苏 州 大 学 实 验 报 告

院、系	计算机科学与技术	年级 专业	2020 软件工程	姓名	高歌	学号	2030416018
课程名称	信息检索综合实践					成绩	
指导教师	贡正仙	同组实验者	无		实验日期	2023 年 3 月 15 日	

实 验 名 称　　实验 3 页面信息抽取

一. 实验目的

了解爬虫与网页信息抽取的相关知识，学习并实现从 HTML 网页中提取需要的内容，构建搜索系统的文档（即数据）部分，供后续实验使用。

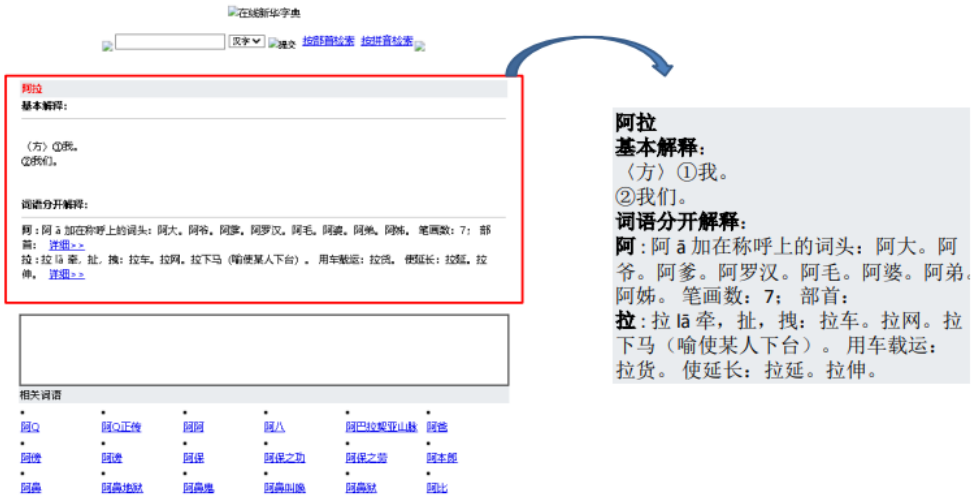
二. 实验内容

任务 1：单页抽取。从 data1 文件夹中的 Ng.htm 中提取标题与 body 的文本内容（不需要链接），并输出。

任务 2：多页抽取。

输入：一个指定的文件夹，其中包含 1000 个网页文件。假设文件夹为 data2.

输出：每个页面的纯文本数据问题。为 data2 里的每一个网页文件产生一个 txt 文件，包含文件中的主要内容（不需要链接，如下图所示），将所有生成的 txt 文件放到程序当前路径下的 output 文件夹中。



三. 实验步骤和结果

注：代码使用 TypeScript 编写，运行时使用 ts-node。使用 Prettier 与 ESLint 作为代码格式化工具，代码风格遵从 TypeScript ESLint Recommended 标准。

（一）实验步骤

1. 在`./src/`目录下创建`utils.ts`文件，用于导出辅助函数`extractText()`。

```
import * as cheerio from 'cheerio';

/**
 * Extract text from node.
 * @param $
 * @param node
 * @returns
 */
export const extractText = (
  $: cheerio.CheerioAPI,
  node: cheerio.AnyNode,
): string => {
  if (node.type === 'tag' && node.name === 'br') {
    return '\n';
  } else if (node.type === 'text') {
    if (/^\s+$/.test(node.data)) {
      return '';
    }
    return `${node.data.replace(/^\n+/, '').replace(/\n/g, ' ')}${
      node.data.at(-1) === '\n' ? '\n' : ''
    }`;
  } else {
    return $(node)
      .contents()
      .map((_, el) => extractText($, el))
      .get()
      .join('');
  }
};
```

该函数的作用是输入一个 cheerio Node（类似于 jQuery 的 Node 对象），并将其中的文本一行行地提取出来，最终输出一个字符串。例如，输入一个 body Node 就能得到<body>…</body>中的文本：`extractText($, $('body'))`.

这个函数的逻辑稍有些复杂，这里解释一下。

首先判断当前 node 是否是一个
，如是，则在结果中添加一个'\n'.

然后判断当前 node 是否是一个文本节点，如果是，再使用`/^\s+$/.test(node.data)`判断它是否是一个没有内容的空串，如果是，则直接返回空字符串，不在结果中添加不必要的空白字符；如果不是空白字符，则对文本稍作处理后返回，具体的处理过程如下：

①使用`replace(/^\n+/, '')`替换文本开头的若干个换行符，这是为了处理下面这样的元素：

```
<br>
13 Computing Drive<br>
```

cheerio 读取 HTML 时，对于这样的元素会把开头的空行读进去，而这是多余的，因此在这一步把开头的若干个换行符拿掉。

②使用`replace(/\n/g, ' ')`将文本中的换行符替换为空格（注意区别于上一步，这里替换为空格而不是空字符串），这是为了处理下面这样的元素：

```
School
of Computing<br>
```

对于这种元素，cheerio 也会将换行符读进去，然而 HTML 渲染时不会渲染这里的换行符，因此在这里将换行符替换为空格。

③如果文本本就以换行符结尾，则使用`${node.data.at(-1) === '\n' ? '\n' : ''}`保留**最后的**换行符。

这是由于 cheerio 在读取 … 、…等元素时，会认为它们的文本最后包含一个换行符。在上一步中，我们将所有换行符都替换成了空格，这会导致下面这样的元素最终也挤在一起，中间没有换行符：

```
conferences including ACL, EMNLP, SIGIR, AAAI, and IJCAI.</span></p>
<p>At NUS, he currently serves as
```

因此，为了保留这里的换行符，要检查文本原本在最后有没有换行符，如有，则原样保留。

最后，如果 node 既不是
也不是文本，则获取该 node 下的所有子 node，对它们递归地执行 extractText，最终再连接起来。

以上就是`extractText()`函数的大致逻辑。

2. 然后开始做练习 1。首先在`./src/`目录下创建`exercise1.ts`文件，它的框架如下：

```ts
import * as cheerio from 'cheerio';
import fs from 'node:fs/promises';
import { extractText } from './utils.js';

/**
 * Process a html file and save the result to another file.
 * @param sourcePath
 * @param outputPath
 * @param logOutput
 */
const exercise1 = async (
  sourcePath: string,
  outputPath: string,
  logOutput: boolean = false,
) => {
  ...
};

export default exercise1;
```

然后补全这里的`exercise1()`函数。它接受要处理的 HTML 文件的路径、结果输出的文件路径，和一个可选的`logOutput`变量，表示该函数运行时是否向终端打印信息。

```ts
/**
 * Process a html file and save the result to another file.
 * @param sourcePath
```

```
 * @param outputPath
 * @param logOutput
 */
const exercise1 = async (
  sourcePath: string,
  outputPath: string,
  logOutput: boolean = false,
) => {
  const html = await fs.readFile(sourcePath, 'utf-8');
  const $ = cheerio.load(html);

  // Title
  const title = $('title').text();
  fs.writeFile(outputPath, `title:\n${title}\n\n`, 'utf-8');
  if (logOutput) {
    console.log(`title:\n${title}\n`);
  }

  // Body
  const body = $('body')
    .contents()
    .map((_, el) => extractText($, el)) // Extract text from each node.
    .get()
    .join('')
    .replace(/\n{2,}/g, '\n') // Remove redundant newlines.
    .trim()
    .split('Google Scholar', 2)[0]; // Remove words after "Google Scholar".
  await fs.appendFile(outputPath, `body:\n${body}\n`, 'utf-8');
  if (logOutput) {
    console.log(`body:\n${body}`);
  }
};
```

该函数比较简单。首先读取 HTML 文件，使用 cheerio 加载该文件，然后选择<title>…</title>元素，获取标题内容并写入到文件，同时打印到终端（若 logOutput 为 true）。

然后，选择<body>…</body>元素，对它的所有子元素调用`extractText()`，然后将得到的字符串数组组合到一起。这里使用了`replace(/\n{2,}/g, '\n')`将多个换行符替换成一个换行符，防止结果中出现不必要的空行，并使用了`trim()`去除字符串前后的若干个空白字符，还使用`split('Google Scholar', 2)[0]`去除出现在“Google Scholar”之后的不必要的链接。同理，在此之后，将结果写入文件，并打印到终端（若 logOutput 为 true）。

3. 继续做练习 2。在`./src/`目录下创建`exercise2.ts`文件，它的框架如下：

```
import * as cheerio from 'cheerio';
import fs from 'node:fs/promises';
import path from 'node:path';
```

```
import { extractText } from './utils.js';

/**
 * Process all html files in a folder and save the result to another folder.
 * @param sourceFolder
 * @param outputFolder
 * @param logProgress
 */
const exercise2 = async (
  sourceFolder: string,
  outputFolder: string,
  logProgress: boolean = false,
) => {
  ...
};

export default exercise2;
```

同理，补全这里的`exercise2()`函数。它接受要处理的包含 HTML 文件的**文件夹**的路径、结果输出的**文件夹**路径，和一个可选的`logOutput`变量，表示该函数运行时是否向终端打印信息。

```
/**
 * Process all html files in a folder and save the result to another folder.
 * @param sourceFolder
 * @param outputFolder
 * @param logProgress
 */
const exercise2 = async (
  sourceFolder: string,
  outputFolder: string,
  logProgress: boolean = false,
) => {
  // Get all file names in the source folder.
  const fileNames = await fs.readdir(sourceFolder);

  for (const fileName of fileNames) {
    const sourcePath = path.join(sourceFolder, fileName);
    const outputPath = path.join(outputFolder, `${fileName.slice(0, -4)}txt`);
    if (logProgress) {
      console.log(`Processing "${sourcePath}" to "${outputPath}"...`);
    }

    // Load html file.
    const html = await fs.readFile(sourcePath, 'utf-8');
    const $ = cheerio.load(html);

    // Extract the main part of the page.
```

```
      const body = $('table[bgcolor="#C0C0C0"] tr td') // Find the main table.
        .contents()
        .not('font[color="#FFFFFF"], a') // Filter out advertisement and links.
        .map((_, el) => extractText($, el)) // Extract text from each node.
        .get()
        .join('')
        .replace(/\n{2,}/g, '\n') // Remove redundant newlines.
        .trim() // Remove leading and trailing whitespace.
        .split('相关词语', 2)[0] // Remove words after "相关词语".
        .split('基本解释', 2)
        .join('\n 基本解释'); // Add a line break before "基本解释".

      // Save the result to a text file.
      await fs.writeFile(outputPath, body, 'utf-8');

      if (logProgress) {
        console.log('Done.\n');
      }
    }
  };
```

练习 2 的代码和练习 1 大体上是相似的。在最开始，读取文件夹中所有文件的文件名，然后迭代这些文件名，获取 HTML 文件的路径和输出 txt 文件的路径（sourcePath 和 outputPath），接下来的处理和练习 1 类似，主要区别在选取的 HTML 元素不同：

练习 1 直接选取了 body 元素，而在这里，通过`'table[bgcolor="#C0C0C0"] tr td'`选取包含主要内容的表格中的所有 td 元素。

此外，使用`not('font[color="#FFFFFF"], a')`去除了文字颜色为白色的反爬虫元素，和不必要的链接。

最后，还需要对结果稍作修饰。一是删除“相关词语”及之后的不必要文字，二是为“基本解释”前面添加一个换行符。

其余部分均同练习 1.

4. 在`./src/`目录下创建`main.ts`文件，导入`exercise1()`和`exercise2()`，依次运行它们。

```
import exercise1 from './exercise1.js';
import exercise2 from './exercise2.js';

const EX1_SOURCE_PATH = './data/data1/Ng.htm';
const EX1_OUTPUT_PATH = './output/data1/Ng.txt';
const EX1_SHOW_OUTPUT = true;
const EX2_SOURCE_FOLDER = './data/data2/';
const EX2_OUTPUT_FOLDER = './output/data2/';
const EX2_SHOW_OUTPUT = false;

const main = async () => {
  console.log('Exercise 1:');
```

```
  await exercise1(EX1_SOURCE_PATH, EX1_OUTPUT_PATH, EX1_SHOW_OUTPUT);
  if (!EX1_SHOW_OUTPUT) {
    console.log('Done.');
  }

  console.log();

  console.log('Exercise 2:');
  await exercise2(EX2_SOURCE_FOLDER, EX2_OUTPUT_FOLDER, EX2_SHOW_OUTPUT);
  if (!EX2_SHOW_OUTPUT) {
    console.log('Done.');
  }
};

await main();
```

这段代码的逻辑比较简单，就不再解释了。就是首先定义一些相关常量，然后依次运行两个练习的代码而已。

最后，执行主函数`main()`，本实验就完成了。

（二）实验结果

运行`npm run dev`，使用 ts-node 执行`./src/main.ts`。

练习 1 的输出如下：

```
title:
Home Page

body:
Home Page of Professor NG Hwee TouAffiliation
Provost's Chair Professor
Department of Computer Science
School of Computing
National University of Singapore
13 Computing Drive
Singapore 117417
Tel: (65) 6516 8951
Fax: (65) 6779 4580
Email: nght@comp.nus.edu.sg
Home Page: http://www.comp.nus.edu.sg/~nght
Office: AS6 05-16
Senior Faculty Member
NUS Graduate School for Integrative Sciences and Engineering
National University of Singapore
Education
Ph.D. in Computer Science, 1992
University of Texas at Austin, U.S.A.
```

M.S. in Computer Science, 1987
Stanford University, U.S.A.
B.Sc. in Computer Science for Data Management (High Distinction), 1985
University of Toronto, Canada
Bio
Professor Hwee Tou NG is Provost's Chair Professor of Computer Science at the National University of Singapore (NUS) and a Senior Faculty Member at the NUS Graduate School for Integrative Sciences and Engineering. He received a PhD in Computer Science from the University of Texas at Austin, USA. His research focuses on natural language processing and information retrieval. He is a Fellow of the Association for Computational Linguistics (ACL).
He has published papers in premier journals and conferences, including Computational Linguistics, Journal of Artificial Intelligence Research (JAIR), ACM Transactions on Information Systems (TOIS), ACL, NAACL, EMNLP, SIGIR, AAAI, and IJCAI. His papers received the Best Paper Award at EMNLP 2011 and SIGIR 1997. He is the editor-in-chief of Computational Linguistics, an editorial board member of Natural Language Engineering, and a steering committee member of ACL SIGNLL.
He has also served as the editor-in-chief of ACM Transactions on Asian Language Information Processing (TALIP) (May 2007 − May 2013), an editorial board member of Computational Linguistics (Jan 2004 ◆ Dec 2006) and Journal of Artificial Intelligence Research (JAIR) (Sep 2008 − Aug 2011), the book review editor of Computational Linguistics (Aug 2014 ◆ Jul 2018), and an action editor of the Transactions of the Association for Computational Linguistics (TACL) (May 2012 ◆ Jun 2018). He was an elected member of the ACL executive committee (Jan 2008 ◆ Dec 2010) and a former secretary of ACL SIGNLL. He was program co-chair of EMNLP 2008, ACL 2005, and CoNLL 2004 conferences, and has served as area chair of ACL, NAACL, EACL, EMNLP, SIGIR, AAAI, and IJCAI conferences and as session chair and program committee member of many past conferences including ACL, EMNLP, SIGIR, AAAI, and IJCAI.At NUS, he currently serves as Deputy Director of Temasek Defence Systems Institute (TDSI), and he was formerly Program Co-Chair of the Singapore-MIT Alliance (SMA) Computer Science Program and Vice Dean (Research) of the School of Computing.
Research Interests
Natural Language Processing, Information Retrieval

练习 2 由于文件较多，只摘取“阿拉”对应的文本文件在此展示。全部输出结果已打包在代码文件附件中。

阿拉
基本解释：
〈方〉①我。
②我们。
词语分开解释：

阿 ：阿 ā 加在称呼上的词头：阿大。阿爷。阿爹。阿罗汉。阿毛。阿婆。阿弟。阿姊。 笔画数：7； 部首：

拉 ：拉 lā 牵，扯，拽：拉车。拉网。拉下马（喻使某人下台）。 用车载运：拉货。 使延长：拉延。拉伸。

四. 实验总结

1. 本次实验使用了 Node.js 上的 jQuery 精简移植库 cheerio 进行页面信息抽取。相比于比较流行的使用 Python BeautifulSoup4 进行页面信息抽取的方案，JavaScript 中由于原本就为操作 DOM 树而生，使用体验要更贴近原生 DOM 树操作，能够直接使用 CSS 选择器更简洁地选中节点元素，并且在处理网页中内嵌的 JS 脚本时要更加方便。然而对于纯静态网页的抽取，JS 由于生态不如 Python 成熟，客观上存在一些不便的地方。

2. 通过本次实验，了解并实践了对 HTML 网页的信息抽取，构建了搜索系统的数据部分，为后面的实验做了铺垫。