

Universidade do Minho

DEPARTAMENTO DE INFORMÁTICA

Transformador Publico2NetLang TP1 - Exercício 4 - Grupo 7

Jose Filipe Ferreira (A85504) Jose Filipe Ferreira (A83683) Miguel Solino (A86435)

5 de Abril de 2020

${\bf Resumo}$ O objetivo deste projeto é processar um ficheiro HTML para produzir um JSON organizado, utilizando FLex para gerar um parser eficiente utilizando expressões regulares.

Conteúdo

1	Introdução	2
2	Problema	3
3	Solução 3.1 Contextos de Processamento	4 4 5 5
	3.1.2 THREAD 3.1.3 CONTENT 3.1.4 ID 3.1.5 USER 3.1.6 DATE 3.1.7 REPLY	6 6 6 6 6
4	3.2 Arquitetura do Projeto	6 7
5	Conclusão	8

Introdução

Este projeto tem como objetivo o processamento de um ficheiro HTML contendo os comentários de uma notícia de forma a extrair todos os dados relevantes e transformá-los num formato JSON.

Para o processamento deste ficheiro foi escrito um filtro de texto utilizando o gerador FLex e a linguagem de programação C.

Em primeiro lugar iremos analisar o problema, determinar que funcionalidades implementar e identificar os desafios que serão necessários ultrapassar para implementar essas funcionalidades.

Em seguida iremos analisar a nossa solução, a estrutura do *parser*, como foram escolhidos os subcontextos, que expressões regulares foram escolhidas para os construir e a estrutura do projeto onde iremos falar das estruturas de dados utilizadas.

Para concluir, iremos apresentar um pequeno manual de utilização do programa.

Problema

O programa tem que cumprir os seguintes requisitos:

- Converter o ficheiro de input para UTF-8;
- Identificar no ficheiro de input os diversos comentários;
- Separar os diferentes elementos dos comentários;
- Garantir que as respostas a um comentário são guardadas de forma aninhada;
- Contar o número de respostas a cada comentário;

O ficheiro de input apresentou vários problemas que precisa de ser tratados para conseguir cumprir os requisitos propostos.

O primeiro sendo a codificação do ficheiro que está em latin1 e é preciso converter para UTF-8 de forma a que o output do programa não apareça com caracteres ilegíveis.

O segundo foi o facto de alguns dos comentários apresentados conterem mais do que uma linha, invalidando assim o JSON de output se não forem tratados.

Solução

3.1 Contextos de Processamento

Para processar o ficheiro com os comentários foram implementados 7 subcontextos: DEFAULT, THREAD, CONTENT (identificado a amarelo), ID (identificado a vermelho), USER (identificado a azul), DATE (identificado a verde) e REPLY (identificado a roxo).

```
<div class="comment inner">
<span class="avatar comment_
<span class="avatar_pad">
<img alt="" data-interchange="https://static.publicocdn.com/files/homepage/img/avatar_74x74.png, small], [https://static.publicocdn.com/files/homepage/img/avatar_74x74.png, small], [https://static.publicocdn.com/files/homepage/img/avatar_74x74.png]</pre>
</span>

</pan>

# rel="nofollow">Joao Vieira de Sousa 

<p
<span class="comment_reputation comment_reputation-r2" title="Experiente"><i aria-hidden="true" class="i-check"></i></span>
<!--<span class="comment_location">Cruz Quebrada Dafundo

class="dateline comment_dateline" datetime="2019-10-02T22:50:07.08">
<a class="comment_permalink">02.10.2019</a> 22:50</a>
</div>
>
                                                            Pois foi o Rio tirou-lhe a teta e ele agora tenta vingar-se, mas palpita-me que não terá muita sorte
</div>
</div
class="comments_list">
class="comment" data-comment-id="93506ff9-5244-4c37-455e-08d7471e83ae">
<div class="comment_inner">
<div class="comment_meta">
<fiv class="comment_meta">
<fiv class="comment_meta">
</fiv class="comment_inner">
</fiv class="comment_meta">
</fiv:
</tr>
 <span class="avatar comment__avatar">
 <img alt="" data-interchange="https://static.publicocdn.com/files/homepage/img/avatar_74x74.png, small], [https://static.publicocdn.com/files/homepage/img/avatar_74x74.png, small], [https://static.publicocdn.com/files/homepage/img/avatar_74x74.png], [https://static.publicocdn.com/files/homepa
</span>
<h5 class="comment_author">
<a href="/utilizador/perfil/ad91ec31-3d7e-45f5-afd3-01905b95c359" rel="nofollow">Vieira </a></a>
<span class="comment_reputation comment_reputation-r4" title="Moderador"><i aria-hidden="true" class="i-check"></i></span>
<!--<span class="comment_location"></span>
<time class="dateline comment dateline" datetime="2019-10-03T00:10:15.203">
<a class="comment_permalink">03.10.2019 00:10</a>
</time>
<div class="comment__content">
                                                                                          A central de corrupcao do PSD so da' a mama aos apaniguados, Rui Rio nao vai fugir 'a regra ate porque a
</div>
</div>
</ol>
```

Figura 3.1: Comentário e respostas com os contextos identificados

3.1.1 DEFAULT

Toda a zona correspondente ao contexto DEFAULT é ignorada, pois é a zona que contem a informação irrelevante ao programa.

3.1.2 THREAD

No THREAD $\acute{\rm e}$ onde se encontra a informação relevante sobre os comentários, e onde estão contidos todos os subcontextos descritos abaixo.

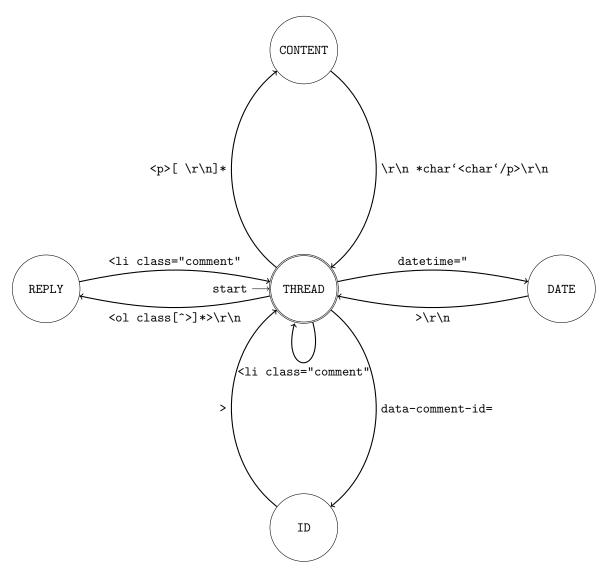


Figura 3.2: Maquina de Estados do Parser

Na Figura 3.2 é descrito como é feita a mudança de estados é efetuada, e que expressões regulares causam estas mudanças.

3.1.3 CONTENT

Neste contexto é onde o conteúdo de cada resposta é tratado e formatado, removendo espaços repetidos e escapando os caracteres n, de forma a assegurar que o output do programa é válido.

3.1.4 ID

Aqui é onde é identificado o ID de cada resposta, ignorando todo o resto presente neste contexto.

3.1.5 USER

À semelhança do contexto acima descrito, o USER é onde são identificados os detalhes do utilizador criador do respetivo comentário.

3.1.6 DATE

No contexto DATE é feito o processamento tanto da data do comentário como a timestamp do mesmo.

3.1.7 REPLY

No contexto REPLY estamos dentro das respostas de um comentário. Aqui, como o processamento é recursivo, voltamos a ter os contextos acima descritos, pois respostas são comentários normais.

3.2 Arquitetura do Projeto

Graças à simplicidade do projeto, foi-nos possível resolver o problema num só módulo não comprometendo a legibilidade da solução. Neste módulo é tratado o input, selecionada a informação relevante presente e formatada para ir de encontro ao formato pretendido.

Quanto a estruturas de dados, apenas mantemos em memória uma stack de contadores, para ser possível armazenar o número de respostas de um comentário, respeitando a recursividade do processamento no campo das respostas. Para a restante informação não nos foi necessário armazenar nada em memória, dando flush desta mal é processada.

Manual de Utilização

Este programa é de utilização simples, recebendo como argumento o ficheiro HTML a ser processado, podendo ser chamado desta forma: ./html2json path_to_input_file.

Depois de processado, o JSON resultante irá ser retornado pelo stdout. Como o JSON não é formatado de uma forma muito legível o programa pode ser encadeado com ferramentas processamento de JSON como, por exemplo, o jq, para facilitar a leitura do mesmo. Isto pode ser feito com o comando exemplo ./html2json path_to_input_file | jq. O resultado da formatação deste será mais uma vez retornado pelo stdout.

Conclusão

Para concluir, fazemos um balanço bastante positivo do projeto, conseguindo atingir todos os requisitos definidos. A utilização de expressões regulares para tratamento de strings mostrou ser algo extremamente poderoso, permitindo simplificar tarefas que de outra forma seriam muito mais trabalhosas e demoradas.

Como trabalho futuro, gostaríamos de trabalhar mais a formatação do output para facilitar a leitura do mesmo.

Apêndice A

FLex

```
CDEFAULT,THREAD>\<li\ class=\"comment\"\ { BEGIN THREAD; printf(",{"); rsc[cursor-1]++; } <REPLY>\<li\ class=\"comment\"\ { BEGIN THREAD; rsc[cursor-1]++; }
                                                       { BEGIN THREAD; printf("{"); rsc[cursor-1]++; } 
{ char *bool = rsc[cursor] ? "true" : "false"; 
   char *reply = rsc[cursor] ? " " : "\"replies\": [], "; \
<*>\<li\ class=\"comment\"\</pre>
<THREAD>\<\/ol\>
                                                          printf \
                                                            reply, bool, rsc[cursor]); \
rsc[cursor--] = 0; }
{ char *bool = rsc[cursor] ? "true" : "false"; \
char *reply = rsc[cursor] ? " " : "\"replies\": [], "; \
BEGIN DEFAULT; }
                                                       BEGIN ID;
printf("\"id\": %s,", yytext);
BEGIN THREAD;
<THREAD > data - comment - id=
<ID>[^\>]+
{ BEGIN CONTENT; printf("\"commentText\": \""); }
<CONTENT > [^\"\r\n]
<CONTENT > \"
                                                       ECHO;
                                                       printf("\'");
<CONTENT>\r
                                                       {;}
                                                       { BEGIN THREAD; printf("\""); } { printf("\\n"); }
<CONTENT>\r\n\ *\<\/p\>\r\n
<CONTENT>\n
\THREAD > \ \ href[^\>] + \ \
                                                       BEGIN USER;
                                                       printf("\"user\": \"%s\",", yytext);
<USER > [^\<] +/\
<USER>\<\/a\>\r\n
                                                       BEGIN THREAD;
<THREAD > datetime = \"
                                                       printf("\"date\": \"%s\",", yytext);
printf("\"timestamp\": \"%s\",", yytext);
<DATE > [0-9\-]+[^T]
<DATE > [0-9:.]+[^\"]
\begin{tabular}{ll} $<$DATE>$\r\n \end{tabular}
                                                       BEGIN THREAD;
                                                       { BEGIN REPLY; printf(",\"replies\": [{"); cursor++; }
{ char *bool = rsc[cursor] ? "true" : "false"; \
   char *reply = rsc[cursor] ? " " : "\"replies\": [], "; \
\label{thread} $$ \THREAD > \class[^\>] * \r\n $$
<*>\<\/li\>\r\n/\<li
                                                            (",%s\"hasReplies\": %s, \"numberOfReplies\": %d, \"likes\": 0}", \
                                                                    reply, bool, rsc[cursor]); }
<*>.|\n
                                                       {;}
```