数据结构lab3实验报告

作者：卢虹宇-2023202269
日期：2024年12月9号

1. 需求分析

本实验旨在通过实现HTML CSS Selector，巩固和复习线性表、栈等基础数据结构，练习树相关数据结构的应用，熟悉在实际问题中合理设计和使用数据结构，并训练学生阅读和理解课外资料的能力。

1.1 输入的形式和输入值的范围

- **输入形式**：
  - **文件输入**：用户可以通过指定文件路径读取本地HTML文件。
  - **URL输入**：用户可以输入一个网址，程序将通过网络爬取该网页的HTML内容。
- **输入值范围**：
  - **HTML文件或URL指向的网页内容**应符合标准的HTML格式。
  - `CSS` **选择器输入**应符合 `CSS` 选择器的语法规范，仅支持实验要求中标注为“是”的选择器。

以下表格列出了支持的 CSS 选择器及其解释。部分选择器行已合并，以便更好地展示选择器的用途。

例子	Explanation
`.class`	选择 `class="intro"` 的所有元素。
`.class1.class2`	选择 `class` 属性中同时有 `class1` 和 `class2` 的所有元素。
`.class1 .class2`	选择作为类名 `class1` 元素后代的所有类名 `class2` 元素。
`#id`	选择 `id="firstname"` 的元素。
`#firstname`	选择 `id="firstname"` 的元素。
`*`	选择所有元素。
`p`	选择所有 `<p>` 元素。
`p.intro`	选择 `class="intro"` 的所有 `<p>` 元素。
`div, p`	选择所有 `<div>` 元素和所有 `<p>` 元素。
`div p`	选择 `<div>` 元素内的所有 `<p>` 元素。
`div > p`	选择父元素是 `<div>` 的所有 `<p>` 元素。
`div + p`	选择紧跟 `<div>` 元素的首个 `<p>` 元素。
`p ~ ul`	选择前面有 `<p>` 元素的每个 `<ul>` 元素。

例子	Explanation
`[target]`	选择带有 `target` 属性的所有元素。
`[target=_blank]`	选择带有 `target="_blank"` 属性的所有元素。
`[title~=flower]`	选择 `title` 属性包含单词 "flower" 的所有元素。
`[lang|=en]`	选择 `lang` 属性值以 "en" 开头的所有元素。
`a[href^="https"]`	选择其 `href` 属性值以 "https" 开头的每个 `<a>` 元素。
`a[href$=".pdf"]`	选择其 `href` 属性以 ".pdf" 结尾的所有 `<a>` 元素。
`a[href*="w3schools"]`	选择其 `href` 属性值中包含 "w3schools" 子串的每个 `<a>` 元素。

- **输出形式**：
  - **控制台输出**：程序将结果以文本形式输出到控制台。
  - **查询结果列表**：符合条件的HTML节点信息将以结构化的列表形式展示。
  - **操作结果**：用户可以选择对查询结果进行进一步的操作，如提取文本、HTML代码或链接地址。

1.3 程序所能达到的功能

- **功能1**：解析HTML内容，构建DOM树结构。
- **功能2**：支持基本的CSS选择器查询，包括类选择器、ID选择器、标签选择器、后代选择器、子选择器、相邻兄弟选择器和通用兄弟选择器。
- **功能3**：提取符合选择器条件的节点的内部文本、HTML代码以及链接地址（对于 `<a>` 标签）。
- **功能4**：支持对查询结果进行二次查询，以实现更复杂的数据抽取。
- **功能5**：提供用户交互界面，允许用户选择输入方式、执行查询并查看结果。

2. 概要设计

本程序主要由HTML解析器和CSS选择器两大模块组成，分别负责解析HTML内容并构建DOM树，以及基于CSS选择器进行节点查询。程序整体结构如下：

2.1 抽象数据类型定义

- `html_node`：
  - `parent`：指向父节点的指针。
  - `tag_name`：标签名，如`div`、`p`等。
  - `tag`：标签的HTML原文。
  - `son_node`：子节点的链表。
  - `attr`：标签的属性集合，存储键值对形式。
- `Stack`：
  - 基于链表实现的栈，用于在HTML解析过程中维护当前节点的层次关系。

- `Link_list`：
    - 双向链表实现，用于存储子节点或其他需要链式存储的数据。

2.2 主程序流程

程序的主流程包括以下几个步骤：

1. 用户选择输入方式（文件或URL）。
2. 程序读取并解析HTML内容，构建DOM树。
3. 用户输入CSS选择器查询语句。
4. 程序根据选择器查询符合条件的节点，并展示查询结果。
5. 用户可对查询结果进行进一步操作，如提取文本、HTML代码或链接地址。
6. 用户可选择重新查询或退出程序。

3. 详细设计

本节将详细描述程序中各类和函数的实现，包括HTML解析器和CSS选择器的具体算法。

3.1 数据类型实现

3.1.1 `html_node` 结构体

用于表示HTML文档中的每一个节点，包括标签名、属性、子节点等信息。

```
struct html_node
{
    html_node *parent;                // 父节点
    string tag_name;                  // 标签名
    string tag;                       // 标签的HTML原文
    Link_list<html_node *> son_node; // 子节点链表
    map<string, string> attr;         // 属性集合

    html_node(string t) : tag(""), tag_name(t), parent(nullptr), son_node(),
attr() {};
    html_node() : tag(""), tag_name(""), parent(nullptr), son_node(), attr() {}
};
```

3.1.2 `Html_parser`

负责解析HTML内容，构建DOM树结构，并提供提取文本和HTML代码的功能。

```
class Html_parser
{
private:
    html_node *root;

public:
    Html_parser() : root(nullptr) {};
    ~Html_parser() {
        clear();
    }
```

```
    html_node *get_root() { return root; }
    bool parse(string &htmlcontent); // 解析HTML文本
    bool parseFromURL(const string &url); // 通过URL解析HTML
    void file_input(); // 文件输入与解析
    void printall(); // 打印整个HTML树
    void clear(); // 清空解析树

    // 提取文本和HTML的辅助函数
    string extract_text(html_node *node, string parent_tag);
    string extract_html(html_node *node, string parent_tag);

    // 判断标签类型的辅助函数
    bool isSelfClosingTag(const string &tagname);
    bool isBlockTag(const std::string &tagName);
    bool isInlineTag(const std::string &tagName);
    bool is_SpecialBlockTag(const std::string &tagName);

private:
    // 解析标签和属性的辅助函数
    void parseTag(const string &tagcontent, string &tagname);
    void ParseAttr(const string &tagcontent, map<string, string> &attr);

    // 树遍历函数
    void tree_travel(html_node *Hnode, Node<string> *tag_name, std::string
&result, std::string sign);

    // 递归删除树节点
    void deleteTree(html_node *node);

    // 打印树的辅助函数
    void printTree(html_node *node, int depth);

    // 辅助函数：去除多余空格
    string trim_and_add_spaces(const string &input);

    // CURL回调函数
    static size_t WriteCallback(void *contents, size_t size, size_t nmemb, string
*userp);
};
```

3.1.3 `Css_selector` 类

负责解析CSS选择器语法，并在DOM树中执行查询操作，返回符合条件的节点集合。

```
class Css_selector
{
private:
    Html_parser *parser;
    vector<Node<html_node *> *> *answer;

public:
    Css_selector() : parser(new Html_parser()), answer(new vector<Node<html_node
*> *>) {}
    ~Css_selector() {
        delete parser;
```

```
        delete answer;
    }

    Html_parser *get_parser() { return parser; }
    vector<Node<html_node *> *> *get_answer() { return answer; }

    void query(string query, html_node *start);
    void print_result();
    void clear_answer();

private:
    tag *process_tag(string name);
    vector<Node<html_node *> *> *select_operation(tag *a, vector<Node<html_node
*> *> *b, char opt);
    void global_search(tag *a, vector<Node<html_node *> *> *b, Node<html_node *>
*node);
    string trim(const std::string &str);
};
```

3.2 伪码算法

3.2.1 主程序伪码

```
Procedure Main
    初始化 Css_selector 对象
    调用 file_input()
    While 用户未选择退出
        显示主菜单
        读取用户选择
        Switch (选择)
            Case 1:
                重新选择文件进行解析
            Case 2:
                输入查询语句
                调用 query(查询语句, root节点)
                显示查询结果
                While 用户未选择重新查询
                    显示查询结果操作菜单
                    读取用户选择
                    Switch (选择)
                        Case 0:
                            输入节点索引
                            输入新的查询语句
                            调用 query(新的查询语句, 选择的节点)
                            显示新查询结果
                        Case 1:
                            输入节点索引
                            输出节点的内部文本
                        Case 2:
                            输入节点索引
                            输出节点的内部HTML
                        Case 3:
                            输入节点索引
                            输出节点的 href 属性
                        Case 4:
```

```
                        重新打印查询结果列表
                    Case -1:
                        退出查询结果操作菜单
                    Default:
                        提示无效输入
            Case 3:
                打印解析出的HTML文件
            Case -1:
                退出程序
            Default:
                提示无效输入
End Procedure
```

3.2.2 CSS选择器查询伪码

```
Procedure query(query_str, start_node)
    分割多个子查询（逗号分隔）
    For 每个子查询
        初始化栈
        解析子查询字符串，分离标签和运算符
        While 栈不为空且运算符栈不为空
            获取当前标签和运算符
            根据运算符执行相应的选择操作
            将符合条件的节点加入结果集
    去重结果集
    保存到 answer
End Procedure
```

4. 调试分析

4.1 调试过程中遇到的问题及解决方法

在开发过程中，遇到了以下几个主要问题：

- **自闭合标签识别**：部分HTML标签是自闭合的，如`<img>`、`<br>`等。实现自闭合标签的识别函数`isSelfClosingTag`，并在解析过程中正确处理这些标签，避免将其错误地压入栈中。
- **属性解析**：HTML标签中的属性解析较为复杂，尤其是属性值可能包含引号或不包含引号。通过编写`ParseAttr`函数，使用状态机逻辑逐个解析属性名和值，确保正确提取所有属性。
- **CSS选择器解析**：CSS选择器的多样性使得解析过程较为复杂，特别是嵌套选择器和复合选择器。通过分步骤解析选择器，先以“`,`”为标志分割出子查询，再从后往前逐一解析标签名和运算符，确保每种选择器类型都能被正确处理。
- **内存管理**：在动态创建节点和标签时，确保所有分配的内存都能被正确释放，避免内存泄漏。通过在析构函数中**递归**删除所有节点，保证内存的正确回收。

4.2 算法的时空分析

4.2.1 HTML解析算法

- **时间复杂度**：O(n)，其中n为HTML内容的长度。解析过程中每个字符最多被扫描一次。
- **空间复杂度**：O(m)，其中m为HTML标签的数量。主要用于存储DOM树结构。

4.2.2 CSS选择器查询算法

- **时间复杂度**：，查询操作的时间复杂度为O(m)，其中m为DOM树中节点的数量。
- **空间复杂度**：O(k)，其中k为符合选择器条件的节点数量。用于存储查询结果。

4.3 改进设想

1. **支持更多CSS选择器**：目前仅支持基本的选择器类型，未来可以扩展支持属性选择器、伪类选择器等更复杂的选择器。
2. **优化查询性能**：通过索引或缓存机制，提升查询效率，尤其是在处理大型HTML文档时。
3. **增强错误处理能力**：提供更详细的错误信息和恢复机制，允许在遇到部分错误时继续解析。
4. **内存管理优化**：引入智能指针（如`std::shared_ptr`或`std::unique_ptr`），进一步提高内存管理的安全性和效率。

4.4 经验和体会

在实现本项目的过程中

5. 用户使用说明

本程序用于解析HTML内容并基于CSS选择器进行节点查询。以下是详细的使用步骤：

5.1 安装步骤

1. **下载程序文件**：获取项目的源代码文件，包括`Html_parser.h`、`Css_selector.h`、`main.cpp`等。
2. **安装所需依赖库**：
   - 安装`lcurl`库，用于网络请求。
   - 安装C++编译器（如g++）。

5.2 运行步骤

1. **编译程序**：

   ```
   g++ -o css main.cpp
   ```

2. **运行程序**：

   ```
   ./css
   ```

3. **选择输入方式**：
   - 输入文件地址：选择[1]，然后输入本地HTML文件的路径。
   - 输入URL地址：选择[2]，然后输入目标网页的URL。
4. **执行查询**：

   用户菜单如图所示：

用户主菜单	子菜单
`` ========== 主菜单 ========== 请选择以下操作： [1]：重新选择文件进行解析 - 加载一个新的 HTML 文件进行解析操作。 [2]：查找符合条件的 HTML 节点 - 根据条件查询并返回相关节点信息。 [3]：打印解析出的 HTML 文件 - 显示整个解析后的 HTML 文件结构。 [-1]：退出程序 - 结束程序并退出。 ========== 请输入操作编号： ```	``` ========== 查询结果操作菜单 ========== 请选择以下操作： [0]： 对节点进行二次查询 - 允许对已选节点执行进一步的查询操作 [1]： 输出节点的内部文本 - 提取并显示节点的纯文本内容。 [2]： 输出节点的内部 HTML - 显示节点及其子节点的完整 HTML 结构。 [3]： 输出节点的 href 属性（仅适用于 <a> 标签） - 提取并显示超链接地址。 [4]： 重新打印查询结果列表 - 查看上一次查询的结果列表。 [-1]：重新输入查询条件 - 开始新的查询操作。 ========== 请输入操作编号： ```

    - 在主菜单中选择[2]进行CSS选择器查询。
    - 输入符合CSS选择器语法的查询语句。
5. **查看查询结果**：
    - 查询结果将以列表形式展示，每个节点带有索引号。
    - 选择相应的操作编号，如提取文本、HTML或链接地址。
6. **退出程序**：选择[-1]即可退出程序。

6. 测试结果

本节展示了程序在不同测试用例下的运行结果，包括输入和输出情况。以下列出部分测试用例：

文件：`cvpr.html`

6.1 测试用例1

- **输入**：

```
.dropdown-item.dropdown.pe-3 .nav-link.dropdown-toggle.border-3.btn.btn-primary.text-white.p-1
```

- **输出**：

```
[
{0}<a class="nav-link dropdown-toggle border-3 btn btn-primary text-white p-1" style="background-color: #070bff; font-size: 1.2em;" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
]
```

6.2 测试用例2

- **输入**：

```
button+div
```

- **预期输出**：

```
[
{0}<div class="collapse navbar-collapse" id="navbarToggler1">
{1}<div class="collapse navbar-collapse" id="navbarToggler1029">
]
```

6.3 测试用例3

- **输入**:

```
[href=/Conferences/2025/ProgramCommittee]
```

- **输出**:

```
[
{0}<a class="nav-link p-1" href="/Conferences/2025/ProgramCommittee">
]
```

6.4 测试用例4

- **输入**:

```
a[href^="/Conference"]
```

- **输出**:

```
[
{0}<a class="nav-link p-1" href="/Conferences/2024/CodeOfConduct">
{1}<a class="nav-link p-1" href="/Conferences/2024/PrivacyPolicy">
{2}<a class="dropdown-item p-1" href="/Conferences/2025">
{3}<a class="dropdown-item p-1" href="/Conferences/2024">
{4}<a class="dropdown-item p-1" href="/Conferences/2023">
{5}<a class="nav-link p-1" href="/Conferences/2025/Dates">
{6}<a class="nav-link p-1" href="/Conferences/2025/CallForPapers">

...
{36}<a class="nav-link p-1" href="/Conferences/2025/PRProfessionals">
{37}<a class="nav-link p-1" href="/Conferences/2025/PressLandingPage">
{38}<a class="nav-link p-1" href="/Conferences/2025/MediaPass">
{39}<a class="nav-link p-1" href="/Conferences/2025/NewsAndResources">
{40}<a class="nav-link p-1" href="/Conferences/2025/Organizers">
{41}<a class="nav-link p-1" href="/Conferences/2025/ProgramCommittee">
]
```

6.5 测试用例5

- **输入**:

```
a[href*="?"]
```

- **输出**:

```
[
{0}<a href="/accounts/login?nextp=/Conferences/2025/CallForPapers "
class="navbar-brand">
{1}<a href="https://openreview.net/group?id=thecvf.com/CVPR/2025/Conference">
]
```