

Web-Informationssysteme, WS 2009/10

Seminar-Übung 5: XML, Anfragen auf Text- und HTML Dokumenten

Besprechung am Mi 09.12.2009

— SÜ-5.1 —

Was Ihr wissen müßt über “XML (Basics)”

Wiederholung

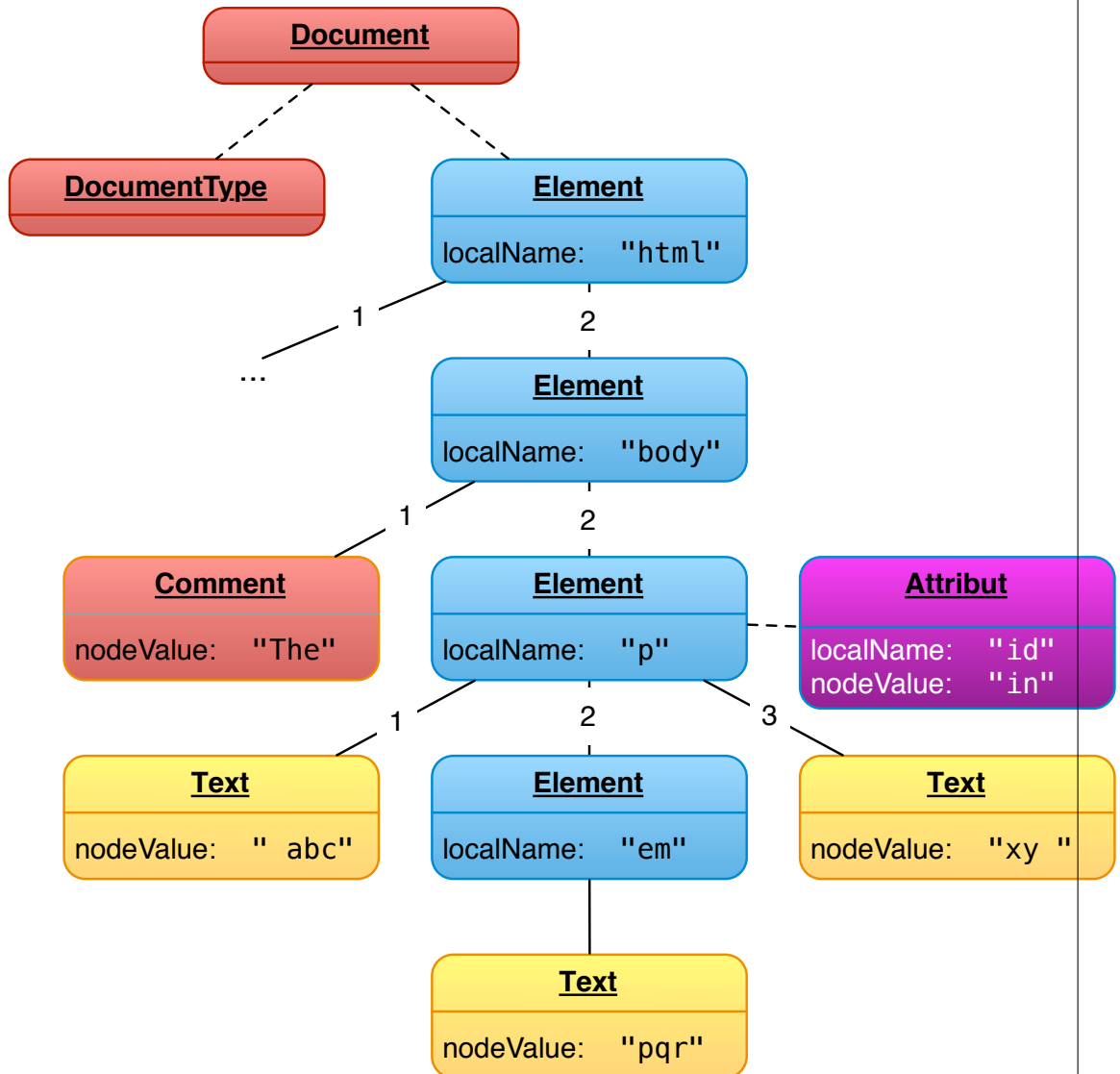
- Beschreiben Sie die **Grobstruktur** eines XML Dokuments! Welche Art von XML Information darf auf dem obersten Level in einem XML Dokument vorkommen? In welcher Reihenfolge?
- Charakterisieren Sie die **Unterschiede zwischen Elementen und Attributen** in XML?
- Was bedeutet **“Wohlgeformtheit”** und welche Bedingungen muss ein XML Dokument erfüllen, damit es “wohlgeformt” ist.
- Was ist der Unterschied zwischen einem **“tag”** und einem “element”?
- Geben Sie für das folgende XML Dokument eine **Baumdarstellung** an, die ausreichend detailliert ist, um genau dieses XML Dokument wieder aus der Baumdarstellung zu gewinnen (Tipp: Sie können sich am DOM oder am *XML Information Set* orientieren.)

```
1 <!DOCTYPE html ...>
  <html>
3   ...
  <body>
5   <!-- The -->
  <p id='in'>
7     abc<em>pqr</em>xy
  </p>
9   ...
  </body>
11 </html>
```

```

<!DOCTYPE html ...>
<html>
  ...
  <body>
    <p>
      abc<em>pqr</em>xy
    </p>
    ...
  </body>
</html>

```



— SÜ-5.2 —

Pipe-Separated Values vs. HTML

Programmierung

Immer noch werden tabellarisch oder anders strukturierte Daten oft einfach in delinierten Text-Dateien oder `<pre>` Blöcken in Web-Seiten abgelegt. Solche Dateien werden auch, je nach Separator, *comma / semi-colon / pipe separated value* (kurz CSV, SSV, PSV) Dateien genannt. Sie enthalten pro Zeile einen Datensatz dessen Attribute durch Separatoren (oder *delimiter*), voneinander getrennt abgespeichert werden:

EXAMPLE FLATFILE:

```
# Nachname, Vorname <E-Mail>      | Matrikeln | StR  | Vorl.  | Note
#-----+-----+-----+-----+-----
Markmann, Jana <mm@gmail.net>      | 12399201  | Inf  | IN2003 | 1.3
Mut, Hans <mut@uni-tuebingen.de>    | 20923898  | Inf  | IN2003 | 2.3
Ahr, Benjamin <ben86@gmx.net>      | 23894333  | Math | IN2041 | 1.7
Markmann, Jana <mm@gmail.net>      | 12399201  | Inf  | IN2041 | 1.0
Koller, Matthias <zorro@web.de>     | 12232112  | Inf  | IN2062 | 2.0
Mut, Hans <mut@uni-tuebingen.de>    | 20923898  | Inf  | IN2062 | 2.3
```

Die darin vorhandenen Informationen werden durch | voneinander getrennt. Ein # am Anfang einer Zeile leitet einen Kommentar ein, der für den Rest der Zeile gilt. Die erste Kommentarzeile wird als Header der Tabelle betrachtet.

In einer Reihe von Programmieraufgaben auf diesem und folgenden Themenblättern werden wir untersuchen, wie einfach die Verarbeitung solcher tabellarischer Daten in unterschiedlichen Darstellungen (PSV, HTML, XML, etc.) und mit unterschiedlichen Techniken (DOM, SAX, XPath, etc.) ist.

- Konvertieren Sie dieses Flatfile in ein HTML Dokument, in dem die Informationen mit Hilfe des HTML elements table repräsentiert sind.

Sowohl auf dem ursprünglichen Flatfile als auch auf dem erstellten HTML Dokument sollen Sie **die folgenden Recherche-Aufgaben mit einem Werkzeug Ihrer Wahl implementieren.**

1. Geben Sie den Vor- und Nachname, sowie Matrikelnummer der Studenten aus, die in einer Vorlesung eine Note besser als 1.7 erzielt haben.

Auf obigen Daten sollte beispielsweise folgende Ausgabe produziert werden:

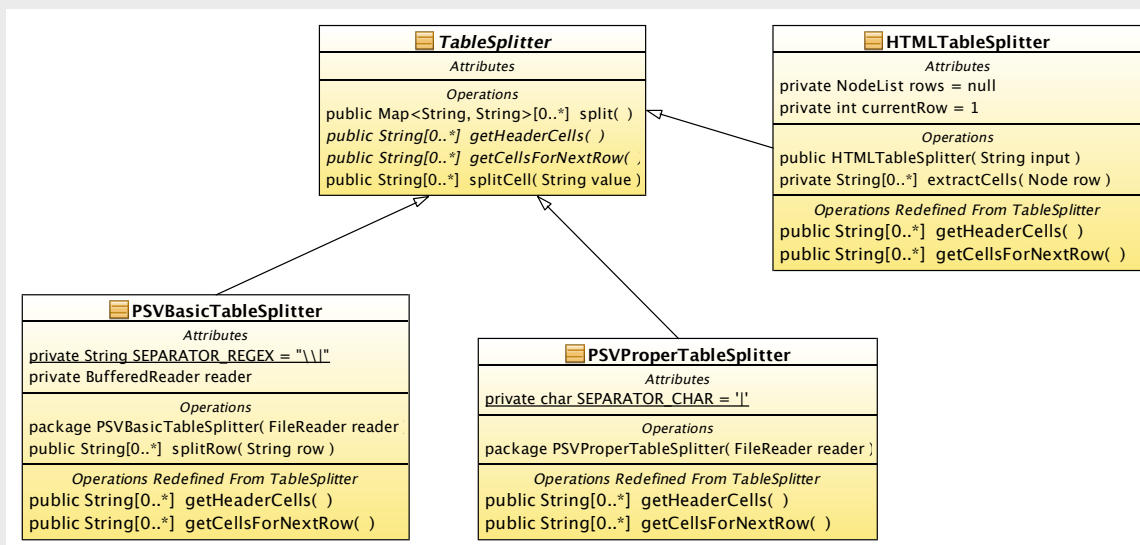
```
1 Basic PSV splitter with input emails.psv!
  Jana Markmann, MNr: 12399201
3 Benjamin Ahr, MNr: 23894333
  Jana Markmann, MNr: 12399201
```

2. Geben Sie die E-Mail Adressen aller Mathematikstudenten aus, deren Matrikelnummer mit 238 beginnt.
3. Überprüfen Sie, ob der Student Hans Mut in allen Vorlesungen, die er besucht hat, eine Note besser als 3.0 erzielt hat.
4. Berechnen Sie den Notendurchschnitt des Studenten Hans Mut.
5. Berechnen Sie den Notendurchschnitt aller Studenten.

Diskutieren Sie die Probleme, welche Sie dabei hatten.

Tip: Für die Verarbeitung des Flatfile bieten sich *reguläre Ausdrücke* an. Falls Sie Java verwenden wollen, könnte sich ein Blick auf `java.lang.String#split` lohnen. Für den HTML Fall werden Sie sich mit regulären Ausdrücken schon schwerer tun. Am einfachsten ist die Verwendung eines HTML oder XML Parsers (siehe z.B. `javax.xml.parsers` für Java).

Hinweis: Sie finden schon ein Gerüst von .java-Dateien in den Anlagen. Das folgende UML-Modell gibt einen Überblick über den Lösungsvorschlag:



Wie immer finden sich einige mit @TODO markierte Stellen, an denen noch Code ergänzt werden muss.

Command-Line Driver:

```

import java.io.IOException;
import java.io.PrintStream;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

/**
 * Command-line driver for solutions of topic sheet 03.
 *
 * @author Tim Furch
 */
public class t04CLIDriver {

    /**
     * Contains the code for the five problems. solvers[i] solves the (i+1)-th problem.
     * Each member of solvers is an anonymous class derived from {@link Solver}.
     */
    private static final Solver[] solvers = {

```

```

new Solver() {
20     @Override
    protected void processRow(Map<String, String> row, PrintStream out) {
22         if(Double.parseDouble(row.get("Note")) <= 1.7) {
            out.printf("%s %s, MNr: %s\n", row.get("Vorname"), row.get("Nachname"),
24                 row.get("Matrikeln"));
        }
    }
},
new Solver() {
28     @Override
    protected void processRow(Map<String, String> row, PrintStream out) {
30         if(row.get("Matrikeln").startsWith("238")) {
            out.println(row.get("E-Mail"));
32         }
    }
},
new Solver() {
36     private boolean busted = false;
    @Override
38     protected void processRow(Map<String, String> row, PrintStream out) {
        if(row.get("Nachname").equals("Mut") && row.get("Vorname").equals("Hans") &&
40         Double.parseDouble(row.get("Note")) >= 3.0 ) {
            this.busted = true;
42         }
    }

    @Override
44     protected void finish(PrintStream out) {
        out.printf("Hans Mut hat %san allen Vorlesung mit besser 3.0 teilgenommen.\n",
46             (busted ? "nicht " : ""));
    }
},
new Solver() {
50     private Double summe = 0.0;
    private int count = 0;
    @Override
52     protected void processRow(Map<String, String> row, PrintStream out) {
        if(row.get("Nachname").equals("Mut") && row.get("Vorname").equals("Hans") ) {
54             this.summe += Double.parseDouble(row.get("Note"));
            this.count++;
56         }
    }

    @Override
60     protected void finish(PrintStream out) {
        out.printf("Hans Mut hat einen Notendurchschnitt von %1.1f erreicht.\n",
62             this.summe / this.count);
    }
},
new Solver() {
66     private Double summe = 0.0;
    private int count = 0;
    @Override
68     protected void processRow(Map<String, String> row, PrintStream out) {
70         this.summe += Double.parseDouble(row.get("Note"));

```

```

72         this.count++;
73     }
74
75     @Override
76     protected void finish(PrintStream out) {
77         out.printf("Die Studenten haben einen Notendurchschnitt von %1.1f erreicht.\n",
78             this.summe / this.count);
79     }
80 }
81
82 };
83
84 public static void main(String[] args) throws IOException {
85     // Read command line -----
86     FileReader input = new FileReader(args[0]);
87     int aufgabe = Integer.parseInt(args[1]);
88     List<Map<String,String>> cells = null;
89
90     if(args.length < 3 || args[2].equals("--basic")){
91         System.out.printf("Basic PSV splitter with input %s!\n", args[0]);
92         cells = new PSVBasicTableSplitter(input).split();
93     } else if(args[2].equals("--proper")) {
94         System.out.printf("Proper PSV splitter with input %s!\n", args[0]);
95         cells = new PSVProperTableSplitter(input).split();
96     } else if(args[2].equals("--html")) {
97         System.out.printf("HTML table splitter with input %s!\n", args[0]);
98         cells = new HTMLTableSplitter(args[0]).split();
99     }
100
101     //      System.out.println(cells);
102
103     solvers[aufgabe].solve(cells, System.out);
104 }
105
106 }

```

Solver:

```

1 package t05;
2
3 import java.io.PrintStream;
4 import java.util.List;
5 import java.util.Map;
6
7 /**
8  *
9  * @author timfu
10 */
11 public abstract class Solver {
12
13     /**
14      * Solves the problem on the given list of rows. Outputs the result to
15      * the given {@link PrintStream}.
16      */
17     public void solve(List<Map<String, String>> rows, PrintStream out) {

```

```

19     for(Map<String, String> row : rows) {
20         this.processRow(row, out);
21     }
22     this.finish(out);
23 }
24
25 /**
26  * Processes a single row.
27  */
28 protected abstract void processRow(Map<String, String> row, PrintStream out);
29
30 /**
31  * Outputs aggregated values.
32  */
33 protected void finish(PrintStream out) { }
34 }

```

TableSplitter:

```

1 import java.io.IOException;
2 import java.util.HashMap;
3 import java.util.Iterator;
4 import java.util.LinkedList;
5 import java.util.List;
6 import java.util.Map;
7
8 /**
9  * Abstract class providing a common interface (and some common functions)
10  * for all table splitters, regardless of the data format they operate on.
11  * @author Tim Furche
12  */
13 public abstract class TableSplitter {
14
15     /**
16      * Splits all rows in the file of the given {@link FileReader}.
17      */
18     public List<Map<String, String>> split() throws IOException {
19         List<String> headers = new LinkedList<String>();
20         List<Map<String, String>> result = new LinkedList<Map<String, String>>();
21
22         // Process the header row
23         String[] cells = this.getHeaderCells();
24         for (String cell : cells) {
25             String[] subCells = this.splitCell(cell);
26             for (String subCell : subCells) {
27                 if (subCell.trim().length() != 0)
28                     headers.add(subCell.trim());
29             }
30         }
31
32         // Process all remaining rows
33         while ((cells = getCellsForNextRow()) != null) {
34             Map<String, String> rowMap = new HashMap<String, String>();
35             Iterator<String> currHeaders = headers.iterator();
36             for (String cell : cells) {
37                 String[] subCells = this.splitCell(cell);
38                 for (String subCell : subCells ) {

```

```

39         if(!currHeaders.hasNext()) break;
40         if (subCell.trim().length() != 0)
41             rowMap.put(currHeaders.next(), subCell.trim());
42     }
43 }
44 result.add(rowMap);
45 }
46 return result;
47 }

48 /**
49  * Returns an array of the header cells.
50  */
51 public abstract String[] getHeaderCells() throws IOException;

52 /**
53  * Returns an array of the cells of the next row or <code>null</code> if
54  * there is no more row.
55  */
56 public abstract String[] getCellsForNextRow() throws IOException;

57 /**
58  * Splits a single cell (in form of the value parameter). The cell is splitted
59  * at each ",", "<"", ">".
60  */
61 public String[] splitCell(String value) {
62     String[] values = value.split("(,|<|>");
63     return values;
64 }
65 }
66 }
67 }
68 }
69 }

```

PSVBasicTableSplitter:

```

2 package t05;

3
4 import java.io.BufferedReader;
5 import java.io.FileReader;
6 import java.io.IOException;

7 /**
8  * Provides frame for splitting a pipe-("|")-separated-values file.
9  * @author Tim Furch
10  */
11
12 public class PSVBasicTableSplitter extends TableSplitter {

13     private static final String SEPARATOR_REGEX = "\\|";
14     private final BufferedReader reader;

15     PSVBasicTableSplitter(FileReader reader) {
16         this.reader = new BufferedReader(reader);
17     }

18     /**
19      * Splits a row using {@link String#split(java.lang.String)}.
20      */
21 }

```



```

24     public String[] splitRow(String row) {
25         return row.split(SEPARATOR_REGEX);
26     }

28     /**
29      * Here we assume that the header row is always the first row and always starts
30      * with "#".
31      */
32     @Override
33     public String[] getHeaderCells() throws IOException {
34         String row = this.reader.readLine().substring(1).trim();
35         return this.splitRow(row);
36     }

38     @Override
39     public String[] getCellsForNextRow() throws IOException {
40         String row = null;
41         while ((row = this.reader.readLine()) != null && row.trim().startsWith("#")) { }
42         return row == null? null : this.splitRow(row);
43     }
44
46 }

```

HTMLTableSplitter:

```

1     package t05;

3

4     import java.io.FileReader;
5     import java.io.IOException;
6     import java.util.LinkedList;
7     import java.util.List;
8     import javax.xml.parsers.DocumentBuilderFactory;
9     import org.w3c.dom.Document;
10    import org.w3c.dom.Element;
11    import org.w3c.dom.Node;
12    import org.w3c.dom.NodeList;

13
14    /**
15     *
16     * @author Tim Furche
17     */
18    class HTMLTableSplitter extends TableSplitter {

19
20        private NodeList rows = null;

21
22        private int currentRow = 1;

23
24        public HTMLTableSplitter(String input) {
25            try {
26                this.rows = DocumentBuilderFactory.newInstance().newDocumentBuilder().
27                    parse(input).getElementsByTagName("tr");
28            } catch (Exception e) {
29                e.printStackTrace();
30                System.exit(1);
31            }

```

```
    }  
33  
    /**  
35     * Here we assume that the header row is always the first row  
    */  
37    @Override  
    public String[] getHeaderCells() throws IOException {  
39        return this.extractCells(this.rows.item(0));  
    }  
41  
    @Override  
43    public String[] getCellsForNextRow() throws IOException {  
        if (currentRow >= rows.getLength()) return null;  
45        return this.extractCells(this.rows.item(currentRow++));  
    }  
47  
    /**  
49     * Extracts the String values of all cells in a row.  
    */  
51    private String[] extractCells(Node row) {  
        NodeList tds = row.getChildNodes();  
53        // We do not know whether there are non-td child nodes  
        List<String> cells = new LinkedList<String>();  
55        for (int j = 0; j < tds.getLength(); j++) {  
            Node td = tds.item(j);  
57            if (td.getNodeType() == Node.ELEMENT_NODE) {  
                if (((Element) td).getTagName().equals("td")) {  
59                    cells.add(td.getChildNodes().item(0).getNodeValue());  
                }  
61            }  
        }  
63        return (String[]) cells.toArray(new String[cells.size()]);  
    }  
65  
}
```