3、结合多个使用案例对系统中生成的服务进行了分析检验，证明了整个服务封装系统的实用性、通用性和高效性。

4、介绍了在高分辨率对地观测系统重大专项（民用部分）的服务网格平台上由智能化服务封装系统生成的高分相关的服务的部署和运行情况，将高分相关的 Web 应用中的数据通过系统整合到了高分重大专项平台，一定程度上支撑了国家级重大专项的运作。

1.3 本文结构安排

本文的章节安排如下：

第一章：绪论。针对智能化服务封装系统的研究背景和意义进行了介绍，同时提出了本文的主要工作，最后介绍了本文的结构安排。

第二章：国内外研究现状及相关理论基础。针对系统中涉及到的国内外相关工作进行了总结，并对系统实现中用到的相关技术的基本概念进行了介绍，如 Web 服务体系架构，Google Chrome 扩展程序的架构等。

第三章：智能化服务封装系统需求分析与总体设计。为了使读者更好的理解系统的功能模块，本章对系统的需求进行了分析，同时对系统执行流程中涉及到的相关概念以及用户能够执行的相关操作进行了介绍，并阐述了系统的操作流程，最后给出了系统的总体设计。

第四章：简单数据采集分系统的设计与实现。针对简单数据采集分系统的整体架构图以及各模块之间的关系进行了介绍，并详细描述了各个模块内涉及到的算法及其他细节。最后，本章给出了简单数据采集场景下的案例分析。

第五章：复杂数据采集分系统的设计与实现。针对复杂数据采集分系统的整体架构图以及各模块内部的算法细节进行了描述。最后，本章给出了复杂数据采集场景下的案例分析。

第六章：系统测试与应用。对两种数据采集场景下生成的服务及内部流程进行了测试，同时在高分服务网格系统上对封装好的高分相关服务的运行和部署情

况进行了介绍，证明了各算法的有效性以及系统在不同场景下的高效性、实用性。

第七章：总结与展望。总结本文的重要工作，并针对未来的研究工作进行了规划。

1.4 本章小结

本章介绍了课题的研究背景，同时对高分网格平台国家重大工程进行了介绍，并提出了本文主要的研究内容，然后提出了本文的结构安排，梳理了文章的组织结构，最后是本章小结。

第2章 国内外研究现状及相关理论基础

本章主要介绍了面向 Web 应用的智能化服务封装系统中涉及到的国内外相关工作，同时介绍了系统实现过程中使用到的相关技术知识和相关术语的定义，最后是本章小结。

2.1 国内外研究现状

面向 Web 应用的智能化服务封装系统结合 Web 页面分块和流程图转化为可执行程序等技术，实现对 Web 应用中数据的采集，并将采集任务以 Web 服务的形式将呈现给用户，以方便用户进行业务对接和二次开发。因此，本小节将对系统中涉及的 Web 数据提取技术、Web 页面分块技术、Web 服务生成技术和流程图转化为可执行程序技术的国内外相关研究进行概述。

2.1.1 Web 数据提取技术

Laender, A.H.F. 等人[2]详细列举了在 Web 数据提取领域中的各种包装器（Wrapper）生成方法；Ferrara E 和 Zhang S 等人[3][4]也对 Web 数据提取方法进行了详尽的阐述，包括对深度 Web 爬虫技术的研究[5]。许多 Web 数据提取方法都倾向于去定义一套自己设定的特殊规则，并使用这套规则来完成网页数据的提取任务，如 Ristoski P 等人[6]使用了他们自己定义的语法来构建 Web 知识图谱并从中提取数据，这些方法通常对非专业人士不够友好，因为他们没有提供图形用户界面，因此，只有具备计算机领域知识的专业人士才能够使用这些工具。

许多方法在致力于简化 Web 数据的提取过程[7]。XWRAP[8]是一个监督化交互的包装器生成工具，它使用了 HTML 的 DOM 树结构来在 Web 页面中定位元素。Wiccap 数据模型[9]则将网页信息映射到了一种有组织的逻辑概念，并提供可视化的工具[10]供用户使用。Potvin B 等人 [11]使用无监督可视化验证的方法来增强 Web 数据提取中的鲁棒性，但这些方法通常都不能够做到完全脱离专业人士的干预。

也有许多研究人员尝试通过机器学习的方法对网页数据进行提取。如 Xuan H.P 等人[12]尝试使用最大熵马尔科夫模型来从 Web 页面中提取最有效的信息（即正文内容）。Wien[13] 和 Stalker[14]等算法通过从大量的网页数据中训练自己的模型，使他们可以自动化的学习提取规则并进行数据采集。然而，机器学习模型对

数据的要求量通常较大，对计算资源的要求较高，因此，它们并不适合通用性的网页提取场景。Lixto[15]是一个可以在 IMDB 视频评论网站上生成包装器的系统，同样的，用户需要知道如何编写它特定的 Elog 规则才可以使用，Lixto 同时提供了一个深度 Web 页面导航工具，但是仍然需要手动编写大量的配置规则来驱动系统的运行。因此，这些方法都不能做到简单通用化的从 Web 页面中提取数据。

Alison S 等人[16]设计了一种无需浏览器即可自动化从页面中提取数据的方法，但无法实现复杂数据提取流程的设计功能；八爪鱼采集器[17]是一个可以对 Web 数据采集任务进行可视化操作的系统，然而，八爪鱼采集器提供的 API 接口的功能有限，不能随意的修改数据采集的需求，从而无法作为直接扩展到其他业务中去，二次开发的难度较大。

2.1.2 Web 页面分块技术

Web 页面分块是一种将网页按照语义内容进行划分的技术，将 Web 页面进行分块，找到重要的正文块并生成提取规则是自动化网页信息提取任务的通用解决思路，许多学者对网页分块技术也有过很深的研究。蔡登等人[18]提出了一种基于视觉的方法，该方法可以模拟人的感知行为对页面进行划分，从而找到关键信息块。Andrew J 等人[19]提出了非视觉感知略读情况下的 Web 页面分块方案，从而将 Web 页面中相同类型的元素成功聚类。Jun Zeng 等人[20]使用了网页块的可视化特征来划分一个 Web 页面，但这种方式仅适用于页面中元素可见的情形。Msos[21]使用了混合（hybrid）的方法去划分一个面向多屏幕的网页页面。BERYL[22]是一个可以将 Web 块进行分类的系统，以便用户快速找到主题信息块。Feng H 等人[23]提出了一个基于流程分析的框架来理解网页的结构。C Liao 等人[24]则介绍了一种可以分析网页中事件（event）数据的方法来将网页中具有相同结构的块提取出来。这些方法解决了不同场景下的网页分块需求，为 Web 数据采集任务提供了技术支撑。

2.1.3 Web 服务生成技术

Sheng Q Z 等人[25]介绍了以数据为中心的大数据服务的基本概况，有部分研究者尝试将 Web 数据采集任务转化为 Web 服务[26]。H2W 框架[27]通过人工描述工作流的方式从已存在的 Web 应用中提取 Web 服务，而 Zhou, J J 等人[28]则尝试将 Web 应用直接迁移到 Web 服务上去，但是他们需要手动从 Web 页面中收集数据然后再手工生成包装器。Jarrett J 等人[29]使用服务体系架构来预处理推特的情感

分析数据。Sarkar A 等人[30]提出了一种在聊天应用程序中对 Web 链接生成智能预览的 Web 服务，用于从预览的 URL 中提取数据并展示。Ali K 等人[31]提出了一种面向服务的方法来可视化和分析社交媒体平台中的 Web 数据。Cao H 等人[32] 提出了一种从纯 HTML 源码中自动生成 REST API 文档的模型，但对 HTML 所在页面的结构提出了限制，对复杂结构的 Web 页面的解析效果较差。Zhang F 等人[33]设计了一个提取地理信息数据的服务封装系统，但是该模型只能对 Web 环境中的地理信息数据进行采集，具有较大的局限性。

2.1.4 流程图转换为可执行程序技术

本文实现了将服务流程图转化为可执行程序的功能，这种自动化编程技术也被广泛的研究。Xiang-Hu Wu 等人[34]描述了一种将结构化的流程图转换为程序代码的算法，并提出了一套完备的代码问题分析和转换的方案[35]。Supaartagorn C 等人[36]提供了一个使用结构化流程图作为自动代码生成器的工具来自动执行 Web 应用中的程序。Qu M 和 Gong W J 等人[37]提出了一种算法，可将流程图转换为代码，并具有错误检查的功能；同时，他们详细定义了流程图识别和错误检查过程中的相关策略[38]。

Yokoyama T 等人[39]提出了可逆流程图的基本架构，并使用示例语言来具体实现可逆流程图。Palaniyappan S 等人[40]实现了从单维度流程图中生成自动执行代码的功能。Hussein B M 等人[41]设计了一套框架，负责将流程图转化为基于模型的代码等。Flowgen[42]是一个可以从 C++代码中反向转换为流程图的工具。借助以上代码和流程图之间转换的设计思路，我们完成了 Web 数据采集场景下将数据采集流程转换为可执行程序的功能。

2.2 相关技术基础与概念

本小节将对面向 Web 应用的智能化服务封装系统中使用到的相关技术和架构，如 Web 服务体系架构、网络爬虫技术等概念进行介绍。

2.2.1 面向服务的架构与 Web 服务体系结构

2.2.1.1 面向服务的架构

面向服务的架构（service-oriented architecture，SOA）[43]是一种分布式执行的软件设计方法，软件的部分组件（调用者）可以通过网上的通用协议（如 HTTP）操纵另一个应用软件组件的运行，从而获得相应的服务。图 2.1 描述了 SOA 的层

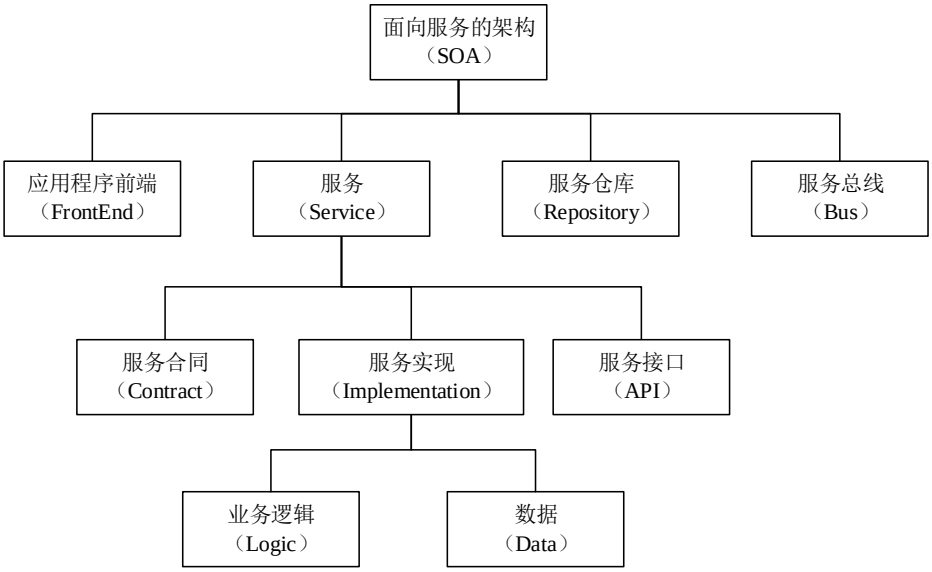


图 2.1 面向服务的架构（SOA）层次结构图

次结构图，服务架构由应用程序前端、服务、服务仓库和服务总线组成，应用程序前端负责展示服务信息及示例操作；服务仓库负责对服务进行持久化的处理；服务总线负责对服务仓库中的服务进行统一管理，集成及协议交换，从而完成复杂的业务需求；而一个服务中包含服务合同、服务实现和服务接口三部分，分别代表服务的需求、实现和调用方式，服务实现中包含服务的实际业务逻辑和服务对应的数据两部分，两者结合实现服务需要完成的功能。

一个 SOA 中的服务应该具有以下特性：

1) 服务之间通过简单、精确定义的接口进行通信，不涉及底层编程逻辑和通信模型。

2) 粗粒度性：服务只提供一项特定的业务功能，采用粗粒度服务接口使得使用者和服务层之间不必进行多次往复的交互。

3) 松耦合性：不同服务之间保持相对独立无依赖的关系，使系统具有灵活性，且当组成整个应用程序的服务内部结构和实现逐步地发生变化时，系统可以继续地独立存在。

4) 位置透明性：SOA 系统中的所有服务对于其调用者来说都是位置透明的，即服务调用者只需要知道想要调用的是哪一个服务，而并不需要知道所调用服务的物理位置。

5) 协议无关性：每一个服务都可以通过不同的协议来调用。

2.2.1.2 Web 服务概念

Web 服务（Web Service）[44]是一种 SOA 的技术，通过标准的 Web 协议提供服务，它有两种表现形式：

1、一个电子设备向另外一个电子设备提供服务，两者通过互联网进行交互；

2、服务器侦听网络上特定端口上的请求，提供 Web 文档（如 HTML，JSON，XML），并创建 Web 应用程序服务，这些服务可解决 Web 上的特定领域问题。

本文中封装和使用的服务属于第二种类型。Web 服务技术使得运行在不同设备和平台上的上的应用无需借助任何附加的第三方软硬件，即可进行交互并交换想要的数据。依照 Web 服务规范进行的业务，不论内部协议、平台或语言，均可以相互交换数据。

Web 服务通常向外界暴露出一个能够通过 Web 进行调用的 API，使用户可以通过直接调用 API 的方式来得到想要的数据，而无需知道具体的服务实现细节。

通常，用户调用 API 时以 GET 或 POST 方式提供输入参数给 Web 服务，服务读取 API 输入参数并进行相应处理后，会以结构化的数据文档格式（如 XML，JSON，CSV）将输出数据返回给用户。

本文生成的面向 Web 应用采集任务的 Web 服务的形式化定义如下：

一个 Web 服务可以被形式化描述为一个六元组：

$$WS =< id, Desc, In, Out, Graph, QoS >,$$

其中：

1) *id* 是服务的唯一标识码，在服务生成阶段由系统自动指定。

2) *Desc* 是服务的名称和描述的集合。

3) *In* 是服务所需要的输入参数的集合。

4) *Out* 是服务所提供的输出参数的集合。

5) *Graph* 是服务实现逻辑中的工作流程图，记录了 Web 数据采集任务的操作步骤和流程。

6) *QoS* 是服务质量参数集，本文中的服务质量包含服务响应时间，有效性和吞吐量等指标。

2.2.1.3 Web 服务体系结构模型

Web 服务体系结构包含三种角色：服务提供者、服务注册中心和服务请求者，他们之间相互进行交互（发布、发现、绑定）来完成业务。如图 2.2 所示，服务提供者定义 Web 服务并将其发布到服务注册中心（发布），服务请求者从服务注

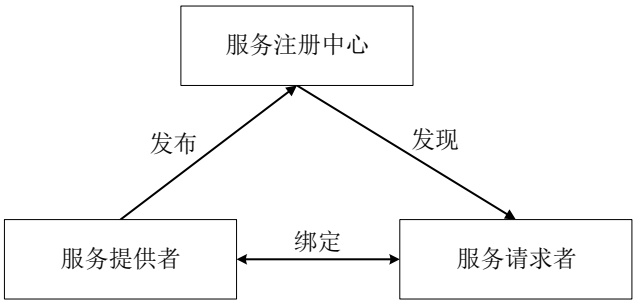


图 2.2 Web 服务体系架构图

册中心中查找服务（发现），通过服务描述与服务提供者进行绑定并最终调用 Web 服务来得到自己想要的信息（绑定）。

本文中的 Web 服务开发周期描述如下：

1) 构建：开发和测试 Web 服务的实现，定义服务接口描述和定义服务实现描述。本文中服务的实现即用户使用智能化服务封装系统定义采集流程；服务的接口描述和实现描述由系统自动生成，并可被用户修改。

2) 部署：发布服务接口和服务实现的定义，将可执行文件部署到执行环境中。本文中服务的部署由系统自动完成。

3) 运行：服务提供者通过服务注册中心提供服务，服务请求者可以进行绑定和查找服务的操作。本文中服务的运行即系统按照用户设定的流程自动化的采集 Web 数据并存储到相应位置的过程。

4) 管理：持续的管理和经营 Web 服务应用程序，解决安全性、可用性、性能、服务质量和业务流程等问题。本文中服务的管理包括服务的增加、删除、查询和对服务描述的修改以及对数据采集流程的修改等。

2.2.2 网络爬虫相关技术

2.2.2.1 网络爬虫概览

网络爬虫（Web Crawler）[45]，又名网络蜘蛛，是一种可以自动化的在互联网上进行浏览和数据提取的机器人。最初，网络爬虫是为了对 Web 应用中的数据进行索引而存在，从而使搜索引擎可以实时更新自己的搜索结果，为用户提供更好的服务。通用网络爬虫经常从一个网站的首页 URL 出发，对首页中的数据进行采集，并循环点击页面中的超链接，递归的爬取一个网站的所有页面和数据，并将采集到的数据存储到自己的数据库中。图 2.3 给出了经典的网络爬虫技术的架构。

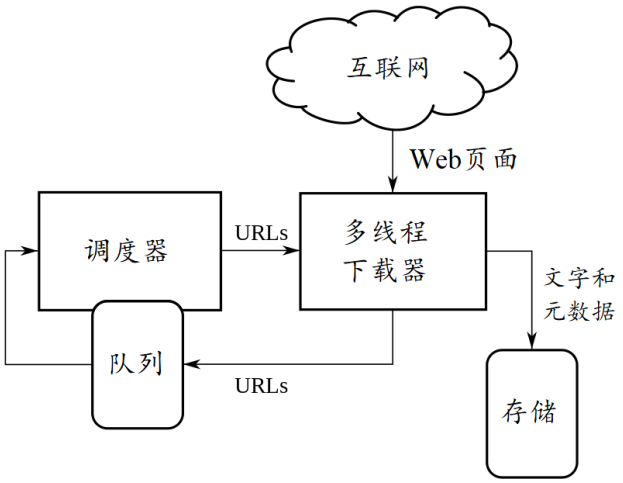


图 2.3 网络爬虫技术架构图

聚焦网络爬虫[46]是一种定向抓取相关网页资源的定制化爬虫程序，它通常只采集与特定主题内容相关的网页，而不追求大而广的覆盖。聚焦网络爬虫需要根据采集者的需求，定向的过滤掉与主题无关的链接并根据设定的搜索策略选择下一步要抓取的页面进行采集。对于大多数业务来说，数据采集任务均是通过定制聚焦网络爬虫程序来实现。

2.2.2.2 网络爬虫的传统实现方式与 Selenium WebDriver

聚焦网络爬虫中的输出数据是通过将 Web 页面的超文本标记语言（HTML）源码进行格式化的处理后的结果。如图 2.4 所示，HTML 以 DOM 树状结构图的形式呈现，HTML 中的每一个元素都以一个节点的形式在 DOM 树中展示。图中的元素<h1>即为<body>元素下的子元素，它的父元素为<body>，子元素为文本元素“我的标题”，同时含有一个兄弟元素<a>。

在对 HTML 源码进行格式化处理的过程中，通常需要通过元素的 XPath 来定位到元素，并采集元素的相关数据。XPath[47]是一种可以在 XML 文档和 HTML 文档中查找信息的语言，通过对文档的元素和属性进行遍历，便可以得到自己想要的节点内容。XPath 使用路径表达式在 HTML 和 XML 文档中进行导航，并选取文档中的节点或者节点集。如：

1) */html/body/h1* 即定位到了图 2.4 中的<h1>元素节点，注意，如果图 2.4 中的 body 节点含有多个<h1>节点，则所有的<h1>节点会被全部匹配到，即通过一条 XPath 可能会定位到一个 Web 页面中的多个元素。

2) *//*[@id="search"]*即定位到当前 Web 页面 id 属性为 search 的元素节点。

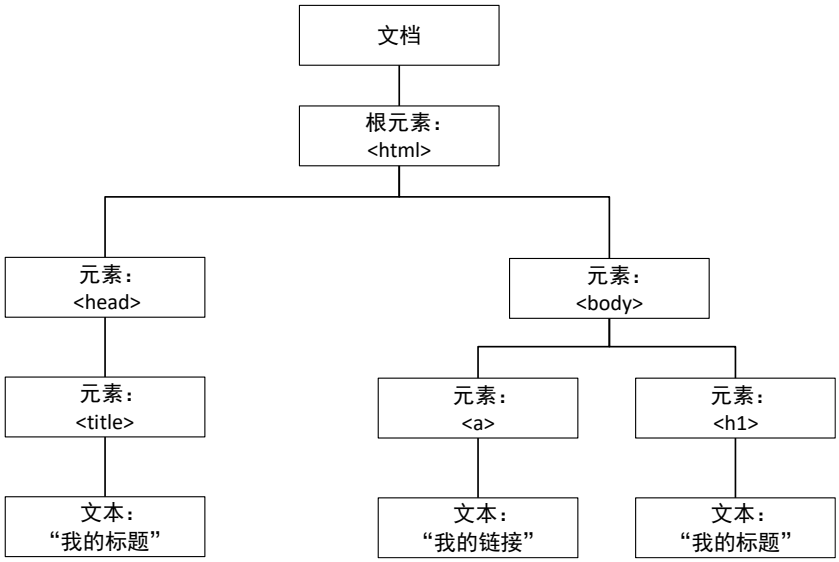


图 2.4 HTML DOM 树结构

3) *//a[text()='下一页']*即匹配页面中所有文字内容为下一页的超链接元素。

由于 XPath 的语法结构较为简单，用户在短时间内便可以熟练掌握，因此适合作为本文中系统的查询语言来供用户使用。

这种通过 URL 得到 HTML 源码并使用 XPath 进行定位和数据采集的模式正被爬虫编写者广泛使用。然而，随着网络带宽速度的提升以及富媒体时代的到来，网页结构和展现形式发生了巨大的变化，如今，只通过提取网页源码通常很难得到想要的数据。Ajax[48]技术的产生使得用户在不刷新的情况下更新 Web 页面的内容成为了可能，这些通过 JavaScript（JS）技术动态得到的数据无法直接从网页源码处得到。同时，许多 Web 数据通常需要用户与网页进行交互才能得到，如在某些购物网站中，用户需要在页面搜索框中输入关键词“电脑”，并点击“搜索”按钮才能得到电脑相关的商品列表，而直接访问网页 URL 则无法得到商品数据；再比如，许多网站页面设置了权限控制，用户通常需要先登录之后才能访问相应的 Web 页面，否则将采集到错误的数据（如 Access Denied）。因此，如何得到这些由 JS 控制而动态生成的数据成为了一个巨大的挑战。

Selenium [49]是一个便携式的 Web 应用程序测试框架，它提供了一种回放工具，用于对 Web 页面进行功能测试，而无需学习测试脚本语言。它的测试语句可以用多种流行的编程语言编写，包括 C＃，Python，Ruby 和 Scala。它可以在大多数现代的 Web 浏览器上运行测试，通过模拟普通用户的行为，如点击链接，输入文本框等操作来打开网页并提取想要的数据。通过 Selenium 的 WebDriver 可以对

Web 文档中的 DOM 元素进行控制和操作，自动化的执行自定义的操作流程并进行 JS 的脚本控制。WebDriver 运行时会打开一个浏览器实例，然后通过控制浏览器进行点击鼠标和输入文字等行为来实现网页浏览，因此，使用 WebDriver 进行网页浏览时，其效果和人工浏览一致。

本文中的智能化服务封装系统在对 Web 应用中的数据进行采集的过程中，通过操作 Selenium WebDriver 的浏览器实例来对预定义的用户操作行为进行复现并进行数据采集，通过这种方式，可以得到与用户手工浏览网页时相同的 Web 数据，包括由 JS 动态生成的数据，并可以通过网站的权限控制。

2.2.3 Google Chrome 扩展开发技术

在用户定义 Web 数据采集任务的过程中，智能化服务封装系统在用户操作的每个 Web 页面上都提供了一个可视化的操作模块，使得用户可以通过直接在 Web 页面上进行点选的方式选择元素，并定义 Web 数据的采集流程。想要实现上述功能，就需要用到 Web 浏览器的脚本注入技术。

如今市面上，67%的用户使用 Google 的 Chrome 浏览器进行网页浏览，而市面上存在的国产浏览器，如 360 极速浏览器，搜狗浏览器等，均是基于 Chromium 内核[50]（Chrome 浏览器的开源引擎）进行构建。2018 年年底，微软公司宣布将自己的 Edge 浏览器的内核从 IE 核更换为 Chromium 内核，意味着 Google 的 Chromium 内核已全面占领浏览器市场。Chromium 内核同时兼容新老版本的 Web 页面，因此，如今的 Web 页面均可以使用 Chrome 浏览器进行访问。本文使用 Google Chrome 浏览器的扩展开发技术，来实现对 Web 浏览器的脚本注入，以构建用于记录操作流程的图形用户界面。

Chrome 扩展（Chrome Extension）[51]是一种定制 Web 浏览体验的小型软件程序，使用户能够根据个人需求或偏好调整 Chrome 浏览器的功能和行为。它建立在诸如 HTML、JavaScript 和层叠样式表（CSS）之类的 Web 技术上，可以修改用户看到并与之交互的 Web 内容，或者扩展和更改浏览器本身的行为。本文中使用到的 Chrome 扩展中的各模块作用如下：

1、manifest：扩展的入口配置文件，记录了扩展的名字，版本号，描述，图标等基本信息。同时记录了扩展的其他子模块，如后台脚本，内容脚本的配置信息，以及扩展运行所需要请求的权限信息等。

2、background-script：后台脚本，是一个可以常驻浏览器后台的页面，其生命周期从浏览器打开持续到浏览器关闭，而不会因为 Web 窗口或标签页的关闭而

关闭。因此，需要一直运行的操作以及跨页面的操作，都需要放在后台页面中进行。后台页面具有非常高的权限，可以调用 Chrome 扩展程序提供的所有 API，并且可以无视网站的跨域权限要求而访问任何网站。

3、content-script：内容脚本，是 Chrome 扩展程序向 Web 页面注入脚本的一种形式。通过在入口配置文件中对内容脚本进行配置，就可以实现向任意 Web 页面注入指定的 JavaScript 和 CSS 文件，从而实现例如页面定制、广告屏蔽等功能。

内容脚本的生命周期自网页加载持续至网页卸载。网页的加载和卸载行为通常发生在以下情形下：

1) 通过浏览器地址栏输入 URL 打开页面时，网页加载；关闭页面所在标签页时，网页卸载。

2) 刷新页面时，刷新前的页面卸载，刷新后的页面加载。

3) 点击页面超链接或者按钮时，如果发生了页内跳转，即没有新的标签页被打开，而是当前标签页所在页面的 URL 从 URL_1 跳转到 URL_2 时，URL_1 所属页面卸载，URL_2 所属页面加载；如果打开了新的标签页，则 URL_1 所在页面没有被卸载，而 URL_2 所在页面进行了加载。

4) 通过 Ajax 技术动态获取数据时不存在页面卸载行为，因为使用 Ajax 请求数据后页面并没有发生 URL 跳转和页面刷新行为。

内容脚本在网页加载时将入口配置文件中指定的 JS 和 CSS 文件注入页面，在页面本身脚本执行完毕后，会继续执行内容脚本中所含的脚本并加载相应的样式表。内容脚本和页面脚本共享页面 DOM，因此，可以直接在内容脚本中通过 JS 来操控页面元素，实现对元素的增删改查。

如图 2.5 所示，本文通过内容脚本技术向 Web 页面中进行了元素注入，实现了在浏览器打开任意页面时，将自己定义的图形化操作提示框（操作台）嵌入到页面中的功能，为记录用户操作流程奠定了基础。

图 2.5 内容脚本注入示例

4、Message：消息。如第 1、2 条所述，打开浏览器后，无论发生多少行为和操作，浏览器只维护一个后台脚本实例，而每一个加载的 Web 页面都会产生一个内容脚本实例，且网页卸载后与之对应的内容脚本实例将会被释放。因此，两者之间需要进行消息通讯，来将内容脚本产生的信息，如用户的操作，页面指定元素的数据等，传送到后台，从而保存到全局并进行下一步的处理。

内容脚本通过调用 Chrome 扩展提供的接口 *chrome.runtime.sendMessage* 向后台传送消息；同样的，后台脚本调用 *chrome.runtime.onMessage.addListener* 接口的方式添加监听器来接收内容脚本的消息，并可以通过回调函数的方式向内容脚本回复消息。

2.2.4 WebSocket 技术

如第 2.2.3 节所述，Google Chrome 扩展中的后台脚本可以全局的记录用户的操作行为，但由于浏览器的安全性限制，无法直接将后台脚本实例中维护的信息在用户本地进行存储，因此需要借助其他方法来将这些信息保存到浏览器外部（如服务器中）。使用 JS 的 Ajax 技术，结合服务器后台开发的数据存储接口，可以一定程度上解决此问题。然而，频繁的进行 Web 交互会降低程序的执行效率，网络带宽的变化也会对程序的稳定性产生影响。

同时，本文中的智能化服务封装系统是由包括浏览器在内的多个外部进程组成，这些进程需要与浏览器进行相关的交互，如服务生成模块想要询问浏览器此时的加载状态，或者告知浏览器此刻要执行的操作等。如果使用 Ajax 技术，则只能通过轮询机制来对服务器进行状态的查询和写入，这种操作方式效率很低，而且容易出现因时间差导致的异步错误。因此，系统需要一种机制，使得各个进程和浏览器之间可以监听彼此的消息，从而实现实时的交互。

Socket[52]，又名套接字，是计算机之间进行通信的一种约定。计算机之间可以通过 Socket 接收其他计算机的数据，也可以向其他计算机发送数据。操作系统通常会为各应用程序之间提供一组 Socket 接口，从而实现操作系统内部各进程之间的交互。在网络通信中，Socket 使用 TCP 或者 UDP 作为传输协议，利用三元组（ip 地址，协议，端口）标识网络中的进程并绑定相应的端口，从而实时监听端口中的信息，并可以向端口中传送消息，典型的 Socket 通信流程如图 2.6 所示。

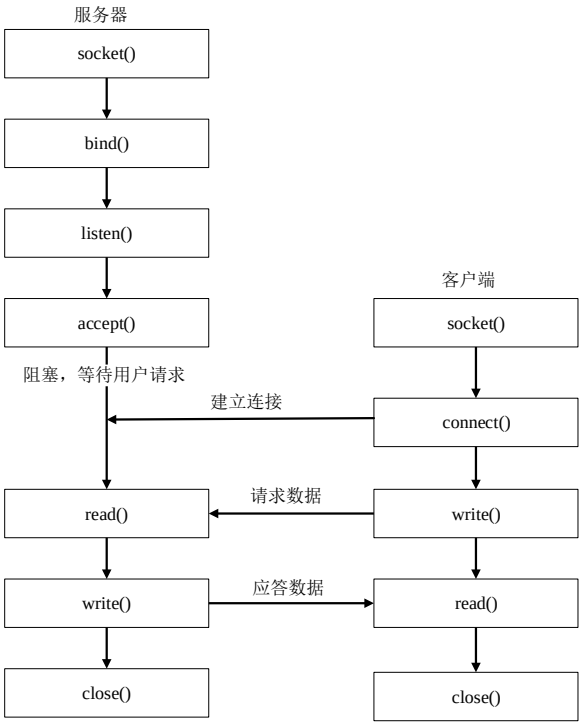


图 2.6 Socket 通信流程图

2011 年，国际互联网工程组（IETF）将 Web 上的 Socket 技术定为标准 RFC 6455，并命名为 WebSocket。WebSocket[53] 是 HTML5 规范提供的一种可以在单个 TCP 连接上进行全双工通讯的协议，如图 2.7 所示，WebSocket 技术使得服务器可以主动向客户端推送数据，在 WebSocket API 中，浏览器和服务端只需要完成一次握手，两者之间就可以建立持久的连接，并进行双向的数据传输。相比于 Ajax 轮询机制，WebSocket 协议能更好的的节省服务器带宽和资源，并进行实时的通讯。

在 WebSocket 协议中，只需要通过访问服务端提供的 URL 地址即可建立 WebSocket 连接，通过 JS 的 *WebSocket* 对象的 *Open*，*Close*，*Send*，*OnMessage* 方法，即可对 WebSocket 连接执行打开，关闭，发送和接收消息的操作。

本文使用 WebSocket 作为智能化服务封装系统中各个模块和进程之间的消息通讯协议，来实现各进程之间，及进程与浏览器之间的交互。

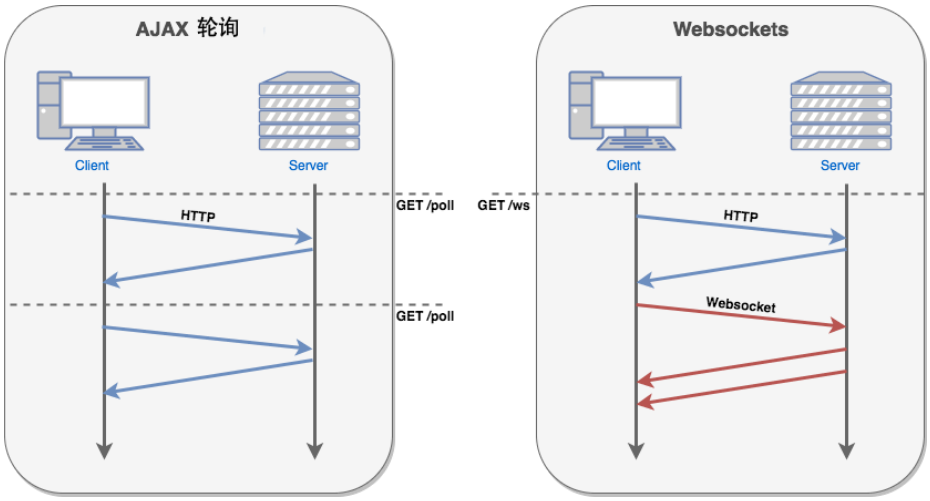


图 2.7 传统 Ajax 轮询和新的 WebSocket 机制的通讯方式对比

2.3 本章小结

本章对智能化服务封装系统中涉及到国内外相关研究进行了总结，并对系统实现过程中的相关理论和概念进行了介绍，包括 Web 服务与网络爬虫的定义，Google Chrome 扩展开发技术的架构，WebSocket 技术的规范等。

第3章 智能化服务封装系统需求分析与总体设计

为了更好的理解面向 Web 应用的智能化服务封装系统的功能，为接下来讲述各分系统的架构设计和关键实现打下基础，本章对系统的功能需求进行了细致的归纳和总结，并同时对系统执行流程中涉及到的相关概念以及用户能够执行的相关操作进行了定义，最后，本章给出了系统的总体设计。

3.1 Web 应用架构简介

由 Web 应用的开发模式可知，Web 应用分为后台程序和前端程序两部分。如图 3.1 所示，后台程序负责对 Web 应用中的业务逻辑进行处理，并负责对用户的会话信息（session）进行控制等，后台程序结合数据库对用户的数据做持久化的保存和处理。后台程序可以由各种各样的语言和框架实现，如 ASP、PHP、Spring 等，并结合如 MySQL、MongoDB 等数据库完成后台整体操作逻辑。前端程序即 Web 应用展示给用户的可视化操作界面，通过 HTTP 协议，前端页面可以显示在浏览器中，并结合 JavaScript 等技术完成与后台程序的交互。

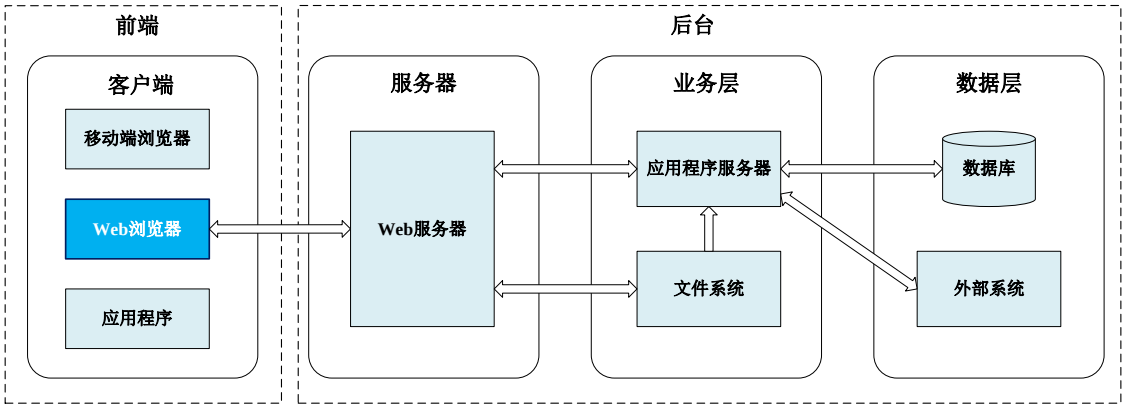


图 3.1 Web 应用开发架构图

如今，Web 应用的开发分为前后端分离和前后端不分离两种模式，前后端分离模式即后台只提供数据接口，前端负责通过 JS 从接口中读取数据从而渲染页面；前后端不分离模式指后端既负责提供数据，也负责渲染前端页面，从而将数据展示给用户。但无论哪种开发模式，以及后台程序使用哪种语言或框架，最终要采集的 Web 应用中的数据均以 HTML 源码的形式展示。同时，通过 JS 动态生成的数据，也会以 HTML 源码的形式出现在浏览器的 Web 页面中，而最新的

HTML5 规范兼容了之前的 HTML 的源码格式，因此，我们只需要按照 HTML5 的规范对图 3.1 中 Web 浏览器中的 Web 应用前端页面进行解析，即可完成 Web 数据采集任务。由于任意 Web 页面的源码均开源，因此大大降低了从 Web 应用中采集数据的难度。

本文中的智能化服务封装系统通过模拟人工浏览网页的行为，来实现对用户操作流程的自动复现。本文根据对 Web 应用中前端页面的操作复杂程度的不同，设计了简单数据采集和复杂数据采集两类 Web 数据采集场景，以满足用户不同的数据采集需求。

3.2 简单数据采集场景分析

为了使读者更好的理解简单数据采集场景的概念，本节首先对场景下涉及的块与子块、数据页面等概念进行了解释，然后分析了服务体系架构中各类用户的实际需求，最后对简单数据采集场景下 Web 应用数据采集服务的生成和调用流程进行了阐述。

3.2.1 相关概念定义

简单数据采集场景：简单数据采集场景指用户在输入一个 Web URL 后，只需要在页面中执行有限的前置操作（如填写表单并点击提交按钮）即可得到想要采集的数据的场景，且数据在 Web 页面中具有下列特征之一：

1、数据以表格形式展现，在 HTML 源码中以<table>标签存在。

2、数据以列表形式展现，在 HTML 源码中通常以<ul>和<li>标签的形式存在。

3、数据为大段文本，在 HTML 源码中以<p>标签或 标签的形式存在。

满足以上特征的数据所在的 Web 页面的结构较为简单清晰，因此无需过多人工干预，即可通过分块算法自动的识别到数据所在块，从而满足数据采集的需求。

即在简单数据采集场景下，Web 应用的数据采集流程最多包含两步：

1、表单填写（可选）和 **2、数据提取**。

简单数据采集场景下涉及到的相关概念如下：

1、块和子块（Block 和 Sub Block）：一个**块**指 Web 页面的一个区域，其中包含信息化的数据。一个块在 HTML 中以 DOM 子树的形式表示，使用 Web 页面分块算法，我们可以将一个 Web 页面分割成多个具有不同语义内容的块。一个块通常由多个**子块**组成，这些子块彼此之间的结构是相同的，子块结构中的每一

个元素的值都将对应最终生成服务中的一个输出参数。在数据采集过程中，系统将提取块中每一个子块的信息，并按照各子块内的元素进行归类。

2、表单：许多 Web 页面中的数据并不是直接打开 URL 地址就可以得到，而是需要通过提交表单的形式获得。一个表单通常由多个输入框（包括下拉框、多选框等）和一个提交按钮组成。用户通过在表单的输入框中填写查询条件后，通过点击提交按钮得到数据。每一个输入框将对应最终生成服务中的一个输入参数。

3、查询页面：查询页面指表单所在的 Web 页面。在部分情况下，用户并不需要进行表单查询操作，因此，查询页面在某些案例下并不存在。

4、数据页面：数据页面指最终采集的 Web 数据所在的页面。

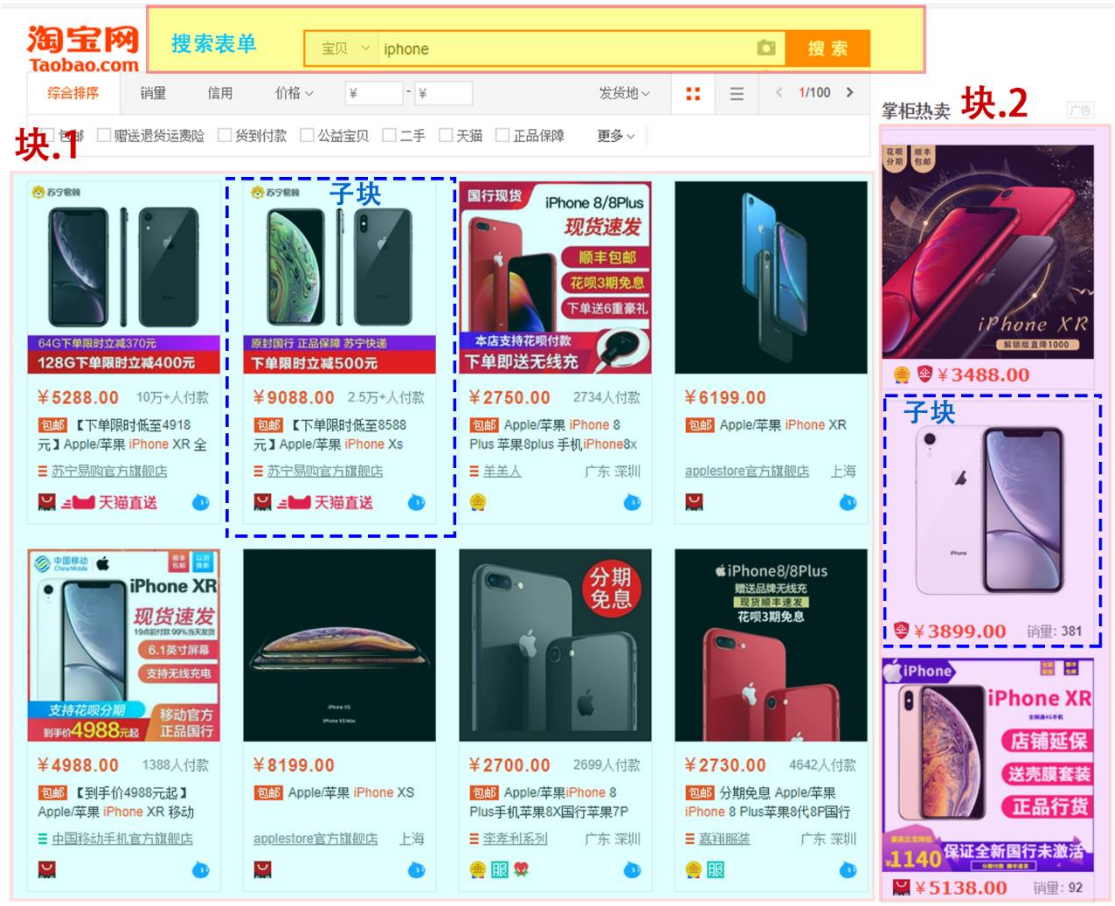


图 3.2 淘宝商品列表页面分块和表单示例图

如图 3.2 所示，在淘宝商品列表页面中，我们可以将页面的主要信息区域分为两个块，块 1 为主要商品信息块，其中包含 8 个子块，每个子块展示一条商品信息，子块中包含文本、图片、超链接等元素。块 2 为广告信息块，其中包含 3 个子块，每个子块中均包含手机图片、手机价格和销量三个主要元素。同时，图

中含有一个搜索表单，表单中有一个输入框和一个提交按钮（即搜索按钮）。在图 3.2 的 Web 页面中提交搜索表单后，相应的商品信息会通过 Ajax 技术显示在同一个页面中，因此，图 3.2 中的查询页面和数据页面为同一个页面。

3.2.2 需求分析

由第 2.2.1 节可知，Web 服务体系结构包含三种角色：服务提供者，服务请求者和服务注册中心，针对简单数据采集场景下的 Web 数据采集服务，各角色所能执行的操作如下：

1、服务提供者：服务提供者负责设计和管理系统中的 Web 数据采集服务。服务提供者发起服务生成请求后，在系统提供的 GUI 界面上设计 Web 数据采集的流程，并提供服务基本信息，交互式的完成 Web 数据采集服务的创建。同时，服务提供者可以在服务注册中心中对自己生成的服务进行管理，如增加，删除，查看，修改服务等。

2、服务请求者：服务请求者可以发起服务调用的请求，从而得到想要的 Web 应用中的数据。服务请求者以 Web 请求的方式访问服务提供的 API 从而发起服务调用请求，等待系统自动执行 Web 数据的采集任务，系统根据 Web 请求中定义的输入参数，即服务请求者设定的数据处理需求对提取到的数据进行格式化的处理，并最终返回给服务请求者。同时，服务请求者还可以对服务注册中心中公开的服务进行查看，并根据需求购买、调用和测试所有可用的 Web 数据采集服务。

3、服务注册中心：服务注册中心负责对服务提供者生成的 Web 数据采集服务进行管理和维护。服务注册中心中记录了所有服务的基本信息，如服务名称，服务描述，服务的输入输出参数格式以及服务的示例返回值等。同时，服务注册中心中记录了各 Web 数据采集服务的具体实现逻辑和流程，以供系统在服务调用阶段对流程进行复现。服务注册中心管理员可以查看、修改和测试服务注册中心中存储的所有 Web 数据采集服务，定期维护服务的有效性。

本文将 Web 应用中的数据采集服务的操作过程总结成两个主要阶段：**服务生成阶段**和**服务调用阶段**。服务提供者在服务生成阶段定义数据采集服务的操作流程并生成 Web 服务；服务请求者在服务调用阶段以 Web 请求的方式调用服务，并得到采集的数据。

在服务生成阶段，系统自动执行的操作包括：检测页面的表单信息、页面分块、页面块排序、服务参数分析及生成；需要服务提供者手工执行的操作包括：提供 Web 页面 URL、选择要填写的表单、选择要提取的数据块、修改输入输出参

数值以及提供服务基本信息。

在服务调用阶段，系统自动执行的操作包括：分析服务请求参数、读取 Web 数据采集流程、执行 Web 数据采集和数据的格式化处理；需要服务请求者手动执行的操作包括：提供服务输入参数。

本文中的服务提供者和服务请求者通常是同一批用户，即：用户在智能化服务封装系统中对 Web 数据的采集流程进行定义，并以 Web 服务的形式发布到服务注册中心，然后定期的对生成的 Web 服务进行调用来获得想要的 Web 数据。

3.2.3 流程设计

图 3.3 展示了简单数据采集场景下系统的执行流程。

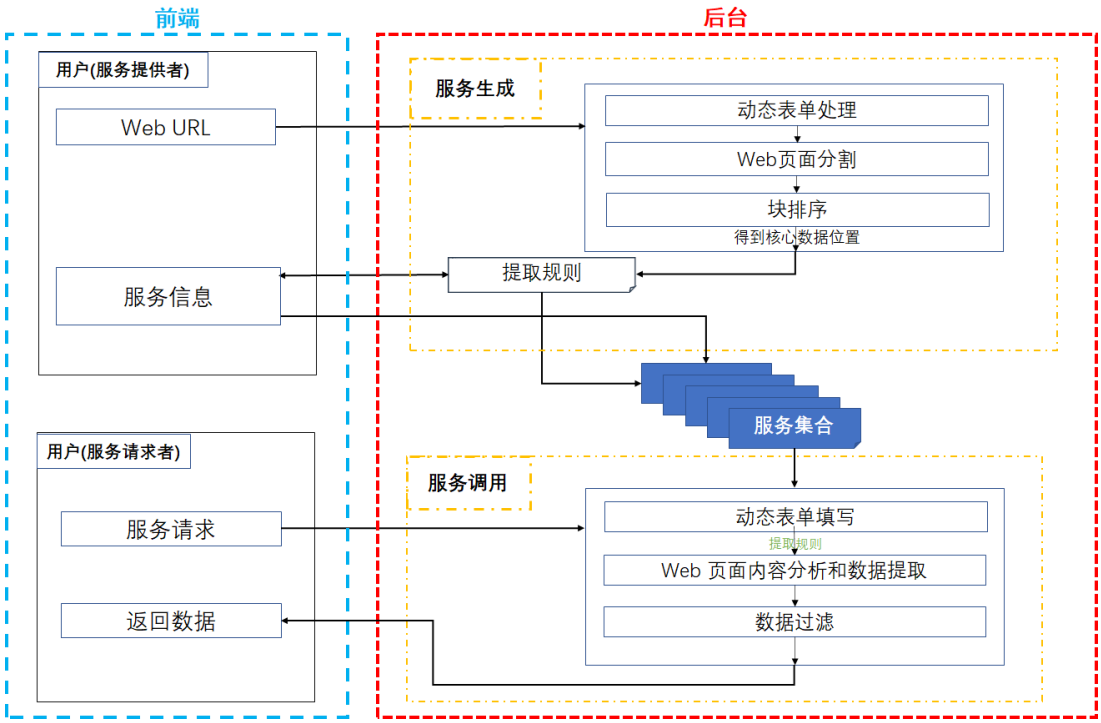


图 3.3 简单数据采集场景下系统的操作执行流程

在服务生成阶段，各流程的简单描述如下：

（1）用户（服务提供者）输入自己想要输入表单的查询页面的 URL（如果无需表单查询操作，则直接输入数据页面的 URL）进行服务生成请求。

（2）系统后台对 Web 页面进行动态表单检测，并将页面中找到的所有表单信息返回供用户选择。

（3）用户选择自己要填写的表单，并提供字段样例值。

（4）系统操纵浏览器提交表单，并获得数据页面。

（5）系统对数据页面进行分割，将页面按信息内容分割成不同的块，每个块内包含多个子块，子块之间共享相同的结构。

（6）系统对分割好的块按照重要程度进行排序后，将块排序结果返回给用户。

（7）用户选择自己想要提取的数据块（可多选）。

（8）系统按照用户选择的块和内部的子块中包含的元素信息生成服务提取规则以及服务输出参数，由用户提供服务名称和描述后，保存 Web 服务到服务注册中心。

在服务调用阶段，各流程的简单描述如下：

（1）服务请求者以 Web 请求的方式调用 Web 数据采集服务。

（2）系统根据 Web 请求中输入的 ID 号从服务注册中心的服务集合中找到相应服务，并得到服务流程中定义的 Web 数据提取规则。

（3）系统根据 Web 请求中的表单相关的输入参数自动填写并提交查询页面中指定的表单，得到数据页面。

（4）系统根据步骤（2）中得到的 Web 数据提取规则，对数据页面中的数据进行提取。若数据存在多页，还可进行翻页操作。

（5）系统按照服务请求者在 Web 请求中设定的数据处理规则相关的输入参数，对提取到的数据进行过滤，并根据服务中的输出参数信息将数据以格式化的方式返回给服务请求者，完成 Web 数据采集任务。

3.3 复杂数据采集场景分析

由第 3.2 节可知，简单数据采集场景下，系统的功能较为有限，只适用于 Web 数据采集任务步骤较少、Web 页面结构简单清晰的情形。而多数情况下，用户需要根据自己的意愿设计较为复杂的操作流程来进行数据采集，因此，本文同时开发了复杂数据采集分系统，来支持复杂数据采集场景下更为多样化的 Web 数据采集需求。

3.3.1 相关概念和模块定义

复杂数据采集场景：用户可以自行配置操作规则，以满足多样化的数据采集需求的场景。由第 3.2 节可知，简单数据采集场景下，用户无法自定义操作规则，只能按照“填写表单”→“提取数据”的顺序执行数据采集任务，具有较大的局限性。而复杂数据采集场景下，用户可以任意的定义数据采集过程中涉及到的各

个操作，如点击元素、输入文字等，并配合循环和判断条件来完成更加复杂和多样化的数据采集需求。如用户可以设置先登录后采集的数据采集服务，或循环对页面中的链接进行点击以采集链接对应页面信息的服务等。在复杂数据采集场景下，服务实现流程中所有的规则由用户自行配置，并在服务调用阶段由系统自动执行。

复杂数据采集场景仍然使用第 2.2.1 节中的 Web 服务体系结构对 Web 数据采集任务进行定义和执行，因此，复杂数据采集场景下仍然包含服务提供者、服务请求者和服务注册中心三种角色，并同样包含服务生成和服务调用两个阶段。

在复杂数据采集场景下的服务生成阶段，服务提供者需要在以下两个模块中进行操作：

1、服务流程定义模块：服务流程定义模块是一个加载了本文设计的 Google Chrome 扩展程序的定制化浏览器，负责对用户的操作流程进行定义。如图 3.4 所示，加载了扩展的浏览器会出现一个蓝色边框的服务提示框（又称操作台），用户可以根据提示框中的提示选项对选中的元素执行第 3.3.2 节提到的各种操作。服务流程定义模块会改变 Web 页面的浏览模式，使得元素的背景颜色随着鼠标移入移出而变化，以方便用户快速定位和选中想要进行操作的元素。在图 3.4 中，鼠标当前所在位置为第一个商品子块，而第二个商品块为已经被选中的子块。同时，第一、三个子块与被选中的第二个子块类型相同，因此被用蓝色边框标出。

图 3.4 服务流程定义模块示例图

注意，这里的块与子块定义与第 3.2.1 节中的定义稍有不同，当两个节点的结构相似（不是必须要完全相同）时，我们就将其视为同一个块内的不同子块，如图中第三个子块不包含“险”图片，我们也将其视为京东商品列表中的一个子块。

2、服务流程管理模块：服务流程管理模块是一个可以自定义数据采集流程的

操作台，负责对 Web 数据采集流程中的各个操作进行管理。服务提供者在服务流程定义模块定义的每一步操作，都会同步显示在服务流程管理模块中。服务提供者可以对自己定义的操作中涉及的相关参数进行修改，如元素定位的 XPath，操作名称等；同时，服务提供者还可以对各操作进行增删改查和剪切复制的控制。

如图 3.5 所示，服务流程管理模块由三个部分组成，最左侧工具箱中定义了各种操作选项，以及调整锚点（即调整下一步要插入操作节点的位置），剪切，复制，删除，取消操作的功能选项。中间的服务操作流程图可视化的展示了服务提供者定义的服务流程，并配合工具箱和参数面板，实现对服务流程的可视化管理。右侧参数面板负责对每一个操作节点中所包含的参数进行展示和修改，如图 3.5 中可在参数面板中修改“打开网页”操作的 URL，或者执行后的等待秒数等，从而改变数据采集过程中实际打开的网页，以及打开网页后需要等待的时间。

最后，服务流程管理模块还包含一个可以查看和修改服务信息的模态框，负责将服务信息传给服务注册中心，从而实现服务的新增和修改。

图 3.5 服务流程管理模块示例图

3.3.2 需求分析

复杂数据采集场景下，系统中各个角色所能执行的操作定义如下：

1、服务提供者：服务提供者需要在服务流程定义模块中定义自己想要执行的操作，并可以在服务流程管理模块中修改各操作顺序及相关参数。在一个复杂的 Web 数据采集服务中，服务提供者可以定义的相关操作包括：

1) **打开网页：**根据 URL 地址集合加载网页。URL 地址集合可以由多行组成，系统将循环的对每一行的 URL 对应页面进行操作和数据采集。URL 地址集合可作为最终生成服务的一个输入参数。

2) **选中元素：**选中 Web 页面中的元素，从而进行下一步的处理。

3) **点击元素：**点击 Web 页面上的任意类型元素，通常情况下，被点击的元素为超链接和按钮，但同时也包括被前端开发人员定义的能触发 JS 操作的其他 HTML 元素。

4) **输入文字：**在可输入的文本框中输入文字，常见于填写表单的操作中。每个输入文字操作的输入值将对应最终生成服务的一个输入参数。

5) **切换下拉选项：**对于可切换的下拉选项，切换到指定索引。每个下拉选项的索引值对应最终生成服务的一个输入参数。

6) **鼠标移动到元素：**将鼠标移动到特定元素以显示需要将鼠标移动过去才能够显示的页面内容。

7) **识别验证码：**识别 Web 页面登陆过程中需要填写的验证码。

8) **提取数据（采集元素）：**提取页面中指定位置元素的数据，数据类型可包含纯文本、超链接、图片地址等。每个提取数据步骤中要提取的元素值，都对应最终生成服务中的一个输出参数。

9) **选中全部元素：**选中与该元素同等类型的元素。如图 3.6 所示，在京东商品列表页中，当选中了商品列表块中第一个商品子块的商品名称对应的超链接元素（蓝色背景文本）后，其他商品子块的商品名称对应的超链接元素被检测到与第一个超链接属于同一类型的元素，因此它们被用蓝色边框标出。选中全部元素操作将会将所有同等类型商品链接同时加入已选中元素列表，并进行下一步的处理，如选中子元素操作或提取数据操作。

图 3.6 同类型元素匹配示例

10) **选中子元素：**选中一个子块内所有的元素，通常配合选中全部元素使用。如图 3.7 所示，商品列表块包含三个子块，当执行选中第一个子块元素→选中全部元素→选中子元素操作后，所有子块内的所有子元素全部被选中（蓝色框标识），并在右侧操作台中生成了要提取的数据字段参数（如参数 5_文本为商品的价格）。

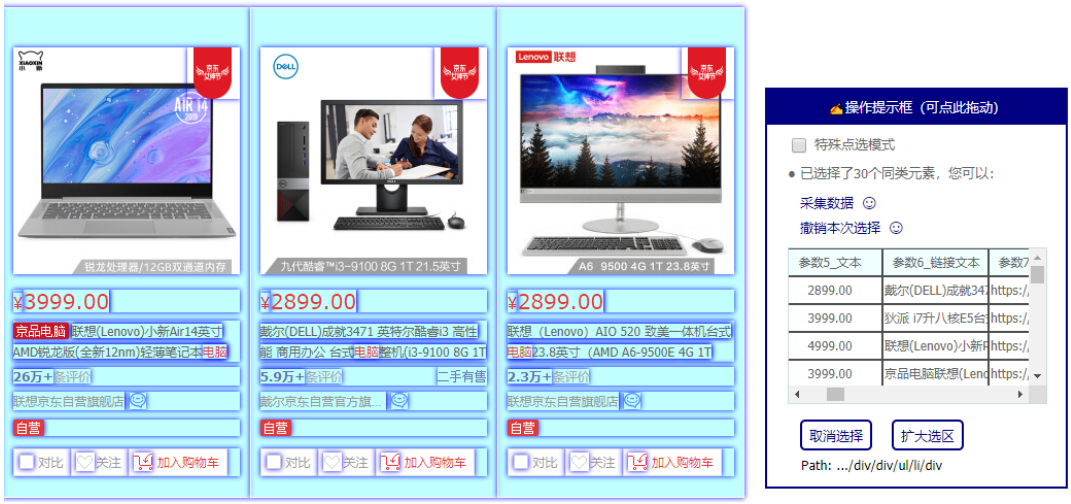


图 3.7 选中全部+子元素操作示例

11) **撤销选择：**取消选择上一次选中的元素。

12) **扩大选区：**将当前选中的元素改选为其父元素。该操作常用于当某元素无法被直接选中的情况（视觉上难以定位）。此时，可以先选择该元素的一个子元素，然后使用扩大选区功能选中该元素。图 3.7 中右下角的元素 Path 在执行扩大选区操作后，将会改为：*.../div/div/ul/li*。

13) **循环：**循环执行某些操作。在 Web 数据采集服务中，有许多操作需要循环执行，如循环点击下一页的按钮以实现翻页功能；循环点击图 3.6 中的每个商品链接进入商品详情页面以采集商品的详情数据；循环的在图 3.2 中的搜索框中输入不同的文字以搜索不同的商品等。使用循环功能，结合其他操作，可以极大的扩展服务流程的定义空间，实现自定义的 Web 数据采集。

14) **条件判断：**根据不同的条件执行不同的分支操作，每个条件判断操作内包含多个条件分支。条件判断适用于需要筛选信息的任务中，如在图 3.6 中可以设置条件判断，从而实现只采集带有京东自营商品图标的商品数据。

通过以上操作，可以让服务提供者自由的定义 Web 数据采集服务的操作流程，以满足多样化的 Web 数据采集需求。

同时，服务提供者还可以对服务流程中的各个操作节点进行增加、删除、剪切、复制以及修改节点参数的操作，同时提供服务基本信息，从而将服务保存到

服务注册中心。

最后，同第 3.2.2 节，服务提供者可以在服务注册中心中对自己生成的服务进行管理和维护。

2、服务请求者：同第 3.2.2 节，服务请求者以调用服务 API 的方式发起服务请求，等待系统自动执行 Web 数据的采集任务，并获得服务返回的结果。

3、服务注册中心：同第 3.2.2 节，服务注册中心负责对生成的服务信息，包括服务实现的业务逻辑进行保存和管理，以实现对服务的维护。

3.3.3 流程设计

复杂数据采集场景下，其服务生成阶段的执行流程的简单描述如下：

（1）服务提供者输入 Web 页面的 URL 地址，发起服务生成请求。

（2）服务提供者在服务流程定义模块中的 Web 页面上以鼠标点击的方式选中鼠标所在位置的元素，并在操作台中选择执行相应的操作，从而在服务流程管理模块中生成对应的操作节点。

（3）服务提供者在服务流程管理模块中对生成的操作节点进行处理，如剪切、复制节点等，同时修改操作节点中相关的参数，如元素定位的 XPath 等。

（4）重复步骤（2）（3）以定义复杂的 Web 数据采集流程。

（5）服务提供者输入服务名称和描述，保存服务。

复杂数据采集场景下，其服务调用阶段的执行流程的简单描述如下：

（1）服务请求者以 Web 请求的方式调用 Web 数据采集服务。

（2）系统根据 Web 请求中服务的 ID 号从服务注册中心的服务集合中找到相应服务，并得到服务流程中各可输入参数的信息。

（3）系统根据 Web 请求中的输入参数对服务操作流程中的各可输入参数的默认值进行修改，并生成一个任务 ID 号，返回给服务请求者。

（4）服务请求者将步骤（3）中生成的任务 ID 号以 Web 请求的方式传递给远程采集程序，或通过进程调用的方式将 ID 号传递给本地采集程序，从而开始 Web 数据的采集。

（5）系统根据服务请求者在步骤（1）Web 请求中设定的数据处理规则相关的输入参数，对提取到的数据进行过滤，并根据服务中的输出参数信息将数据以格式化的方式返回给服务请求者，完成数据采集任务。

3.4 智能化服务封装系统总体设计

面向 Web 应用的智能化服务封装系统简称“智能化服务封装系统”，其英文名为：OKAPI。OKAPI 为霍加皮（一种非洲鹿）的英文单词的复数表达形式，意为系统的服务运行和处理能力如非洲鹿反应速度一样高效，复数形式代表用户可用生成大量的 Web 服务，且具有服务内的各 API 均可正常运行（OK）的寓意。

图 3.8 展示了智能化服务封装系统的总体设计架构。

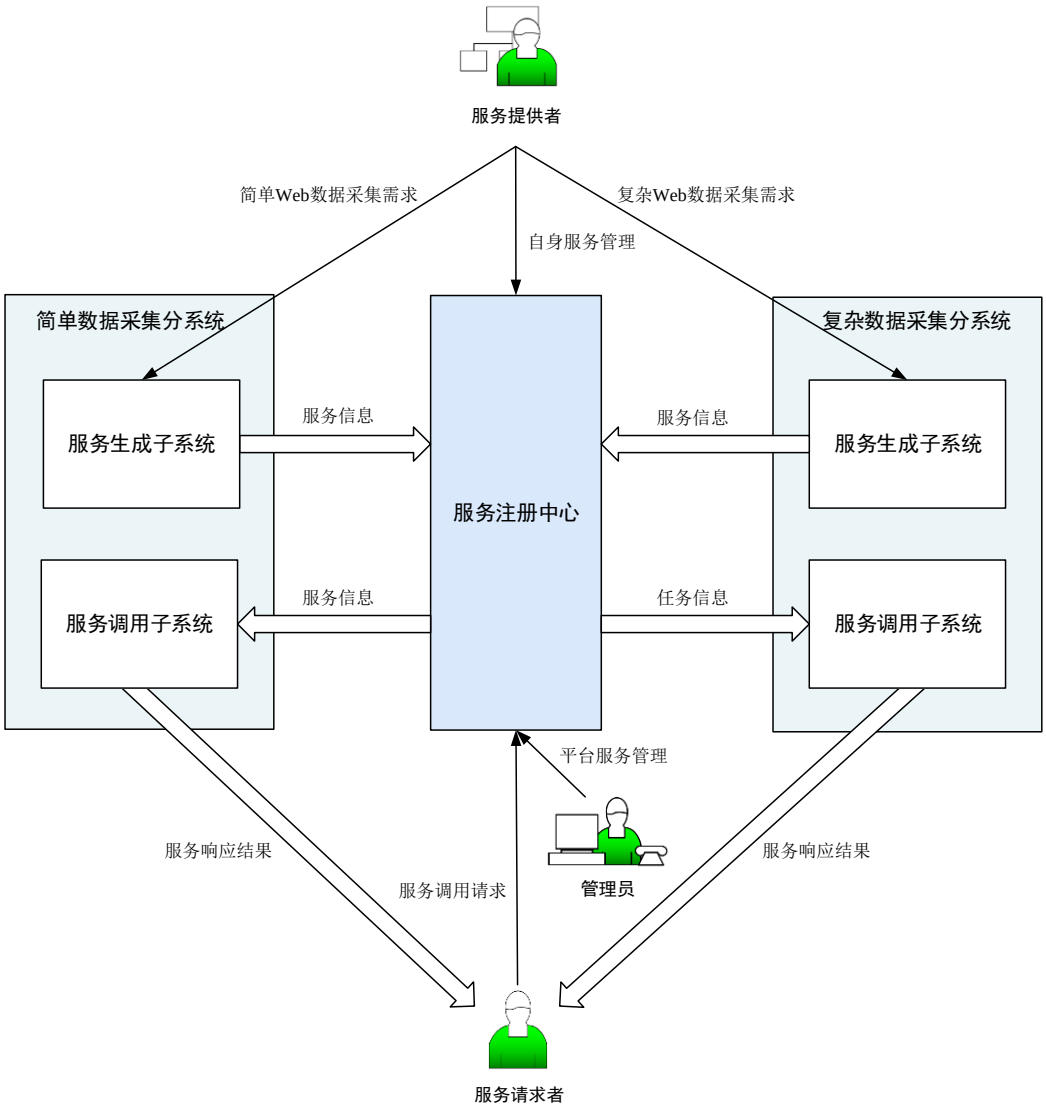


图 3.8 面向 Web 应用的智能化服务封装系统总体设计架构图

如图 3.8 所示，智能化服务封装系统由三部分组成：简单数据采集分系统、服务注册中心和复杂数据采集分系统。简单数据采集分系统和复杂数据采集分系统中均包含一个服务生成子系统和一个服务调用子系统。

服务提供者在定义 Web 应用数据采集服务时，需要根据自身对服务流程复杂度的需求（简单或复杂），在简单数据采集场景下使用简单数据采集分系统，或在复杂数据采集场景下使用复杂数据采集分系统中的服务生成子系统进行服务生成操作。服务生成子系统将生成的服务信息保存到服务注册中心，从而实现对生成服务的维护。服务提供者可对自身生成的服务进行管理，服务注册中心管理员可对服务注册中心中所有的服务进行管理。

服务请求者从服务注册中心中寻找想要的服务并以 Web 请求的方式进行服务调用，在提供服务输入参数后，服务注册中心会将相应服务的服务信息（简单数据采集场景）或任务信息（复杂数据采集场景）传递给对应场景分系统下的服务调用子系统，从而进行 Web 应用数据的采集。最后，服务调用子系统在服务流程执行完成后，将服务响应结果返回给服务请求者，完成服务调用。

系统同时负责对简单数据采集分系统、复杂数据采集分系统和服务注册中心内发生的各项操作行为进行监控和日志记录，以方便系统开发人员进行代码调试和项目维护。系统使用 MongoDB 数据库对服务注册中心中的服务进行持久化处理，同时，服务监控系统使用 Redis 进行缓存处理，并使用 MySQL 数据库对系统中产生的日志进行处理和维护。系统使用 Python 语言执行 Web 应用中的数据处理任务，在服务生成阶段，本文使用 JavaScript 语言，结合 Chrome 扩展开发技术，实现对服务生成流程的控制；最后，系统以 C# 语言和.Net Framework 作为载体，提供 WebSocket 服务，从而实现复杂数据采集分系统中各模块的消息传递。

本文中使用了高分服务网格平台的服务注册中心对高分相关 Web 应用的数据采集服务（如地理信息遥感服务）进行管理和维护，同时使用普通服务注册中心对普通 Web 应用的数据采集服务（如微信公众号文章提取服务）进行管理和维护。因此，任何服务注册中心平台的管理人员都可以使用智能化服务封装系统在自己的服务发布平台上发布 Web 应用数据采集服务，以方便用户快速的检索以及按类别整合 Web 应用的数据和资源，从而提高业务的执行效率。

3.5 本章小结

本章首先介绍了 Web 应用的基本架构，然后详细介绍了智能化服务封装系统的简单数据采集场景和复杂数据采集场景的概念和相关术语的定义，并对两个场景下用户的需求进行了分析，阐述了系统在不同场景下的服务生成阶段和服务调用阶段的执行流程。最后，本章对智能化服务封装系统的总体设计进行了介绍，加深了读者对系统功能模块的理解。

第4章 简单数据采集分系统的设计与实现

为了使读者更好的理解简单数据采集场景的功能和作用，本章针对简单数据采集分系统的整体架构进行了描述，展示了系统的服务生成阶段和服务调用阶段内部和两阶段之间的各项模块的操作关系。同时，本章对简单数据采集场景下的动态表单处理、静态页面分割、块排序等模块的实现细节进行了介绍，并对第三章没有涉及到的相关概念进行补充说明。最后，本章介绍了中国地震台网地震信息的采集案例，以描述系统的具体使用方式和操作流程。

4.1 系统整体设计架构

简单数据采集分系统的整体设计架构如图 4.1 所示。

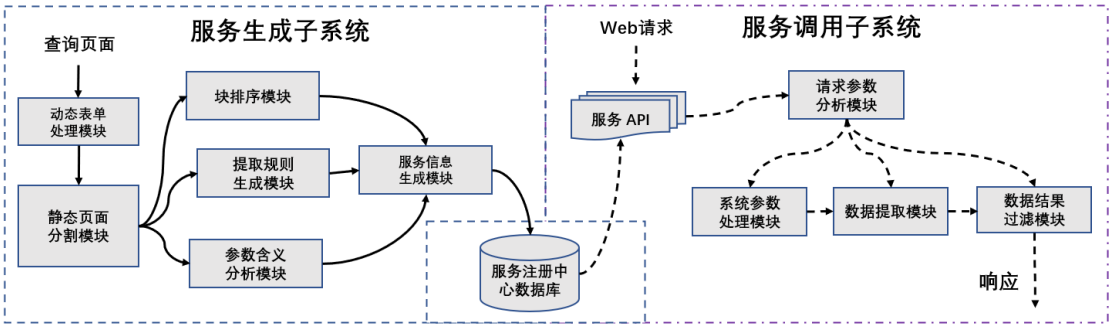


图 4.1 简单数据采集分系统整体设计架构图

图 4.1 中的各模块之间相互配合，以完成图 3.2 中的服务生成和服务调用流程。服务提供者提供查询页面给服务生成子系统的动态表单处理模块进行处理，并最终由服务信息生成模块生成服务并存储到服务注册中心；在服务调用阶段，服务请求者以 Web 请求的方式调用服务提供的 API，并由系统自动的执行 Web 数据的采集流程，并将采集结果作为服务响应返回给服务请求者。

4.2 系统关键技术与关键模块设计

4.2.1 服务生成子系统相关模块设计与实现

4. 2. 1. 1 动态表单处理模块

动态表单处理模块负责检测一个查询页面中存在的所有可填写表单，根据用

户提供的参数示例值对表单进行填写和提交，并记录下表单中的输入参数信息。

在 HTML 源码中，表单通常以<form>节点的形式存在，其中的输入框的常见类型包括：文本输入框、（下拉）选择框、单选框和多选框，它们在 HTML 源码中以<input>标签和<select>标签的形式存在。而表单的查询按钮通常以<button>标签或<input type='button'>标签的形式存在。随着现代前端开发技术的发展，许多前端开发人员选择使用自定义的节点类型设计表单，并使用 JS 动态提交表单，如使用 Ajax 技术提交以非<form>形式存在的表单，这一定程度上增加了表单检测的难度。然而，我们发现，在大多数 Web 页面中，其内部表单都包含以下特性：

1、表单包含<input>标签形式的输入框。

2、表单中的提交按钮绑定了 JS 的相关事件，具有 *EventListener* 属性。

因此，我们可以利用以上特性对页面中存在的表单进行查找和定位。同时，许多 Web 页面使用<iframe>标签在页面中嵌入另一个页面，这种页面的元素是无法直接访问的，因此需要做特殊的嵌入框架的处理。

对一个 Web 页面中执行表单检测的算法执行步骤如下：

（1）遍历当前 Web 页面的 DOM 树，找到其中所有的<form>标签对应的节点，包括在<iframe>标签中的节点。

（2）对于步骤（1）中的每个表单，遍历表单中的元素，并记录下所有可填写的输入框和提交按钮。

（3）除去步骤（1）中找到的表单，以 input[type=submit], input[type=button], <button>, <a>, <img>的顺序遍历其余 HTML DOM 树下的节点，并记录下绑定了 *Click EventListener* 属性的元素。

（4）如果步骤（3）中记录的元素所在的父节点下存在输入框，则将其父节点标记为一个表单。

表单检测算法执行结束后，模块将检测到的所有表单信息展现给用户等待选择，在用户选择其中一个表单并提供输入示例值后，模块将按照示例值填写和提交表单，并将表单信息和用户提供的参数信息一同记录下来。表 4.1 给出了表单提取模块中记录下的表单及操作信息的一个示例片段。

表 4.1 表单提取模块信息示例

```
{
    "url": "https://www.se.com",
    "main_form_index": -1, //要选择的表单索引，-1 为不选择表单
    "forms": //记录了页面内所有的表单信息
    [{
```

表 4.1 表单提取模块信息示例（续）

```
        "id": 0,   "XPath": "/html/form", "main_btn_index": 0,   //提交按钮索引
        "input_list": //输入参数列表
        [{
            "id": 0, "type": "text", "name": "start_time", "value": "2020-01-09",
            "XPath": "/html/form/input", "index": "T1"
        }],
        "query_button_list":   //提交按钮列表
        [{"id": 0, "type": "a", "index": "b1", "XPath": "html/form/a[0]",}]
    }]
}
```

4.2.1.2 静态页面分割模块

静态页面分割模块负责将一个给定的数据页面使用页面分块算法分割成多个语义不同的块。由第 3.2.1 节块和子块的定义可知，在页面分割完成后，各个块彼此之间的语义不同，而块内的各个子块的结构相同。

本文设计了一个基于 HTML DOM 的语义分块算法，如算法 1 所示。

算法 1 Web 页面分块算法

输入: 页面 HTML DOM 树的根节点 E_R .

初始化: $Q \leftarrow \{E_R\}, i \leftarrow 0$
while $Q[i]$ **is not** $null$ **do**
  $E_t \leftarrow Q[i]$
  $i \leftarrow i + 1$
  $C_t \leftarrow \text{CHILDS}(E_t)$
  if $E_t.type == "iframe"$ **then**
    $\text{SWITCH\_TO}(E_t)$
    $C_t \leftarrow \text{ROOT}(E_t)$
  else if $E_t.\text{type} == "th"$ **then**
    $\text{CHANGE\_TYPE}(E_t, <td>)$
  end if
  if $same(E_t, E_t{}^L)$ **or** $same(E_t, E_t{}^R)$ **then**
    $\text{MARK}(E_t)$
  else
    $Q \leftarrow Q + C_t$
  end if
end while

在算法 1 中，E_R是 HTML DOM 树的根节点；Q是一个搜索用的队列；*Child*s 方法负责从指定节点元素中获得其所有子节点（不包括孙节点）；*Root* 方法负责拿到一个<*iframe*>节点对应页面中 HTML DOM 树的根节点；*switch_to* 方法负责

将当前浏览器操作的上下文从当前页面切换到指定 iframe 页面；*change_type* 方法负责改变一个 HTML 标签的标签类型；*same* 方法负责比较给定的两个元素对应的 DOM 树的结构是否相同，最后，*mark* 方法负责将 DOM 树中的指定元素节点标记为一个子块。

如算法 1 所示，我们首先将 Web 页面的根节点E_R加入搜索队列Q，并设置Q的开始索引号i为 0。接下来，我们使用广度优先搜索的方式对 HTML DOM 树中的每一个节点进行遍历。如第 12 到 16 行所示，在每次循环中，若当前节点E_t的结构和其左右兄弟节点相似，我们就将E_t标记为一个子块；否则，我们就将E_t的所有子节点集合C_t放入搜索队列Q尾部进行下一步的搜索。第 6 到 8 行，我们保证了对<iframe>标签中含有的元素的处理；第 9 到 10 行将<th>标签类型替换成了<td>标签，保证表格检索时的一致性。

接下来我们介绍 *same* 方法的实现原理，$same(E_1, E_2)$负责比较两个元素E_1和E_2对应结构是否相同。我们对E_1和E_2对应的元素 DOM 子树进行遍历，并分别将遍历到的每个节点对应的标签类型放入有序数组S_1和S_2中，遍历结束后，如果S_1和S_2完全相同，则E_1和E_2结构相同，否则，结构不同。

算法 1 执行完成后，Web 页面中存在的所有子块将会被标记，此时，我们按照标记值，将所有具有相同结构的子块的父节点标记成一个块，如某个子块找不到和它具有相同结构的其他子块，则将其单独划分成一个块。如页面中划分了四个子块，其中前三个子块具有相同的结构，则将前三个子块的父节点标记为块 1，将第四个子块标记为块 2。注意，算法 1 只能将页面中结构完全相同的节点标记为同样的子块，而当出现如第 3.3.2 节图 3.6 所示的情况，即两个子块中有细微元素的差别（有的不包含自营图片，有的包含）时，算法 1 将不把两个子块合并成一个块。此时，需要使用复杂数据采集场景下的子块识别算法进行操作。

算法 1 基于 HTML DOM 子树进行操作，成功实现了对 Web 页面的分割。由于算法 1 不是基于视觉元素进行的页面分割，因此其不受元素是否可见的影响，可实现对页面中暂时不可见的元素的分割和处理，体现了算法的适应性、广泛性。

4.2.1.3 块排序模块

块排序模块负责将分割好的块按照重要程度进行排序，使得用户可以快速定位到自己想要选择提取的块。尽管 Web 页面的表现形式多种多样，我们也可以找到通常情况下，这些页面中的重要块的共有特征，如：

1、重要程度高的块中的子块数量较多，如京东商品列表页面中商品块中子块

的数目最多。

2、重要程度高的块中文本的数目较大，如新闻类页面的正文块字数最多。

3、重要程度高的块所占的页面面积最大。

针对以上三个特征，本文设计了一个页面块排序算法，从而保留页面中最重要的 *n* 个块（*n* 由用户指定，如 10）。其步骤如下：

（1）根据页面中分好的所有块中的子块数量、文本数目和所占面积，将排序结果以降序方式存入三个数组中，分别表示为：*BlockList*, *BlockLen*, *BlockSize*。

（2）保留步骤（1）中三个数组的前 *2n* 个块，并删除剩余的块。

（3）取三个数组的交集，将其结果放入数组 *Candidates* 中，并保留 *Candidates* 数组中的前 *n* 个块。

以上算法通过较为简单的方式，满足了大部分场景下的块排序需求，并弥补了只通过单个指标来进行块重要度排序算法的短板。如页面中通常含有的广告块占有的面积一般较大，但通常内部文字较少，因此，算法不会将广告块的重要度上调。

图 4.2 展示了对豆瓣电影排行榜页面进行分块并排序后的效果，如图可知，我们认为最重要的电影内容块被选择为第 1 块，而右侧的各类排行榜所属块分别被排到第 2,3,4 块，满足了排序要求。注意，由于对哪个块是重要的块的判断具有较大的主观性，因此，本模块的排序结果只是给出用户一种建议，真正选择哪一块进行提取，还需要用户自行选择。块排序算法的准确率将在第 6 章中讨论。

图 4.2 豆瓣电影排行榜页面分块和排序效果图

4.2.1.4 提取规则生成模块

提取规则生成模块负责生成 Web 数据的提取规则，供服务调用阶段提取数据使用。在一个 Web 页面中，用户可能提取的数据的主要类型包括：

1、文本：最常见提取类型，文本信息存在于 HTML 文档中的任何标签中。

2、超链接：超链接以<a>标签的形式存在与 HTML 文档中，在实际的采集需求中，超链接的文本和地址往往会被同时采集，其地址存在于 *href* 属性中。

3、图片地址：在 Web 页面中，图片地址存在于<img>标签的 src 属性和 CSS 中的 *backgroundImageURL* 属性中。

4、输入框中的输入值：在某些含有大量表单的 Web 页面中，用户往往需要采集输入框中的已存在值。如学生管理系统信息采集服务中学生的基本信息通常存在于 Web 页面<input>标签的 *value* 属性中。

5、OuterHTML 和 InnerHTML：对于一些特殊的需求，用户可能需要直接将当前节点的 HTML 源码提取出来并进行下一步的处理，如获得 HTML 元素标签中绑定的特殊属性值。此时，系统提供了直接提取当前元素 HTML 源码的功能。OuterHTML 和 InnerHTML 分别代表元素本身和其内部元素的 HTML 源码，如某 标签的 OuterHTML 为：

* start example end *，

则其 InnerHTML 为：

start example end。

Web 数据的提取规则中常见的项包括：ID、元素类型、元素位置、提取的数据类型等。表 4.2 给出了提取规则生成模块生成的数据提取规则的一个示例片段。

表 4.2 Web 信息提取规则示例

```
[
  { "id": 0, "nodeType": "h3", "XPath": "/html/p[0]/h3", "dataType": "text" },
  { "id": 1, "nodeType": "img", "XPath": "html/p[4]/img", "dataType": "backgroundImage" },
  {"id": 2, "nodeType": "a", "XPath": "html/p[2]/a", "dataType": "OuterHTML",},
]
```

4.2.1.5 参数含义分析模块

参数含义分析模块负责给输出参数一个初始的名称，以简化用户修改输出参数名称的工作量。在某些 Web 应用中，系统可根据 Web 页面中的表格标签的表头信息来确定输出参数的名称，如 Hancock B 等人[54]使用序列神经网络模型来生

成 HTML 表格中的标题。但大多数情况下，Web 页面不包含参数名称的显式定义，因此参数名称通常需要用户自行设定。系统默认生成名称的格式为：参数_ID，这种形式不能直接表达参数的实际含义，因此，如 Novgorodov S 等人[55]使用深度提取模型从商品评论中生成商品的描述一样，参数含义分析模块负责分析参数的示例值，从而给参数一个具有合理语义的参数名称，如某参数的所有示例值均为 11 位数字时，参数的名称即被设置为：*手机号码*。

本文使用知识图谱作为工具来分析输出参数示例值的可能含义，并给出参数的名称。知识图谱（Knowledge Graph）[56]是一种揭示实体之间关系的语义网络，图 4.3 展示了一个表示国家信息各实体之间关系的知识图谱可视化网络模型。知识图谱中的知识以三元组的形式存储，一个三元组的表示如下：

（实体 1，关系，实体 2），

如三元组：（阿里巴巴，竞争对手，腾讯）表达了实体“阿里巴巴”的实体“腾讯”之间的关系为“竞争对手”。

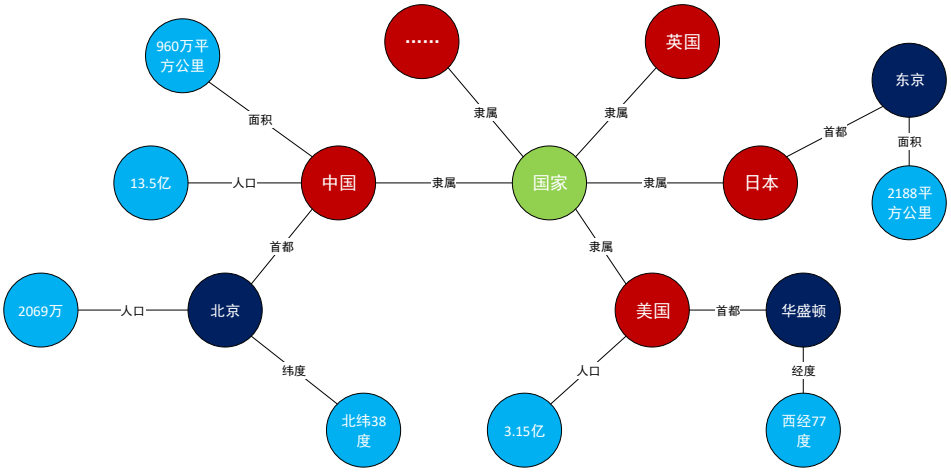


图 4.3 国家信息知识图谱示例

三元组的常用表示还包括：

（实体，属性，属性值），

上述三元组表达了实体的相关属性值。

openKG.cn 是一个开放的中文知识图谱库，其中包含了各行各业中各类型的知识图谱的资源，如常识知识图谱、城市知识图谱等，本文中使用了其中的*中文通用百科*知识图谱数据集对参数含义进行分析。该数据集包含 900 万+的百科实体以及 6700 万+的三元组关系，其标签信息数量为 1980 万+，适合作为参数含义分析的基础数据集。我们在该数据集上对参数的示例值做以下的三元组预测：

（参数示例值，类型，？），

即推测参数示例值可能的类型，由于服务中的一个输出参数可能存在多个示例数据，因此，我们对这多个数据同时进行类型预测，并将预测最多的类型作为参数的默认名称使用。

4.2.1.6 服务信息生成模块

服务信息生成模块负责将服务的信息，包括服务的名称、描述，以及由其他模块生成的输入输出参数信息和数据提取规则整合成一个文档，上传到服务注册中心。服务的信息将被存储到服务注册中心的数据库中，供服务请求者使用。

4.2.2 服务调用子系统相关模块设计与实现

服务调用子系统中各模块的具体实现较为简单，服务请求者在发起服务调用请求后，全部的 Web 数据采集流程均在服务器端实现，并最终将采集到的数据返回给服务请求者。

4.2.2.1 请求参数分析模块

请求参数分析模块负责分析 Web 请求中的各项参数，并按照类型将参数分配给不同的处理模块进行处理。我们将 Web 请求中的参数分为以下三类：

1、系统级别请求参数：系统为所有的 Web 数据采集服务提供的公共的参数，如服务调用所需的 API Key，最大翻页的次数，最大超时时间等。

2、应用级别请求参数：即服务提供者在服务生成阶段定义的输入参数。

3、过滤参数：有些时候用户需要对服务返回的数据进行过滤处理，以满足定制化的数据处理需求，因此，系统提供了过滤参数实现对数据进行定制化的过滤和处理。

4.2.2.2 系统参数处理模块

系统参数处理模块负责对 4.2.2.1 节中提到的系统级别请求参数进行处理，各项参数对应的处理逻辑由服务提供者在服务注册中心中定义和实现。如：当用户提供的 API Key 正确时，继续服务调用流程，否则，停止服务调用。如果服务需要收费，还将进行扣费等操作。系统参数处理模块可以动态的进行扩展，以满足服务提供者针对 Web 数据采集任务之外的各项功能的需求。

4.2.2.3 数据提取模块

数据提取模块是服务调用子系统的核心模块，负责复现用户的操作行为并提取数据，即：打开查询页面，根据服务请求者提供的应用级别请求参数进行表单填写，提交表单并获得数据页面。在数据页面中根据提取规则生成模块定义的数据提取规则对 Web 数据进行指定类型的提取，并将结果提交给数据结果过滤模块进行数据过滤。

数据提取模块使用 Python 语言，搭载 Selenium Webdriver 库实现。模块可以模拟普通用户浏览网页的行为并对表单进行填写和提交；在数据提取阶段，Selenium Webdriver 可通过 XPath 定位的方式定位到元素，然后调用中 *getText* 方法获得元素的文本数据。同时，针对一些特殊的数据类型，如元素的 OuterHTML，Selenium Webdriver 可以控制浏览器执行 JavaScript 脚本（如：*return document.getElementbyTagName("body").innerHTML*）来获得这些数据。

4.2.2.4 数据结果过滤模块

数据结果过滤模块负责按照 4.2.2.1 节中定义的过滤参数对采集到的数据进行过滤和格式化的处理。例如，如果某个输出参数为：*price*，表示商品的价格，则服务请求者可以通过在 Web 请求中提供请求参数：*filter!price!larger=35* 来提取所有商品价格高于 35 的数据，并将商品价格在 35 及以下的数据条目过滤掉。

数据结果过滤模块提供的数据过滤操作包含：数据值大于某值，数据值小于某值，数据值等于某值，数据中包含某值等。在对采集的数据进行过滤之后，模块将会把最终的处理结果返回给服务请求者，完成 Web 数据采集流程。

4.3 案例分析

本节以中国地震台网的历史查询页面为例，描述了简单数据采集分系统的服务生成和调用过程。中国地震台网的历史查询页面包含一个地震信息的搜索表单，可以按照如时间和经纬度的输入指标来查询地震的基本信息。在提交表单后，页面下方将会以表格形式出现地震信息的查询结果，因此，对该页面的地震信息进行采集，符合简单数据采集场景的页面结构和操作流程的定义。

本案例想要查询固定时间段内的地震信息。**其服务生成流程如下：**

（1）服务提供者在系统中输入中国地震台网历史查询页面的 URL：*http://www.ceic.ac.cn/history*，并点击开始按钮发起服务生成请求。

（2）如图 4.4 所示，系统定位到了页面的表单结构，并自动生成了 10 个输

入参数，分别从 T1 到 T10，并且定位到了一个提交按钮 B1。

图 4.4 中国地震台网历史查询页面表单定位示例

（3）如图 4.5 所示，用户选择要输入的表单编号，修改输入参数并提供样例值。

图 4.5 表单选择与参数信息修改示例

（4）如图 4.6 所示，系统根据用户提供的表单索引和输入参数的样例值，填写相应的表单并提交，从而得到数据页面，对页面进行分块和块排序后，将块信息返回给用户。

图 4.6 页面分块和排序结果示例

（5）与图 4.5 类似，用户选择自己想要提取的数据块，这里是块 1，并配置服务的输出参数（如震级）以及服务 API 描述，提交系统后台。

（6）系统后台根据用户指定的配置生成了中国地震台网历史查询服务，服务信息中同时包含了表单定位的信息以及数据提取的规则。因为地震信息为高分相关信息，因此，我们将生成的服务保存到高分服务网格平台，图 4.7 展示了在高分服务网格平台中生成的地震信息采集服务的相关信息。

中国地震台网——历史查询

服务描述: 中国地震台网——历史查询

API : http://localhost:8000/call_service/78

Method : HTTP GET

Example : http://localhost:8000/call_service/78?__max_page=1

输入参数

Name	Type	Required	Example	Description
__max_page	int	yes	1	System Request Parameter
time_start	date	yes	2019-01-03	开始时间

输出参数

Name	Type	Example	Description
震级(M)	text	3.8	地震的级别，单位 M
发震时刻(UTC+8)	text	2018-11-24 04:41:10	地震时间

图 4.7 中国地震台网历史查询数据采集服务信息（省去了部分参数）

本案例使用 Web 请求中的 GET 方式调用生成的服务，其过程如下：

（1）如图 4.7 所示，服务的 API 地址为：*http://localhost:8000/call_service/78*，假设调用者想要查询从 2020 年 1 月 1 日到 31 日的地震数据，则调用 URL 为：

http://localhost:8000/call_service/78?start_time=2020-01-01&end_time=2020-01-31&__max_page=3&api_key=a1f6de13

其中，*start_time* 与 *end_time* 为服务提供者在服务生成阶段定义的参数名，这里对应图 4.4 的 T1 和 T2 参数，即开始时间和结束时间；*__max_page* 是系统提供的输入参数，表示最大采集的页数，这里为 3 页；*api_key* 为用户提供的调用服务所需要的密钥，由服务提供者在系统中购买服务后得到。

（2）服务调用后台收到 Web 服务请求后，会根据请求的服务 ID 号查找服务信息，然后在控制的浏览器实例中打开历史查询页面，按照请求参数中的时间值填写并提交表单，得到相应的地震信息数据页面。

（3）服务调用后台按照提取规则，提取页面表格中的数据，并自动点击翻页

```
[{
        "record_id": 0,
        "发震时刻": "2020-03-07 13:43:46",
        "深度（千米）": "10",
        "参考位置": {
            "文本": "瓦努阿图群岛",
            "链接地址": "http://news.ceic.ac.cn/CC20200307134347.html"
        }
    },
    {
        "record_id": 1,
        "发震时刻": "2020-03-07 11:52:02",
        "深度（千米）": "10",
        "参考位置": {
            "文本": "墨西哥",
            "链接地址": "http://news.ceic.ac.cn/CC20200307115203.html"
        }
    }
]
```


图 4.8 服务调用返回 JSON 样例

按钮采集下一页的数据，在 3 页数据全部提取完成后，系统将提取结果以 JSON 格式返回给用户，如图 4.8 所示。

4.4 本章小结

本章介绍了简单数据采集分系统的设计架构图与具体实现方案，详细阐述了服务生成和调用子系统中各个模块内部相关算法的实现。最后，本章通过中国地震台网的地震信息采集案例，描述了系统的功能和具体使用方式。

第5章 复杂数据采集分系统的设计与实现

为了使读者更好的理解复杂数据采集场景的功能和作用，本章针对复杂数据采集分系统的整体架构进行了描述，展示了系统中服务生成子系统、服务注册中心和服务调用子系统内各项模块之间的操作关系。同时，本章对复杂数据采集分系统中的关键技术点进行了介绍，包含 Web 流程定义、同类型元素匹配和服务流程执行等算法的实现以及使用到的相关数据结构，并对第三章没有涉及到的相关概念进行补充说明。最后，本章通过 58 同城房源信息采集案例，展示了系统的使用方式和具体操作流程。

由于简单数据采集分系统的部分模块和操作同样适用于复杂数据采集分系统，因此，在第四章描述过的相关模块和功能，本章将不再赘述。

5.1 系统整体设计架构

图 5.1 展示了复杂数据采集分系统的整体设计架构。

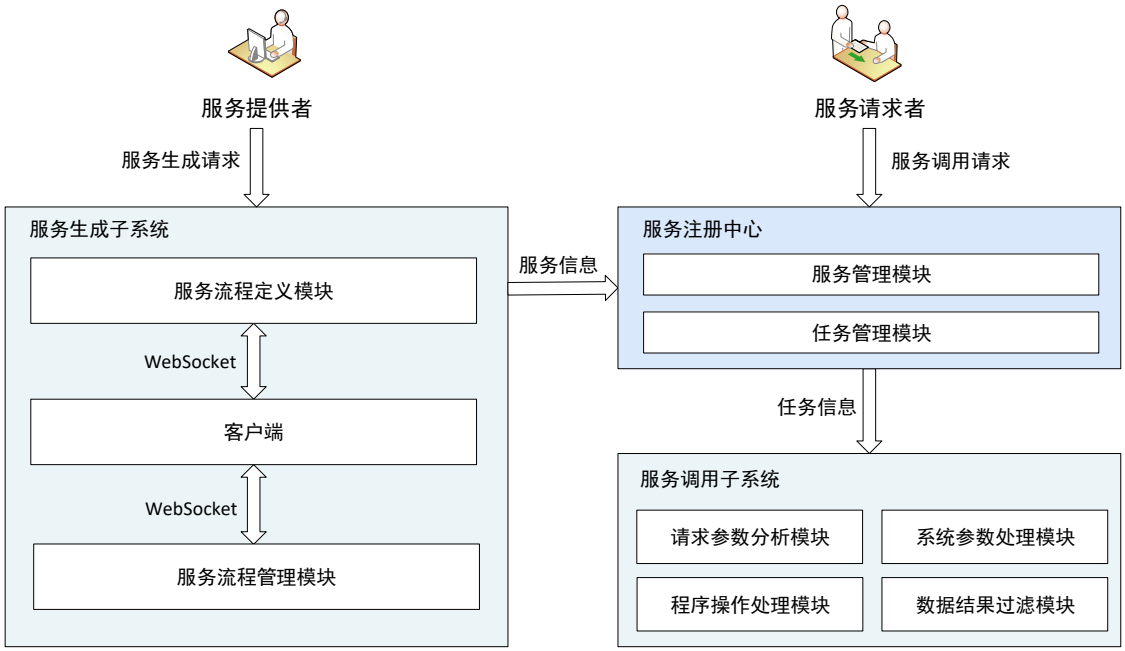


图 5.1 复杂数据采集分系统的整体架构图

由图 5.1 可知，复杂数据采集分系统的服务生成子系统中，服务流程定义模块和服务流程管理模块之间由一个客户端作为中介，使用 WebSocket 技术进行消息传递和交互。相比于简单数据采集分系统，这里的服务调用子系统中新增了程

序操作处理模块对用户自定义的操作流程进行解析，如点击元素，提取数据和循环、判断等操作，并根据各操作的配置参数复现这些操作。

接下来，本文将对第四章中没有涉及到的系统中的主要模块的功能和实现细节进行介绍。

5.2 服务流程定义模块设计与实现

由第 3.3.1 节可知，服务流程定义模块是一个加载了定制化的 Google Chrome 扩展的浏览器，可供服务提供者可视化的对 Web 页面执行第 3.3.2 节中描述的各种操作。服务流程定义模块中使用的扩展程序的操作架构如图 5.2 所示。

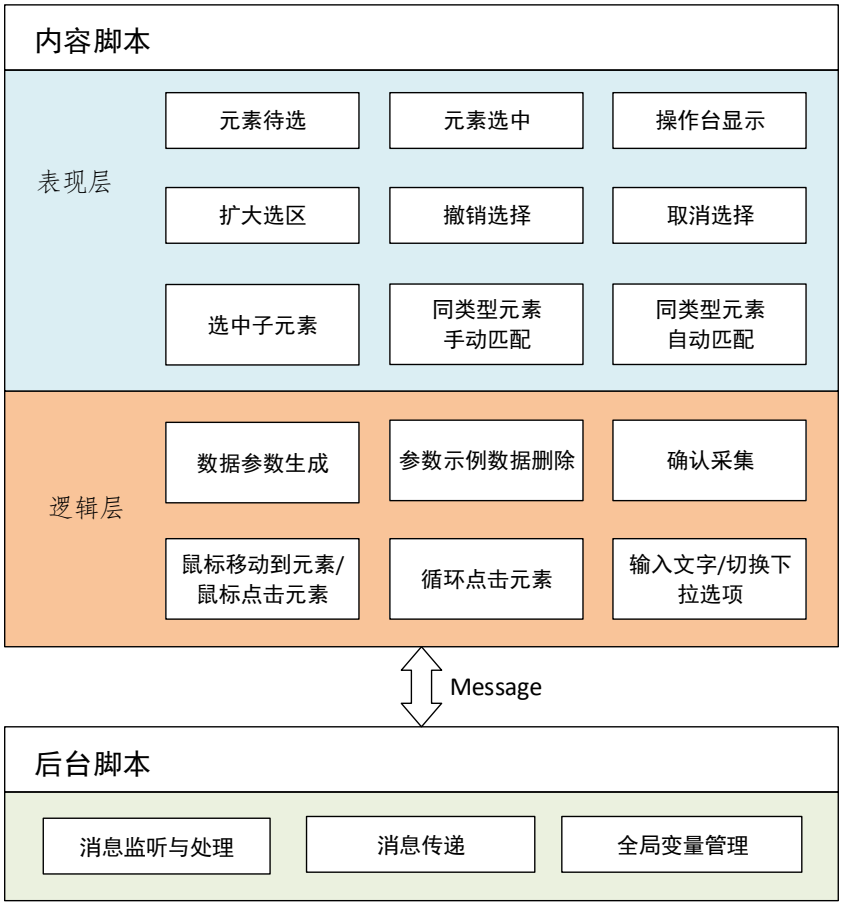


图 5.2 服务流程定义模块操作架构图

由图 5.2 结合第 2.2.3 节所述的 Chrome 扩展开发技术可知，服务流程定义模块的扩展程序由内容脚本和后台脚本两个组件组成，两组件之间通过扩展提供的方法进行消息传递。内容脚本将操作台嵌入到浏览器中加载的任意页面中，并提供各种操作的执行方法；后台脚本负责维护一个全局的环境，如记录当前已经添

加的参数的数目；同时，后台脚本负责和整个操作流程中任意页面内的内容脚本进行交互，并将需要记录的操作信息通过 WebSocket 传递给客户端，并最终由客户端传递给服务流程管理模块。最后，后台脚本监听来自客户端的消息，从而进一步的通知相关 Web 页面中的内容脚本执行相应的操作。

5.2.1 内容脚本表现层设计与关键实现

由图 5.2 可知，内容脚本由表现层和逻辑层两部分组成。表现层负责实现对界面的各种可视化操作，逻辑层则负责处理相关操作的底层逻辑。本节将对内容脚本表现层中涉及到的各个方法的功能，以及相关算法的实现细节进行介绍。

5.2.1.1 功能介绍

内容脚本的表现层中包含的方法有：

1) 元素待选：元素待选指用户在 Web 页面中移动鼠标时，鼠标所在的位置对应的元素的背景颜色会变成指定的颜色 A，以辅助用户快速的定位到自己想要选择的元素；当鼠标移开时，元素的背景恢复成原来的颜色。

2) 元素选中：当元素被选中时，元素的背景颜色变为指定的颜色 B，以提示用户哪些元素被选中，被选中元素的背景颜色在取消选中之前不会发生变化。

3) 操作台显示：在 Web 页面中嵌入操作台，并绑定相关的处理逻辑供用户进行操作。同时，该操作台可被拖动到页面中的任意位置。注意，在操作台中，屏蔽了元素待选和元素选中的功能。

4) 扩大选区：扩大当前元素的选区。

5) 撤销选择：撤销最后一次选中的元素。

6) 取消选择：取消所有选中的元素。

如图 5.3 所示，鼠标当前位置为新闻标题，因此新闻标题元素的背景颜色变为淡紫色；下方的新闻发布时间，来源，作者以及查看人数四个元素的背景颜色为天蓝色，表示四个元素已被选中。当鼠标从标题移动到发布时间时，标题背景将恢复成白色，而发布时间的背景颜色不变。当执行撤销选择操作后，最后一次被选中的元素会被取消选择；当执行取消选择后，所有被选中的元素将被取消选择；当执行扩大选区操作后，最后一次选中的元素将会变为其父元素。

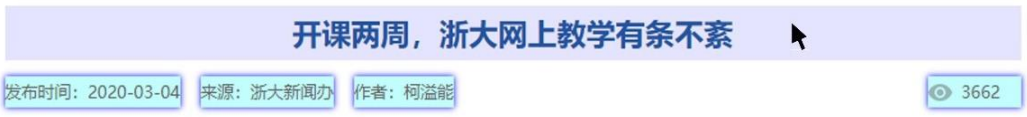


图 5.3 元素待选和元素选中示例

表现层的方法还包括：

7) 选中子元素：选中当前所有选中的元素内的所有子元素，并进行标识。

8) 同类型元素自动匹配：在没有元素被选中时，任意选中页面中的一个元素，系统将会自动检测到与被选中元素同类型的元素，并将这些同类型的元素以蓝色框标记，以方便用户进行“选中全部元素”操作，节省用户点选元素的时间。

9) 同类型元素手动匹配：在某些情况下，由于元素的层级关系较为复杂，而同类型元素自动匹配方法遵循就近优先的原则并没有将和已选中元素同类型的元素全部选中，因此需要用户手动选中尚未被标记的同类元素，系统此时将会进行再一次的同类型元素检测，扩大检测范围以检测到与当前所有选中的元素同类型的元素，并做好标记向用户展示。

图 5.4 展示了淘宝网站首页的商品类别的超链接元素列表，用户首先点击了第一行第一个元素“女装”，则如 A 图所示，根据同类型元素自动匹配算法，匹配到了“男装”和“内衣”两个元素；此时，用户选择“选择全部元素”操作，则如 B 图所示，系统选中了第一行的所有元素；然后，用户再次点击第二行第一个元素“鞋靴”，则如 C 图所示，系统根据同类型元素手动匹配算法匹配到了所有行中的所有超链接元素；最后，用户再次进行“选择全部元素”操作，即如 D 图所示，所有行中的所有商品超链接被全部选中。

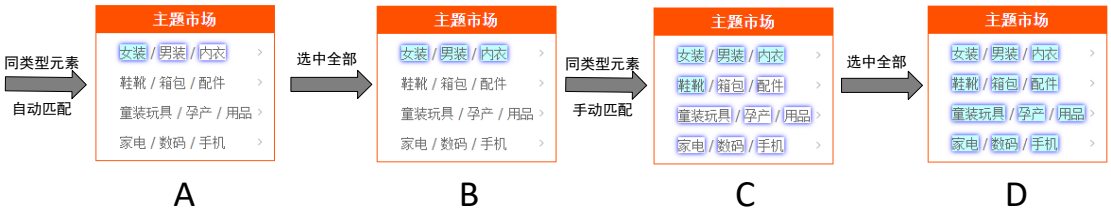


图 5.4 同类型元素自动和手动匹配方法示例

5.2.1.2 元素选中相关操作实现

系统通过对 Web 页面的根节点的鼠标移动事件进行监听，实现了实时监控鼠标移动变化并得到当前鼠标所在位置元素的功能。通过 JavaScript 的 *createElement* 方法，实现了动态的在页面中创建操作台并将其插入到当前 Web 页面中的功能，并通过 jQuery[57]中的 *drag* 方法，实现了对操作台的拖动功能。

在 Web 页面中的元素被选中后，我们需要对元素的基本信息进行记录，以实现对页面元素的操控。在本系统中，一条元素的信息包含以下主要字段：

1) *node*：元素的对应的 DOM 节点，可直接通过 JS 操作。

2) ***XPath***：元素的 XPath。

3) ***styles***：元素原本的样式，如背景颜色和边框颜色。

4) ***step***：元素被添加时的步数，以实现撤销操作。

系统通过维护数组 *elementList* 来记录已经选中元素的信息，同时，维护数组 *readyList* 来记录与已选中元素同类型的元素信息，并准备在下一步将它们放入 *elementList* 中。

算法 2 展示了元素选中相关的部分操作的实现方法，以展示对元素信息对应变量的基本操作方式。

算法 2 表现层元素选中相关操作实现

输入： 元素信息列表 $elementList$, 上一次的待选元素信息 $lastElement$, 鼠标当前位置对应的元素信息 $nowElement$, 操作台元素 $toolKit$, 当前操作步数 $step$, 待选元素背景颜色 $readyColor$, 选中颜色背景颜色 $selectedColor$.

```
//元素待选处理函数
function elementWait(nowElement)
    //如果鼠标当前位置对应元素为操作台中的元素，不进行任何操作
    if nowElement in toolKit then
        return
    end if
    //如果鼠标当前位置对应元素已被选中，不进行任何操作
    if nowElement in elementList then
        return
    end if
    //如果上一次的待选元素已被选中，不进行任何操作
    if lastElement in elementList then
        return
    end if
    //恢复上一次的待选元素的原始样式，并将当前元素的样式改变
    lastElement.node.styles = lastElement.styles
    nowElement.node.styles = readyColor
    //将当前元素标记为上一次的待选元素
    lastElement ← nowElement
end function

//选中元素操作
function selectElement(nowElement)
    //元素没有被选中过才去选中
    if nowElement in elementList then
        return
    else
        elementList ← elementList + nowElement
        nowElement.node.styles = selectedColor
```

```
    end if
end function

//撤销选择操作
function revokeSelect
    tstep ← step
    step ← step - 1
    //删掉最近一次选中的所有元素
    while tstep == elementList[-1].step do
        elementList[-1].node.styles = elementList[-1].styles
        elementList ← elementList - elementList[-1]
    end while
end function
```

算法 2 中的对象属性以非斜体表示，由算法 2 可知，鼠标移动事件发生时，*elementWait* 方法被执行以处理元素待选行为，如果出现鼠标移动事件被屏蔽的情况，我们还可以使用特殊点选模式，即在元素上方加入遮罩层来实现元素选择；每次鼠标按下时，*selectElement* 方法会被执行以完成选中元素的操作；每一次对元素的操作都涉及到对其样式的修改和恢复，以达到最佳的展示效果。

其他的元素选中相关的操作，如扩大选区，取消选择等，实现方式与算法 2 中的函数类似，即通过对 *elementList* 数组中的元素进行替换和清除，并实时的处理被替换和清除的元素的样式来实现相关功能。

5.2.1.3 同类型元素匹配算法实现

本节结合具体示例来分析同类型元素自动匹配和手动匹配算法的实现。使用同类型元素匹配算法，结合选中全部元素和选中子元素操作，可以实现多种复杂的参数生成需求。

判断两个元素为同类型元素的标准为：两元素的节点类型，以及其父节点元素和所有祖先元素的节点类型均相同。反映在 XPath 上，即两个元素的 XPath 中去掉谓语部分的路径是相同的。如图 5.5 所示，A，B，C，D 四个超链接元素为同类型元素，因为他们元素类型相同，且父元素节点、祖先元素节点类型均相同。反映到 XPath 上，四个元素的 XPath 分别为：

A：*/html/body/div[1]/ul[1]/li[1]/a*，

B：*/html/body/div[1]/ul[1]/li[2]/a*，

C：*/html/body/div[1]/ul[2]/li[1]/a*，

D：*/html/body/div[1]/ul[2]/li[2]/a*。

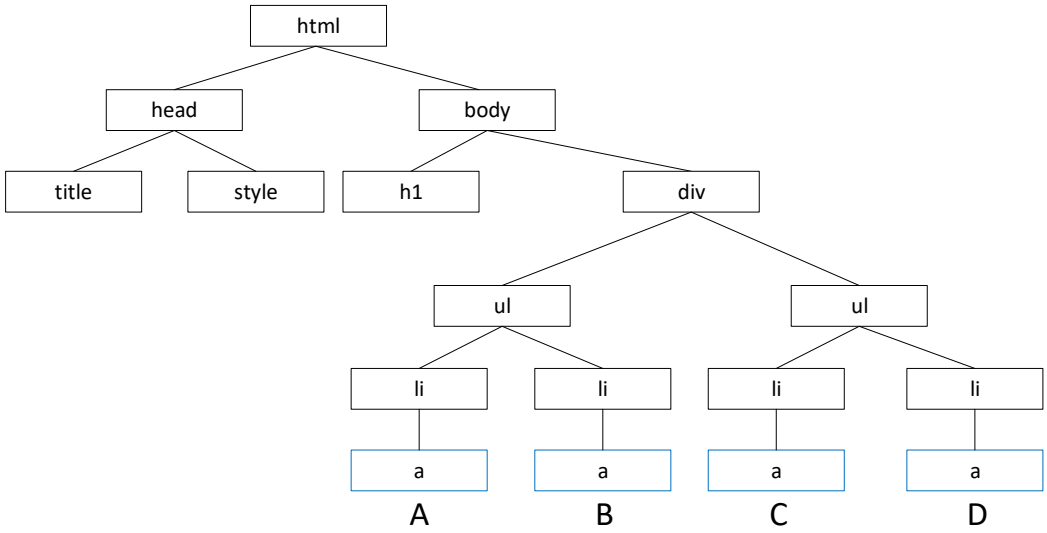


图 5.5 同类型元素结构示意图

四个元素的 XPath 去除谓语的部分均为：

/html/body/div/ul/li/a

即四个元素属于同类型元素。

由第 5.2.1.1 节中的描述可知，同类型元素自动匹配算法是在只选择了一个元素，而没有其他参考的情况下进行的同类型元素匹配，因此，为了提高效率，算法将遵循就近原则，在搜索到与当前选中的元素最近的符合同类型元素标准的元素的父节点时，搜索即停止。即如果在用户没有选中页面中其他元素的情况下选中了图 5.5 中的元素 A，则系统会匹配到与元素 A 相同类型的元素 B，而并不会匹配到 C 和 D，此时，用户继续选中元素 C，则系统会根据同类型元素手动匹配算法，匹配到元素 D。算法 3 给出了同类型元素自动和手动匹配算法的实现。

算法 3 同类型元素匹配算法实现

输入： 元素信息列表 $elementList$, 待选元素信息列表 $readyList$, 最后一次选中的元素信息 $nowElement$.

```
1: //同类型元素自动匹配算法
2: function autoFindRelatedElements
3:     //将 xpath 按照路径中各个元素节点进行分离，并得到元素名称和索引号
4:     testPath ← nowElement.XPath.split
5:     nodeNameList, nodeIndexList ←[ ],[ ] //记录元素的名称和索引号
6:     for i = 0 → testPath.length do
7:         //存入路径中各元素的名称和索引号
8:         nodeNameList ← nodeNameList + testPath[i].name
9:         nodeIndexList ← nodeIndexList + testPath[i].index
10:    end for
11:    //从上到下遍历节点路径中的元素
12:    for i = nodeIndexList.length − 1 → 0 do
```

```
        tempIndexList ← nodeIndexList //复刻一个 index 数组
        DeleteIndex(tempIndexList[i]) //删除元素索引, 即将 div[3] 变成 div
        //根据设置的索引号生成新的 xpath
        tempPath ← combineXpath(nodeNameList, tempIndexList)
        elements ← getElementsByXPath(tempPath) //根据新的 XPath 查找元素
        if elements.length >1 then //如果能枚举到第二个元素，说明存在同类元素
            readyList ← readyList + elements
            break
        end if
    end for
end function

//同类型元素手动匹配算法
function artificialFindRelatedElements
    testList = [ ]
    for i = 0 → elementList.length do
        //如果已选中的元素不是同类型的元素，取消匹配
        if i>0 and elementList[i].Xpath != elementList[i − 1].Xpath then
            return -1
        end if
        testPath ← elementList[i].XPath.split
        for j = 0 → testPath.length do
            nodeNameList ← nodeNameList + testPath[i].name
            nodeIndexList ← nodeIndexList + testPath[i].index
        end for
        testList ← testList + nodeIndexList
    end for
    indexList ← [ ] //记录新生成的 xpath 索引
    //如果选中的元素属于同样的序列，则计算出序列的最佳 xpath 表达式
    for j = 0 → testList[0].length do
        indexList ← indexList + testList[0][j]
        for i = 1 → testList.length do
            if testList[i][j] != testList[i − 1][j] then
                DeleteIndex(indexList[j])
                break
            end if
        end for
    end for
    finalPath ← combineXpath(testpath, indexList)
    elements ← getElementsByXPath(finalPath)
    readyList ← readyList + elements
end function
```

算法 3 中的 *split* 函数将一段完整的 XPath 按照路径中每个节点的元素类型和索引值分割，并赋值给两个独立的数组，如 XPath：*/html/body/div[1]/span[2]*在经过分割后会生成两个数组，分别为：*["html", "body", "div", "span"]*，*[-1,-1,1,2]*，其中-1 代表无索引值。*combineXPath* 函数负责将上述两个数组合并还原成 XPath。

由算法 3 可知，匹配到同类型元素后，这些待选元素的信息会被加入 *readyList* 中等待进一步的处理，需要注意的是，如果当前已经选中了多个元素，且这些元素没有相关性，则无法进行选中全部和选中子元素的操作。

选中全部元素即将 *readyList* 数组中的信息全部放入 *elementList* 尾部。对于选中子元素的操作，本文使用深度优先搜索算法，对已选中的所有元素中的子元素进行提取和处理，对于已选中元素之间的有细微不同的子元素，系统将会在服务参数生成阶段将该子元素单独设置成一个输出参数字段，供用户选择是否提取。由于选中子元素操作同时涉及到数据参数的生成，因此，选中子元素操作的具体实现细节将在第 5.2.2 节中配合数据参数生成操作进行描述。

5.2.2 内容脚本逻辑层设计与关键实现

5.2.2.1 功能介绍

内容脚本逻辑层中的主要方法包括：

1) 数据参数生成：生成最终要提取的数据参数字段，即服务的输出参数。如图 5.6 所示，数据参数生成时，可能存在的情况包括：

A. 当前只选中了一个元素，并提取元素本身的数据，如元素文本。此时系统生成 1 个参数字段，并带有 1 个示例值，即元素本身的值。

B. 当前选中了 n（n>1）个元素，各元素类型相同，并提取同类型元素本身的数据，与图 3.6 相同，图 B 中表示了用户选中了商品列表中所有的链接，并提取它们的地址。此时系统生成 1 个参数字段，并带有 n 个示例值。

C. 当前选中了 n（n>1）个元素，各元素类型相同，并提取元素的子元素的数据，与图 3.7 相同，图 C 中表示用户选中了商品列表中所有的商品所在的块，并提取每个块内的子元素，如商品标题，图片地址，销售商家等。如果所有块内的所有不同类型子元素数目为 m，则此时系统生成 m 个参数字段，并带有 n 个示例值；如果某个参数字段对应的子元素（如京东自营图片）在某些已选中的元素中不存在，则其字段示例值为空。

D. 当前选中了 n（n>1）个元素，各元素类型不同，此时，只能提取元素类型本身的数据。此时，系统生成 n 个参数字段，每个字段带有 1 个示例值。

图 5.6 数据参数生成各情况示例

2) 参数示例数据删除：某些情况下，用户并不想提取一个字段中所有的示例值对应位置的元素的数据，因此，系统提供了删除参数示例数据的功能，图 5.6 中所有的情况下均存在示例数据删除功能，在图 5.6 B 中，如果删除了第一行的字段示例数据“联想（Lenovo）小新 Pro”，则在服务调用阶段，当前页面商品列表中第一个商品的名称将不会被提取。

3) 确认采集：将生成的字段参数传递给后台脚本，并最终传递给服务流程管理模块，以生成采集流程中的“提取数据”操作节点，同时带有相关的参数信息。

4) 鼠标移动到元素/鼠标点击元素：进行鼠标点击元素和移动到元素的操作，并将操作信息传递给后台脚本。

5) 循环点击元素：为了执行循环点击已选中元素列表中的每个元素的操作，需要在服务流程定义模块操作台中点击“循环点击元素”选项。选项执行后，系统将会把循环点击元素的操作信息传递给服务流程管理模块，从而生成相应的执行操作节点，并同时点击第一个需要循环点击的元素，供用户进行下一步的操作。

6) 输入文字/切换下拉选项：进行输入文字或切换下拉选项操作，并将操作信息（包括输入的值）传递给服务流程管理模块。

5.2.2.2 关键实现

本系统通过调用 JavaScript 的 *Click* 方法来实现点击页面元素的功能，通过设置 *select* 元素的 *Value* 值，来实现切换下拉选项的功能。对于输入文字的模拟操作，本文通过将输入文字操作的信息传递给客户端，并由客户端调用 Windows API[58]集合中的 *SendKeys* 方法来从操作系统底层模拟键盘的输入操作。

表 5.1 给出了数据参数生成方法中生成参数的数据结构示例。当参数示例值数目超过 1 个时，需要在服务调用阶段使用循环操作来提取同一个字段的不同的

元素的数据，此时，需要设置 *relative* 字段为 *true*，并配置 XPath 字段值为相对于循环元素内的 XPath，有关循环操作的细节将在第 5.4 节中讲述。

当执行了参数示例数据删除操作后，*XPath* 字段将由单条 XPath 改为多条 XPath 的列表，同时，恢复被删除的示例数据对应的元素的原本样式。

表 5.1 数据提取参数结构示例

```
[{
    "name": "商品类别_链接文本",
    "elementType": "a", //元素类型
    "contentType": "outerHTML", //数据类型
    "relative": true, //是否使用相对循环的 XPath
    "XPath": "/a", //根据 relative 的值设置绝对 XPath 或相对 XPath
    "exampleValues": [{ //参数示例值
        "id": 0,
        "value": "女装",
    }],
}]
```

在数据参数生成方法中，如果提取的是元素本身的参数值，则只需要将 *elementList* 中包含的元素信息转化为表 5.1 中的数据参数即可；如果提取的是已选中元素的子元素的数据，则需要使用深度优先搜索算法，找到所有已选中元素中的所有子元素，并将每个不同位置的子元素数据对应为一个输出参数。算法 4 给出了选中子元素并生成数据参数方法的实现细节。

算法 4 选中子元素并生成数据参数

输入: 元素信息列表 $elementList$, 待生成的参数列表 $parameters$, 已经添加的参数的数目 n.

```
1: //对已选中的每一个元素进行遍历
2: for i = 0 → elementList.length do
3:     element ← elementList[i]
4:     stack ← new Stack()
5:     //从元素节点根部开始深度优先搜索遍历元素
6:     stack.(element)
7:     while stack is not null do
8:         t_element ← stack.POP // 从栈中取出元素
9:         //只插入那些不在参数列表中的元素
10:        if t_element not in outputParameters.elements then
11:            para ← parameters.PUSH(NEWPARAMETER(t_element))
12:            para.relative = true
13:            para.XPath = GETRELATIVEXPATH(t_element,elementList[i])
14:            parameters[i].exampleValues.PUSH({ "id": i, "value": t_element.Data })
```

```
15:         else//如果元素节点已经存在，则只需要插入示例值就可以了
16:             parameters[i].exampleValues.PUSH({ "id": i, "value": t_element.Data })
17:         end if
18:         for j = t_element.children.length - 1 → 0 do
19:             stack.PUSH(t_element.children[j])
20:         end for
21:     end while
22: end for
```

在算法 4 中，*newParameter* 方法根据传入的元素信息生成表 5.1 中形式的数据参数；*getRelativeXPath(element, parent_element)*方法则用来获取 *element* 元素在 *parent_element* 元素内的相对 XPath。*t_element.Data* 表示元素的数据（如图片源）。

最后，“确认采集”操作将生成的数据参数以及操作信息传递给后台脚本。

5.2.3 后台脚本设计与关键实现

服务流程定义模块后台脚本的主要方法如下：

1) 消息监听与处理：监听来自内容脚本和客户端中发出的消息，并解析消息内容，进行对应的操作处理，如更新全局变量，控制页面刷新等。

2) 消息传递：将用户操作过程中执行的各个操作的消息传递给客户端，并最终传递给服务管理模块，从而实现对流程节点的管理。同时，对于接收到的输入文字操作的请求，后台脚本向客户端传递输入文字的消息，并由客户端调用 Windows API 实现。

3) 全局变量管理：管理全局变量，如已存在的参数的数目，当前激活窗口的 ID 等。

后台脚本使用 Chrome 扩展中的 *OnMessage* 方法监听来自内容脚本的消息，使用 WebSocket 的 *onmessage* 方法监听来自客户端的消息，并通过 WebSocket 的 *send* 方法向客户端传递消息；最后，后台脚本和内容脚本均使用 Chrome 扩展中的 *storage.local.set* 和 *storage.local.get* 方法设置和得到全局变量。

表 5.2 展示了后台脚本向客户端发送消息的一个片段。

表 5.2 消息片段示例

```
{
    "type": 3, //消息类型，3 代表增加一项操作
    "from": 0, //0 代表从浏览器到流程图，1 代表从流程图到浏览器
    "message": {
        "operationType": "singleClick", //操作类型，这里是点击元素
        "useLoop": false, //是否循环点击元素
```

表 5.2 消息片段示例（续）

```
        "XPath": "//*[@id=\"lg\"]"}"
    }
}
```

5.3 客户端设计与实现

客户端的设计与实现较为简单，其功能包括：

1) 消息转接：作为服务流程定义模块和服务流程管理模块的消息传递的管道，从而实现两个模块之间的交互。

2) 进程操作：实现对进程的基本操作，如将服务流程定义模块的浏览器窗口高度调整为当前屏幕的一半，并放置在当前屏幕的下方；将服务流程管理模块所在进程的窗口放置在屏幕的上方，以供用户同时对两个模块进行管理。

3) 交互行为模拟：实现模拟键盘输入文字的操作。

4) 服务信息可视化展示及管理：通过嵌入 Chrome 内核浏览控件对服务注册中心网站门户进行浏览，同时，作为服务流程管理页面的载体存在。

客户端使用 C# 和.Net Framework 框架实现，搭载 WebSocket 和 JSON 解析的动态链接库，实现消息的转接功能；通过调用 Windows API，实现进程的相关操作；通过在 C# 窗体中嵌入 CefSharp[59]浏览器控件，实现在客户端中对服务注册中心中存在服务的管理，以及加载服务流程管理模块操作页面的功能。

5.4 服务流程管理模块设计与实现

由第 3.3.1 节可知，服务流程管理模块是一个可以自定义 Web 数据采集流程的操作台，使服务提供者可以通过可视化的方式对流程中的各个操作选项进行增删改查，剪切复制的控制，同时修改各个选项中涉及的各项参数。服务流程管理模块的操作架构如图 5.7 所示。

由图 5.7 可知，服务流程管理模块同样由表现层和逻辑层构成。表现层负责根据用户的操作实时渲染流程图。如当用户在服务流程定义模块中点击“确认采集”操作后，流程图中应出现“提取数据”的操作节点，并在参数面板中显示用户提取的数据参数的信息。逻辑层负责对客户端传来的消息进行监听与处理，并协助用户修改操作参数和服务信息，以生成 Web 数据采集服务。

我们知道，计算机程序通常由选择，顺序，循环三种语句组成，因此，针对 Web 数据采集任务，我们同样定义了这三种类型的操作：

1) 顺序操作。在 Web 数据采集流程中，涉及到的顺序操作包含：打开网页，

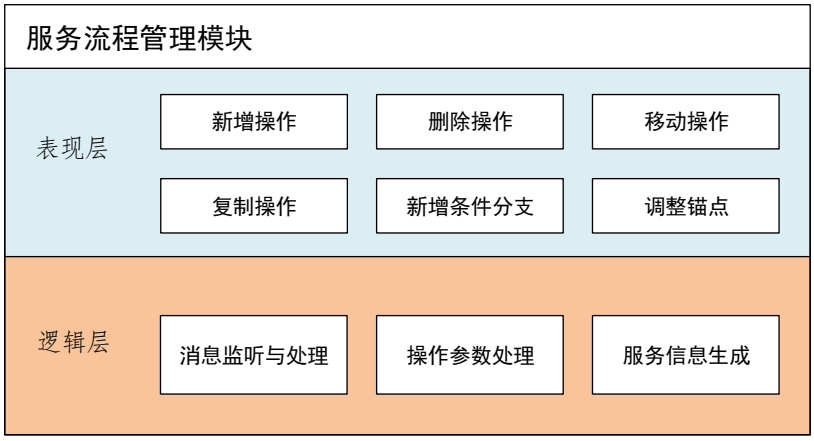


图 5.7 服务流程管理模块操作架构图

点击元素，提取数据，输入文字，识别验证码，切换下拉选项，移动到元素。

2) 选择操作。即条件判断操作，一个条件判断内可以包含多个条件分支，每个条件分支内可以包含任意的其他操作，包括选择，顺序和循环操作。在 Web 数据采集任务中，条件分支的条件包含：当前页面包含文本，当前页面包含元素，当前循环项包含文本，当前循环项包含元素，无条件（else）。

3) 循环操作：一个循环操作中同样可以包含任何的其他操作，配合循环内部操作中的“使用相对循环内的元素/XPath”选项，来满足各种各样定制化的需求。在 Web 数据采集任务中，循环方式包含：单个元素循环，不固定元素列表循环，固定元素列表循环，文本列表循环。

“单个元素”的循环方式指每次循环中定位到的元素为固定的单个元素，如一个循环中包含一个“点击元素”的操作，循环方式为“单个元素”，并在循环中的“点击元素”参数设置中勾选了“使用循环中的元素”选项，则程序会根据循环中设置的 XPath，不停地点击相应的元素，直到找不到元素或达到设定的退出条件为止。

由第 2.2.2 节可知，一条 XPath 语句可以定位到页面中的多个元素，因此，“不固定元素列表”的循环方式只需要输入单条 XPath，就可以匹配到页面中所有对应的元素。系统中各个模块的 XPath 由服务流程定义模块自动生成，并可被用户修改。

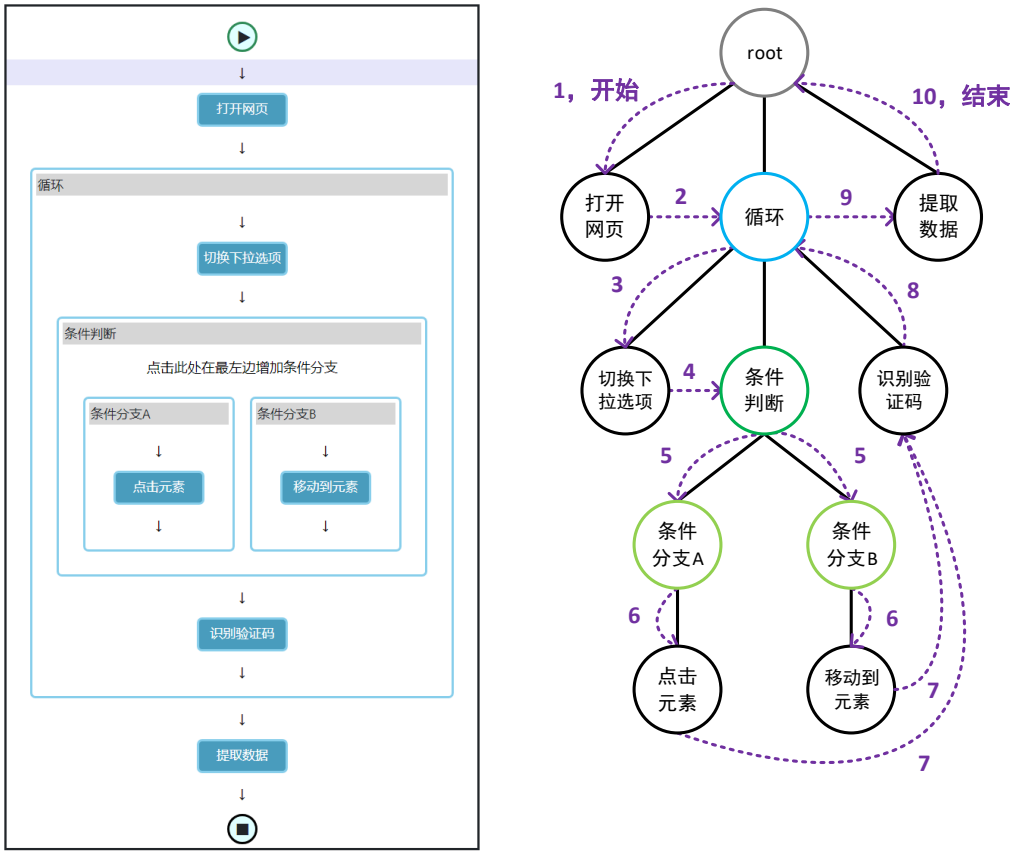
当循环方式为“固定元素列表”循环时，系统会生成多行 XPath，每次循环定位一行 XPath 对应的第一个元素，如某固定元素列表循环的 XPath 有两行，分别为：*/html/body/**div[1]**/ul* 和 */html/body/**div[2]**/ul*，循环中的“提取数据”操作含有一个数据字段，字段中勾选了“使用相对循环内的 XPath”选项，并设置

XPath 为：/li[1]/img，则程序会循环执行两次“提取数据”的操作，第一次提取的数据对应元素的 XPath 为：*/html/body/**div[1]**/ul/li[1]/img*，第二次为：*/html/body/**div[2]**/ul/li[1]/img*，即实现了循环提取每个元素的对应位置子元素数据的功能。

系统默认以不固定元素列表循环，即一条 XPath 来定位所有的同类型元素，但当执行了任意的参数示例数据删除操作后，系统将改为使用固定元素列表循环方式来逐个定位元素，列表中的每条 XPath 只对应一个元素。

文本列表循环即循环的在输入文字操作中输入文本列表中的每个文本值。

在本系统中，我们将所有操作选项抽象化为统一的数据结构进行处理，流程中任意的操作选项被称为一个“节点”，整个流程以树状结构存储。在程序执行阶段，程序操作处理模块将会从根节点开始依次读取并执行其子节点对应操作，以还原整个操作执行流程。如图 5.8（a）为用户定义的一个操作流程图，其中所有的顺序操作节点以深蓝色为背景的长方形表示，其内部不可再添加元素；而循环



（a）服务流程图　　（b）程序树状结构及执行顺序示意

图 5.8 服务流程图及其树状结构和执行顺序示例

节点、条件判断节点及条件分支节点以无背景颜色的长方形表示，其内部可以添加其他操作节点。

图 5.8（b）中展示了将（a）图中的流程存储结构具象化后的树状图，其中的箭头和序号展现了程序的执行顺序。在服务流程模块中生成的流程图遵循**“从上到下，从内到外”**的执行顺序，从上到下即操作从上到下顺次执行，从内到外指在多层嵌套的循环/判断操作中，程序先执行最内层循环/判断中包含的操作，当最内层循环中的元素遍历完毕或满足其他退出条件时，程序会跳出内层循环，继续向下执行其外层循环的操作，依此类推。在图 5.8 中，每次条件判断只会执行一个分支中的操作；**而 3 到 8 的操作会一直循环进行，直到满足循环退出条件。**

接下来，本文将介绍如何维护上述树状结构，包含相关增删改查操作的实现。

服务流程管理模块通过设置数组 *nodeList* 来记录程序执行流程中各个节点的信息以维护树状结构，每次新建任务时，都将在 *nodeList* 中首先插入一个 root 节点，表示程序执行的根节点。每个操作节点信息的数据结构如表 5.3 所示。

表 5.3 操作节点数据结构示例

```
{
    "index": 0, //索引号
    "id": -1, //渲染时的 id 号
    "parentId": 0, //渲染时父节点的 id 号
    "position": 1, //操作节点在父元素中的位置
    "type": 1,   //节点类型，-1 到 3 分别表示 root、顺序、循环、条件判断和条件分支
    "option": 8, //操作选项值，8 代表循环操作，如 2 代表点击元素操作
    "title": "循环", //操作名称
    "sequence": [3,5,8], //所包含操作序列索引
    "isInLoop": false, //操作是否位于循环中
    "parameters": { //操作参数列表，不同节点的操作参数不同
        "XPath": "//*/h3/a",
        "wait": 0, //执行完成后等待时间
        "useLoop": false, //是否使用循环中的元素
        "loopType": 1, //循环类型，1 代表不固定元素列表循环
        "value": "电脑", //示例值
    }
}
```

在表 5.3 中，*sequence* 数组负责有序的记录当前操作内包含的操作节点信息

在 *nodeList* 中的索引号，表 5.3 中的节点对应的操作为图 5.8 中的循环操作，其内部包含三个操作节点：切换下拉选项，条件判断，识别验证码，三个节点的操作信息分别存储在 *nodeList* 中的第 4，6，9 个位置（0 为索引第一位）。同时，*parameters* 对象存储了只在循环节点中存在的参数信息，如循环类型等。

算法 5 展示了流程图的具体渲染过程及流程中节点的部分操作的实现细节。

算法 5 服务流程管理模块流程操作算法实现

输入: 节点信息列表 $nodeList$, 图结构遍历队列 $queue$, 流程图操作序列 $actionSequence$, 当前操作锚点信息 $nowArrow$, 当前选中的节点信息 $nowNode$

```
//流程图绘制
function renderFlowChart
    //初始化操作，清空队列，并将 root 节点放入队列中进行广度优先搜索
    actionSequence.clearAndPush(0)
    queue.clearAndPush(0)
    n ← 1 //用于记录当前操作的索引位置
    while queue is not null do
        nodeId ← queue.pop //取出父节点并建立对子节点的链接
        for i = 0 → nodeList[nodeId].sequence.length do
            nodeList[nodeList[nodeId].sequence[i]].parentId ← nodeList[nodeId].id
            nodeList[nodeList[nodeId].sequence[i]].position ← i
            nodeList[nodeList[nodeId].sequence[i]].id ← n++
            queue.push(nodeList[nodeId].sequence[i])
            actionSequence.pushnodeList[nodeId].sequence[i]
        end for
    end while
    //渲染流程图，第一个节点不渲染
    for i = 1 → actionSequence.length do
        parentId ← nodeList[actionSequence[i]].parentId
        //通过 getNodeHTML 方法来获得不同的操作选项下节点的 HTML 代码
        childNode ← getNodeHTML(nodeList[actionSequence[i]])
        //将节点插入到当前锚点位置所在节点 HTML 中的最后一个位置
        getElementbyId(parentId).append(childNode)
    end for
end function

//新增节点操作
function addNewNode(option,parameters)
    //根据不同的选项值和用户指定的参数创建一个新的选项节点
    node ← newNode(option,parameters)
    node.index = nodeList.length //记录节点索引号
    nodeList.push(node)
    if option == 9 then //如果添加的是条件判断节点
        //生成两个条件分支节点，并生成默认的参数
```

```
35:         branchA, branchB ← NEWNODE(10,defaultParameters)
36:         nodeList.PUSH(branchA,branchB)
37:         branchA.index, branchB.index = nodeList.length
38:         node.sequence.PUSH(branchA.index,branchB.index)
39:     end if
40:     //nowArrow 中记录了当前锚点的位置以及对应元素的父元素的索引
41:     position, pId ← nowArrow.position,nowArrow.pId
42:     //在相应位置添加新节点
43:     nodeList[actionSequence[pId]].sequence.INSERTAT(position + 1, node.index)
44:     //锚点位置设置
45:     nowArrow ← { "position": node.position, "pId": node.parentId}
46:     RENDERFLOWCHART //重新渲染流程图
47: end function
48:
49: //剪切当前选中节点
50: function CUTNODE
51:     pId, id, position, positionB ← nowNode.pId,nowNode.id, nowNode.position, nowArrow.position
52:     pIdB ← nowArrow.pId
53:     //一个节点不能移动到自己内部的操作节点中
54:     if DETECTSELF(pIdB,id) is true then
55:         return
56:     end if
57:     //在相应位置删除节点，并得到被删除的节点
58:     node = nodeList[actionSequence[pId]].sequence.DELETEAT(position, 1)
59:     //如果要移动的位置属于同一层并且是从前往后移动，注意需要控制数组插入位置向前错位
60:     if pId == pIdB and position < positionB then
61:         positionB ← positionB - 1
62:     end if
63:     //在相应位置添加被删除的节点
64:     nodeList[actionSequence[pIdB]].sequence.INSERTAT(positionB + 1, node)
65:     nowArrow ← { "position": node.position, "pId": node.parentId }
66:     RENDERFLOWCHART //重新渲染流程图
67: end function
```

算法 5 以节点的新增和剪切操作为例，展示了服务流程管理模块对节点数据结构的操作过程。在流程中的删除、复制和剪切操作中，系统使用 *nowNode* 对象来存储用户当前选中的节点信息，使用 *nowArrow* 对象存储用户想要将节点放置的位置信息。新增条件判断操作时，会同时新增两个条件分支；删除循环节点时，循环内包含的所有节点都将被标记为不可用，从而避免在服务参数生成阶段生成已删除的输入输出参数；注意，在执行剪切操作时，节点不可以被剪切到自己的子孙节点中；复制节点时，需要对节点对象进行深复制，从而将节点内的所有参

数拷贝给新节点；每次执行对节点的操作后，都会根据情况调整锚点的位置，并重绘流程图。

接下来，本文简单介绍服务流程管理模块中逻辑层的功能和实现方案。

服务流程管理模块的逻辑层负责监听来自客户端的消息，从而将从服务流程定义模块中传递来的操作信息转化为操作节点插入到服务执行流程中，如算法 5 中的 *addNewNode* 方法接收两个参数，分别表示新增操作节点的选项类型及操作包含的相关参数（如提取数据操作中的数据字段信息），如果是用户直接在流程图中增加的节点，则参数将设置为默认值。最后，服务流程管理模块对 *nodeList* 中的各个节点进行遍历，从而将操作中涉及到的输入输出参数提取出来，作为最终生成服务的输入输出参数，并生成服务信息，提交给服务注册中心。

5.5 程序操作处理模块设计与实现

在服务调用阶段，服务注册中心负责将用户提供的输入参数与服务流程中各个操作的输入参数进行对应，并将各操作中的输入值替换成用户指定的输入值，如在 Web 请求中指定 *inputText=浙江大学*，则在程序执行阶段，输入的文字将由原来设定的默认值改为输入“浙江大学”。

程序操作处理模块负责执行用户设定的 Web 数据采集流程，由程序执行方式可知，程序执行阶段会维护一个程序调用栈，当有函数执行操作时，PC 指针会跳转到函数中执行，函数调用完成后，PC 指针跳回原来的程序调用栈，以此完成程序的执行。因此，参照以上原理，本文实现了自定义服务流程的执行功能。

在拿到一个任务信息中的数据采集流程的 *nodeList* 数组后，程序操作处理模块首先进行了以下预备操作：

（1）将打开网页操作对应 URL 列表的值按行分割，从而记录需要执行服务流程的所有 URL，对 URL 列表中每个 URL 对应的 Web 页面，程序都会执行一次数据采集流程并提取数据。

（2）维护一个 *OUTPUT* 的二维数组，用来记录最终的提取结果，每次执行提取数据的操作时，OUTPUT 数组中将会增加一个字符串数组，包含 n 列，n 为服务输出参数的个数。

（3）设置一个 *outputParameters* 对象，对象中的每一个 key 对应被调用服务中的一个输出参数，value 值设置为空，用来在数据提取阶段匹配参数使用，每次提取数据行为发生时，都会将 *outputParameters* 中对应 key 中的 value 值设置为定位到元素的数据值，而当前数据操作节点中不包含的参数值则维持上一次的赋值

结果不变，从而实现对全局输出参数的维护。

在程序执行过程中，如果找不到要定位的元素，程序会跳过当前操作并继续执行，如找不到要点击的元素时，退出循环；找不到要提取的数据字段中对应的元素时，使用用户定义的默认值来填补此字段。

程序操作处理模块通过调用天聚地合数据公司提供的验证码识别服务来识别验证码，由于验证码识别功能由第三方实现，因此，本文中将不对验证码识别功能做更多介绍和实验测试。

如果程序执行过程中的循环操作内含有点击元素的操作，则在点击元素后，会存在两种情况：**发生了页内跳转**或**打开了新标签页**。

而在一次循环执行完成后，程序需要回到循环未开始执行时的页面继续进行下一次的循环操作，如在循环点击京东商品列表中的商品链接并提取用户评论的服务中，在点击第一个商品链接并提取数据后，程序需要跳转回商品列表页面，继续点击第二个商品的链接。对于以上不同的情况，程序需要在每个循环操作维护的变量环境中记录跳转前后标签页的句柄和 Web 页面的历史记录长度来实现页面回退，并在每次发生点击元素操作后，记录下上述两个参数以进行数值匹配。

由于很多 Web 页面的加载速度很慢，且可能出现页面上的主要元素都已经存在而页面还没有加载完毕的情况，此时，需要程序手动停止页面加载；由于程序执行时的网络环境可能不够稳定，因此，程序需要在必要的操作前设置一定的等待时间，来保证数据采集任务的稳定性。最后，部分 Web 页面的数据需要在用户向下滚动屏幕时才会将所有的数据加载出来，因此，程序还需要模拟滚动条滚动的操作来满足用户的特定采集需求。

同样，所有对 Web 页面交互的操作均是通过调用 Selenium Webdriver 中的方法实现，这里将不再赘述，对于程序流程执行的部分实现细节，详见算法 6。

算法 6 程序操作处理模块方法实现

输入： 流程节点信息列表 $nodeList$, 历史信息 $history$, 浏览器实例 $browser$.

```
1: //执行任意操作节点
2: function EXCUTENODE(nodeId, loopValue="", clickPath="", index=0)
3:     //nodeId 表示节点在 nodeList 中的索引号
4:     //loopValue 表示由循环操作传递给该节点的当前循环项
5:     //clickPath 和 index 分别表示点击操作中元素的 XPath 及匹配多个元素情况下的具体索引号
6:     node ← nodeList[nodeId]
7:     //根据不同选项执行不同操作
8:     if node.option == 0 or node.option == 10 then //root 操作和条件分支操作
9:         for all i in node.sequence do //从根节点开始向下读取节点，并顺序执行
10:            EXCUTENODE(i, loopValue,clickPath,index) //调用本身
11:        end for
```

```
    else if node.option == 8 then //循环操作
        loopExcute(node, loopValue,clickPath,index)
    else if node.option == 9 then //条件判断操作
        judgeExcute(node, loopValue,clickPath,index)
    else//任何的顺序操作，如打开网页，点击元素等
        //执行操作, 根据节点选项执行不同的操作流
        excuteOperation(node,loopValue,clickPath,index)
    end if
    //执行完之后进行等待
    wait(node.parameters.waitTime)
end function

//对循环操作的处理方法
function loopExcute(node, loopValue,clickPath="",index=0)
    thisHandle ← browser.currentWindowHandle //记录本次循环内的标签页的句柄
    //记录本次循环内的 Web 页面历史记录的长度
    thisHistoryLength ← browser.currentHistoryLength
    if node.parameters.loopType == 0 then //单个元素循环
        count ← 0 //执行次数
        while true do
            //findElementByXpath 方法只得到 XPath 对应的第一个元素
            element ← browser.findElementByXpath(node.parameters.xpath)
            if element is null then //找不到元素就退出循环
                break
            end if
            for all i in node.sequence do //挨个执行操作
                excuteNode(i, element, node.parameters.xpath,0)
            end for
            count ← count + 1
            //达到设置的最大循环次数时，退出循环
            if node.parameters.exitCount == count then
                break
            end if
        end while
    else if node.parameters.loopType == 1 then //不固定元素列表循环
        //findElementsByXpath 方法可得到 XPath 对应的所有元素
        elements ← browser.findElementsByXpath(node.parameters.xpath)
        for index = 0 → elements.length do
            for all i in node.sequence do //依次执行循环内的操作
                excuteNode(i, elements[index], node.parameters.xpath, index)
            end for
            handleLoopTabsAndHistory(thisHandle,thisHistoryLength) //回到循环执行前页面
        end for
    else if node.parameters.loopType == 2 then //固定元素列表循环
```

```
56:         for all path in node.parameters.XPath.SPLIT("\n") do //分割多行 XPath
57:             element ← browser.FINDELEMENTBYXPATH(path)
58:             for all i in node.sequence do //依次执行循环内的操作
59:                 EXCUTENODE(i, element, path, 0)
60:             end for
61:             HANDLELOOPTABSANDHISTORY(thisHandle,thisHistoryLength)
62:         end for
63:     else if node.parameters.loopType == 3 then //固定文本列表
64:         textList ← node.parameters.textList.SPLIT("\n")
65:         for all text in textList do
66:             for all i in node.sequence do //依次执行循环内的操作
67:                 EXCUTENODE(i, text, "",0)
68:             end for
69:         end for
70:     end if
71:     history.index ← thisHistoryLength
72:     history.handle ← browser.currentWindowHandle
73: end function
```

在算法 6 中，我们将所有的操作抽象化为一个统一操作 *excuteNode*，其中含有所有操作都必须执行的公共逻辑，如执行后等待，根据循环项和元素设置的相对 XPath 获得元素相对于根节点的真正的 XPath 等，并将各类操作的具体实现逻辑分配给其他具体的函数来执行，*excuteOperation* 方法抽象了所有顺序操作函数的执行逻辑。*judgeExcute* 函数负责条件判断操作的实现，程序会按顺序依次判断条件判断节点中包含的各个条件分支节点内定义的条件是否被满足，并执行第一个满足条件的分支中的操作，即在 *judgeExcute* 函数的最后会执行：

excuteNode(branchId,loopElement,clickPath,index)

其中，*branchId* 代表满足条件的第一个条件分支的索引号，后面三个参数由循环操作传入，并继续向下传递给条件分支内的其他操作。

对于循环操作中的不同循环类型，程序将执行不同的循环处理逻辑；每次循环操作执行时，都会首先操纵浏览器跳转回循环执行前的 Web 页面，即使用算法 6 中的 *handleLoopTabsAndHistory* 方法，通过比较程序执行上下文和所在循环内记录的句柄和历史记录的差值，来实现页面回退的功能，从而保证元素定位和数据采集操作的正确性。

在主函数中，我们在对每一条 URL 列表中的 URL 执行一次：*excuteNode(0)* 语句，即将服务流程的 root 节点放入程序执行序列，然后顺序执行 root 序列中包含的操作节点，即可完成数据采集的功能；在进行数据过滤后，将最终结果返回

给服务请求者。

如果服务请求者选择在云端调用服务并执行采集操作，本系统可以同时打开多个浏览器实例对 URL 列表中的不同的 URL 进行并行化的采集，从而提高数据采集的效率，我们将在采集结束后将所有浏览器实例中采集的数据合并，并将合并后的数据返回给服务请求者。

5.6 案例分析

本节以 58 同城网站的房源信息采集案例为例，详细介绍了如何使用复杂数据采集分系统以可视化的方式生成自定义的 Web 数据采集服务。

5.6.1 采集需求定义

如图 5.9 所示，58 同城杭州地区房源列表信息页面包含一个搜索表单，多条房源信息以及一个下一页的按钮。点击房源链接后，浏览器将会跳转到如图 5.10 所示的房源信息详情页面（图片经过简化），页面中包含房源的描述信息。

图 5.9 58 同城杭州地区房源列表页面关键信息展示

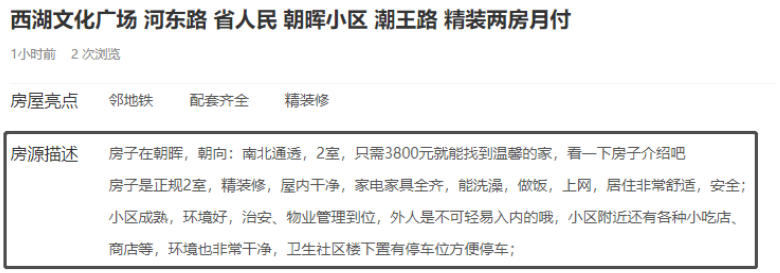


图 5.10 58 同城房源信息详情页关键信息展示

本次想要完成的采集任务需求如下：

1、循环的在图 5.9 中的搜索表单中输入地段名，如“阿里巴巴”、“网易”等，

得到不同区域的房源信息，搜索结果将以图中多页的信息列表形式展现。

2、针对房源列表中所有标题中含有“单间”文字的房源链接，点击房源链接进入如图 5.10 所示的房源信息详情页面，并采集其房源描述等信息。

3、针对步骤 1 中的每个地段名搜索得到的多页列表，连续采集 5 页符合第 2 条中描述的房源详情信息。

5.6.2 服务流程示例

针对 58 同城房源信息采集案例中的服务生成流程，其简单描述如下：

（1）服务提供者在系统中输入 58 同城杭州地区房源搜索页面的 URL：*https://hz.58.com/chuzu/*，并点击开始按钮发起服务生成请求。

（2）服务流程定义模块和服务流程管理模块会同时启动。如图 5.11 所示，服务提供者使用服务流程管理模块中的工具箱，在“打开网页”节点下拖入一个循环操作，并设置循环类型为：文本列表，并在下方的文本内容框中输入“阿里巴巴”、“网易”两行文字。这里输入的文字可以在服务调用阶段被其他值替换。

图 5.11 循环操作配置

（3）如图 5.12 所示，在服务流程定义模块中，鼠标选中搜索框，并点击操作台中的“输入文字”操作，并输入任意文本，如“支付宝”，然后点击确定按钮。此时，搜索框中将被填写“支付宝”字样。接下来，在服务流程管理模块的参数面板中，勾选新生成的“输入文字”操作节点的“使用循环中的文本”选项，使得该操作中接收的文字由循环操作指定，即在执行阶段“支付宝”将不被输入，而是输入循环中的“阿里巴巴”等文字。

（4）鼠标选中图 5.12 中的“搜房源”按钮，并点击操作台中的“点击元素”操作，系统将会执行点击元素的操作，并展示房源信息，同时在服务流程管理模块中生成“点击元素”的操作节点。

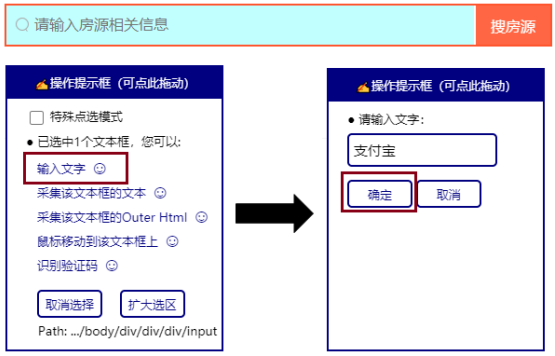


图 5.12 输入文字操作配置

（5）鼠标选中图 5.9 中所示的下一页按钮，并在操作台中点击“循环点击该元素”操作。此时，服务流程定义模块中将会出现如图 5.13 所示的循环操作节点，这里，我们将参数面板中右下角的参数“最多执行循环次数”设置为 5。

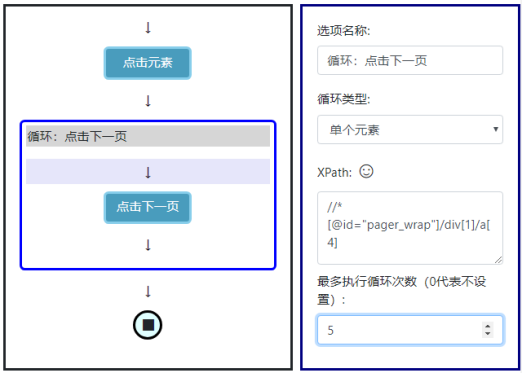


图 5.13 循环点击下一页操作配置示例

（6）如图 5.14 所示，选中页面中第一个房源信息链接，服务流程定义模块将会自动匹配到其他房源信息的链接并作出标记，此时，可以通过点击操作台中的“选中全部”操作，或者鼠标直接选中第二个房源信息链接元素，来选中所有的房源信息链接。

图 5.14 选中全部链接示例

（7）点击图 5.14 右侧的操作台中的“循环点击每个链接”选项。系统将会自动打开第一个链接对应的房源信息页面。注意，如果这里直接点击“采集数据”操作，系统将会对每个房源链接的文本和地址进行采集，即图 5.14 右下角所示的样例。

（8）如图 5.15 所示，从服务管理模块的工具箱中向当前锚点拖入一个条件判断操作，并将第一个条件分支的条件选项设置为“当前循环项包含文本”，下方的“包含的文字/元素 XPath”输入框中的值设置为“单间”。

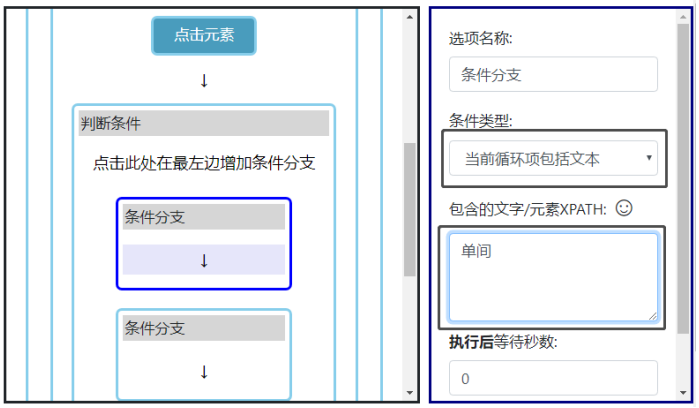


图 5.15 条件分支配置示例

（9）如图 5.16 所示，在服务流程定义模块中，选中页面的房源信息块，并可同时选中页面的其他想要采集的信息（如经纪人信息），点击“采集该元素的文本”→“确认采集”选项。此时，在服务流程管理模块的第一个条件分支内生成了“提取数据”的节点。

图 5.16 采集数据操作示例

（10）服务提供者在服务流程管理模块对相关参数进行修改，如各操作节点

的选项名称，数据提取操作中各输出字段参数名等，最后，点击工具箱中的保存服务按钮，在输入服务的名称和描述后，将生成的 Web 服务提交给服务注册中心。

由于 58 同城房源数据采集服务不属于高分相关服务，因此，我们将服务保存到普通服务注册中心平台，最终生成的 58 同城房源信息采集服务的基本信息如图 5.17 所示。

服务名称：58同城房源数据采集服务

服务描述：https://hz.58.com/chuzu/

服务示例URL：https://hz.58.com/chuzu/

输入参数：

ID	参数名称	调用名称	参数类型	示例值	参数描述
1	打开网页	urlList_0	string	https://www.jd.com	要采集的网址列表,多行以\n分开
2	循环搜索	loopText_1	string	阿里巴巴 网易	要输入的文本/网址,多行以\n分开

输出参数：

ID	参数名称	参数类型	示例值	参数描述
1	房源标题	string	房租月付 配套齐全 无中介 绿化好 紫晶时代	
2	房源描述	string	1、交通便利，700米轻轨无缝对接地铁5号线南湖站...	
3	经纪人_文本	string	潘洋(经纪人)	
4	经纪人_链接	string	https://houserent.58.com/landlord/center?infold=4...	经纪人主页

修改服务 调用服务

图 5.17 58 同城数据采集服务基本信息展示

由图 5.17 中生成的服务信息中可知，服务含有两个输入参数：*打开网页*和*循环搜索*，分别代表要打开的网页 URL 列表和循环搜索中使用到的搜索值，这里即地段名。在服务调用阶段，两个输入参数对应的 Web 请求中的请求参数名分别为 *urlList_0* 和 *loopText_1*。

图 5.17 中生成的服务包含四个输出参数，其含义分别是房源标题、房源描述、经纪人名称以及经纪人主页的链接地址，这些参数名称可被服务提供者修改。

如图 5.17 下方所示，本系统可以对生成的 Web 服务进行修改，并提供了一个服务调用页面供服务请求者使用。

最终生成的 58 同城房源信息采集服务的操作执行流程如图 5.18 所示。

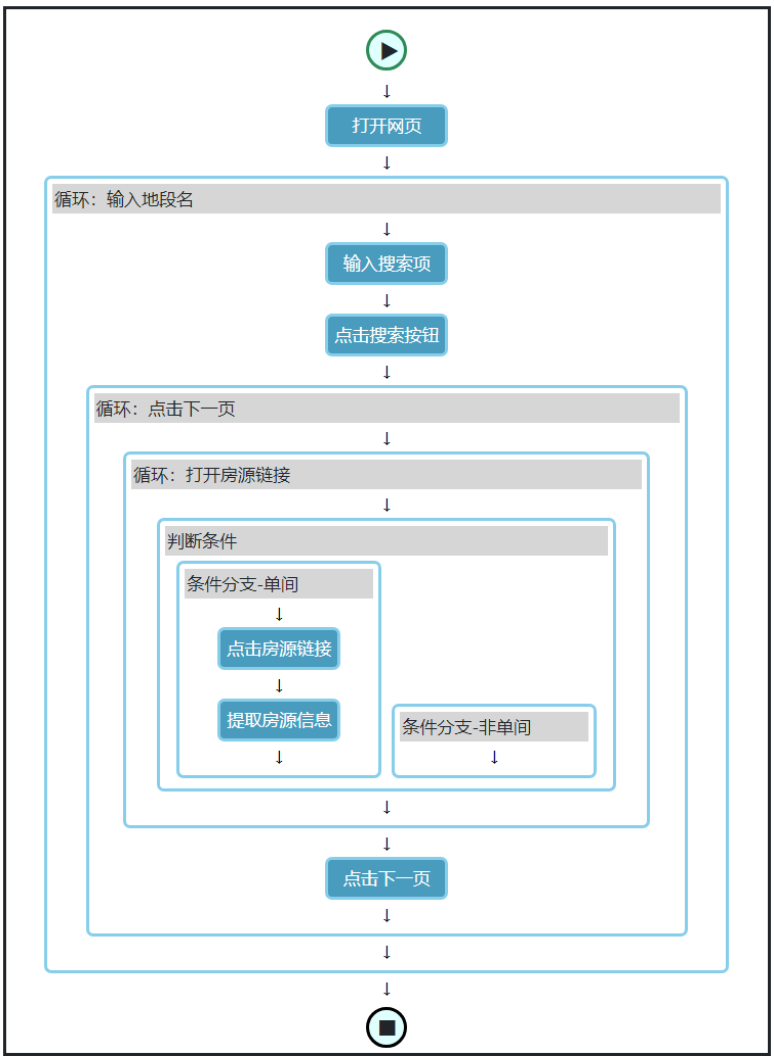


图 5.18 58 同城房源信息采集服务操作执行流程图

如图 5.18 所示，本次 58 同城房源信息采集任务由 3 个循环，1 个条件判断，2 个条件分支和 6 个顺序操作节点组成。相关操作的解释如下：

1、最外层的循环负责循环的在搜索框中输入值并提交表单，即图 5.11 中所示的文本列表循环类型，文本由用户在服务调用阶段指定，其默认值为图 5.11 中输入的值。

2、中间的循环负责循环点击下一页的操作，当满足指定的循环次数或者找不到下一页按钮的时候，退出循环。

3、最内层的循环负责循环点击每个房源信息链接并采集数据。

4、在最内层的循环里包含一个条件判断操作，即图 5.15 所示的操作，该条件判断节点内包含两个条件分支，执行时按照从左到右的顺序进行条件分支的判

断，即当第一个条件分支的条件“当前循环项中包含‘单间’二字”被满足时，执行第一个条件分支中的操作；否则，执行第二个条件分支中的操作。本例中只有第一个条件分支内含有操作，因此，该流程成功实现了只采集具有单间标识的房源描述信息。

图 5.19 展示了图 5.18 中生成的 Web 数据采集服务的实际执行流程。

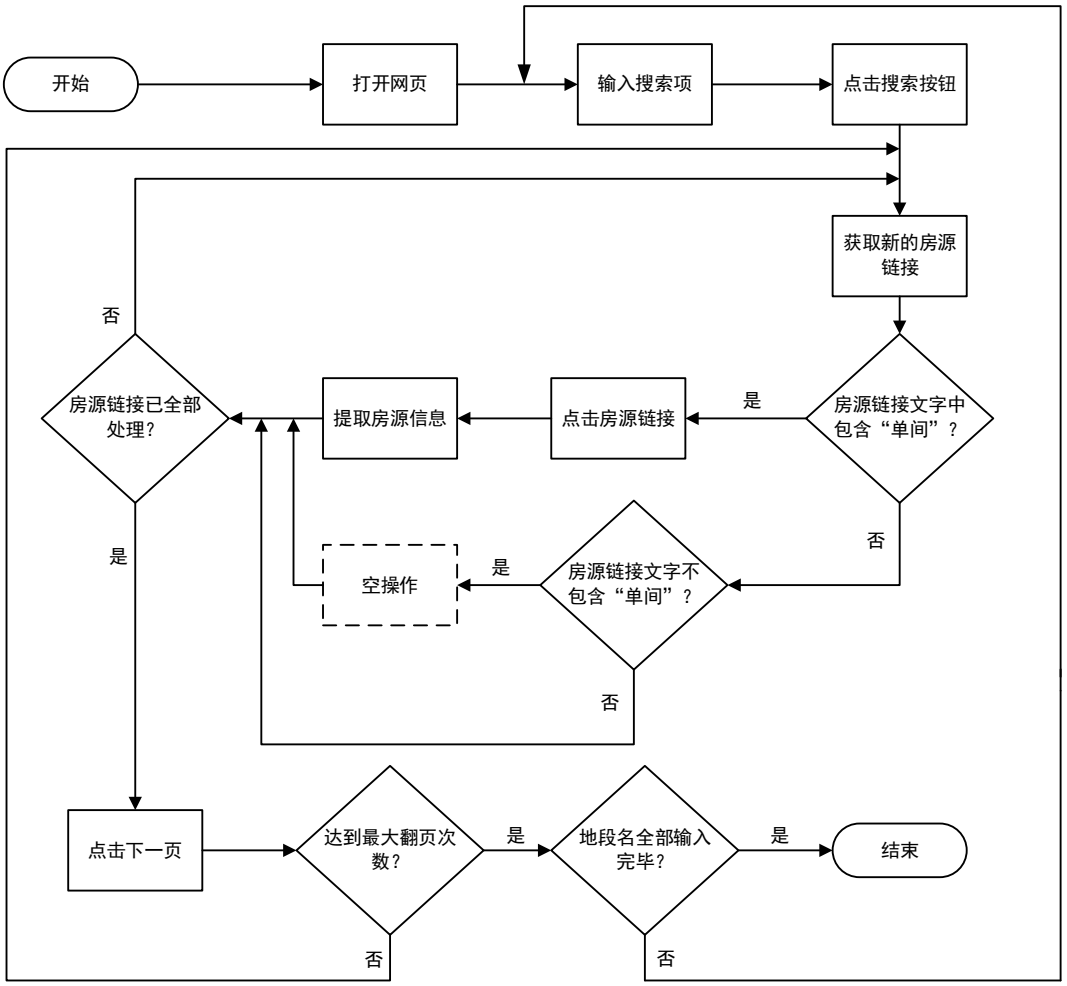


图 5.19 58 同城房源信息采集服务执行流程图

至此，第 5.6.1 节定义的数据采集需求已被全部满足，并以 Web 服务的形式存储到服务注册中心，等待服务请求者调用。

针对上述生成的 58 同城房源信息采集案例中的服务调用流程，其简单描述如下：

（1）本系统为服务请求者提供了如图 5.20 所示的调用操作界面。这里，我们将循环搜索参数的值修改为“云栖小镇”和“西溪湿地”，以查找云栖小镇和西

图 5.20 58 同城房源信息采集服务调用示例

溪湿地地区的房源信息。

（2）服务请求者点击图 5.20 下方的“获得任务 ID”按钮，向服务注册中心提交服务调用请求。这里，系统以 POST 方式将图中参数提交给服务注册中心，用户也可以自行以 POST 的请求方式提交调用请求。服务注册中心后台根据用户提供的服务 ID 和输入参数，定位并修改服务参数后，生成一条 Web 数据采集任务，并将任务 ID 返回给服务请求者。图 5.20 中获得的任务 ID 号为 44。

（3）服务请求者拿到任务 ID 号后，可根据自身需求，进行本地数据采集或者云采集。点击图 5.20 中下方的本地执行任务或者云端执行任务按钮，即可等待程序自动的按照服务请求者的需求，采集相应的数据，并存储到指定位置。

图 5.21 展示了本次 58 同城房源信息采集案例中采集到的部分信息，数据以 CSV 格式存储在用户本地。

至此，58 同城房源信息采集案例的服务生成和调用流程已全部介绍完毕，通过本例，读者可以更好的理解复杂数据采集场景的定义和相关功能的作用，以及系统展现出的易用性、实用性。

房源标题	房源描述	经纪人_文本	经纪人_链接
中大银座 云栖小镇 经济科技区	1．房屋亮点：押一付一朝南	马庆美(经纪人)	https://houserent.58.com/landlo
阿里巴巴 良户家苑 云栖小镇	0中介押一付一1．房屋亮点	刘幸(经纪人)	https://houserent.58.com/landlo
单身公寓|阿里云旁 押一付一	【龙湖冠寓杭州云栖小镇店	金嘉明(经纪人)	https://houserent.58.com/landlo
雄镇楼，凤山花苑，建国南苑	建筑面积为171平米。独立卫	邹静静(经纪人)	https://houserent.58.com/landlo

图 5.21 58 同城房源信息采集服务部分采集结果展示

5.7 本章小结

本章详细介绍了复杂数据采集分系统的设计与实现，包括服务流程定义模块、客户端、服务流程管理模块和程序操作处理模块的架构、功能及相关操作和算法的实现细节。最后，本章通过对 58 同城房源信息采集案例的介绍，证明了系统的易用性和实用性。

第6章 系统测试与应用

本章将分别对简单数据采集场景和复杂数据采集场景下的 Web 应用数据采集服务进行测试，对两个场景对应分系统中的服务本身和服务逻辑中的算法进行评估，评估指标包括服务可达性、服务准确率等。最后，本章介绍了系统在高分服务网格平台中封装的 Web 服务的部署和运行情况，证明了系统在不同场景下的高效性、实用性。

6.1 简单数据采集分系统测试与分析

6.1.1 测试环境配置

简单数据采集场景下，根据 Web 页面的不同类别[60]，本文定义了以下三种类型的 Web 页面对简单数据采集分系统进行测试：

1) 列表结构页面。主要信息块为列表块的页面，列表结构页面是 Web 数据采集任务中最常见的采集页面类型。系统测试中使用到的列表结构页面数目为：1800。图 6.1 展示了列表结构页面的类别占比图，系统对图中各类别页面进行了预处理，以保证页面各块中子块的结构完全相同。

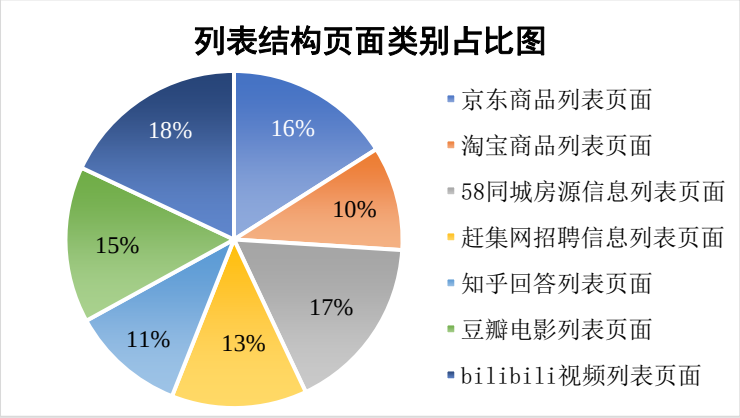


图 6.1 列表结构页面类别占比图

2) 文本信息页面。主要信息块为大块文本信息的页面，大多数新闻类页面属于文本信息页面。系统测试中使用到的文本信息页面数目为：2600。图 6.2 展示了文本信息页面的类别占比图。

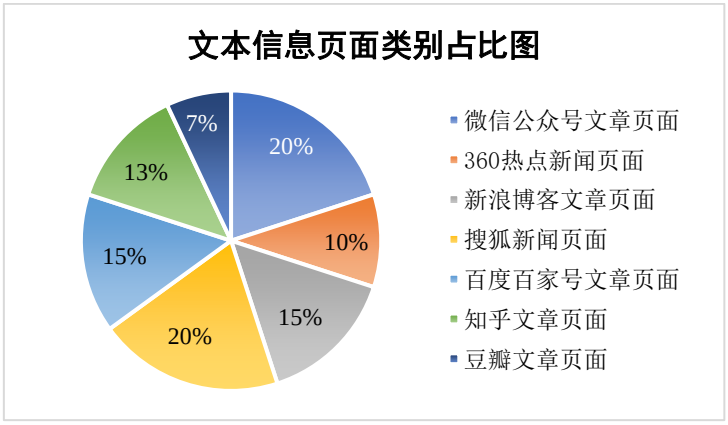


图 6.2 文本信息页面类别占比图

3) 查询表单页面：含有动态表单的页面，几乎所有的门户网站都存在搜索表单，因此对动态表单处理模块性能的测试也十分必要。对查询表单页面的表单进行提交后，系统还会同时测试其数据页面的准确率等指标。系统测试中使用到的动态表单页面数目为：300。图 6.3 展示了查询表单页面的类别占比图。

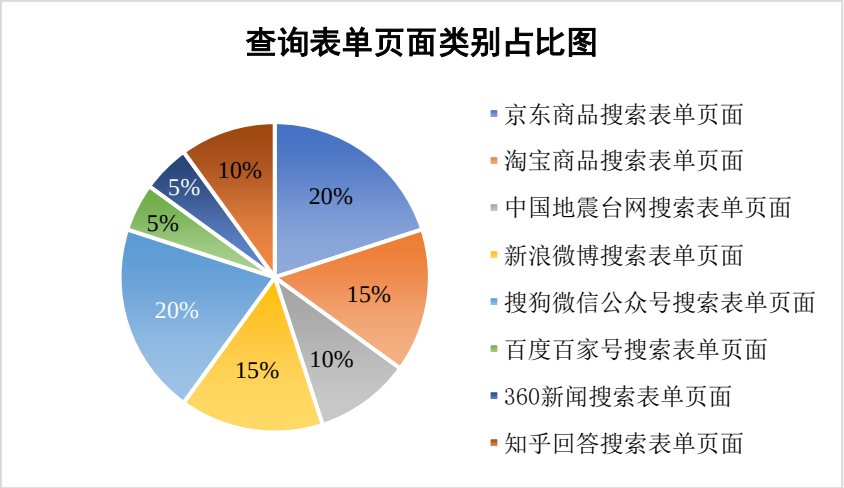


图 6.3 查询表单页面类别占比图

上述三种类型的表单页面以大规模的数据量覆盖了第 3.2.1 节描述的简单数据采集场景下的不同类别的特征，同时覆盖了对表单页面的测试，体现了测试集合的完备性。

本文将分别对简单数据采集分系统的服务生成和服务调用阶段的性能进行测试。结合第 2.2.1 节定义的 Web 服务的服务质量 *QoS* 中的定义以及简单数据采集场景下各模块的功能，我们对系统中生成服务的**可达性、返回结果准确率**及服务生成和调用过程中涉及到每一个模块的**处理时间，可用性和采集速率**等指标进

行了测试和分析，且所有的测试结果均为各个类型页面测试后的平均值，具体测试结果在第 6.1.2 节中讲述。

本次测试的环境为：32 核心的 Intel(R) Xeon(R) E5-2609 v3 CPU 和 128 GB 内存，测试操作系统为 Ubuntu 18.04，带宽为 100Mbps。

6.1.2 测试结果展示与分析

6.1.2.1 服务性能测试

表 6.1 展示了简单数据采集分系统下服务的性能测试结果。如表 6.1 所示，系统内各种类型页面的 Web 数据采集服务的生成操作的成功率均为 99%以上，只要页面可以正常加载，即可对页面进行服务生成的操作。在服务调用阶段，三种类型页面下的数据采集服务的平均响应时间均不超过 300ms，可达性为 99%以上，且在页面结构没有发生变化的情况下，返回的 Web 数据的准确率为 99%以上，只有少部分情况下因为网络带宽问题导致页面无法加载，此时，系统会自动重试，并成功采集到指定的数据。

表 6.1 简单数据采集分系统服务性能测试结果

测试指标	列表结构页面	文本信息页面	查询表单页面
服务生成成功率	99.48%	99.54%	99.67%
服务响应时间	231ms	215ms	244ms
服务调用可达性	99.97%	99.89%	99.91%
服务返回结果准确率	99.23%	99.15%	99.76%

6.1.2.2 Web 数据采集服务生成阶段测试

表 6.2 展示了简单数据采集分系统下服务生成阶段的性能测试结果。

表 6.2 服务生成阶段性能测试结果

测试指标	列表结构页面	文本信息页面	查询表单页面
表单查找时间（s）	/	/	9.0
页面分块时间（s）	59.9	47.2	50.6
块排序时间（s）	9.3	2.6	5.3

表 6.2 服务生成阶段性能测试结果（续）

测试指标	列表结构页面	文本信息页面	查询表单页面
信息生成时间（s）	0.15	0.11	0.13
提取规则长度（行）	3180	2854	2941
Pr.F(%)	/	/	97.6
Pr.1(%)	88.8	73.1	77.3
Pr.3(%)	96.3	92.3	93.4
Pr.5(%)	98.8	96.2	96.8

由表 6.2 可知，在服务生成阶段，页面分块操作所占时间最久，其他操作时间占用较少。总体来看，一个 Web 服务的生成时间不超过两分钟，相比于如 XWRAP 等工具需要花费至少三十分钟来生成一个服务的系统，本系统大大降低了用户生成服务所需要耗费的时间，提高了服务生成的效率。

我们同时对页面分块和块排序算法的准确率进行了评估。表 6.2 中，Pr.F 表示定位到用户想要的搜索表单的准确率；Pr.1, Pr.3, Pr.5 分别代表页面分块并排序后，最重要的块排在第一位，前三位和前五位的概率。由于如何定义哪个表单是用户想要使用的表单，以及哪个块是最重要的块具有较大的主观性，本文将按照普遍的认知，如在新闻页面中人们普遍认为正文内容是最重要的块，商品页面中普遍认为商品列表块是最重要的块等，人工的对各项准确率进行判断。

由表 6.2 可知，97.6%的情况下，系统可以成功的识别和定位到用户想要填写的表单；而在页面分块和块排序操作执行结束后，三种类型的页面中最重要的块排在前五位的几率均超过 96%，证明了页面分块算法和排序算法的有效性。

对于参数含义分析模块是否成功的生成了与参数示例值的语义相关的参数名称，本文同样进行了测试，结果证明，在 74.3%的情况下，通过 *openKG.cn 中文通用百科*知识图谱工具推测出的参数类型符合参数示例值的语义。知识图谱仍然是正在发展的技术，相信在不久的将来，这一准确率会有较大提升。

6.1.2.3 Web 数据采集服务调用阶段测试

表 6.3 展示了简单数据采集分系统下服务调用阶段的性能测试结果。

表 6.3 服务调用阶段性能测试结果

测试指标	列表结构页面	文本信息页面	查询表单页面
文档大小（byte）	285881	187382	228064
文档 DOM 树长度	1075	878	1023
提取结果 JSON 大小（byte）	9992	22094	18203
页面加载时间（ms）	1601	1966	1869
表单处理时间（ms）	/	/	988
数据提取时间（ms）	4486	5385	5145
数据过滤时间（ms）	535	327	382
总时间（ms）	6623	7678	8384
数据采集速率（条/分钟）	642	574	617

表 6.3 中记录了不同页面的平均 HTML 文档大小和 DOM 树长度（即树中的节点个数），以及最终提取到数据的 JSON 文件的平均长度，并记录了相应的表单处理和数据的平均提取时间。由表 6.3 可知，文档树长度和文档大小成正比；而在通常情况下，由于文本信息页面中的文本在 HTML 源码中所占比例偏高，而列表结构页面中的文本在 HTML 源码中所占比例相对较低，因此提取到的数据 JSON 长度与文档大小并无直接比例关系。

由表 6.3 可知，三种类型页面的加载时间基本相同，而列表结构页面的数据提取时间相对其他两种类型页面较短，这是由于列表结构页面中的单条数据量通常较小，在程序执行阶段提取单个元素的时间会随着元素的长度增加而显著增加导致。而列表结构页面的数据过滤时间相比于其他类型页面要长，这是因为列表结构页面中的信息条目数通常较多，因此需要较长的处理时间。处理表单的时间大约为 1 秒左右，而完成一个数据页面中所有数据的采集的总时间不超过十秒，系统对不同类型页面的数据平均采集速率约为 600 条/分钟。

由系统测试结果可知，在简单数据采集场景下，系统可在较短的时间内生成 Web 数据采集服务，并以较快的速度进行数据采集工作并返回采集结果。简单数据采集场景在一定程度上满足了用户采集 Web 数据的要求，使用户在没有任何专

业经验的情况下，实现了 Web 数据采集的目标。

6.2 复杂数据采集分系统测试与分析

6.2.1 测试环境配置

为了证明复杂数据采集分系统的高可用性、可扩展性和高效性，本节设计了以下 8 个数据采集服务用于系统测试，以覆盖复杂数据采集场景下服务生成和调用流程中涉及到的所有操作。S 代表服务（Service）：

1) S1：百度进入地震台网的地震信息采集服务。服务流程为：打开百度首页，在搜索框中输入“中国地震台网”，并点击第一个展示的链接，进入中国地震台网首页，然后点击“历史查询”按钮，进入“历史查询”页面。接下来，和第 4.3 节中的流程类似，在表单中输入经纬度，点击搜索按钮得到地震数据，并采集地震数据信息。此服务主要对点击元素操作在跨站点之间操作的可行性进行验证。

2) S2：京东商品列表页面商品信息采集服务。服务流程为：打开京东网站首页，循环的在搜索框中输入商品名称，如“电脑”,“手机”等，然后采集京东商品列表中含有“京东自营”标识的商品的基本信息，如商品名称，链接，店铺名称，价格等。此服务主要针对第 5.2.2 节中的选中子元素并生成数据参数的算法（算法 4）进行验证。

3) S3：浙江大学学生信息管理系统信息采集服务。服务流程为：打开浙江大学学生信息管理系统的学生列表页面，并循环的打开页面中每一个学生的信息详情页，采集页面内学生的各项基本信息，这些信息存在于页面表单输入框内。此服务主要针对表单标签页面采集的可行性进行验证。

4) S4：浙江大学 CC98 论坛心灵贴主要内容和评论采集服务。服务流程为：打开浙江大学 CC98 论坛的登陆页面，输入用户名和密码进行登陆，并跳转到心灵之约板块，挨个采集每个帖子的标题，主要信息以及主要评论。此服务主要针对登陆流程的控制以及循环打开链接进入详情页的数据采集情形进行验证。

5) S5：多 URL 地址服务集市信息采集服务。服务流程为：循环的对多条服务集市 URL 中每一个 URL 对应的 Web 页面中的服务信息进行采集。此服务主要针对多 URL 地址情形下系统采集的稳定性进行验证。

6) S6：微博数据采集服务。服务流程为：打开微博网站中某博主的微博页面，向下拉滚动条直到显示全部的微博数据，然后对每条微博的正文内容进行采集。此服务主要针对复杂结构页面下的同类型元素匹配算法（算法 3）进行验证。

7) S7：微信公众号文章内容采集服务。服务流程为：打开某微信公众号的文

章列表，并循环点击每个文章链接，然后采集每一篇文章的标题及正文内容，包括文字和图片信息。此服务主要针对文章类型页面的采集准确率进行验证。

8) S8：58 同城房源信息采集服务。即第 5.6 节中定义的服务。此服务主要对第 5.5 节的程序执行操作模块在复杂任务流程下的执行鲁棒性进行验证。

由上述服务的描述可知，复杂数据采集场景下的所有操作节点均被覆盖，具有测试集合完备性的特征。对于每个数据采集服务，系统将会连续采集最大 1000 页的 Web 数据，同时，系统将对每个服务的生成和调用流程测试 100 次，并取测试的平均结果，以满足测试过程的规模性需求。

本节将对上述服务的**可达性、返回结果准确率**及服务生成和调用流程中在算法 2 到 6 中定义的各个操作的**执行准确率和执行时间**进行测试，以分析和证明系统的可用性和高效性。

本次测试的环境为：32 核心的 Intel(R) Xeon(R) E5-2609 v3 CPU 和 128 GB 内存，测试操作系统为 Windows 10 v1903，带宽为 100Mbps。

6.2.2 测试结果展示与分析

6.2.2.1 服务性能测试

表 6.4 展示了复杂数据采集分系统中服务的性能测试结果。如表 6.4 所示，系统中各类数据采集服务生成的成功率均为 99%以上，同样只要页面可以正常加载，即可生成服务。在服务调用阶段，各种数据采集服务的可达性同样为 99%以上，部分情况下页面由于网络问题无法加载，需要系统重试后进行数据采集。最后，各服务返回结果的准确率为 99%以上，证明了服务的鲁棒性。

表 6.4 复杂数据采集分系统服务性能测试结果（%）

测试指标	S1	S2	S3	S4	S5	S6	S7	S8
服务生成成功率	99.99	99.99	99.99	99.99	99.99	99.99	99.99	99.99
服务调用可达性	99.89	99.68	99.91	99.96	99.95	99.32	99.58	99.13
服务返回结果准确率	99.91	99.95	99.94	99.95	99.96	99.91	99.94	99.97

图 6.4 展示了复杂数据采集分系统中的服务响应和任务生成时间。由图 6.4 可知，在服务调用阶段，所有服务的平均响应时间均不超过 300ms，系统处理服务请求生成数据采集任务的平均时间均不超过 350ms，即系统从收到服务请求到任

务生成操作结束所耗费的时间共 600ms 左右，短暂的服务响应时间证明了系统执行的高效性，这极大的提升了用户进行二次开发的效率。

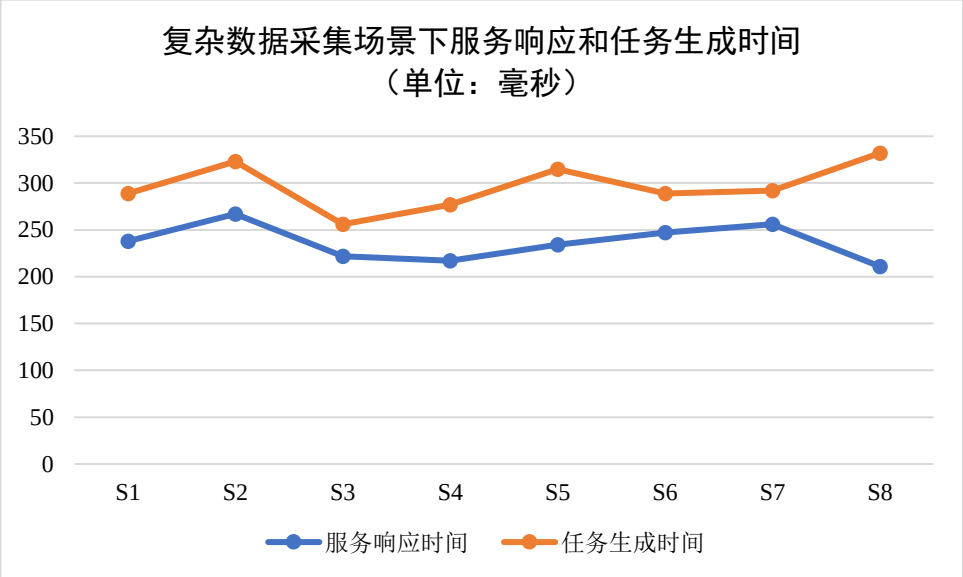


图 6.4 复杂数据采集场景下服务响应和任务生成时间（ms）

6.2.2.2 Web 数据采集服务生成阶段测试

表 6.5 展示了服务生成阶段同类型元素自动匹配和手动匹配算法在各个服务下的执行结果，其中：C1（情形 1，Condition 1）表示匹配到了所有想要匹配的同类型元素，且没有任何多余元素被匹配到；C2 表示匹配到了所有想要匹配的同类型元素，但有多余元素被同时匹配到；C3 表示只匹配到了部分同类型元素，另有部分同类型元素未被匹配到，C4 表示没有匹配到任何同类型元素。

表 6.5 同类型元素自动匹配和手动匹配算法执行结果

操作名	S1	S2	S3	S4	S5	S6	S7	S8
同类型元素自动匹配算法	C2	C1	C1	C2	C1	C2	C3	C1
同类型元素手动匹配算法	C1	/	/	C1	/	C1	C1	/

由表 6.5 可知，通过同类型元素自动匹配算法，所有服务情景下均可以匹配到部分同类型元素，但会出现匹配元素过多或过少的情况，如 S1 中会将地震数据的表头同时匹配到，S4 多匹配了分页元素块，S6 中多匹配了广告块，S7 中文

章所有段落没有一次性匹配完全等。此时，需要使用同类型元素手动匹配算法（配合参数示例数据删除操作）删掉多匹配的元素，并匹配未被自动匹配算法匹配到的同类型元素。

同时，所有服务在生成阶段的元素待选、元素选中、扩大选区、操作台显示、撤销选择、取消选择、选中子元素、选中全部元素、数据参数生成、示例数据删除、移动到元素/点击元素、循环点击元素、输入文字/切换下拉选项、消息交互、增加节点、删除节点、剪切节点、复制节点和服务生成操作均成功执行，没有出现执行错误。

表 6.6 展示了服务生成阶段各操作在各服务下的平均执行时间。

表 6.6 服务生成阶段各操作的平均执行时间（ms）

操作名	S1	S2	S3	S4	S5	S6	S7	S8
同类型元素自动匹配算法	**13.4**	**12.9**	**11.6**	**13.6**	**14.7**	**15.5**	**20.1**	**13.2**
同类型元素手动匹配算法	**6.4**	**/**	**/**	**7.3**	**/**	**7.8**	**6.9**	**/**
元素待选	<10	<10	<10	<10	<10	<10	<10	<10
元素选中	<10	<10	<10	<10	<10	<10	<10	<10
扩大选区	<10	<10	<10	<10	<10	<10	<10	<10
操作台显示	<100	<100	<100	<100	<100	<100	<100	<100
撤销选择	<20	<20	<20	<20	<20	<20	<20	<20
取消选择	<20	<20	<20	<20	<20	<20	<20	<20
选中全部元素	<20	<20	<20	<20	<20	<20	<20	<20
示例数据删除	<10	<10	<10	<10	<10	<10	<10	<10
移动到元素/点击元素	<20	<20	<20	<20	/	<20	<20	<20
循环点击元素	/	/	<25	<25	/	/	<25	<25

表 6.6 服务生成阶段各操作的平均执行时间（ms）（续）

操作名	S1	S2	S3	S4	S5	S6	S7	S8
输入文字/切换下拉选项	<50	<50	<50	<50	/	<50	/	<50
消息交互	<30	<30	<30	<30	<30	<30	<30	<30
增加节点	<30	<30	<30	<30	<30	<30	<30	<30
删除节点	<10	<10	<10	<10	<10	<10	<10	<10
剪切节点	<30	<30	<30	<30	<30	<30	<30	<30
复制节点	<30	<30	<30	<30	<30	<30	<30	<30
服务信息生成	<50	<50	<50	<50	<50	<50	<50	<50

由表 6.6 可知，绝大多数操作的执行时间在 100 毫秒以内，包括同类型元素自动/手动匹配算法，且自动匹配算法的耗时要高于手动匹配算法，这是因为自动匹配算法需要对 HTML DOM 树进行递归搜索，最坏情况下将遍历整个 HTML DOM 子树；而手动匹配算法只需要根据已选中元素的路径进行一次遍历，即可找到与当前已选中元素同类的元素。

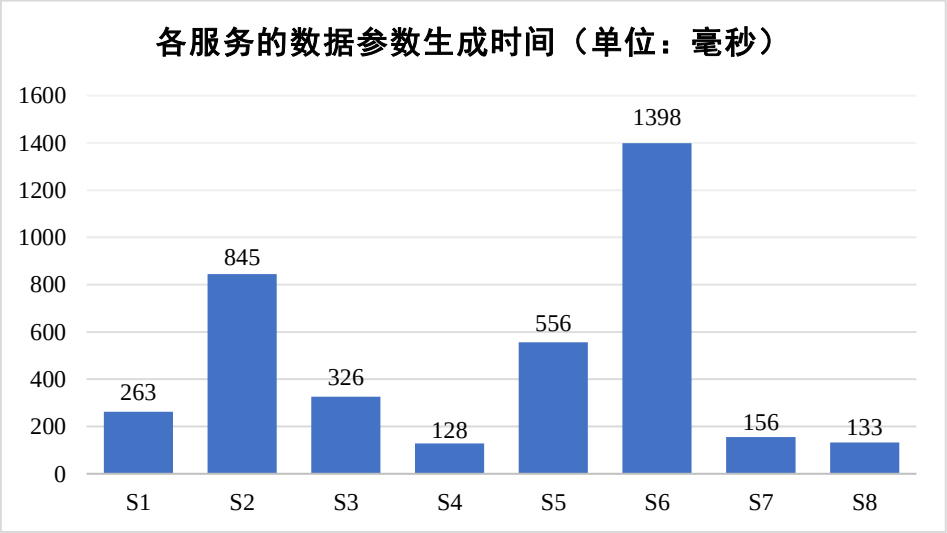


图 6.5 各服务的数据参数生成时间（ms）

在服务生成阶段，最耗时的操作为数据参数生成操作。如图 6.5 所示，不同服务的参数生成时间有明显不同，其原因为数据参数生成的操作分为第 5.2.2.1 节

中讲述的 ABCD 四种情况，即当生成 m 个参数，每个参数含有 n 个示例值时，参数的生成时间与 m×n 成比例关系。由于 S2、S5、S6 服务中包含选中子元素+选中全部的操作，因此其数据参数的生成时间相对较长，且 S6 服务中每条微博信息中子元素数目相比于其他服务多，因此，S6 服务的数据参数生成时间最长。

相比于八爪鱼采集器，本系统对 S1 到 S8 服务进行同类型元素自动匹配和手动匹配时匹配元素的准确率更高，所耗时间更短；在 S6 微博数据采集服务中，八爪鱼采集器的数据参数生成算法平均耗时为 34 秒，而本系统的耗时为 828 毫秒，速度显著提升，且本系统可以检索到所有选中元素中的所有子元素，包含不同位置的文本节点元素，而八爪鱼采集器则无法做到子元素的全部匹配功能，证明了本系统各匹配算法的准确性和高效性。

结合表 6.6 可知，对于一个可以熟练使用本系统的用户来说，生成 S1 到 S8 任意一个 Web 数据采集服务的时间将不超过五分钟，相比于手工编写爬虫程序来完成 Web 数据采集服务的方式，使用面向 Web 应用的智能化服务封装系统极大的简化了用户的操作流程，并节省了用户的时间，提高了用户的工作效率。

6.2.2.3 Web 数据采集服务调用阶段测试

在服务调用阶段，本文对每个服务调用流程中涉及到的所有操作，即服务请求处理、打开网页、点击元素、提取数据、输入文字、切换下拉选项、鼠标移动到元素、页面滚动操作，以及在四种不同类型下的循环操作、五种不同类型下的条件判断和分支操作进行了测试，结果表明，程序可以按照设定好的参数信息正确的执行上述所有的操作，并得到期待的采集结果。然而，在 S1 服务中，系统无法在第 2 页后正确的执行点击下一页的操作，这是因为在服务生成阶段，由服务流程定义模块定位到的下一页按钮的 XPath 为：

```
//ul[@id="paging"]/li[10]
```

即认为下一页按钮处于 ul 标签的第 10 个位置，而如图 6.6（a）所示，在第一页中，下一页按钮的位置位于当前页码按钮列表的第 10 位，因此可以正确的进行翻页操作；而在第二页中，如图 6.6（b）所示，下一页按钮的位置处于当前页码按钮列表的第 12 位，第 10 位的元素为第 8 页的按钮，因此，系统将直接跳转到第 8 页进行数据采集。在此流程中，系统本身按照设定好的 XPath 执行点击操作并无问题，但由于不同页面中的翻页按钮位置不同，所以导致数据采集服务无法按照预期进行。

（a） 第一页中下一页按钮位置示意 （b） 第二页中下一页按钮位置示意

图 6.6 中国地震台网地震信息采集页面分页元素定位示意

此时，我们需要手动的在服务流程管理模块中修改流程中翻页循环操作参数中的 XPath 为：

//ul[@id="paging"]/li[text()="»"]

即每次匹配元素都匹配文本内容为"»"的元素，这种 XPath 的写法不是根据位置定位，而是根据文本来定位，即修复了无法正常翻页的问题。因此，如果在循环操作的执行过程中出现网页结构变化的情况，系统可能无法按照自动生成的 XPath 进行正确的元素定位，此时就需要用户手动的去排查可能存在的问题，适当的修改相关操作的 XPath 以使程序可以正确的执行。相比于直接编写代码来进行数据提取的情形，用户使用本系统时只需了解基本的 XPath 知识就可以对服务进行全方位的设置及排错，这很大程度上降低了用户进行 Web 数据采集的门槛。

表 6.7 展示了各个服务在服务调用阶段中涉及到的相关操作的平均执行时间。

表 6.7 服务调用阶段各操作的平均执行时间（ms）

操作名	S1	S2	S3	S4	S5	S6	S7	S8
打开网页	**1287**	**3756**	**5733**	**2365**	**1923**	**4562**	**2745**	**8796**
点击元素	**1365**	**3842**	**5626**	**2311**	**/**	**4489**	**2862**	**7832**
页面滚动	/	<30	/	/	/	<30	/	/
输入文字/切换下拉选项	**604**	**886**	**/**	**374**	**/**	**/**	**/**	**656**
循环	<20	<20	<20	<20	<20	<20	<20	<20
条件判断	/	<30	/	/	/	/	/	<30
数据平均采集速度（条/分钟）	**2239**	**480**	**479**	**779**	**468**	**729**	**833**	**634**

表 6.7 中记录的时间均为操作的单次处理的平均时间。由表 6.7 可知，在服务调用过程中，比较耗费时间的操作有：打开网页、点击元素和输入文字/切换下

拉选项，其中，打开网页和点击元素操作都发生了网页加载的行为，因此受网络带宽和页面本身脚本和内容的影响，两者耗时最长。如在 S8 服务中，打开 58 同城任意房源信息页面后，要采集的元素一般会在 5 秒之内出现，但整个页面完全加载则需要 20 秒以上的时间。为了避免无谓的时间浪费，本系统设置了最大加载时间为 10 秒，即 10 秒后不论页面是否加载完成，均停止加载并进行下一步操作。由于表 6.7 中的循环、条件判断以及其他相关操作的执行时间较短，因此基本上可被忽略。本系统对各个操作的处理时间与八爪鱼采集器大致相同。

由表 6.7 可知，不同服务的数据平均采集速度不同，这是由于不同的服务要采集的 Web 页面的复杂程度不同，因此元素定位时，需要检索和进行元素定位的时间就不同。如 S1 服务中的中国地震台网历史查询页面 HTML DOM 树结构比 S2 服务中的京东商品列表页面简单，因此其数据采集速度较 S2 服务快。同时，每行数据包含的参数字段数目不同，S1 服务中每行数据只有 6 个字段，而 S2 服务中有 24 个字段，因此 S2 服务的单条数据采集速度慢于 S1 服务。本系统的数据采集速率与八爪鱼采集器大致相同，采集一行数据的时间大约为 100 毫秒，即 600 条/分钟左右。最后，系统针对大规模数据采集任务执行的稳定性在 95%以上。

由系统的测试结果可知，复杂数据采集场景下，系统可在较短的时间内完成了服务请求和任务生成的处理，并以较快的采集速率和准确率保证了服务的高质量性，证明了系统处理服务请求的高效性和稳定性。

6.3 高分服务网格应用展示

面向 Web 应用的智能化服务封装系统是依托于国家重点研发计划专项《高分辨率对地观测系统重大专项（民用部分）高分网格平台（二期）》中的高分服务网格平台开发，因此，本节介绍了智能化服务封装系统在高分服务网格平台的使用和部署情况，从而证明系统针对高分相关 Web 应用进行封装的适用性。

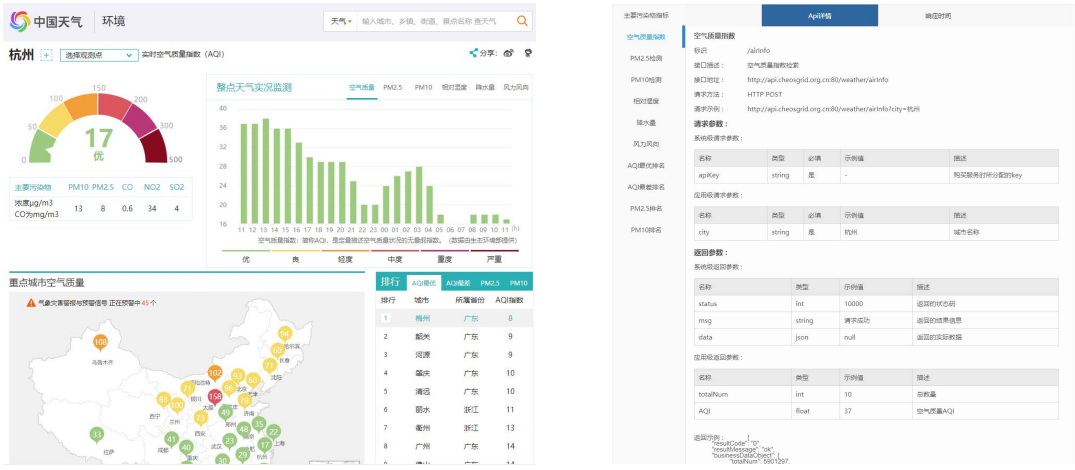
高分服务网格是高分二期项目中致力于推动高分数据的服务化而开发的平台，从而达到深化高分数据的大众应用，促进高分数据的产业化发展的作用。服务网格提供服务集成与接入、服务软件开发、服务运维管理等功能，从而实现高分应用服务的统一集成与接入、快速开发与测试及低成本的部署与运营。高分服务网格系统提供了针对高分相关服务的服务注册中心，如图 6.7（a）所示，高分服务网格平台门户首页包含多种服务类别，如基础卫星遥感影像服务、旅游气象服务、区域线下服务等，点击服务类别链接即可查看服务类别下的所有服务。如图 6.7（b）所示，我们使用智能化服务封装系统在高分服务网格平台上生成了多

种高分遥感信息服务，如新疆棉花种植区空间分布服务、安徽矿山开采检测服务等，这些服务的原始数据均来自各国家部委（如农业部）和地方资源管理局的 Web 网站，在对这些 Web 应用使用智能化服务封装系统进行封装后，即可统一集中管理和部署在高分服务网格平台上，以供服务请求者快速的检索和使用与高分业务相关的服务，获得自己想要的数据和信息，提高了行业用户的工作效率。

如图 6.8（a）所示，中国天气网环境信息页面包含某地区（如杭州）的环境质量信息，如 PM2.5 指数信息、降水量信息等。在使用智能化服务封装系统对图 6.8（a）中所有的图表进行封装后，即生成了图 6.8（b）左侧所示的 11 个 API 接口，每个 API 接口负责提取中国天气网 Web 页面中不同图表中的数据。用户可按照如第 3.2.3 节中的服务调用过程描述，以 Web 请求的方式调用服务响应的 API，以得到杭州地区相关环境信息数据。

（a）高分服务网格平台门户首页　　（b）由智能化服务封装系统生成的高分相关服务

图 6.7 高分服务网格平台门户界面



（a）环境信息 Web 页面　　（b）环境信息服务接口示例

图 6.8 中国天气网环境信息页面及服务接口示例

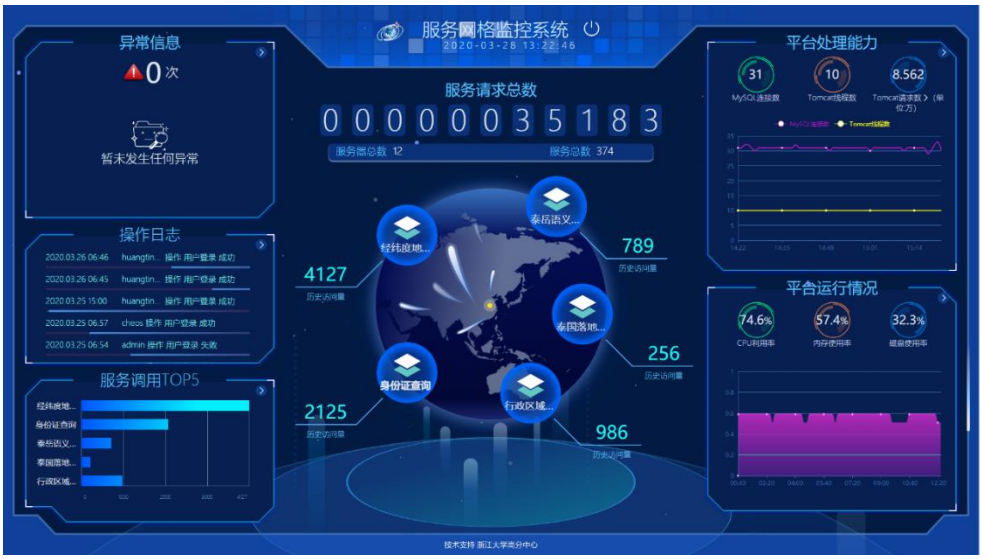


图 6.9 高分服务网格平台监控系统主页

图 6.9 展示了高分服务网格平台监控系统的主页，图中显示了高分服务网格平台中各服务和系统的部署和运行情况。由图 6.9 可知，高分服务网格平台的服务总数为 374 个，服务调用总次数已超过 3 万次。同时，监控系统中还记录了平台中各类用户（管理员、服务提供者、服务请求者）的操作日志，以及平台中各设备的运行状态，磁盘 IO 使用状态等，这有利于系统开发人员对系统进行实时的维护和监控。

图 6.9 中由智能化服务封装系统生成的高分相关服务数量为 128 个，涵盖土地资源、基础卫星遥感影像、旅游气象、地理热力信息等多种类别，同时，系统在高分服务网格平台生成了常用的第三方服务，如身份证信息查询服务，实时股票查询服务等。这些高分服务网格平台中的 Web 应用数据采集服务的总调用次数已超过 1 万次，总检索次数超过 1000 次，体现了面向 Web 应用的智能化服务封装系统对国家级重大专项的一定程度的支撑作用，为相关行业用户的工作提供了便利。

6.4 本章小结

本章对两种数据采集场景下的生成的服务及服务流程中涉及到相关算法进行了测试，详细介绍了不同场景下测试环境的配置过程，测试了服务的准确性、可达性及内部算法的准确率等指标，证明了各算法的有效性。最后，本章介绍了在高分服务网格平台中使用智能化服务封装系统生成的高分相关服务的运行和部署情况，体现了系统为相关行业用户提供的便利性。

第7章 总结与展望

7.1 全文总结

本文针对 Web 应用中的数据采集任务，设计了一个面向 Web 应用的智能化服务封装系统，使用户可以在没有任何编程经验的情况下，使用图形用户界面定制化的设计和生成 Web 数据采集服务。本文的主要工作如下：

（1）介绍了 Web 数据提取、Web 页面分块、Web 服务生成技术的相关工作，以及将流程图转换为可执行程序技术的相关研究。

（2）针对智能化服务封装系统涉及到的关键技术点进行介绍。

（3）介绍了简单数据采集场景下系统的需求、设计与实现，对系统的整体架构以及相关模块的实现进行了详细的介绍。

（4）介绍了复杂数据采集场景下系统的需求、设计与实现，对系统的整体架构以及相关模块中涉及的主要算法的实现细节进行了阐述。

（5）通过中国地震台网地震信息采集案例和 58 同城房源信息采集案例，展示了系统在简单数据采集场景和复杂数据采集场景下的操作流程和功能。

（6）对两种场景下系统的功能进行了测试与分析，同时介绍了系统在高分服务网格平台上封装的高分相关服务的运行和部署情况，证明了系统的高可用性、高效性和可扩展性，以及对行业用户提供的便利性。

7.2 未来展望

面向 Web 应用的智能化服务封装系统中的部分功能是通过调用第三方的服务来实现。为了让系统更具完备性，我们还可以在以下方面继续进行研究：

（1）设计针对服务的知识库，并生成对应的知识图谱网络，以完成服务参数名称的推理功能，提升参数含义分析模块的准确性。

（2）基于机器学习技术，构建图形验证码识别模型，并实现对更多类型验证码的处理，从而满足更多情况下的数据采集需求。

（3）基于 Hadoop 技术构建分布式的数据采集系统，满足在云端进行大规模的数据采集的需求，并解决高并发采集任务下的负载均衡问题。

我们相信，通过不断的完善与改进，面向 Web 应用的智能化服务封装系统一定可以满足各种类型的数据采集需求，从而帮助到各类具有数据采集需求的用户。

参考文献

[1] 中共中央网络安全和信息化委员会办公室, 第 44 次《中国互联网络发展状况统计报告》[EB/OL] 2019,8, http://www.cac.gov.cn/2019-08/30/c_1124939590.htm. Accessed February 28, 2020.

[2] Laender A H F, Ribeiro-Neto B A, Da Silva A S, et al. A brief survey of web data extraction tools[J]. ACM Sigmod Record, 2002, 31(2): 84-93.

[3] Ferrara E, De Meo P, Fiumara G, et al. Web data extraction, applications and techniques: A survey[J]. Knowledge-based systems, 2014, 70: 301-323.

[4] Zhang S, Balog K. Web Table Extraction, Retrieval, and Augmentation: A Survey[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2020, 11(2): 1-35.

[5] Hernández I, Rivero C R, Ruiz D. Deep Web crawling: a survey[J]. World Wide Web, 2019, 22(4): 1577-1610.

[6] Ristoski P, Gentile A L, Alba A, et al. Large-scale relation extraction from web documents and knowledge graphs with human-in-the-loop[J]. Journal of Web Semantics, 2020, 60: 100546.

[7] Fayzrakhmanov R R, Sallinger E, Spencer B, et al. Browserless web data extraction: challenges and opportunities[C]//Proceedings of the 2018 World Wide Web Conference. 2018: 1095-1104.

[8] Liu L, Pu C, Han W. XWRAP: An XML-enabled wrapper construction system for web information sources[C]//Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073). IEEE, 2000: 611-621.

[9] Liu Z, Li F, Ng W K. Wiccap data model: Mapping physical websites to logical views[C]//International Conference on Conceptual Modeling. Springer, Berlin, Heidelberg, 2002: 120-135.

[10] Liu Z, Ng W K, Li F, et al. A visual tool for building logical data models of websites[C]//Proceedings of the 4th international workshop on Web information and data management. 2002: 92-95.

[11] Potvin B, Villemaire R. Robust Web Data Extraction Based on Unsupervised Visual Validation[C]//Asian Conference on Intelligent Information and Database Systems. Springer, Cham, 2019: 77-89.

[12] Phan X H, Horiguchi S, Ho T B. Automated data extraction from the web with conditional models[J]. International Journal of Business Intelligence and Data

Mining, 2005, 1(2): 194-209.
[13] Kushmerick N, Weld D S, Doorenbos R. Wrapper induction for information extraction[M]. Washington: University of Washington, 1997.
[14] Muslea I, Minton S, Knoblock C. A hierarchical approach to wrapper induction[C]//Proceedings of the third annual conference on Autonomous Agents. 1999: 190-197.
[15] Baumgartner R, Flesca S, Gottlob G. Visual web information extraction with lixto[J]. 2001.
[16] Alison S, Engle R, Riesterer J, et al. Automated extraction of data from web pages: U.S. Patent Application 16/567,367[P]. 2020-1-2.
[17] 八爪鱼，八爪鱼采集器，[EB/OL], https://www.bazhuayu.com/, Accessed August 23, 2019.
[18] Cai D, Yu S, Wen J R, et al. Vips: a vision-based page segmentation algorithm[J]. 2003.
[19] Andrew J, Ferrari S, Maurel F, et al. Web page segmentation for non visual skimming[C]. 2019.
[20] Zeng J, Flanagan B, Hirokawa S, et al. A web page segmentation approach using visual semantics[J]. IEICE TRANSACTIONS on Information and Systems, 2014, 97(2): 223-230.
[21] Sarkis M, Concolato C, Dufourd J C. Msos: A multi-screen-oriented web page segmentation approach[C]//Proceedings of the 2015 ACM Symposium on Document Engineering. 2015: 85-88.
[22] Kravchenko A. BERYL: A System for Web Block Classification[M]//Transactions on Computational Science XXXIII. Springer, Berlin, Heidelberg, 2018: 61-78.
[23] Feng H, Zhang W, Wu H, et al. Web page segmentation and its application for web information crawling[C]//2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, 2016: 598-605.
[24] Liao C, Hiroi K, Kaji K, et al. An event data extraction method based on HTML structure analysis and machine learning[C]//2015 IEEE 39th Annual Computer Software and Applications Conference. IEEE, 2015, 3: 217-222.
[25] Sheng Q Z, Xu X, Chang R N, et al. Guest Editorial: Data-Centric Big Services[J]. IEEE Transactions on Services Computing, 2019, 12(3): 341-342.
[26] Baumgartner R, Campi A, Gottlob G, et al. Web data extraction for service creation[M]//Search Computing. Springer, Berlin, Heidelberg, 2010: 94-113.
[27] Tatsubori M, Takashi K. Decomposition and abstraction of web applications for

web service extraction and composition[C]//2006 IEEE International Conference on Web Services (ICWS'06). IEEE, 2006: 859-868.

[28] Zhou J J, Zou X, Shen B J, et al. Design of Service Wrapper Environment for Web Legacy System[J]. Computer Engineering, 2011, 37(19): 73-75.

[29] Jarrett J, Hemmings-Jarrett K, Blake M B. Towards a Service-Oriented Architecture for Pre-processing Crowd-Sourced Sentiment from Twitter[C]//2019 IEEE International Conference on Web Services (ICWS). IEEE, 2019: 163-171.

[30] Sarkar A, Bose J. A Web Service to Generate Intelligent Previews of Web Links[C]//2018 IEEE International Conference on Web Services (ICWS). IEEE, 2018: 327-330.

[31] Ali K, Hamilton M, Thevathayan C, et al. Social Information Services: A Service Oriented Analysis of Social Media[C]//International Conference on Web Services. Springer, Cham, 2018: 263-279.

[32] Cao H, Falleri J R, Blanc X. Automated generation of REST API specification from plain HTML documentation[C]//International Conference on Service-Oriented Computing. Springer, Cham, 2017: 453-461.

[33] Zhang F, Chen M, Ames D P, et al. Design and development of a service-oriented wrapper system for sharing and reusing distributed geoanalysis models on the web[J]. Environmental modelling & software, 2019, 111: 498-509.

[34] Wu X H, Qu M C, Liu Z Q, et al. Research and application of code automatic generation algorithm based on structured flowchart[J]. Journal of Software Engineering and Applications, 2011, 4(09): 534.

[35] Wu X H, Qu M C, Liu Z Q, et al. Automatic Conversion of Structured Flowcharts into Problem Analysis Diagram for Generation of Codes[J]. JSW, 2012, 7(5): 1109-1120.

[36] Supaartagorn C. Web application for automatic code generator using a structured flowchart[C]//2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS). IEEE, 2017: 114-117.

[37] Qu M, Cui N, Wu X, et al. A Novel Algorithm of Error Check and Code Generation for Structured Flowchart[J]. Journal of Harbin Institute of Technology (New Series), 2017, 24(4).

[38] Gong W J, Qu M C, Wu X, et al. The Verification of Structure Identification Algorithm and Error Detection Strategies for Structured Flowchart[J]. IERI Procedia, 2012, 2: 880-887.

[39] Yokoyama T, Axelsen H B, Glück R. Fundamentals of reversible flowchart languages[J]. Theoretical Computer Science, 2016, 611: 87-115.

[40] Ilayaranimangammal I, Palaniyappan S, Anbuselvan S. Generation of Self Acting code from Single dimension flow chart[J]. International Journal of Modern Trends in Engineering and Research (IJMTER), 2015, 2(3).

[41] Hussein B M, Salah A. A Framework for Model-Based Code Generation from a Flowchart[J]. International Journal of Computing Academic Research, 2013, 2(5): 167-181.

[42] Kosower D A, Lopez-Villarejo J J. Flowgen: Flowchart-based documentation for C++ codes[J]. Computer Physics Communications, 2015, 196: 497-505.

[43] Erl T. Service-oriented architecture (SOA): concepts, technology, and design. 2005[J].

[44] Gustavo A, Casati F, Kuno H, et al. Web services: concepts, architectures and applications[J]. 2004.

[45] 冯俐.爬虫技术综述[J].电脑知识与技术,2017,13(27):213-214.

[46] 周中华, 张惠然, 谢江. 基于 Python 的新浪微博数据爬虫[J]. 计算机应用, 2014, 34(11): 3131-3134.

[47] Clark J, DeRose S. XML path language (XPath) version 1.0[J]. 1999.

[48] J. J. Garrett et al., “Ajax: A new approach to Web applications” [J], 2005.

[49] Selenium Inc, automates browsers. That's it! [EB/OL]. https://www.selenium.dev, Accessed May 5, 2019.

[50] Google Inc, The Chromium Project [EB/OL]. https://www.chromium.org/, Accessed April 14, 2019.

[51] Google Inc., Chrome Extension Development [EB/OL]. https://developer.chrome.com/extensions/extension, Accessed February 14, 2020.

[52] Gay W W. Linux socket programming: by example[M]. Que Corp., 2000.

[53] Fette I, Melnikov A. The Websocket protocol[J]. 2011.

[54] Hancock B, Lee H, Yu C. Generating titles for web tables[C]//The World Wide Web Conference. 2019: 638-647.

[55] Novgorodov S, Guy I, Elad G, et al. Generating product descriptions from user reviews[C]//The World Wide Web Conference. 2019: 1354-1364.

[56] A. Singhal, “Introducing the knowledge graph: things, not strings,” Official google blog, vol. 5, 2012.

[57] jQuery, jQuery [EB/OL]. https://jquery.com, Accessed July 17, 2018.

[58] Microsoft, Windows API Index - Win32 apps | Microsoft Docs [EB/OL]. https://docs.microsoft.com/zh-cn/windows/win32/apiindex/windows-api-list, Accessed November 16, 2019.

[59] Cefsharp, CefSharp: .NET (WPF and Windows Forms) bindings for the Chromium Embedded Framework[EB/OL]. https://github.com/cefsharp/CefSharp, Accessed January 15, 2020.

[60] Ceri S, Fraternali P, Matera M. Conceptual modeling of data-intensive Web applications[J]. IEEE Internet Computing, 2002, 6(4): 20-30.

攻读硕士学位期间主要的研究成果

论文：

[1] Xi, M., Li, Y., Wei, Y., Wang, N., Yin, Y., Luo, Z., ... & Yin, J. (2019, July). A Scenario-Based Requirement Model for Crossover Healthcare Service. In 2019 IEEE World Congress on Services (SERVICES) (Vol. 2642, pp. 252-259). IEEE.

[2] Wang N, Luo Z, Lyu X, et al. Service Wrapper: a system for converting web data into web services[J]. arXiv preprint arXiv:1910.07786, 2019.

[3] Xi M, Luo Z, Wang N, et al. A Latent Feelings-aware RNN Model for User Churn Prediction with Behavioral Data[J]. arXiv preprint arXiv:1911.02224, 2019.

专利：

[1] 一种基于网页分割和搜索算法的服务封装方法，专利申请号：2019104474480

主要参与项目：

[1] 高分服务网格平台，主要负责服务开放平台的前端开发，以及智能化服务封装系统的设计与研究工作。

[2] 网易 UNO 流失检测分析项目，主要负责模型的搭建与实现，实验数据代码的编写及数据验证，论文的校验等工作。

致谢

不知不觉间，一个西安电子科技大学的本科生，已经接受了三年的系统训练，开始有能力自己去探索科学的世界。在浙江大学的三年，我接触到的人和事，让我从各个方面体验了一个社会的运作流程，也收获了宝贵的人生阅历。在此离别之际，我想感谢每一位为我这段人生带来收获的人。

首先，我要感谢我的导师尹建伟教授，尹老师从大四尚未入学时便与我保持联系，给我学业和工作上的指导，在研究生三年期间，尹老师从论文写作、专利申请、报告撰写、小组管理等各个方面给了我全方位的指导和照顾，尤其在我论文写作期间，尹老师对我的论文进行了四轮修改，使我能够毫无悬念的通过审核，顺利毕业，尹老师对学生认真负责的态度值得我一辈子去学习。

我还要感谢曾在实验室的罗智凌老师，罗老师是我在研究生科研阶段的启蒙老师，对我进行了严格的科研训练，同时亲自指导我的论文写作，并认真定期的检查我们的工作，从各个方面对我进行监督，使我对未来的科研工作充满了信心。此外，我要感谢李莹、沈正伟、邓水光、尚永衡和张鹿鸣老师对我工作的指导。

感谢我的父母，对你们的感恩存在于我人生中的各个阶段，无论我在何处，无论我在干什么，你们永远会在背后支持我，你们的角色永远不可替代。

十分感谢实验室的廖翔勇师兄和席萌同学，给我的生活带来了极大的欢乐，感谢唐文博、林博、郑邦蓬师兄对我的指导，以及汪俊祥师兄对我的人生教导，感谢陈锋、吕西亚、张金迪、张欢、蒋芳乔、谭哲越、崔颖华、程冠杰、王涛、陈裕、靳婷、蔡钰祥等同学三年来的互帮互助，我们的友谊长存。

尤其感谢杨子桐同学，你和我一起走过了近两年的青春岁月，一起收获，一起成长，不知我们的故事还能延续多久？

感谢新加坡国立大学的何丙胜老师和各位同学，我们的故事才刚刚开始。

最后，我要特别感谢所有为抗击新冠肺炎疫情付出时间甚至生命的医务工作者及各行各业负责协调的战疫英雄们，你们与病毒争分夺秒的抗争保护了千千万万中华儿女的生命，也为国家的发展赢得了时间，因为你们不顾一切的努力，才让我有机会坐在这里，写下这段话。祝愿每一位中华儿女身体健康，祝愿伟大的祖国繁荣富强，我在此立下誓言，将来定要学成归来，报效祖国。

王乃博

二零二零年六月于浙江大学玉泉校区