计算机科学
COMPUTER SCIENCE

一种用于 RPA 系统的 DOM 对象快速搜索与定位算法

孟媛，秦云川，蔡宇辉，李肯立

引用本文

孟媛，秦云川，蔡宇辉，李肯立. 一种用于 RPA 系统的 DOM 对象快速搜索与定位算法[J]. 计算机科学, 2022, 49(10): 252-257.

MENG Yuan, QIN Yun-chuan, CAI Yu-hui, LI Ken-li. Fast DOM Object Search and Location Algorithm for RPA System[J]. Computer Science, 2022, 49(10): 252-257.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210900210

一种用于 RPA 系统的 DOM 对象快速搜索与定位算法

孟 媛 秦云川 蔡宇辉 李肯立
湖南大学计算机科学与工程学院 长沙 410082
(mengyuan2020@hnu.edu.cn)

摘 要 机器人流程自动化(RPA)是以软件机器人及人工智能为基础的业务过程自动化科技,能够代替或协助人类在计算机等设备中完成重复性工作。在应用 RPA 软件对浏览器页面元素进行自动化操作时,在保证准确的前提下快速对目标 DOM 元素进行定位和搜索是完成一个完整自动化流程的关键技术难点。现有的定位方法,如 Xpath 和 Css-Selector,面对结构复杂的网页会出现路径过长的问题,导致定位速度慢或路径定位不准等。为解决上述问题,提出一种用于 RPA 系统的 DOM 对象快速搜索与定位算法——最优 XPATH 路径算法。该算法分析元素的属性等信息生成最优路径,用于在自动化操作时对元素进行唯一定位。实验结果表明,使用最优路径对元素进行定位所需时间仅为使用完整 XPATH 路径定位耗时的 23.14%,说明所提算法具有降低路径生成难度,加快元素定位速度等优点,提高了自动化效率。

关键词:机器人流程自动化;DOM 元素搜索;DOM 元素定位;自动化;网页结构
中图法分类号 TP312

Fast DOM Object Search and Location Algorithm for RPA System

MENG Yuan,QIN Yun-chuan,CAI Yu-hui and LI Ken-li
School of Computer Science and Engineering,Hunan University,Changsha 410082,China

Abstract Robot process automation(RPA) is a business process automation technology based on software robot and artificial intelligence. It can replace or assist human beings to complete repetitive work in computers and other equipments. When applying RPA software to automate the browser page elements,how to quickly locate and search the target DOM elements on the premise of ensuring accuracy is the key technical difficulty to complete a complete automation process. The existing location methods,such as XPath and Css-Selector,will have problems such as slow location speed or inaccurate path location in the face of web pages with complex structure. In order to solve the above problems,a fast DOM object search and location algorithm for RPA system is proposed:the optimal XPATH path algorithm,which analyzes the attributes of elements and generates the optimal path to uniquely locate elements during automatic operation. Experimental results show that the time required to locate elements using the optimal path is only 23.14% of that using the complete XPATH path. It has the advantages of reducing the difficulty of path generation and improving the element positioning speed,and improves the automation efficiency.

Keywords Robot process automation,DOM element search,DOM element positioning,Automation,Web page structure

1 引言

机器人流程自动化(Robotic Process Automation)是以软件机器人及人工智能为基础的业务过程自动化科技。RPA 软件在技术上类似图形用户界面测试工具[1],可以自动和图形用户界面进行互动,并且能够通过示范性编程来复现使用者进行的一系列操作,协助人类完成重复性工作[2]。RPA 软件允许资料在不同应用程序之间交换,例如接收电子邮件可能包括接收付款单、取得其中资料、输入到簿记系统中等步骤[3]。RPA 软件的目标是使符合适用性标准的基于桌面的业务流程和工作流程实现自动化操作[4]。一般来说,这些操作在很大程度上是重复的,数量也较多,并且可以通过严格的规则和结果来定义。

在应用 RPA 软件对浏览器页面元素进行自动化操作时,通过使用浏览器插件将 js 代码注入需要自动化操作的页面,使用 js 代码来定位 DOM 元素,即可对特定位置进行特定的操作,从而达到动态的效果。

为了定位到需要操作的 DOM 元素,RPA 软件需要设计一种 DOM 对象快速搜索与定位算法,使用树型结构管理 DOM 元素,每个 DOM 元素可用一个从 HTML 根节点出发的路径进行唯一表达,定位时将路径传入 js 端[5-8],通过遍历 path 中的节点,在页面中依次搜索对应的 DOM 元素,最终

到稿日期:2021-09-24 返修日期:2022-03-22
基金项目:国家重点研发计划(2017YFB0202201)
This work was supported by the National Key Research and Development Program of China(2017YFB0202201).
通信作者:秦云川(qinyunchuan@hnu.edu.cn)

即可定位到需要操纵的元素。

但是，实际应用 RPA 软件时，面对一些结构复杂的浏览器页面，一个元素的完整路径长度极有可能达到十几层或数十层，要求用户根据实际的网页结构从所需元素依次向上搜索父节点并连接生成需要操作的元素的完整路径十分困难，并且极易产生路径出错的问题。仅依靠获取完整路径实现元素定位的方法对用户的操作要求极高，如果无法获取正确且完整的元素路径，RPA 软件将无法搜索和定位到正确的 DOM 元素。

为了解决上述问题，本文提出一种用于 RPA 系统的 DOM 对象最优 XPATH 路径算法，该算法通过重新设计路径结构来降低路径生成难度，加快元素定位速度。

2　已有的 DOM 对象搜索与定位算法

本节主要介绍 DOM 对象的最优 XPATH 路径算法的相关技术，具体包括 RPA 系统中 DOM 对象搜索与定位的常用的几种算法，并对比分析现有算法的优缺点，为下一步算法的设计提供理论基础。

2.1　使用元素的属性定位

可以使用 DOM 元素的某些属性值对 DOM 元素进行定位，通常较为常用的属性有 id，class，name 等。使用 id 可以对元素进行唯一定位，但是实际网页中很多元素没有 id 属性，而 class 和 name 等属性值的重复率较高，通常无法对元素进行唯一定位。

2.2　使用 Css-Selector 进行定位

Css-Selector 的格式如下：

“HTML＞body＞div＞div＞div＞div＞div＞form＞span＞input”

使用 Css-Selector 对元素进行定位通常需要获取元素在 DOM 树中的完整路径，在一些结构复杂的网页中生成和定位耗时较长[9-10]。同时，Css-Selector 路径中只包含元素的类型信息，这可能会导致同时定位到 DOM 树中多个结构相同的元素，无法保证定位的唯一性。

2.3　使用 XPATH 进行定位

XPATH 的格式如下：

“/ HTML/body/div[1]/div[3]/div[1]/div[3]/div[7]/form/span/input[1]”

使用 XPATH 对元素进行定位通常需要获取元素在 DOM 树中的完整路径，其与 Css-Selector 的不同之处在于可以通过索引进行精确定位，避免出现同时定位多个元素的问题[11]。针对页面结构复杂的网站，使用 XPATH 对元素进行定位会出现元素路径过长的问题，导致定位速度慢。

3　最优 XPATH 路径算法的基本思想

本节主要介绍 DOM 对象最优 XPATH 路径算法的设计思路。上一节阐述了 RPA 系统中 DOM 对象定位上的需求以及若干定位方法的优缺点。在此基础上，提出了一种 DOM 对象的最优 XPATH 路径算法，并将算法分为搜索算法和定位算法两部分进行介绍。

3.1　最优 XPATH 路径搜索算法

在搜索算法中，目标元素的最优路径是在 XPATH 路径的基础上生成的。在获得最优路径之前，需要对 DOM 树进行 DFS 获取元素的 XPATH 路径，即获取从根节点到目标节点的路径上的所有节点，并将这些节点存储在一个列表中。在得到有关节点的列表之后计算最优路径。搜索算法的流程如图 1 所示。

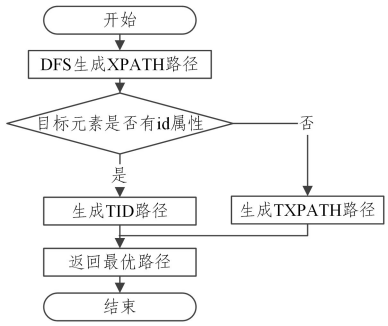


图 1　搜索算法流程图

Fig. 1　Flow chart of search algorithm

最优解分为两种情况：

(1)被点击的元素本身具有 id 属性。这种情况下节点的表示只要头部和 id，路径的样例如下：

TID/chrome. exe title＝xxx/id：xxx

(2)被点击的元素本身不具有 id 属性。向上遍历完整的 XPATH 路径，直到搜索到距离节点最近的具有 id 属性的祖先节点。元素的最优路径为该祖先节点到目标元素的完整路径，其中除了祖先节点由 id 属性表示以外，其余路径上的节点格式与 XPATH 路径的节点格式相同。如果目标元素在 IFRAME 内部，则需要将 IFRAME 节点的类型及其 id 或 src 属性补充在路径头部。路径的样例如下：

TXPATH/chrome. exe title＝xxx/IFRAME@src：xxx/IFRAME@id：xxx /…/tagName@id：xxx/完整路径

3.2　最优 XPATH 路径定位算法

定位算法的流程如图 2 所示。

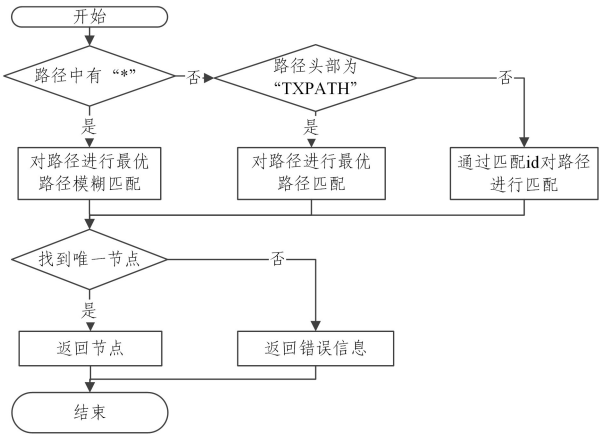


图 2　定位算法流程图

Fig. 2　Flow chart of location algorithm

获取 json 文件中的元素路径，如果判断路径类型为“TXPATH”，则路径为最优路径。首先找到距离叶子节点最近的

具有 id 的祖先节点，之后按照完整路径依次搜索下一个子节点，直至遍历全部路径找到叶子节点。如果头部为“TID”，该路径是通过 DOM 元素自己的 id 属性进行定位，直接使用元素自己的 id 属性值即可搜索到唯一确定 DOM 元素。在搜索到唯一确定的元素后返回该节点。

4 最优 XPATH 路径算法的实现

本节主要介绍 DOM 对象最优 XPATH 路径算法的实现方法。根据上一节设计的算法的主要流程，本节对算法的具体实现方法进行设计。所提算法主要分为搜索算法和定位算法。

4.1 最优 XPATH 路径搜索算法

搜索算法的实现如算法 1 所示。

算法 1 XPATH 路径生成算法

```
输入：根节点 DOM 元素 root，遍历过的路径 path，需要拾取的 DOM
      元素 aimDOM
输出：元素的 XPATH 路径 selectorPath
1. nextpath ← getXpath(path, root);
2. ok ← flag that indicates whether the target node is found
3. if isEqual(root, aimDOM)
4.    push root node to DOMList;
5.    selectorPath ← getSourcePath("XPATH")+nextpath;
6.    ok ← 0;
7. for each node in root.childNodes do
8.    if node.nodeType ! =target nodeType or ok=0
9.    continue;
10  if node.nodeType="IFRAME"
11.       push node to DOMList;
12.          getSelectorByXpath(next DOM, getXpath(nextpath, root.
             childNodes [i]), aimDOM);
13.    else
14.         getSelectorByXpath (root. childNodes [i], nextpath, aim-
            DOM);
15. output selectorPath
```

在确定需要定位的目标 DOM 元素后开始执行搜索算法，算法使用 DFS 对 DOM 树进行遍历。其中，*root* 代表当前被搜索到的节点，*path* 表示当前节点的父节点的 XPATH 路径，*aimDOM* 表示拾取得到的 DOM 对象。

在 DFS 的过程中，进入路径中的一个节点时，首先执行算法 1 第 1 行，通过 getXpath 方法生成根节点到当前节点的 XPATH 路径，并将该节点保存到用于储存元素路径的列表 *DOMList* 中。

第 2 行表示设置一个标记位 *ok* 用于标记目标节点是否被搜索到，并设定若初始值为 1 则表示没有被搜索到，若值为 0 则表示被搜索到。第 3－6 行表示使用 isEqual 方法判断被搜索到的节点是否为目标节点，若是同一节点，则将该节点加入 *DOMList* 列表和 XPATH 路径中，并设置标志位 *ok* 为 0，退出 DFS。

第 7－9 行表示退出 DFS 进行回溯时，首先检查标志位 *ok*，若 *ok* 为 1，表示没有找到目标节点，最后一次加入 *DOMList* 的节点不在目标路径中，则将该节点移除出列表。

第 10－14 行表示遍历到 IFRAME 节点时，需要获取 IFRAME 中的内联文档，将其中的 HTML 节点当做 IFRAME 节点的子节点进行搜索，若遍历到其他普通节点则无需特殊处理，对元素的子节点继续搜索即可。搜索算法结束后，返回生成的 XPATH 路径 *selectorPath*。

以每个 DOM 元素被搜索到的次数为依据进行时间复杂度分析。设 S_i 为第 i 个 DOM 元素的子节点个数，n 为 DOM 树的节点总数。

$$n=S_1+S_2+S_3+\cdots+S_n \tag{1}$$

由于对 DOM 树进行 DFS 遍历时每个节点只被搜索到一次。

$$T(n)=S_1+S_2+S_3+\cdots+S_n \tag{2}$$

可得：

$$T(n)=n \tag{3}$$

即 XPATH 路径生成算法的时间复杂度为 $O(n)$。

当调用生成 XPATH 路径的方法结束时，*DOMList* 列表存储着从根节点到目标节点的路径的所有节点的 DOM 对象，*selectorPath* 存储着从根节点到目标节点的 Xpath 路径。

在 onClick 方法中，需要判断目标节点的 id 属性是否存在。如果存在，首先调用 isIframe 方法，判断 *DOMList* 中是否存在 IFRAME 节点。若存在 IFRAME 节点，则对 *DOMList* 中的节点进行遍历，将查找到的 IFRAME 节点依次加入最优路径头部。IFRAME 节点使用 src 或 id 属性值进行定位。因此，在生成路径时，将每个 IFRAME 节点的格式设置为“IFRAME@id|src:xxx”，连接后路径格式为“IFRAME@src:xxx/…/IFRAME@id:xxx”。最后在路径尾部加上目标节点即可生成完整的最优路径，目标节点由节点类型和 id 属性值表示。

若目标节点不包含 id 属性，则依次将 IFRAME 节点加入路径，之后由目标节点向上遍历完整的 XPATH 路径，直到搜索到距离节点最近的具有 id 属性的祖先节点。元素的最优路径为该祖先节点到目标元素的完整路径，其中除了祖先节点由 id 属性来表示以外，其余路径上的节点格式与 XPATH 路径的节点格式相同，由节点类型和节点在兄弟节点中的排名来表示。

最优路径生成算法的伪代码如算法 2 所示。

算法 2 最优路径生成算法

```
输入：XPATH 路径 DOMList;
输出：最优路径 selector.
1. if 'IFRAME' in DOMList
2.    temp_path ← get_IFRAME_path();
3. pos ← the position of the node in the path;
4. if _node has no ID attribute
5.    xlocation ← get_after_IFRAME_path(pos);
6.       selector ← getSourcePath("TXPATH")+temp_path+xloca-
         tion;
7. else
8.    temp_csspath ← get_exist_id_path();
9.    selector ← getSourcePath("TXPATH")+temp_path+temp_
      csspath;
10. output selector
```

算法 2 第 1－2 行表示如果路径中包含 IFRAME 节点，则调用 *get_IFRAME_path* 获取路径中所有的 IFRAME 节点并依次加入最优路径。

第 3 行表示获取最后一层 IFRAME 节点后的第一个元素在路径中的位置 *pos*。如果 *pos* 之后的完整路径中不包含具有 id 属性的节点，则根据第 4－6 行调用 *get_after_IFRAME_path* 方法，获取由 *pos* 到目标节点的 XPATH 路径，将生成的路径与之前生成的 IFRAME 节点的路径相连，最后加上路径头部得到完整的最优路径。

第 7－9 行表示若最后一层 IFRAME 节点内的路径中存在具有 id 属性的祖先节点，则调用 *get_exist_id_path* 方法来获取祖先节点到目标节点的路径，除了祖先节点由节点类型和 id 属性值来表示以外，其他节点格式与 XPATH 路径的节点格式相同，最后与之前生成的 IFRAME 节点的路径相连并加上路径头部得到完整的最优路径。

4.2　最优 XPATH 路径定位算法

定位算法的实现流程如算法 3 所示。

算法 3　元素定位算法

```
输入：DOM 元素路径 elem_selector
输出：定位的 DOM 元素 DOM
1. if head is "TXPATH"
2.    for each node in path do
3.       if node is represented in the form of "node type @ attribute.
         value"
4.          DOM ← DOM. querySelector('node type [attribute="va-
            lue"]');
5.       else
6.          tag ← node type of node;
7.             tag_id ← position of nodes among brothers;
8.          for each Node in DOM. children do
9.             if Node. tagName=tag and position=tag_id
10.                DOM ← Node;
11. else
12.                DOM ← document. querySelectorAll ('[id = "no-
                   deid"]');
12. output DOM
```

算法 3 第 1 行表示如果判断路径类型为“TXPATH”，则将 *path* 路径按分隔符进行划分得到节点集合，第 2 行表示遍历路径中节点。

第 3－5 行表示如果当前读取到的节点是用“节点类型@属性名：属性值”的形式来表示的，那么首先将节点按“@”分割，得到节点的类型，之后将后半段用“：”分割，得到节点的属性名和属性值。根据 *querySelector* 需要的格式，将得到的信息组合拼接成“节点类型[属性名＝'属性值']”的格式，最后使用 *querySelector* 定位节点。

第 6－10 行表示如果节点中不包含“@”，则表示节点是使用“节点类型[节点在兄弟中的排名]”来表示的。如果路径包含“[”，则将路径按“[”和“]”来分割，得到节点类型和节点在兄弟中的排名。如果路径不包含“[”，则默认节点在兄弟中的排名为 1。其中第 9－11 行表示遍历父节点的全部子节点，如果节点类型符合，且节点在兄弟中的排名也正确，则返回该节点。

第 11 行表示如果头部为“TID”，那么该路径是通过 DOM 元素自己的 id 进行定位的。将路径按“：”进行分割，得到节点的 id 属性值。将得到的信息拼接为“[id＝'属性值']”，使用 *querySelector* 进行定位。最后，在遍历整条路径后，返回搜索到的节点 DOM。

以每个 DOM 元素被搜索到的次数为依据进行时间复杂度分析。设 k_i 为路径中的第 i 个节点的子节点数，n 为路径上的节点总数。

$$T(n)=k_1+k_2+k_3+\cdots+k_n \tag{4}$$

设 $\bar{k}$ 为路径上节点的子节点数的平均值，可得：

$$T(n)=n*\bar{k} \tag{5}$$

即元素定位算法的时间复杂度为 $O(n)$。

5　实验结果与应用价值

本节主要展示算法的实际效果。在上一节对 DOM 对象最优 XPATH 路径算法进行具体实现后，将算法应用于 RPA 软件的拾取和自动化流程中，提高元素搜索效率。

5.1　RPA 实际应用

图 3 给出了 RPA 软件的部分界面，软件的主要功能是创建一个流程，以实现对浏览器中网页元素的一系列自动化操作。具体操作步骤为：

(1)拾取浏览器页面中需要操作的元素；

(2)将拾取元素按操作顺序连接并根据需求补充操作信息，如输入框填充内容等；

(3)开始自动化操作。

其中，步骤(1)中的拾取操作应用了 4.1 节中的搜索算法，生成了用于对元素进行唯一定位的路径；步骤(3)中的自动化操作应用了 4.2 节的定位算法，在浏览器页面根据步骤(1)中生成的路径定位到需要操作的具体元素。使用最优路径标识和定位元素，在保证正确定位元素的基础上，缩短元素定位的时间，使一个完整的流程可以快速准确地完成，提升用户使用体验，优化流程效率。

图 3　RPA 软件界面

Fig. 3　RPA software interface

使用软件对几种不同类型的元素进行拾取和操作，得到的路径结果与算法的预期结果一致。

5.2　包含 id 属性的元素

如图 4 所示，拾取的元素为测试网页的输入框。元素在 DOM 树中的部分结构如下所示，其中最后一个 INPUT 节点即为拾取对象。

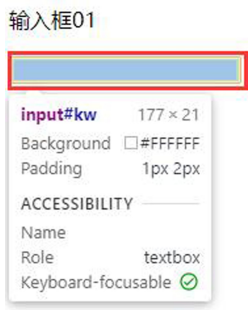


图 4 有 id 属性的元素

Fig. 4 Element with id attribute

```
〈html lang="en"〉
〈head〉…〈/head〉
〈body〉
  〈textarea id="1" hidden〉〈/textarea〉
  〈textarea id="2" hidden〉〈/textarea〉
  〈div id="3"〉
    〈div〉…〈/div〉
    〈div id="4"〉
      〈div id="5"〉〈/div〉
      〈div id="6"〉〈/div〉
      〈div id="7"〉〈/div〉
      〈div〉…〈/div〉
      〈div id="8"〉
        〈div id="9"〉
          〈div id="10"〉
          〈div id="11"〉〈/div〉
          〈a〉…〈/a〉
          〈form id="12"〉
            〈span〉
            〈p〉输入框 01〈/p〉
            〈input id="kw"〉
```

生成的最优路径为：

TID/chrome. exe 测试网页/id:kw

因为该元素带有 id 属性，且 id 属性在当前页面可以唯一标识该元素，判断元素仅通过 id 属性即可定位，所以路径为 TID 类型，路径中包含的信息仅为 id 属性值。

5.3 不包含 id 属性的元素

如图 5 所示，拾取的元素为测试网页的按钮，元素在 DOM 树中的部分结构如下所示，其中最后一个 INPUT 节点即为拾取对象。

图 5 没有 id 属性的元素

Fig. 5 Element without id attribute

```
〈html lang="en"〉
〈head〉…〈/head〉
〈body〉
  〈textarea id="1" hidden〉〈/textarea〉
  〈textarea id="2" hidden〉〈/textarea〉
  〈div id="3"〉
    〈div〉…〈/div〉
    〈div id="4"〉
      〈div id="5"〉〈/div〉
      〈div id=""6"〉〈/div〉
      〈div id="7""〉〈/div〉
      〈div 〉…〈/div〉
      〈div id="8"〉
        〈div id="9"〉
          〈div id="10"〉
          〈div id="11"〉〈/div〉
          〈a〉…〈/a〉
          〈form id="12"〉
            〈span〉
              〈p〉输入框 01〈/p〉
              〈input id="kw"〉
            〈input type="submit" value="按钮 01"〉
```

生成的最优路径为：

TXPATH/chrome. exe 测试网页/|FORM@id:12|SPAN[1]|INPUT[2]

因为元素本身没有 id，不能通过 id 属性进行唯一定位，所以生成的最优路径为 TXPATH 类型，向上遍历得到距离目标元素最近的具有 id 属性的祖先节点为“FORM”元素，因此将最优路径的根节点记录为祖先节点类型“FORM”及其 id 属性“12”，之后的路径为“FORM”元素到目标元素的完整路径，路径中的元素由元素类型和元素在兄弟节点的排名组成。

5.4 IFRAME 内部的元素

如图 6 所示，拾取的元素为测试网页的标签，元素在 DOM 树中的部分结构如下所示，其中最后一个 SPAN 节点即为拾取对象。

图 6 IFRAME 内的元素

Fig. 6 Elements within IFRAME

```
〈iframe frameborder="0" id="iframeResult" name="iframeResult"〉
  〈html〉
    〈head〉〈/head〉
    〈body〉
      〈textarea id="1" hidden〉〈/textarea〉
      〈textarea id="2" hidden〉〈/textarea〉
      〈div id="3"〉
      〈div〉…〈./div〉
```

```
〈form id="searchform"〉
  〈input id="inputo1" hidden〉
  〈input id="inputo2" hidden〉
  〈input id="input03" hidden〉
  〈div class="class1"〉…〈/div〉
  〈div〉
    〈div class="class02"〉
      〈span id="span01"〉标签 01〈/span〉
```

生成的最优路径为：

TXPATH/chrome. exe 测试网页/| IFRAME@id：iframeResult| SPAN@id：span01

首先判断元素在 IFRAME 元素内且元素具有 id 属性，将路径的根节点记录为 IFRAME 节点类型及其 id 属性值“iframeResult”；之后判断目标元素具有 id 属性，因此路径由元素类型“SPAN”及 id 属性值“span01”组合表示。完整的最优路径由以上两部分组成。

5.5　实验结果

表 1 列出了分别使用完整 XPATH 路径和最优路径对上述举例的元素进行定位所需的时间。结果表明，使用最优路径对元素进行定位所需时间仅为使用完整 XPATH 路径定位耗时的 23.14%。使用最优路径对于结构复杂的网页且路径中有 id 属性节点的 DOM 元素定位速度提升较大，节省了元素的搜索时间，提高了使用效率。

表 1　不同方法元素定位所用时间

Table 1　Time spent in different methods

XPATH			Optimal Path		
Start	End	Spend	Start	End	Spend
8.0999	8.5999	0.5000	8.6999	8.7999	0.1000
2.0999	2.5999	0.5000	2.5999	2.7008	0.1009
1.9000	2.2000	0.3000	2.2999	2.4000	0.1001

结束语　针对使用 RPA 系统对浏览器网页中 DOM 元素操作时进行的搜索和定位过程中出现的问题，本文提出了一种 DOM 对象的最优 XPATH 路径算法。在得到用户需要操作的 DOM 元素后，根据元素的属性特征对应生成两种路径格式：如果元素本身有 id 属性，则直接使用元素 id 属性值作为元素的唯一标识；如果元素没有 id 属性，则向上搜索距离叶子节点最近的有 id 的祖先节点，从该节点开始使用完整路径来表示 DOM 元素。两种路径以不同的头部作为区分，定位时通过判断头部选择不同的路径解析方法来定位元素。在面对结构较为复杂的网页时，使用这种搜索和定位算法可以大大缩短路径的长度，降低生成路径的难度，提高搜索元素的速度。

本文提出的方法还存在不足。首先，算法未设计跨域的 IFRAME 节点内 DOM 元素的搜索与定位方法，下一步拟解决有跨域 IFRAME 节点的页面的路径生成问题。其次，算法还可进一步优化，例如在保证属性值唯一确定的情况下，可使用 DOM 元素 id 属性之外的其他属性来定位该元素，进一步缩短生成的路径长度，提高元素搜索效率。

参 考 文 献

[1] CHUONG L V, HUNG P D, DIEP V T, et al. Robotic Process Automation and Opportunities for Vietnamese Market[C]// Proceedings of The 7th International Conference on Computer and Communications Management. 2019: 94-98.

[2] UNAL M A, BOLUKBAS O. The Acquirements of Digitalization with RPA (Robotic Process Automation) Technology in the Vakif Participation Bank[C]// ICISS 2021: 2021 The 4th International Conference on Information Science and Systems. 2021: 68-73.

[3] ISSAC R, MUNI R, DESAI K. Delineated Analysis of Robotic Process Automation Tools[C]// 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC). 2018: 1-5.

[4] XU Y F, LIU Y, WU W P. Research and Application of Social Network Data Acquisition Technology[J]. Computer Science, 2017, 44(1): 277-282.

[5] LI W Q, SUN X, ZHANG C Y, et al. A Semantic Similarity Measure between Ontological Concepts [J]. ACTA Automatica Sinica, 2012, 38(2): 229-235.

[6] WU G Q, HU J, LI L, et al. Online Web News Extraction via Tag Path Feature Fusio[J]. Journal of Software, 2016, 27(3): 714-735.

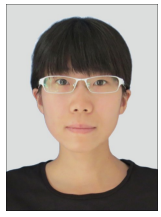
[7] SONG J, YANG X F, LI Y C, et al. Research onRecognition Algorithm for Subject Web Pages Based on Tag Tree Adjacency Matrix[J]. Computer Science, 2016, 43(6): 316-320.

[8] NASSIRI H, MACHKOUR M, HACHIMI M. One Query to Retrieve XML and Relational Data [J]. Procedia Computer Science, 2018, 134: 340-345.

[9] UZUN E. A Regular Expression Generator Based on CSS Selectors for Efficient Extractionfrom HTML Pages [J]. Turkish Journal of Electrical Engineering and Computer Sciences, 2020, 28(6): 3389-3401.

[10] SU Q, LI Z Z, LIU T T, et al. Tree Structure Evaluation Visualization Model for Program Debugging[J]. Computer Science, 2021, 48(5): 68-74.

[11] THACKSTON R. Exploring the Use of XPath Queries for Automated Assessment of Student Web Development Projects [C]// SIGITE 20: The 21st Annual Conference on Information Technology Education. 2020: 255-259.



MENG Yuan, born in 1997, postgraduate. Her main research interests include artificial intelligence and pattern recognition.

QIN Yun-chuan, born in 1983, Ph.D, is a member of China Computer Federation. His main research interests include autonomous unmanned systems and high-performance embedded computing.

（责任编辑：何杨）