FH Aargau: Vorlesung XML Wintersemester 2004/2005

Test XSLT

MUSTERLÖSUNG (Lösungen in grün & underline)

8.2.2005, 8^{30} - 9^{30}

Bitte schreiben Sie *sorgfältig* und *gut lesbar*! Danke!
Für diesen Test dürfen Sie *keinerlei Unterlagen* verwenden, insbesondere auch keine elektronischen Hilfsmittel wie Laptop oder Organizer!

1. XSLT kann nur über Node Sets iterieren (mit der `for-each` Anweisung), so dass die in vielen anderen Programmiersprachen üblichen Iterationen über Werte-Bereiche von Variablen (im Stil von `FOR I=1 TO 10`) nicht möglich sind. Nehmen Sie an, Sie sollen jedoch trotzdem ein Named Template (also ein Unterprogramm) schreiben, das als Parameter eine Zahl akzeptiert und so viele Spaces ausgibt, wie als Zahl angegeben wurden. Wie gehen Sie das Problem an? Sie müssen keinen Code schreiben, es geht um das Konzept!
   Da es in XSLT keine Iterationen über Werte-Bereiche von Variablen gibt, wird das Problem am besten rekursiv programmiert. Das Named Template gibt bei einem Zahlenwert grösser als Null also ein Space aus und ruft danach sich selber mit dem um eins reduzierten Zahlenwert auf.
2. Beschreiben Sie den Unterschied zwischen den Variablen `A`, `B` und `C`.
   ```
   <xsl:variable name="A" select="wert"/>
   <xsl:variable name="B" select="'wert'"/>
   <xsl:variable name="C">wert</xsl:variable>
   ```
   `A` ist höchstwahrscheinlich ein leeres Node Set. Genaugenommen ist der Wert von `A` die Menge der Kinderelemente mit dem Namen `wert` des momentanen Kontextes, aber in den meisten Fällen wird es diese nicht geben und von daher ist `A` ein leeres Nodeset. `B` ist der String `wert`, durch die Apostrophe wird der Wert der Variable als String markiert. `C` ist ein Baum, der Wert der Variable ist der Root Node dieses Baumes, an diesem Root Node hängt als einziges Kind ein Textknoten mit dem Inhalt `wert`. `C` ist also etwas anderes als `B`!
3. Beschreiben Sie die Funktion der einzelnen Attribute in der folgenden XSLT `key` Deklaration und beschreiben Sie eine mögliche Anwendung des `key`, ausgehend von den in `match` und `use` verwendeten Namen.
   ```
   <xsl:key name="wohnort" match="person" use="adresse/zip"/>
   ```
   Das `name` Attribut gibt dem `key` den Namen, mit dem er später in der XPath `key()` Funktion referenziert werden kann. Das `match` Attribut gibt an, dass alle `person` Elemente im Dokument in den `key` aufgenommen werden sollen. Das `use` Attribut gibt an, dass der Wert der in den `key` aufgenommenen Knoten (die `person` Elemente) das `zip` Kind des `adresse` Kindes sein soll, vermutlich also die in einer Adresse angegebene Postleitzahl.

Eine mögliche Anwendung dieses `key` ist es, alle in einem bestimmten Bereich wohnenden Personen zu finden, in dem die Postleitzahl dieses Bereiches in der `key()` Funktion angegeben wird, und als Resultat des Aufrufes dann alle die Personen erwartet werden können, bei denen das `zip` Element im `adresse` Element den angegebenen Wert hat.

4. XSLT basiert auf dem Modell, dass als Eingabe ein Baum existiert (ein Information Set, das in einen XPath Node Tree überführt wird), und dass die Ausgabe ebenfalls ein Baum ist (dessen Serialisierung mittels der `xsl:output` Anweisung gesteuert wird). Welche Konsequenzen hat dies für die Verarbeitung von grossen Dokumenten, und welche Grenze würden Sie in etwa als maximale Dokumentengrösse setzen, wenn es um den Einsatz von XSLT geht?
   Der Baum im Speicher braucht (je nach Implementierung und Optionen) bis zu 10x soviel Speicher wie das entsprechende Dokument, und muss komplett generiert werden, bevor die Abarbeitung beginnen kann. Geht man von einem Arbeitsspeicher von 1GByte aus, könnte man also Dokumente bis zu einer Grösse von ca. 50MByte recht problemlos verarbeiten, darüber können Platzprobleme auftreten, die zu grossen Performanceeinbussen führen können (viel Swapping für den zu grossen Baum). Dies berücksichtigt jedoch noch nicht die Grösse des Ausgabedokuments.
5. Führt man ein leeres Stylesheet aus, so erscheint als Ausgabe der "Text" (genaugenommen der *String Value*) des XML Dokumentes. Attribute erscheinen dabei nicht. Dies ist das Resultat der Built-in Template Rules, die relevanten sind die beiden folgenden:
   ```
   <xsl:template match="* | /">
         <xsl:apply-templates select="node()"/>
   </xsl:template>
   <xsl:template match="text() | @*">
         <xsl:value-of select="."/>
   </xsl:template>
   ```
   Offensichtlich definiert das zweite Template, dass Attributwerte ausgegeben werden sollen. Wieso erscheinen sie dennoch nicht als Ausgabe?
   Weil sie nicht selektiert werden durch das erste Template, das nur die Kinderknoten (mit `select="node()"`) selektiert und damit die Attribute (die in XPath keine Kinder eines Elementes sind) ignoriert.
6. Das Eingabedokument, Variablen und Parameter, und mit der `document()`-Funktion geöffnete Dokumente sind alles separate Bäume im Speicher, die jeweils einen eigenen Root Node haben. Sind diese Bäume miteinander verbunden, wenn ja, wie, und wie gelangt man von einem Baum zum anderen?
   Die Bäume sind nicht miteinander verbunden, und man kann daher nicht auf einem "Pfad" von einem Baum zum anderen gelangen. Es ist daher notwendig, von einem Baum zum anderen zu "springen", und dies macht man am einfachsten durch Variablen, in denen man sich den Root Node der verschiedenen Bäume merkt. Bei document()-Bäumen kann man auch einfach ein weiteres man die document()-Funktion aufrufen.
7. Erklären Sie, welche Software-Komponenten beim Ausführen von XSLT Stylesheets mit dem Saxon XSLT Processor auf Ihrem Rechner aufeinander aufbauen, und weshalb dies wahrscheinlich keine gute Grundlage für eine extrem effiziente Programmausführung ist.
   Saxon ist ein Java Programm. Das XSLT Stylesheet wird also vom Saxon XSLT Prozessor ausgeführt, der seinerseits auf einer JVM läuft, die als Programm des Windows oder Linux Betriebssystems läuft. XSLT und Java

Bytecode sind zudem interpretierte Sprachen (ausser beim Sun HotSpot JIT Compiler, der in der Windows JRE enthalten ist), so dass Initialisierung und Ausführung recht aufwendig sind.