Techniki Kompilacji - Dokumentacja końcowa

Oczyszczanie pliku HTML

Damian Bułak

8 czerwca 2017

1 Wstęp

Celem projektu jest stworzenie programu w języku C++ umożliwiającego oczyszczenie kodu HTML ze wszystkich atrybutów, skryptów, styli w taki sposób, aby kod wynikowy zawierał jedynie strukturę HTML bez elementów formatujących oraz udostępniającej strukturę zagnieżdżeń dokumentu.

2 Funkcjonalność

Program realizuje zadanie przetwarzania wejściowego pliku HTML w celu usunięcia żądanych elementów. Użytkownik ma możliwość wyszczególnienia jakie elementy mają zostać usunięcie spośród następujących:

1. atrybuty
2. style zawarte w tagach `<style>...</style>`
3. skrypty JS zawarte w tagach `<script>...</script>`
4. komentarze `<!- ... ->`
5. doctype `<!DOCTYPE ...>`

Program ma także udostępniać strukturę zagnieżdżeń, która przy każdym uruchomieniu programu zostanie wydrukowana na wyjście standardowe.
Wyjściowy plik HTML zostanie natomiast zapisany pod żądaną przez użytkownika ścieżką.

Aplikacja składa się z trzech modułów:

1. Leksera zczytującego z pliku wejściowego tokeny
2. Parsera analizującego otrzymywane od leksera tokeny i tworząc strukturę dokumentu HTML
3. Generatora kodu przetwarzającego wyprodukowane przez parser struktury i generującego wyjściowy plik HTML w zależności od podanej przez użytkownika konfiguracji tak, że żądane elementy do usunięcia nie znajdą się w pliku wyjściowym

2.1 Lekser

Moduł leksera na wejściu otrzymuje ścieżkę do pliku, który wczytuje strumieniowo i po kolei produkuje na wyjściu tokeny.
Lekser udostępnia następujące funkcje:

- `Token getNextToken()` - pobiera następny token zgodnie ze zdefiniowanymi dla HTML tokenami
- `Token getText()` - pobiera tekst pomiędzy tagami HTML tak długo aż napotka tag otwarcie tagu HTML `<`
- `Token getScript()` - pobiera skrypt JS (JavaScript nie jest parsowany przez aplikację) - wczytuje tak długo aż napotka tag zamykający `</script>`

2.1.1 Tokeny

Lista akceptowalnych tokenów używana przez analizator leksykalny:

- OpenDoctype - otwarcie doctype'u

  `<!DOCTYPE`

- Open - otwarcie taga otwierającego HTML

  `<`

- OpenEnd - otwarcie taga zamykającego

  `</`

- Close - zamknięcie taga

  `>`

- Name - nazwa taga, np. `html` dla `<html...`

  `^[a-zA-Z]([a-zA-Z1-6:-])*`

- Equals - znak równosci (przypisanie wartości atrybutu do klucza)

  `=`

- Value - wartość atrybutu - dowolny ciąg pomiędzy pojedynczym lub podwójnym cudzysłowem
- Text - zwracany po getText lub getScript
- CloseEmpty - zamknięcie taga *inline*, np. `<br/>`

- Comment - dowolny ciąg otoczony znakiem początku i końca komentarza HTML: `<!- ..... ->`
- EndOfFile - koniec pliku HTML
- Empty - zwracany przez getText, gdy wystepują pomiędzy tagami tylko białe znaki
- Undefined - niezdefiniowany nieznany token informujący o niepoprawnym leksykalnie pliku HTML

2.2 Parser

Moduł parsera zależy od leksera i na wejściu otrzymuje (na żądane) kolejne tokeny. Parser udostępnia funkcję `HtmlDocument parse()`, która zwraca strukturę dokumentu HTML w postaci obiektu `HtmlDocument` Zbudowany w aplikacji parser jest parserem RD.

2.3 Gramatyka

```
document = doctype , { [comment] } , element
doctype = "<!DOCTYPE" , name , ">"
comment = "<!--" , text , "-->"
element = tagOpener , ("/>" | ">" | {element}, endingTag | string)
| comment
tagOpener = "<" , name , { [attribute] }
attribute = name , ["=" , value]
value := " \' " , string , " \' "
        | " \" " , string , " \" "
name = a..z | A..Z , {a..z | A..Z | 1..6 | : | - | < } , *
string = * (uwaga niżej)
```

Postać `string` zależy od parsera - gdy wywoła on funcję `getText()` wczytywany jest on aż do znaku nowego taga HTML (`<`), gdy `getScript()` aż do taga zamykającego JavaScript `</script>`, natomiast w przypadku value wszystko pomiędzy cudzysłowami.

2.4 Generator kodu wyjściowego

Generator kodu wyjściowego korzystając z drzewa rozbioru wygenerowanego na wyjściu analizotra składniowego (`HtmlDocument`), produkuje kod HTML oczyszczony z atrybutów, styli, skryptów, komentarzy, doctype'a zgodnie z zadaną konfiguracja wprowadzoną przy uruchamianiu przez użytkownika. Do pliku wyjściowego wysyłane są strumieniową kolejne elementy struktury `HtmlDocument` tylko wtedy jeśli użytkownik nie podał opcji ich oczyszczenia.

3 Wykorzystywane struktury

Aplikacja wykorzystuje do działania nastepujące struktury:

3.1 Token

Informacje o typie tokenu, jego pozycji i wartości:

```
enum TokenType {
    OpenDoctype,
    Open,
    OpenEnd,
    Close,
    Name,
    Equals,
    Value,
    Text,
    CloseEmpty,
    Comment,
    EndOfFile,
    Empty,
    Undefined
};
```

```
class Token {
public:
    Token(TokenType type = Empty, std::string value = "", Position position =
        ↪ Position(0, 0, 0));
    Token(TokenType type, std::string value, Position position, char quoteType)
        ↪ ;
    TokenType getType();
    std::string getValue();
    Position getPosition();
};
```

3.2 Node

Informacja o sparsowanym elemencie HTML - jego typ, atrybuty, dzieci i poziom zagnieżdżenia:

```
enum NodeType {
    SingleTagNode,
    DoubleTagNode,
    InlineTagNode,
    TextNode,
    CommentNode
};
```

```
class Node {
public:
    Node(const std::string &name, NodeType type, const std::vector<Node *> &
        ↪ children,
         const std::vector<Attribute> &attributes, int level);

    std::string getName();
    NodeType getType();
    std::vector<Node*> getChildren();
    std::vector<Attribute> getAttributes();
    int getLevel();
};
```

3.3 Konfiguracja

Wczytane przy uruchomieniu programu opcje wykonania programu, w tym plik wejściowy, ścieżka do pliku wyjściowego i opcje oczyszczania:

```
enum CleanerOption {
    ATTRIBUTES_OPTION,
    STYLES_OPTION,
    SCRIPTS_OPTION,
    DOCTYPE_OPTION,
    COMMENTS_OPTION
};
```

```
class Configuration {
public:
    void addCleaningOption(CleanerOption cleanerOption);
    void setInputFile(const std::string &inputFile);
    void setOutputFile(const std::string &outputFile);
    std::set<CleanerOption> getOptions();
    std::string& getInputFilePath();
    std::string& getOutputFilePath();
};
```

4 Sposób uruchomienia

Program uruchamiany jest z linii komend z podaniem pliku wejściowego, opcjonalnie wyjściowego oraz opcjonalnymi opcjami: `html_cleaner [OPTIONS] input_file [output_file]` Jeśli plik wyjściowy nie zostanie podany wygenerowany zostanie plik o nazwie takiej jak input file bez rozszerzenia z dodaną końcówką: `.out.html`

Dostępne opcje definiują które elementy mają zostać usunięcie w wygenerowanym pliku wynikowym:

- `-attributes` lub `-a` - usuwanie atrybutów
- `-styles` lub `-s` - usuwanie styli CSS
- `-scripts` lub `-S` - usuwanie skryptów JS
- `-doctype` lub `-d` - usuwanie doctype'a
- `-comments` lub `-c` - usuwanie komentarzy
- `-help` lub `-h` - wyświetla informację o tym, jak uruchomić program

W celu wczytania argumentów wejściowych uruchomienia programu użyta została biblioteka `boost`

5 Wykonane testy

W celu sprawdzenia poprawności działania algorytmu wykonanych zostało kilka testów. Efekt jednego z nich znajduje się poniżej. Ponadto oczyszczaniu poddane zostały popularne strony serwisów informacyjnych: **www.onet.pl** oraz **www.interia.pl**. Dla obydwu stron program wyprodukował poprawny plik wynikowy bez błędów parsowania.

Użyte opcje: -asS

Plik wejściowy:

```
<!DOCTYPE>
<!--root comment also work-->
<html>
  <head>
    <!--comment test-->
    <script>
      var x = 0;
      for (var i = 0; i < x; i++) {
          alert('Nothin');
      }
    </script>
    <style>
      p {
        color: black;
      }
    </style>
    <meta charset="UTF-8">
    <title>Sample "Hello, World" Application</title>
  </head>
  <body bgcolor="white">
    <table border="0" cellpadding="10">
      <tr class="classTest" id='idTest'>
        <td>
          <img src="images/springsource.png" />
        </td>
        <td>
          <h1>Sample "Hello, World" Application</h1>
        </td>
      </tr>
    </table>
    <br>
    <p>This is the home page for the HelloWorld Web application.</p>
    <p>To prove that they work, you can execute either of the following links:</p>
    <ul>
      <li>To a <a href="hello.jsp">JSP page</a>.</li>
      <li>To a <a href="hello">servlet</a>.</li>
    </ul>
  </body>
</html>
```

Plik wynikowy:

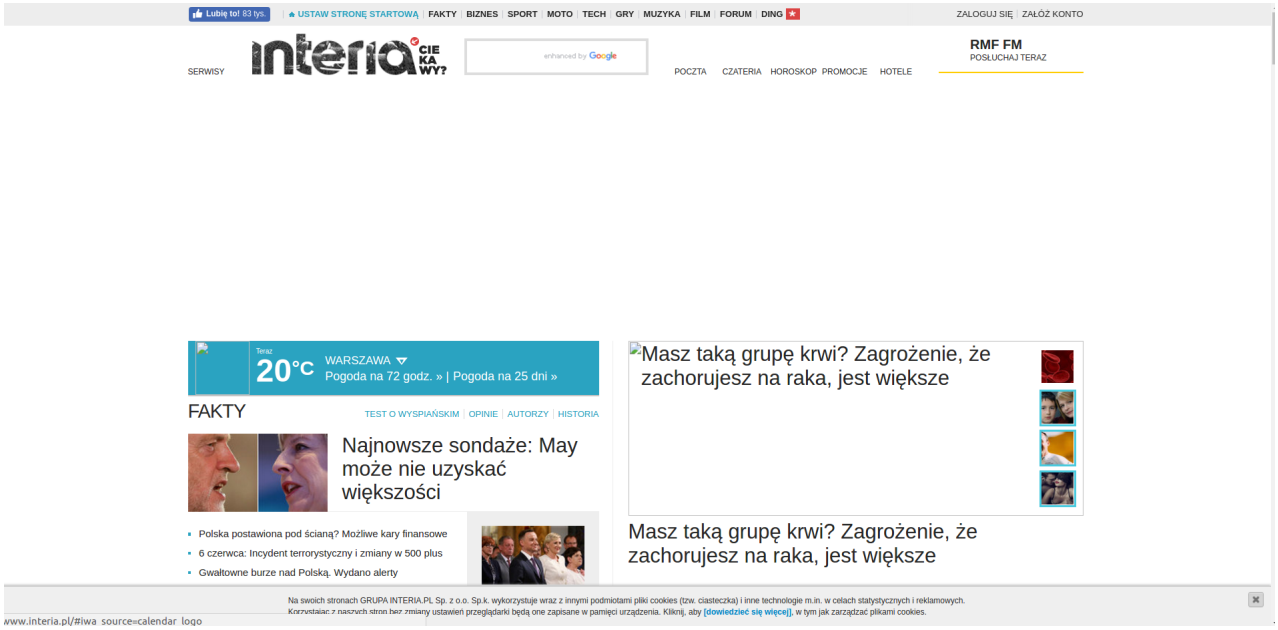
```
<!DOCTYPE>
<!--root comment also work-->
<html>
  <head>
    <!--comment test-->
    <meta>
    <title>
      Sample "Hello, World" Application
    </title>
  </head>
  <body>
    <table>
      <tr>
        <td>
          <img/>
        </td>
        <td>
          <h1>
            Sample "Hello, World" Application
          </h1>
        </td>
      </tr>
    </table>
    <br>
    <p>
      This is the home page for the HelloWorld Web application.
    </p>
    <p>
      To prove that they work, you can execute either of the following links:
    </p>
    <ul>
      <li>
        To a
        <a>
          JSP page
        </a>
        .
      </li>
      <li>
        To a
        <a>
          servlet
        </a>
        .
      </li>
    </ul>
  </body>
</html>
```

Struktura zagnieżdżeń dla pliku wejściowego (bez żadnych opcji)

```
DOM structure:
html -> head -> script
html -> head -> style
html -> head -> meta
html -> head -> title
html -> body -> table -> tr -> td -> img
html -> body -> table -> tr -> td -> h1
html -> body -> br
html -> body -> p
html -> body -> p
html -> body -> ul -> li -> a
html -> body -> ul -> li -> a
```

6 Efekt oczyszczania serwisu Interia

Użyte opcje: -saSc



Rysunek 1: Przed oczyszczaniem

Rysunek 2: Po oczyszczaniu z opcjami -saSc

7 Podsumowanie

Główną trudność projektu stanowi poprawna implementacja parsera. Pomimo, że ten zaimplementowany w dołączonym kodzie źródłowym z pewnością daleko odbiega od tych, które wykorzystują nasze przeglądarki internetowe, można traktować go jako przybliżenie działania używanego w przeglądarkach parsera HTML. O jego dużej skuteczności świadczą wykonane testy na realnych istniejących w sieci stronach WWW takich jak Interia i Onet, gdzie bogactwo różnych elementów HTML jest stosunkowo duże.