HTML CSS 选择器
实验报告

张哲卿 2022201549
人工智能拔尖班

日期：2023 年 12 月 14 日

题目：实现常用 HTML CSS Selector 以及对应的获取文本、HTML、链接等操作。

一、需求分析

1 读取 html 文件，先保存成链表形式，从链表通过栈构建成 DOM 树。采取二叉树存储法存储 DOM 树。

2 本程序的目的是构建一个交互系统，用户先输入读取文件地址，之后进行 CSS 选择器查询，查询结果保存成线性表，并可指定查询结果显示其 innerText, outerHTML, (href 仅针对'a')，并可对指定结点进行查询。

3 测试数据（附后）。

二、概要设计

2.1 树

ADT Tree{

数据对象 D：D 是具有相同特性的数据元素的集合。

数据关系 R：若 D 为空集，则称为空树；

若 D 仅含一个数据元素，则 R 为空集，否则 R={H}，H 是如下二元关系：

(1) 若 D 中存在唯一的称为根的数据元素 root，它在关系 H 下无前驱。

(2) 若 D-{root}$\neq \emptyset$，则存在 D-{root} 的一个划分 $D_1, D_2, \cdots, D_m(m > 0)$,$\forall j \neq k(1 \leq j, k \leq m), D_j \bigcap D_k = \emptyset$, 且 $\forall i(1 \leq i \leq m), \exists! x_i \in D_i, < root, x_i > \in H$.

(3) 对应于 D-{root} 的划分，$H - \{< root, x_1 >, \cdots, < root, x_m >\}$ 有唯一一个划分 $H_1, H_2, \cdots, H_m(m > 0)$, $\forall j \neq k(1 \leq j, k \leq m), H_j \bigcap H_k = \emptyset$, 且 $\forall i(1 \leq i \leq m), H_i$ 是 D_i 上二元关系，$(D_i, \{H_i\})$ 是一棵符合本定义的树，称为根 root 的子树。

基本操作 P：

InitTree(T) // 初始化置空树

CreateTree(T, definition) // 按定义构造树

DestroyTree(T) // 销毁树的结构

ClearTree(T) // 将树清空
TreeEmpty(T) // 判定树是否为空树
TreeDepth(T) // 求树的深度
Root(T) // 求树的根结点
Value(T, cur_e) // 求当前结点的元素值
Assign(T, cur_e, value) // 给当前结点赋值
Parent(T, cur_e) // 求当前结点的双亲结点
LeftChild(T, cur_e) // 求当前结点的最左孩子
RightSibling(T, cur_e) // 求当前结点的右兄弟
TraverseTree(T,Visit()) //遍历
}

2.2 主程序

```
void main(){
    初始化:
    do{
        接受命令;
        处理命令;
    }while("命令"!="退出");
}
```

三、详细设计

3.1 Token

根据数据元素特点，设计 token 类在 token.h，实现在 token.cpp.

```
class token
{
private:
  std::string TagName;  //标签名
  std::string attribute;  //属性
  TokenType type;     //类型  StartTag, EndTag, Comment, Character,
      Uninitialized, DOCTYPE, EndOfFile
  std::string text;   //包含文字

public:
  token() { type = Uninitialized; }
  token(const token& t);
  ~token() {}

  std::string getTagName()const { return TagName; }
  void setTagName(std::string& s) { TagName = s; }
```

```
  std::string getAttribute()const { return attribute; }
  void setAttribute(std::string& s) { attribute = s; }

  TokenType getType() { return type; }
  void setType(TokenType t) { type = t; }

  std::string getText()const { return text; }
  void setText(std::string& s) { text = s; }

  token& operator = (const token & t);
  friend std::ostream& operator <<(std::ostream& out, const token& t);


  void display();   // 打印到控制台

  std::string write();  //  把一般标签写成字符串
  std::string writeEnd();   // 把标签结尾</>写成字符串
};

token::token(const token& t)
{
  TagName = t.TagName;
  attribute = t.attribute;
  type = t.type;
  text = t.text;
}

token& token::operator=(const token& t)
{
  TagName = t.TagName;
  type = t.type;
  text = t.text;
  attribute = t.attribute;
  return *this;
}

/// <summary>
/// 判断是何种类型，结束标签特殊调用，其余正常调用
/// </summary>
void token::display()
{
  std::string show;
  if (!attribute.empty())
    show = '<' + TagName + '␣' + attribute + '>';
  else if (!TagName.empty())
    show = '<' + TagName + '>';
```

```
  std::cout << show << '\t';
}

/// <summary>
/// 按照类型组成html形式
/// </summary>
/// <returns>html形式字符串<   > </returns>
std::string token::write()
{
  std::string res;
  switch (type)
  {
  case(StartTag):
    res += '<' + TagName;
    if (!attribute.empty()) res += '␣' + attribute;
    res += '>';
    break;
  case(Comment):
    res = "<!--␣" + text + "␣-->";
    break;
  case(Character):
    res = text;
    break;
  case(DOCTYPE):
    res = "<!DOCTYPE␣" + attribute + ">";
    break;
  default:
    break;
  }
  return res;
}

/// <summary>
/// 结尾标签特殊实现
/// </summary>
/// <returns>结尾标签html形式字符串</  ></returns>
std::string token::writeEnd()
{
  if (type != StartTag)
    return std::string();
  if (TagName == "br" || TagName == "hr"
    || TagName == "meta" || TagName == "img"
    || TagName == "input" || TagName == "area"
    || TagName == "link" || TagName == "source"
    || TagName == "base") //自闭合标签
    return std::string();
  std::string ret = "</" + TagName + ">";
```

```
    return ret;
  }

  std::ostream& operator<<(std::ostream& out, const token& t)
  {
    out << "tagName: " << t.TagName << ' ';
    out << "type: " << t.type << ' ';
    out << "attribute: " << t.attribute << ' ';
    out << "text: " << t.text << '\n';
    return out;
  }
```

3.2 Tree

树的实现在 tree.h，模板类。根据实验特点，添加指向其物理结构的父亲结点指针。

```
#ifndef TREE_H
#define TREE_H
#include<iostream>
#include "token.h"
#include "sqlist.h"

template<class DataType>
class CSNode {
public:
    DataType data;
  CSNode<DataType>* firstchild;  //第一个孩子结点
  CSNode<DataType>* nextsibling; //下一个兄弟结点
    CSNode<DataType>* parent;   //物理结构的父亲结点，并非存储结构

    CSNode();
    CSNode(const CSNode& csn);
};

template<class DataType>
CSNode<DataType>::CSNode()
{
    firstchild = NULL;
    nextsibling = NULL;
    parent = NULL;
}

template<class DataType>
inline CSNode<DataType>::CSNode(const CSNode& csn): firstchild(csn.firstchild),
    nextsibling(csn.nextsibling)
{
    data = csn.data;
```

```
    firstchild = csn.firstchild;
    nextsibling = csn.nextsibling;
    parent = csn.parent;
}

template <class DataType>
class CSTree
{
public:
    CSNode<DataType>* root;
    CSTree();
    ~CSTree() { }
    void PreOrder() { PreOrder(root); } //前序遍历
    void InOrder() { InOrder(root); }   //中序遍历
    void PostOrder() { PostOrder(root); } //后序遍历

private:
    void PreOrder(CSNode<DataType>* cst);

    void InOrder(CSNode<DataType>* cst);
    void PostOrder(CSNode<DataType>* cst);
};

template<class DataType>
inline CSTree<DataType>::CSTree()
{
    root = new CSNode<DataType>;
}


template <class DataType>
void CSTree<DataType>::PreOrder(CSNode<DataType>* cst)
{
    if (cst == NULL) return;
    else {
        std::cout << cst->data << '\n';
        PreOrder(cst->firstchild);
        PreOrder(cst->nextsibling);
    }
}

template <class DataType>
void CSTree<DataType>::InOrder(CSNode<DataType>* cst)
{
    if (cst == NULL) return;
    else {
        InOrder(cst->firstchild);
```

```
        std::cout << cst->data;
        InOrder(cst->nextsibling);
    }
}

template <class DataType>
void CSTree<DataType>::PostOrder(CSNode<DataType>* cst)
{
    if (cst == NULL) return;
    else {
        PostOrder(cst->firstchild);
        PostOrder(cst->nextsibling);
        std::cout << cst->data;
    }
}


#endif // !TREE_H
```

3.3 Terminal

终端类，主要实现和用户交互。设计在 terminal.h，实现在 terminal.cpp.

为辅助 DOM 树构建，新建结构体 tookenNode，定义在 token.h。

```
struct tokenNode
{
    token t;
    CSNode<token>* ptr;
};
```

```
#ifndef TERMINAL_H
#define TERMINAL_H

#include <iostream>
#include <fstream>
#include "sqlist.h"
#include <string>
#include "linklist.h"
#include "token.h"
#include "tree.h"
#include "linkstack.h"
#include "util.h"


class terminal
{
public:
```

```
  terminal() {}
  void run();   //运行
  void getInstruction(); //获取指令
  void readFile(std::string filepath); //读取文件
  void splitTokens(SqList<std::string>& sql, LinkList<token>& tokenList); //切分文
      件输入进来的token
  int tagOpen(std::string& s, int i, token& t, int len);  //处理输入'<'之后
  void buildDomTree(LinkList<token>& tokenList);    //构建DOMTree

  void query(std::string& input, CSNode<token>* ptr); //处理用户输入query

private:
  CSTree<token> tokenTree;    //DOM Tree
  LinkList<tokenNode> result;   //query按照','切分后单个返回node
  SqList<tokenNode> sqresult;   //query返回node汇总
  std::vector<std::vector<std::string>> Elements; //切分query形成单个匹配的element

  void initializeRes(const std::string& input, CSNode<token>* ptr); //初始化结果,
      初始查找

  void startwithAsterisk(); //从'*'开始 (输入单个'*'返回所有结点)
  void insertTN(const std::string& input, CSNode<token>* ptr);  //找到匹配的node插
      入到答案
  void PreOrderAst(CSNode<token>* ptr); //先序遍历寻找满足'*' node
  void preOrderTag(CSNode<token>* ptr, const std::string& tag); //先序遍历寻找与输
      入tag相匹配 node
  void preOrderClass(CSNode<token>* ptr, const std::string& clss);  //先序遍历寻找
      与输入class相匹配 node
  void preOrderID(CSNode<token>* ptr, const std::string& id);   //先序遍历寻找与输
      入id相匹配 node
  void preOrderAttVal(CSNode<token>* ptr, const std::string& input);  //先序遍历寻
      找匹配的属性键值 node

  void filterTN(const std::string& input, CSNode<token>* ptr,
     std::string (*funcget1)(const tokenNode&), std::string(*funcget2)(const
         tokenNode&));  //遍历找出的node, 根据query过滤不满足的node, 适用于'+','>'
  //两个函数指针, 第一个获取其class/ id / attribute进行匹配, 第二个获取其tag进行匹
      配

  void filterWave(const std::string& input, CSNode<token>* ptr);  //遍历找出的node
      , 根据query过滤不满足的node, 适用于'~'

  void filterSpace(const std::string& input, CSNode<token>* ptr); //遍历找出的node
      , 根据query过滤不满足的node, 适用于' '

  void filter(const std::string& input, CSNode<token>* ptr,
    bool (*func)(const std::string&, const std::string&), std::string(*funcget)(
```

```
        const tokenNode&)); //遍历找出的node，过滤不满足的，第一个函数指针是匹配函
        数，第二个函数指针是获取node信息函数
  void ll2sql(); //把几个query片段找到结果组合起来

  void outQuery(std::string& input, int k); //处理查找第k个node

  std::string outHref(int k); //查找第k个node的href
  std::string outText(int k); //查找第k个node的innerText
  std::string outHTML(int k); //查找第k个node的outerHTML
  void outSelector(int k, std::string& input); //找出第k个node之后继续查找其属性

  void preOrderText(CSNode<token>* ptr, std::string& text); //按照先序遍历查找对应
      node，找出该node下innerText并延续到输入参数text之后
  void preOrderHTML(CSNode<token>* ptr, std::string& HTML); //按照先序遍历查找对应
      node，找出该node的outerHTML并延续到输入参数HTML之后

};

#endif // TERMINAL_H

#include "terminal.h"

void terminal::run()
{
  while (true)
  {
    getInstruction();
  }
}

void terminal::getInstruction()
{
  std::string instruction;
  getline(std::cin, instruction);
  int len = instruction.length();
  if (instruction[0] == 'r' && instruction[1] == 'e' && instruction[2] == 'a' &&
      instruction[3] == 'd' && instruction[4] == '(')
  {
    std::string input;
    for(int i = 5; i < len && instruction[i] != ')'; i++)
    {
      if (instruction[i] == '"')
        continue;
      input.push_back(instruction[i]);
    }
    readFile(input);
  }
```

```
  else if (instruction[0] == 'q' && instruction[1] == 'u' && instruction[2] == 'e'
      && instruction[3] == 'r' && instruction[4] == 'y')
  {
    sqresult.Clear();
    result.Clear();
    Elements.clear();
    std::string input;
    for (int i = 6; instruction[i] != ')' && i < len; i++)
    {
      if (instruction[i] == '"')
        continue;
      input += instruction[i];
    }
    query(input, tokenTree.root->firstchild);
  }
  else if (instruction[0] == 'O' && instruction[1] == 'u' && instruction[2] == 't')
  {
    std::string input;
    std::string num;
    int i = 0;
    for (i = 4; instruction[i] != ']'; i++)
    {
      num += instruction[i];
    }
    int k = std::stoi(num);
    i += 2;
    input = instruction.substr(i, instruction.length() - i);
    if (k < 0 || k >= sqresult.Length())
    {
      std::cout << "Index␣out␣of␣range." << '\n';
      return;
    }
    outQuery(input, k);
  }
}

void terminal::readFile(std::string filepath)
{
  std::ifstream infile;
  infile.open(filepath, std::ios::in | std::ios::binary);
  //std::cout << filepath << '\n';
  if (!infile.is_open())
  {
    std::cout << "读取文件失败" << std::endl;
    return;
  }
  std::string s;
```

```
  SqList<std::string> sl;
  // 把文件一次读入，去除不必要的空格，按行存储在顺序表sl
  while (getline(infile, s))
  {
    while (s[0] == '␣' && s[1] == '␣') s.erase(0, 1);
    sl.push_back(s);
  }

  infile.close();
  LinkList<token> tokenList;
  splitTokens(sl, tokenList);
  buildDomTree(tokenList);
  // tokenTree.InOrder();
}


/// 一行一行处理，读入到'<'，跳转tagOpen函数进行切分，否则按照文字存储
void terminal::splitTokens(SqList<std::string>& sql, LinkList<token>& tokenList)
{
  int size = sql.Length();
  if (!size)
    return;

  std::string html;
  for (int i = 0; i < size; i++)
  {
    html.append(sql[i]);
  }

  int len = html.length();
  for (int i = 0; i < len; i++)
  {
    token t;
    if (html[i] == '<')
    {
      ++i;
      int j = tagOpen(html, i, t, len);
      i = j;
      tokenList.append(t);
      continue;
    }
    else if (html[i] == EOF)
    {
      t.setType(EndOfFile);
      tokenList.append(t);
      break;
    }
```

```
    else
    {
      t.setType(Character);
      std::string text;
      int j = 0;
      for ( j = i; j < len; j++)
      {
        if (html[j] == '<')
          break;
        text += html[j];
      }
      i = j - 1;
      t.setText(text);
      tokenList.append(t);
      continue;
    }
  }
  //std::cout << tokenList;
}


/// tagOpen之后，判断接下来的字符
/// 1. '/'按结束tag处理
/// 2. '!'
///   2.1 '!-- '按照注释处理
///   2.2 '!DOCTYPE 按照DOCTYPE处理
/// 3. 处理成开始tag
int terminal::tagOpen(std::string& s, int i, token& t, int len)
{
  if (s[i] == '/')
  {
    t.setType(EndTag);
    std::string tagName;
    int j = 0;
    for (j = i + 1; s[j] != '>' && j < len; j++)
    {
      tagName += s[j];
    }
    //j++;
    t.setTagName(tagName);
    return j;
  }
  else if (s[i] == '!')
  {
    if (s[i + 1] == '-' && s[i + 2] == '-')
    {
      i += 4;
```

```
      int j = 0;
      t.setType(Comment);
      std::string comment;
      for (j = i; j < len; j++)
      {
        if (s[j] == '-' && s[j + 1] == '-' && s[j + 2] == '>')
          break;
        comment += s[j];
      }
      t.setText(comment);
      j += 2;
      return j;
    }
    else
    {
      i += 9;
      int j = 0;
      std::string doc;
      t.setType(DOCTYPE);
      for (j = i; s[j] != '>' && j < len; j++)
      {
        doc += s[j];
      }
      t.setAttribute(doc);
      return j;
    }
  }
  else if ((s[i] <= 'z' && s[i] >= 'a') || (s[i] >= 'A' && s[i] <= 'Z'))
  {
    std::string tagName;
    t.setType(StartTag);
    int j = 0;
    for (j = i; j < len; j++)
    {
      if (s[j] == '␣')
      {
        std::string attribute;
        int m = 0;
        for (m = j + 1; s[m] != '>' && m < len; m++)
        {
          attribute += s[m];
        }
        //m++;
        t.setAttribute(attribute);
        t.setTagName(tagName);
        return m;
      }
```

```
      else if (s[j] == '>')
      {
        //j++;
        t.setTagName(tagName);
        return j;
      }
      tagName += s[j];
    }
  }
}

/// 建立的DOMTree，有个根节点"document"，不是文件的内容，其兄弟结点是DOCTYPE
/// 遍历建立的链表token
/// 用栈辅助
/// 1. 遇到StartTag，把这个token存储到当前栈顶结点的孩子结点，接着入栈。而自闭合标
    签存储在当前栈顶结点的孩子结点
/// 2. 遇到EndTag，不存储到树里，弹栈直到弹出的与EndTag相匹配
/// 3. 文本存储在当前栈顶结点的孩子结点
void terminal::buildDomTree(LinkList<token>& tokenList)
{
  if (!tokenList.len())
    return;
  //std::cout << tokenList;
  token tokenRoot;
  tokenRoot.setType(StartTag);
  std::string document = "document";
  tokenRoot.setTagName(document);

  tokenTree.root->data = tokenRoot;
  LinkedStack<tokenNode> tokenStack;
  tokenNode tNode = { tokenRoot, tokenTree.root };
  tokenStack.Push(tNode);

  CSNode<token>* pCSNode = tokenTree.root;
  LNode<token>*pLNode = tokenList.getHead()->getNext();
  while (pLNode != NULL)
  {
    token temp = pLNode->getData();
    std::cout << temp << '\n';
    if (temp.getType() == StartTag)
    {
      std::string startTagName = temp.getTagName();
      if (startTagName != "br" && startTagName != "hr"
        && startTagName != "meta" && startTagName != "img"
        && startTagName != "input" && startTagName != "area"
        && startTagName != "link" && startTagName != "source"
        && startTagName != "base")  // 去除自闭合标签
```

```
    {
      tokenNode top = tokenStack.getTopElement();
      CSNode<token>* pTemp = top.ptr->firstchild;
      CSNode<token>* current = NULL;
      if (pTemp == NULL)
      {
        top.ptr->firstchild = new CSNode<token>;
        top.ptr->firstchild->data = temp;
        top.ptr->firstchild->parent = top.ptr;
        current = top.ptr->firstchild;
      }
      else
      {
        while (pTemp->nextsibling != NULL)
        {
          pTemp = pTemp->nextsibling;
        }
        pTemp->nextsibling = new CSNode<token>;
        pTemp = pTemp->nextsibling;
        pTemp->data = temp;
        pTemp->parent = top.ptr;
        current = pTemp;
      }
      tokenNode tnode = { temp, current };
      tokenStack.Push(tnode);

      pLNode = pLNode->getNext();
      pCSNode = current;
      continue;
    }
    // 处理自闭合标签
    else
    {
      tokenNode top = tokenStack.getTopElement();
      CSNode<token>* pTemp = top.ptr->firstchild;
      if (pTemp == NULL)
      {
        top.ptr->firstchild = new CSNode<token>;
        top.ptr->firstchild->data = temp;
        top.ptr->firstchild->parent = top.ptr;
      }
      else
      {
        while (pTemp->nextsibling != NULL)
        {
          pTemp = pTemp->nextsibling;
        }
```

```
        pTemp->nextsibling = new CSNode<token>;
        pTemp = pTemp->nextsibling;
        pTemp->data = temp;
        pTemp->parent = top.ptr;
      }

      pLNode = pLNode->getNext();
    }
  }
  else if (temp.getType() == EndTag)
  {
    tokenNode tempNode = tokenStack.getTopElement();
    while (tempNode.t.getTagName() != temp.getTagName())
    {
      tokenStack.Pop();
      tempNode = tokenStack.getTopElement();
    }
    tokenStack.Pop();
    tempNode = tokenStack.getTopElement();

    pCSNode = tempNode.ptr;
    pLNode = pLNode->getNext();
  }
  else if (temp.getType() == DOCTYPE)
  {
    tokenTree.root->nextsibling = new CSNode<token>;
    tokenTree.root->nextsibling->data = temp;
    pLNode = pLNode->getNext();
    continue;
  }
  else
  {
    tokenNode top = tokenStack.getTopElement();
    CSNode<token>* pTemp = top.ptr->firstchild;
    if (pTemp == NULL)
    {
      top.ptr->firstchild = new CSNode<token>;
      top.ptr->firstchild->data = temp;
      top.ptr->firstchild->parent = top.ptr;
    }
    else
    {
      while (pTemp->nextsibling != NULL)
      {
        pTemp = pTemp->nextsibling;
      }
      pTemp->nextsibling = new CSNode<token>;
```

```
        pTemp = pTemp->nextsibling;
        pTemp->data = temp;
        pTemp->parent = top.ptr;
      }

      pLNode = pLNode->getNext();
    }
  }
}

void terminal::query(std::string& input, CSNode<token>* ptr)
{
  if (input[0] == '*' && input.length() == 1 && ptr == tokenTree.root)
  {
    startwithAsterisk();
    return;
  }
  splitQuery(input, Elements);
  size_t outerSize = Elements.size();
  for (size_t i = 0; i < outerSize; i++)
  {
    int innerSize = Elements[i].size();
    std::string last = Elements[i][innerSize - 1];
    // 从后往前找，逐步缩小范围，提升性能
    initializeRes(last, ptr);
    for (int j = innerSize - 2; j >= 0; j--)
    {

      if (Elements[i][j] == ">")
      {
        filterTN(Elements[i][j - 1], ptr, getParentAtt, getParentTag);
        j--;
      }
      else if (Elements[i][j] == "+")
      {
        filterTN(Elements[i][j - 1], ptr, getSiblingAtt, getSiblingTag);
        j--;
      }
      else if (Elements[i][j] == "~")
      {
        filterWave(Elements[i][j - 1], ptr);
        j--;
      }
      else
      {
        filterSpace(Elements[i][j], ptr);
      }
```

```
    }
    ll2sql();
    result.Clear();
    if (result.getHead() == NULL)
    {
      LNode<tokenNode>* p = new LNode<tokenNode>;
      result.setHead(p);
    }

  }
  int size = sqresult.Length();
  for (int i = 0; i < size; i++)
  {
    sqresult[i].t.display();
  }
  std::cout << '\n' << size << '\n';
}

void terminal::PreOrderAst(CSNode<token>* ptr)
{
  if (ptr == NULL) return;
  token temp = ptr->data;
  if (temp.getType() == StartTag)
  {
    tokenNode tempNode;
    tempNode.t = temp;
    tempNode.ptr = ptr;
    //sqresult.push_back(tempNode);
    result.append(tempNode);
    //temp.display();
  }
  PreOrderAst(ptr->firstchild);
  PreOrderAst(ptr->nextsibling);
}

void terminal::startwithAsterisk()
{
  PreOrderAst(tokenTree.root->firstchild);
  ll2sql();
  int size = sqresult.Length();
  for (int i = 0; i < size; i++)
  {
    sqresult[i].t.display();
  }
  std::cout << '\n' << size << '\n';
}
```

```
void terminal::initializeRes(const std::string& input, CSNode<token>* ptr)
{
  std::vector<std::string> splitedInput = splitElements(input);
  insertTN(splitedInput[0], ptr);
  size_t size = splitedInput.size();
  for (size_t i = 1; i < size; i++)
  {
    filterTN(splitedInput[i], ptr, getMineAtt, getMineTag);
  }
}

void terminal::insertTN(const std::string& input, CSNode<token>* ptr)
{
  if (input[0] == '*')
  {
    PreOrderAst(ptr); //tokenTree.root->firstchild);
  }
  else if (input[0] == '#')
  {
    std::string id = input.substr(1, input.length() - 1);
    preOrderID(ptr, id);// tokenTree.root->firstchild, id);
  }
  else if (input[0] == '.')
  {
    std::string clss = input.substr(1, input.length() - 1);
    preOrderClass(ptr, clss);// tokenTree.root->firstchild, clss);
  }
  else if (input[0] == '[')
  {
    preOrderAttVal(ptr, input);// tokenTree.root->firstchild, input);
  }
  else
  {
    preOrderTag(ptr, input);// tokenTree.root->firstchild, input);
  }
}

void terminal::preOrderTag(CSNode<token>* ptr, const std::string& tag)
{
  if (ptr == NULL)
    return;
  token temp = ptr->data;
  if (temp.getType() == StartTag && matchTag(temp.getTagName(), tag))// && temp.
      getTagName() == tag)
  {
    tokenNode tempNode;
    tempNode.t = temp;
```

```
    tempNode.ptr = ptr;
    result.append(tempNode);
  }
  preOrderTag(ptr->firstchild, tag);
  preOrderTag(ptr->nextsibling, tag);
}

void terminal::preOrderClass(CSNode<token>* ptr, const std::string& clss)
{
  if (ptr == NULL)
    return;
  token temp = ptr->data;
  if (temp.getType() == StartTag && !temp.getAttribute().empty())
  {
    std::string att = temp.getAttribute();
    if (matchClass(att, clss))
    {
      tokenNode tempNode;
      tempNode.t = temp;
      tempNode.ptr = ptr;
      result.append(tempNode);
    }
  }
  preOrderClass(ptr->firstchild, clss);
  preOrderClass(ptr->nextsibling, clss);
}

void terminal::preOrderID(CSNode<token>* ptr, const std::string& id)
{
  if (ptr == NULL)
    return;
  token temp = ptr->data;
  if (temp.getType() == StartTag && !temp.getAttribute().empty())
  {
    std::string att = temp.getAttribute();
    if (matchID(att, id))
    {
      tokenNode tempNode;
      tempNode.t = temp;
      tempNode.ptr = ptr;
      result.append(tempNode);
    }

  }
  preOrderID(ptr->firstchild, id);
  preOrderID(ptr->nextsibling, id);
}
```

```
void terminal::preOrderAttVal(CSNode<token>* ptr, const std::string& input)
{
  if (ptr == NULL)
    return;
  token temp = ptr->data;
  if (temp.getType() == StartTag && !temp.getAttribute().empty())
  {
    std::string att = temp.getAttribute();
    if (matchAttVal(att, input))
    {
      tokenNode tempNode;
      tempNode.t = temp;
      tempNode.ptr = ptr;
      result.append(tempNode);
    }

  }
  preOrderAttVal(ptr->firstchild, input);
  preOrderAttVal(ptr->nextsibling, input);
}

void terminal::filterTN(const std::string& input, CSNode<token>* ptr, std::string(*
    funcget1)(const tokenNode&), std::string(*funcget2)(const tokenNode&))
{
  if (input[0] == '.')
  {
    std::string fit = input.substr(1, input.length() - 1);  // 去除'.'
    filter(fit, ptr, matchClass, funcget1);
    /*LNode<tokenNode>* prev = result.getHead();
    LNode<tokenNode>* cur = result.getHead()->getNext();
    while (cur != NULL)
    {
      std::string att = cur->getData().t.getAttribute();
      if (!matchClass(att, fit))
      {
        result.Delete(prev, cur);
        continue;
      }
      prev = cur;
      cur = cur->getNext();
    }*/
  }
  else if (input[0] == '#')
  {
    std::string fit = input.substr(1, input.length() - 1);  // 去除'#'
    filter(fit, ptr, matchID, funcget1);
```

```
    /*LNode<tokenNode>* prev = result.getHead();
    LNode<tokenNode>* cur = result.getHead()->getNext();
    while (cur != NULL)
    {
      std::string att = cur->getData().t.getAttribute();
      if (!matchID(att, fit))
      {
        result.Delete(prev, cur);
        continue;
      }
      prev = cur;
      cur = cur->getNext();
    }*/
  }
  else if (input[0] == '[')
  {
    filter(input, ptr, matchAttVal, funcget1);
    /*LNode<tokenNode>* prev = result.getHead();
    LNode<tokenNode>* cur = result.getHead()->getNext();
    while (cur != NULL)
    {
      std::string att = cur->getData().t.getAttribute();
      if (!matchAttVal(att, input))
      {
        result.Delete(prev, cur);
        continue;
      }
      prev = cur;
      cur = cur->getNext();
    }*/
  }
  else
  {
    filter(input, ptr, matchTag, funcget2);
  }
}


void terminal::filterWave(const std::string& input, CSNode<token>* ptr)
{
  std::vector<std::string> splitInput = splitElements(input);
  int size = splitInput.size();
  for (int i = 0; i < size; i++)
  {
    if (splitInput[i][0] == '.')
    {
      std::string fit = splitInput[i].substr(1, splitInput[i].length() - 1);  // 去
```

```
        除'.'
    LNode<tokenNode>* prev = result.getHead();
    LNode<tokenNode>* cur = result.getHead()->getNext();
    while (cur != NULL)
    {
      if (!matchWave(fit, cur->getData().ptr, matchClass, getAtt))
      {
        result.Delete(prev, cur);
        continue;
      }
      prev = cur;
      cur = cur->getNext();
    }
  }
  else if (splitInput[i][0] == '#')
  {
    std::string fit = splitInput[i].substr(1, splitInput[i].length() - 1);  // 去
        除'#'
    LNode<tokenNode>* prev = result.getHead();
    LNode<tokenNode>* cur = result.getHead()->getNext();
    while (cur != NULL)
    {
      if (!matchWave(fit, cur->getData().ptr, matchID, getAtt))
      {
        result.Delete(prev, cur);
        continue;
      }
      prev = cur;
      cur = cur->getNext();
    }
  }
  else if (splitInput[i][0] == '[')
  {
    LNode<tokenNode>* prev = result.getHead();
    LNode<tokenNode>* cur = result.getHead()->getNext();
    while (cur != NULL)
    {
      if (!matchWave(splitInput[i], cur->getData().ptr, matchAttVal, getAtt))
      {
        result.Delete(prev, cur);
        continue;
      }
      prev = cur;
      cur = cur->getNext();
    }
  }
  else
```

```
    {
      LNode<tokenNode>* prev = result.getHead();
      LNode<tokenNode>* cur = result.getHead()->getNext();
      while (cur != NULL)
      {
        if (!matchWave(splitInput[i], cur->getData().ptr, matchTag, getTag))
        {
          result.Delete(prev, cur);
          continue;
        }
        prev = cur;
        cur = cur->getNext();
      }
    }
  }
}

void terminal::filterSpace(const std::string& input, CSNode<token>* ptr)
{
  std::vector<std::string> splitInput = splitElements(input);
  int size = splitInput.size();
  for (int i = 0; i < size; i++)
  {
    if (splitInput[i][0] == '.')
    {
      std::string fit = splitInput[i].substr(1, splitInput[i].length() - 1);  // 去
          除'.'
      LNode<tokenNode>* prev = result.getHead();
      LNode<tokenNode>* cur = result.getHead()->getNext();
      while (cur != NULL)
      {
        if (!matchSpace(fit, cur->getData().ptr, tokenTree.root, matchClass, getAtt
            ))
        {
          result.Delete(prev, cur);
          continue;
        }
        prev = cur;
        cur = cur->getNext();
      }
    }
    else if (splitInput[i][0] == '#')
    {
      std::string fit = splitInput[i].substr(1, splitInput[i].length() - 1);  // 去
          除'#'
      LNode<tokenNode>* prev = result.getHead();
      LNode<tokenNode>* cur = result.getHead()->getNext();
```

```
      while (cur != NULL)
      {
        if (!matchSpace(fit, cur->getData().ptr, tokenTree.root, matchID, getAtt))
        {
          result.Delete(prev, cur);
          continue;
        }
        prev = cur;
        cur = cur->getNext();
      }
    }
    else if (splitInput[i][0] == '[')
    {
      LNode<tokenNode>* prev = result.getHead();
      LNode<tokenNode>* cur = result.getHead()->getNext();
      while (cur != NULL)
      {
        if (!matchSpace(splitInput[i], cur->getData().ptr, tokenTree.root,
            matchAttVal, getAtt))
        {
          result.Delete(prev, cur);
          continue;
        }
        prev = cur;
        cur = cur->getNext();
      }
    }
    else
    {
      LNode<tokenNode>* prev = result.getHead();
      LNode<tokenNode>* cur = result.getHead()->getNext();
      while (cur != NULL)
      {
        if (!matchSpace(splitInput[i], cur->getData().ptr, tokenTree.root, matchTag
            , getTag))
        {
          result.Delete(prev, cur);
          continue;
        }
        prev = cur;
        cur = cur->getNext();
      }
    }
  }
}

void terminal::filter(const std::string& input, CSNode<token>* ptr, bool(*func)(
```

```
    const std::string&, const std::string&), std::string (*funcGet)(const tokenNode
    &))
{
  LNode<tokenNode>* prev = result.getHead();
  LNode<tokenNode>* cur = result.getHead()->getNext();
  while (cur != NULL)
  {
    std::string att = funcGet(cur->getData());
    //std::string att = cur->getData().t.getAttribute();
    //std::string att = cur->getData().ptr->parent->data.getAttribute();
    //std::string att = cur->getData().ptr->nextsibling->data.getAttribute();
    if (!func(att, input))
    {
      result.Delete(prev, cur);
      continue;
    }
    prev = cur;
    cur = cur->getNext();
  }
}

void terminal::ll2sql()
{
  LNode<tokenNode>* p = result.getHead()->getNext();
  int size = sqresult.Length();
  while (p != NULL)
  {
    tokenNode temp = p->getData();
    bool repetition = false;
    for (int i = 0; i < size; i++)
    {
      if (temp.ptr == sqresult[i].ptr)
      {
        repetition = true;
        break;
      }
    }
    if (!repetition)
      sqresult.push_back(p->getData());
    p = p->getNext();
  }
}

void terminal::outQuery(std::string& input, int k)
{
  std::string output = R"()";
  if (input == "innerText")
```

```
  {
    output = outText(k);
  }
  else if (input == "outerHTML")
  {
    output = outHTML(k);
  }
  else if (input == "href")
  {
    output = outHref(k);
  }
  else
  {
    std::string query = input.substr(7, input.length() - 9);
    outSelector(k, query);
  }
  std::istringstream iss(output);

  // 使用 std::getline 循环逐行输出
  std::string line;
  while (std::getline(iss, line, '\r')) {
    std::cout << line << std::endl;
  }
  // std::cout << output << '\n';
}

std::string terminal::outHref(int k)
{
  token tokenk = sqresult[k].t;
  if (tokenk.getTagName() != "a")
  {
    std::cout << "This␣is␣not␣'a'␣tag." << '\n';
    return std::string();
  }
  std::vector<std::vector<std::string>> att_val = splitTagAttribute(tokenk.
      getAttribute());
  for (auto& i : att_val)
  {
    if (i[0] == "href")
    {
      return i[1];
    }
  }
  return std::string();
}

std::string terminal::outText(int k)
```

```
{
  tokenNode nodek = sqresult[k];
  std::string text;
  preOrderText(nodek.ptr->firstchild, text);
  //std::cout << text << '\n';
  return text;
}

std::string terminal::outHTML(int k)
{
  tokenNode nodek = sqresult[k];
  std::string HTML;
  HTML += nodek.t.write();
  preOrderHTML(nodek.ptr->firstchild, HTML);
  HTML += nodek.t.writeEnd();
  return HTML;
}

void terminal::outSelector(int k, std::string& input)
{
  tokenNode nodek = sqresult[k];
  sqresult.Clear();
  query(input, nodek.ptr->firstchild);
}

void terminal::preOrderText(CSNode<token>* ptr, std::string& text)
{
  if (ptr == NULL)return;
  token t = ptr->data;
  if (t.getType() == Character)
  {
    text += t.getText();
    //std::cout << text << '\n';
  }
  preOrderText(ptr->firstchild, text);
  preOrderText(ptr->nextsibling, text);
}

void terminal::preOrderHTML(CSNode<token>* ptr, std::string& HTML)
{
  if (ptr == NULL)
    return;
  token t = ptr->data;
  HTML += t.write();
  preOrderHTML(ptr->firstchild, HTML);
  preOrderHTML(ptr->nextsibling, HTML);
  HTML += t.writeEnd();
```

```
}
```

3.3 Util

其他辅助函数定义在 util.h，实现在 util.cpp.

```
#include <vector>
#include <sstream>
#include <string>
#include <iostream>
#include "token.h"
#include "tree.h"

/// 按照指定分隔符切分字符串
std::vector<std::string> split(const std::string& s, char delimiter);

///','连接是或的关系，先切分；接着按空格切分每一个
void splitQuery(const std::string& query, std::vector<std::vector<std::string>>&
    Elements);

///  切分空格出来的每一个小项
std::vector<std::string> splitElements(const std::string& element);

/// 切分小项中的attribute
std::vector<std::string>  splitAttribute(const std::string& input);

/// 切分html文件中token的tag的attribute值
std::vector<std::vector<std::string>> splitTagAttribute(const std::string& input);

/// tag匹配函数
bool matchTag(const std::string& tag, const std::string& tartag);

/// class匹配函数
bool matchClass(const std::string& att, const std::string& clss);

/// id匹配函数
bool matchID(const std::string& att, const std::string& id);

/// attribute匹配函数
bool matchAttVal(const std::string& att, const std::string& input);

/// ~关系下的匹配函数
bool matchWave(const std::string& str, const CSNode<token>* cur,
  bool (*match)(const std::string&, const std::string&), std::string(*get)(const
      token&));

/// 空格关系下的匹配函数
```

```
bool matchSpace(const std::string& str, const CSNode<token>* cur, const CSNode<
    token>* root,
  bool (*match)(const std::string&, const std::string&), std::string(*get)(const
      token&));

std::string getParentAtt(const tokenNode& tn); /// 获取父结点attribute
std::string getParentTag(const tokenNode& tn); ///  获取父结点的tag

std::string getMineAtt(const tokenNode& tn); /// 获取自己的tag
std::string getMineTag(const tokenNode& tn);///获取自己的attribute

std::string getSiblingAtt(const tokenNode& tn);/// 获取兄弟结点的attribute
std::string getSiblingTag(const tokenNode& tn);/// 获取兄弟结点的tag

std::string getTag(const token& t);/// 获取tag
std::string getAtt(const token& t);///获取attribute

#include "util.h"

std::vector<std::string> split(const std::string& s, char delimiter) {
    std::vector<std::string> tokens;
    std::istringstream tokenStream(s);
    std::string token;
    while (std::getline(tokenStream, token, delimiter)) {
        if (token != "␣" && !token.empty())
            tokens.push_back(token);
    }
    return tokens;
}

void splitQuery(const std::string& query, std::vector<std::vector<std::string>>&
    Elements) {
    // Split the query based on comma to separate different selectors
    std::vector<std::string> selectors = split(query, ',');

    // Process each selector
    for (const std::string& selector : selectors) {
        // Split the selector based on space to identify descendant relationships
        std::vector<std::string> elements = split(selector, '␣');

        Elements.push_back(elements);
    }
}

std::vector<std::string> splitElements(const std::string& element) {
    std::vector<std::string> tags;
    int len = element.length();
```

```
    for (int i = 0; i < len; ++i) {
        std::string tag;

        if (element[i] == '.' || element[i] == '#') {
            tag += element[i];
            for (int j = i + 1; (element[j] >= 'a' && element[j] <= 'z') || (
                element[j] >= 'A' && element[j] <= 'Z')
                || element[j] == '_' || element[j] == '-' || (element[j] >= '0' &&
                    element[j] <= '9' && j < len); ++j) {
                tag += element[j];
            }
            tags.push_back(tag);
            i += tag.length() - 1;
        }
        else if (element[i] == '[') {
            tag += '[';
            for (int j = i + 1; element[j] != ']' && j < len; ++j) {
                tag += element[j];
            }
            tag += ']';
            tags.push_back(tag);
            i += tag.length() - 1;
        }
        else if (element[i] == '*')
        {
            tags.push_back("*");
        }
        else{
            for (int j = i; (element[j] >= 'a' && element[j] <= 'z') || (element[j]
                 >= 'A' && element[j] <= 'Z')
                || element[j] == '_' || element[j] == '-' || (element[j] >= '0' &&
                    element[j] <= '9' && j < len); ++j) {
                tag += element[j];
            }
            tags.push_back(tag);
            i += tag.length() - 1;
        }
    }

    return tags;
}

std::vector<std::string>  splitAttribute(const std::string& input)
{
  std::string temp = input.substr(1, input.length() - 2);
    std::vector<std::string> res = split(temp, '=');
```

```
    if (res.size() == 1)
        res.push_back("%");
    else {
        int len1 = res[0].length();
        char a = res[0][len1 - 1];
        if (a == '$' || a == '^' || a == '*')
        {
            res[0].pop_back();
            std::string s;
            s = a;
            res.push_back(s);
        }
        else
            res.push_back("=");
    }
    return res;
}

std::vector<std::vector<std::string>> splitTagAttribute(const std::string& input)
{
    std::vector<std::string> att_val;
    int len = input.length();
    for (int i = 0; i < len; i++)
    {
        std::string single;
        for (int j = i;  j < len; j++)
        {
            single += input[j];
            if (input[j] == '"' && input[j + 1] == '␣')
                break;

        }
        i += single.length();
        att_val.push_back(single);
    }
    std::vector<std::vector<std::string >> res;
    for (auto& i : att_val)
    {
        std::vector<std::string> temp = split(i, '=');
        res.push_back(temp);
    }
    return res;
}

bool matchTag(const std::string& tag, const std::string& tartag)
{
    if (tag == tartag) return true;
```

```
    return false;
}

bool matchClass(const std::string& att, const std::string& clss)
{
    if (att.empty()) return false;
    std::vector<std::vector<std::string>> att_val = splitTagAttribute(att);
    size_t attsize = att_val.size();
    for (size_t i = 0; i < attsize; i++)
    {
        if (att_val[i][0] == "class")
        {
            std::string val = att_val[i][1];
            val = val.substr(1, val.length() - 2);
            std::vector<std::string> valvec = split(val, '␣');
            for (auto& i : valvec)
            {
                if (i == clss)
                    return true;
            }
        }
    }
    return false;
}

bool matchID(const std::string& att, const std::string& id)
{
    if (att.empty()) return false;
    std::vector<std::vector<std::string>> att_val = splitTagAttribute(att);
    size_t attsize = att_val.size();
    for (size_t i = 0; i < attsize; i++)
    {
        if (att_val[i][0] == "id")
        {
            std::string val = att_val[i][1];
            val = val.substr(1, val.length() - 2);

            if (val == id)
                return true;
        }
    }
    return false;
}

bool matchAttVal(const std::string& att, const std::string& input)
{
    if (att.empty()) return false;
```

```
  std::vector<std::string> target = splitAttribute(input);
std::vector<std::vector<std::string>> att_val = splitTagAttribute(att);
size_t attsize = att_val.size();
size_t tarsize = target.size();
  if (target[tarsize - 1] == "%")
  {
      for (size_t i = 0; i < attsize; i++)
          if (att_val[i][0] == target[0])
              return true;
  }
else if (target[tarsize - 1] == "=")
{
  for (size_t i = 0; i < attsize; i++)
  {
    if (att_val[i][0] == target[0])
    {
      std::string val = att_val[i][1];
      val = val.substr(1, val.length() - 2);
              std::string tarval = target[1];// .substr(1, target[1].length() -
                  2);
      if (val == tarval)
        return true;

    }
  }
}
else if (target[tarsize - 1] == "^")
{
  for (size_t i = 0; i < attsize; i++)
  {
    if (att_val[i][0] == target[0])
    {
      std::string val = att_val[i][1];
      val = val.substr(1, val.length() - 2);

              std::string tarval = target[1];// .substr(1, target[1].length() -
                  2);

      if (val.find(tarval) == 0)
      {
                  return true;
      }
    }
  }
}
else if (target[tarsize - 1] == "$")
{
```

```
    for (size_t i = 0; i < attsize; i++)
    {
      if (att_val[i][0] == target[0])
      {
                std::string val = att_val[i][1];
                val = val.substr(1, val.length() - 2);

                std::string tarval = target[1];// .substr(1, target[1].length() -
                    2);

        if (val.find_last_of(tarval) == val.length() - 1)
            return true;
      }
    }
  }
  else if (target[tarsize - 1] == "*")
  {
    for (size_t i = 0; i < attsize; i++)
    {
            if (att_val[i][0] == target[0])
            {
                std::string val = att_val[i][1];
                val = val.substr(1, val.length() - 2);

                std::string tarval = target[1];// .substr(1, target[1].length() -
                    2);

                if (val.find(tarval) != -1)
                    return true;
            }

    }
  }

    return false;
}

bool matchWave(const std::string& str, const CSNode<token>* cur,
    bool(*match)(const std::string&, const std::string&), std::string (*get)(const
        token&))
{
    CSNode<token>* bro = cur->parent->firstchild;
    while (bro->data.getType() != StartTag)
        bro = bro->nextsibling;
    CSNode<token>* sis = bro->nextsibling;
    if (bro == cur) return false;
    while (sis != cur)
```

```
    {
        std::string tomatch = get(bro->data);
        if (match(tomatch, str))
            return true;
        if (sis->data.getType() == StartTag)
            bro = sis;
        sis = sis->nextsibling;
    }

    return false;
}

bool matchSpace(const std::string& str, const CSNode<token>* cur, const CSNode<
    token>* root,
    bool(*match)(const std::string&, const std::string&), std::string(*get)(const
        token&))
{
    CSNode<token>* parent = cur->parent;
    while (parent != root)
    {
        std::string tomatch = get(parent->data);
        if (match(tomatch, str))
            return true;
        parent = parent->parent;
    }
    return false;
}



std::string getParentAtt(const tokenNode& tn)
{ return tn.ptr->parent->data.getAttribute(); }

std::string getParentTag(const tokenNode& tn)
{
    return tn.ptr->parent->data.getTagName();
}

std::string getMineAtt(const tokenNode& tn) { return tn.t.getAttribute(); }

std::string getMineTag(const tokenNode& tn)
{
    return tn.t.getTagName();
}

std::string getSiblingAtt(const tokenNode& tn)
{
```

```
    CSNode<token>* bro = tn.ptr->parent->firstchild;
    while (bro->data.getType() != StartTag)
        bro = bro->nextsibling;

    if (bro == tn.ptr) return std::string(); // 自己就是长兄结点，返回空字符串

    CSNode<token>* sis = bro->nextsibling;

    while (sis != tn.ptr)
    {
        if (sis->data.getType() == StartTag)
            bro = sis;
        sis = sis->nextsibling;
    }
    return bro->data.getAttribute();

}

std::string getSiblingTag(const tokenNode& tn) {
    CSNode<token>* bro = tn.ptr->parent->firstchild;
    while (bro->data.getType() != StartTag)
        bro = bro->nextsibling;
    if (bro == tn.ptr) return std::string(); // 不存在长兄结点，返回空字符串
    CSNode<token>* sis = bro->nextsibling;

    while (sis != tn.ptr)
    {
        if (sis->data.getType() == StartTag)
            bro = sis;
        sis = sis->nextsibling;
    }
    return bro->data.getTagName();
}

std::string getTag(const token& t)
{
    return t.getTagName();
}

std::string getAtt(const token& t)
{
    return t.getAttribute();
}
```

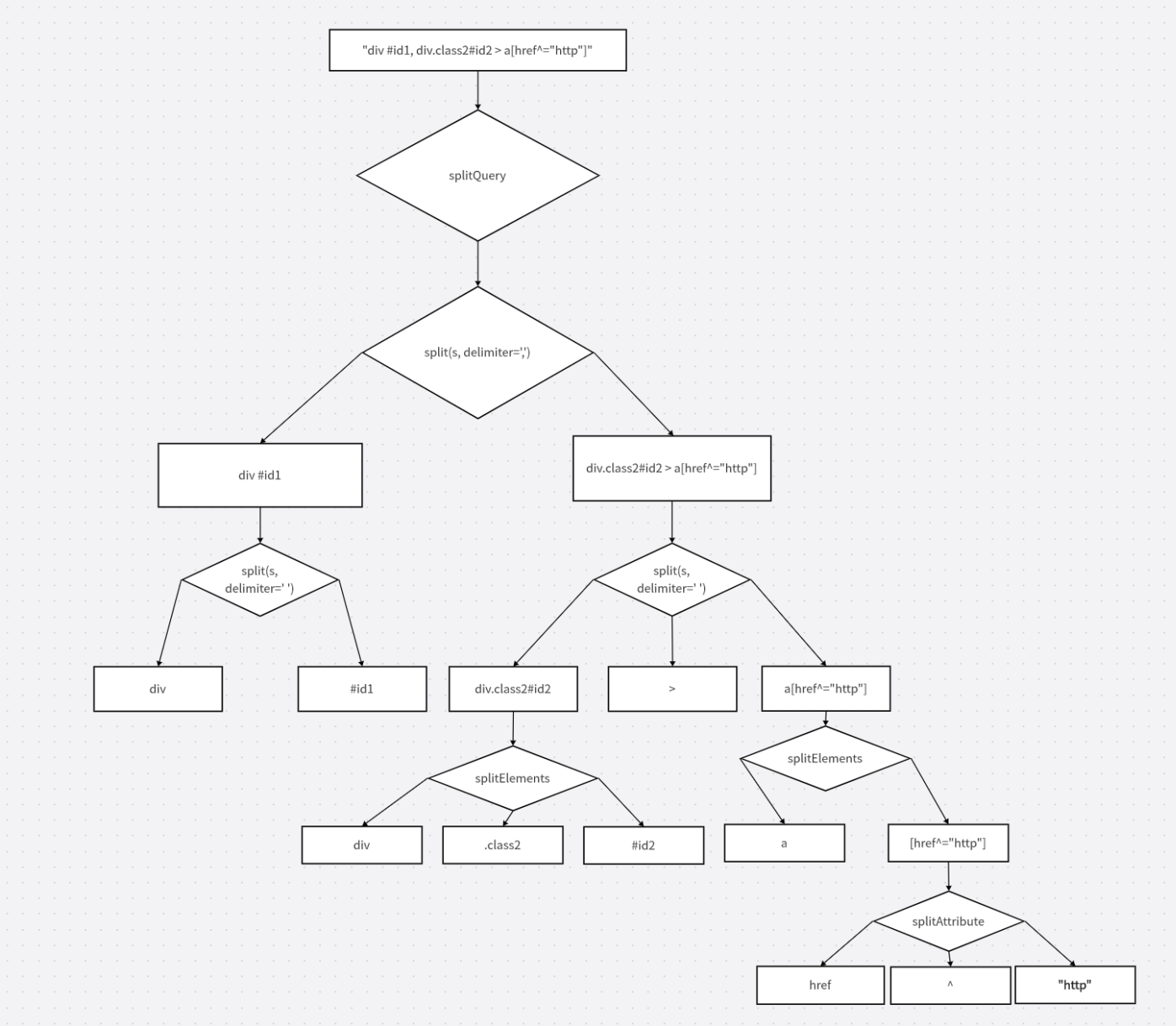

图 1: 处理输入 query 过程

四、调试分析

1 (1) 在实验过程中，一开始处理读入 html 文件并构建 DOM 树出现困难，查阅相关资料后得以解决。

(2) 处理 query 时发现 query 可以包含多个层次，可用类似于广义表的数据结构进行存储，但是没能实现。只采用一层一层处理，可能较为复杂。一开始想按从左到右的顺序一次处理 query 寻找 node，后面阅读 CSS 选择器实际运行原理相关文献，发现从右到左倒序处理更加便捷。故采用。

2 本实验根据实际，设计两个类 token 和 tokenNode。但是后面发现其实不需要 tokenNode，tokenNode 的两个元素指向 token 在树的结点指针 CSNode<token>* ptr 和 token t，而实际上 t = ptr->data. 所以只保留该指针即可，但是因改动成本大，没有实现。

3 时空复杂度分析

假设 html 文件有 m 行，每行有 n 个字符，树总共有 t 个结点，query 切分成 k 个小项，query 查询出来的结果有 s 个。

(1) 构建 dom 树的时间复杂度为 $O(m + m * n)$.

(2) query 查询初始化结构的时间复杂度为 $O(t)$. 之后每次过滤时间复杂度为 $O(s)$，s 的值每次过滤之后都有可能更新。

(3) 程序主要占用辅助空间是树和 query 切分出的小项顺序表和查询结果的顺序表，空间复杂度为 $O(t + k + s)$.

4 对照需求分析，本实验还应该要处理中文输入，但受限于时间未能实现，只能实现英文输入。

五、用户手册

1 本实验运行环境为 Windows 操作系统，执行文件为：HTML_CSS.exe

2 进入用户界面后，直接输入 ‘read(”filepath”)‘即可输入文件并构建 DOM 树。

3 query 处理输入 query(”div,div > class2[a$=”pdf”],注意逗号分割的空格可有可无，而’>’,’∼’,’+’选择时前后要留出空格。attribute 选择器的’=’ 前后不留空格。返回所有满足条件的结点，并在下一行显示个数。

4 对第 k 个 node 进行操作如下，注意大小写敏感。

- Out[k].outerHTML
- Out[k].innerText
- Out[k].href
- Out[k].query(selector)

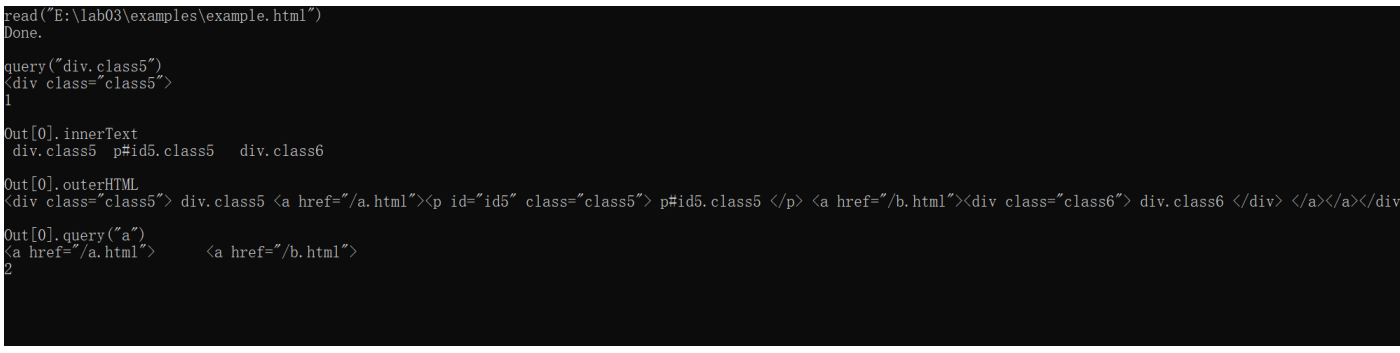

图 2: 用户界面

六、测试结果

6.1 lab3_news.html

Input(1) query(”.channel-nav-li”)

Output(1) <li class=”channel-nav-li”> <li class=”channel-nav-li”> <li class=”channel-nav-li”> <li class=”channel-nav-li”> <li class=”channel-nav-li channel-nav-more”>

5

Input(2) query(”.holding.with-hover.weibo-logo”)

Output(2) <div class=”holding with-hover weibo-logo”>

1

Input(3) query(”.channel-nav-li .sc”)

Output(3) <a href=”/channel_3560” class=”sc” data-name=”sc” name=”/channel_3560”>

1

Input(4) query(”baoliao”)

Output(4) <a href=”/aboutUs_write_uswrite_us” target=”_blank” id=”baoliao”>

1

Input(5) query(”p *”)

Output(5) <a href=”javascript:void(0);”>   <img src=”https://imgcdn.thecover.cn/@ orient/thumbnail/1080xgt;/strip/quality/95/ignore-error/1|imageslim”> <img src=”https://imgcdn.thecover.cn/@/ca orient/thumbnail/1080xgt;/strip/quality/95/ignore-error/1|imageslim”>

8

Input(6) query(”span”)

Output(6)

12

6.2 lab3_newslist.html

Input(1) query(”div,a”)

Output(1)   · · ·

177

Input(2) query(”div a”)

Output(2) <a href=”/” class=”homepage more”> <a href=”http://www.ruc.edu.cn” target=”_blank” class=”more”>

. . .

134

Input(3) query(”div.logo”)

Output(3)

1

Input(4) query(”div > a”)

Output(4) <a href=”/” class=”homepage more”> <a href=”http://www.ruc.edu.cn” target=”_blank” class=”more”>

. . .

32

Input(5) query(”input ∼ input”)

Output(5) <input type=”image” id=”search_btn” name=”commit” class=”submit” value=”” src=”/wp-content/themes/rucnews
bg.png”>
1

七、附录

datadef.h
linklist.h
linkstack.h
lnode.h
sqlist.h
terminal.h
terminal.cpp
token.h
token.cpp
tree.h
util.h
util.cpp
main.cpp