

# Letzte Vorlesung

Themen der letzten Vorlesung:

- XML: Attribute, Entities
- DTD und Überprüfung von XML-Dokumenten
- XML-Namensräume
- Umwandlung von XML für Darstellung: XSLT
- Baumrepräsentation von XML
- XPath
- `xsl:template`

# Templates

Ein Template definieren:

```
<xsl:template match = pattern  
              name = name  
              priority = number>
```

Transformation

```
</xsl:template>
```

- *pattern*: XPath-Ausdruck, der bestimmt welche Elemente transformiert werden
- *name*: Name für die Regel
- *priority*: Reihenfolge der Abarbeitung im Konfliktfall

# Ablauf einer XML-Transformation

Laufe die Baumrepräsentation des Dokuments bei der Wurzel beginnend ab:

- Erzeuge Liste, bestehend aus dem Wurzelknoten
- Verarbeitet eine Liste aus Knoten, die die verarbeiteten Unterlemente darstellen
- Überprüft die Regeln für jedes Listenelement, wähle Regel mit höchster Priorität aus. Die Regel wird mit dem gewählten Knoten instanziiert

Das Template wählt in der Regel weitere Knoten aus, dort wird der Prozess fortgesetzt.

# XSLT-Anweisungen (1)

Innerhalb eines Templates können XSLT-Anweisungen verwendet werden. Auswahl wichtiger Anweisungen:

- **xsl:apply-templates** Das Attribut `select` (XPath-Ausdruck) wählt Kindknoten zur weiteren Verarbeitung aus
- **xsl:attribute** Fügt ein Attribut zum Ergebnisknoten hinzu
- **xsl:value-of** Erzeugt einen Textknoten aus der Stringrepräsentation eines Knotens
- **xsl:text** Erzeugt einen Textknoten aus dem Inhalt des Elements
- **xsl:for-each** Wendet den Inhalt des Elements auf jeden Knoten an, den das Attribut `select` auswählt

## XSLT-Anweisung (2)

- **xsl:if** Führt Template im Inhalt an, wenn der Ausdruck im Attribut **test** wahr ergibt
- **xsl:choose** Unterscheidet durch **xsl:when**-Elemente gegebene Fälle, analog zu **xsl:if**. Falls kein Fall wahr ist, wird evt. vorhandenes **xsl:otherwise** angewendet
- **xsl:sort** Sortiert Knotenliste nach dem im Attribut **select** angegebenen Schlüssel
- **xsl:message** Gibt den Inhalt auf der Konsole aus

## Beispiel: pfi.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE veranstaltung SYSTEM "veranstaltung.dtd">
<?xml-stylesheet href="kommvv.xsl" type="text/xsl"?>
<veranstaltung jahr="2005" art="Vorlesung"
  titel="Programmieren für das Internet">
  <dozent>
    <name>Prof. Dr. H. Klaeren</name>
  </dozent>
  <dozent>
    <name>Eric Knauel</name>
  </dozent>
  <dozent>
    <name>Mandeep Singh</name>
  </dozent>
  <zeit_änderung_nach_release="ja">
    <wochentag>Dienstag</wochentag>
    <uhrzeit>15:15-17:00</uhrzeit>
  </zeit>
  <raum>Sand 6/7, großer Hörsaal</raum>
</veranstaltung>
```

## Beispiel: kommvv-simple.xsl (1)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="html" encoding="ISO-8859-1"/>
  <xsl:template match="veranstaltung">
    <html>
      <body>
        <title><xsl:value-of select="@titel"/></title>
        <h1><xsl:value-of select="@titel"/></h1>
        <p>Dozenten:
          <xsl:apply-templates select="dozent"/>
        </p>
        <xsl:apply-templates select="zeit"/>
        <xsl:apply-templates select="raum"/>
        <hr/>
        <p>Alle Angaben ohne Gewähr</p>
      </body>
    </html>
  </xsl:template>
```

## Beispiel: kommvv-simple.xsl (2)

```
<xsl:template match="dozent">  
  <xsl:value-of select="name"/><xsl:text>#160;</xsl:text>  
</xsl:template>
```

```
<xsl:template match="zeit">  
  <p> Uhrzeit: <em> <xsl:apply-templates/></em></p>  
</xsl:template>
```

```
<xsl:template match="raum">  
  <p>Raum: <em> <xsl:apply-templates/> </em></p>  
</xsl:template>
```

```
</xsl:stylesheet>
```

## Beispiel: kommvv.xsl (1)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <xsl:output method="html" encoding="ISO-8859-1"/>
  <xsl:template match="veranstaltung">
    <html>
      <body>
        <title><xsl:value-of select="@titel"/></title>
        <h1><xsl:value-of select="@titel"/></h1>
        <p>Dozenten:</p>
        <ul>
          <xsl:for-each select="dozent">
            <li>
              <xsl:apply-templates select="self::node()"/>
            </li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

## Beispiel: kommvv.xsl (2)

```
<xsl:apply-templates select="zeit"/>
<xsl:apply-templates select="raum"/>
<hr/>
<p>Alle Angaben ohne Gewähr</p>
</body>
</html>
</xsl:template>
```

## Beispiel: kommvv.xsl (3)

```
<xsl:template match="dozent[position()&lt;last()]">  
  <xsl:apply-templates select="name"/>,  
</xsl:template>
```

```
<xsl:template match="dozent">  
  <xsl:apply-templates select="name"/>  
</xsl:template>
```

## Beispiel: kommvv.xsl (4)

```
<xsl:template match="zeit">
  <p>
    Uhrzeit: <em>
      <font>
        <xsl:choose>
          <xsl:when test="@änderung_nach_release='ja'">
            <xsl:attribute name="color">#FF0000</xsl:attribute>
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name="color">#000000</xsl:attribute>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:apply-templates select="node()" />
      </font>
    </em>
  </p>
</xsl:template>
```

## Beispiel: kommvv.xsl (5)

```
<xsl:template match="raum">  
  <p>Raum: <em> <xsl:apply-templates/> </em></p>  
</xsl:template>  
</xsl:stylesheet>
```

# ***Clientseitige Webprogrammierung***

# Clientseitige Webprogrammierung

Bisher:

- Webserver liefert Daten, Browser stellt diese dar
- Mit XSLT: Einfache Transformationen im Browser

Mit *clientseitigem Code* möglich:

- Überprüfung von Formulareingabe ohne neu zu laden
- Zugriff auf Browserfunktionalität
- Dynamisches HTML
- Asynchrones Nachladen von neuen Informationen

⇒ *Bessere und schnellere Interaktion mit dem Benutzer*

# Programmiersprachen und Browser

Java Applets, ActiveX controls

- Spezielles HTML-Tag mit Verweis auf (Byte-) Code der Anwendung
- Applets: Eingeschränkter Zugriff auf Ressourcen des Client
- Laufen meist in einem klar abgegrenztem Bereich im Browser
- Applets: Verbreitung nimmt ab, da erhebliche Portabilitätsprobleme
- ActiveX controls: Setzt Windows voraus, verbreitete Sicherheitsbedenken

(Wieder) im Aufwind: *JavaScript*

# JavaScript

JavaScript ist eine im Browser eingebaute Programmiersprache.

- 1995 von Netscape entwickelt
- Wird von praktisch allen Browsern unterstützt
- Microsoft-Variante heißt JScript
- Achtung: Es steckt nur sehr wenig Java in JavaScript
- Vorbilder für JavaScript: Scheme und Self
- Sprach-Standard: ECMA 262 (European Computer Manufacturers Association)

# Einbinden von JavaScript

Mit dem `script`-Tag wird JavaScript-Code eingebunden

Attribute:

- `type` ist zu setzen auf: `text/javascript`
- `href` optional, URL zum Code

Der Code im Inhalt als HTML-Kommentar

# Ein erstes JavaScript

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head><title>Test</title>
</head>
<body>
  <script type="text/javascript">
    <!--
      document.write("Dieser Text wurde von " +
        "<em>JavaScript</em> ausgegeben");
    -->
  </script>
</body>
</html>
```

# Datentypen

Primitive Typen in JavaScript:

Undefiniert	Typ des Wertes in einer nicht-initialisierten Variable, Typ des Rückgabewerts einer Funktion ohne Rückgabewert, Literal: <b>undefined</b>
Null	Literal: <b>null</b>
Boolean	Literale: <b>true</b> und <b>false</b>
Zahlen	Ganzzahlen, Fließkommazahlen, NaN
Zeichenketten	Literale in einfachen oder doppelten Anführungszeichen

# Typsystem

JavaScript ist dynamisch getypt und führt automatische Typkonversion durch:

- *Nach Boolean:*  
Liefere `false`: `undefined` `null` `0` `NaN` `""`  
Alles andere ist `true`
- *In eine Zahl:*  
`NaN` für `undefined`, `0` für `null`, `0` für `false`, `1` für `true`, falls String Zahl darstellt, diese Zahl, `NaN` sonst
- *In einen String:*  
`"undefined"`, `"null"`, `"false"`, `"true"`,  
Repräsentation der Zahl

Typkonversion erzwingen: `Number()` `Boolean()` `String()` `void`

Typabfrage mit `typeof`-Operator liefert String

# Funktionen

Funktionen können auf Toplevel und innerhalb von Funktionen definiert werden. Syntax:

```
function name(parameter, ...)  
    body-statement
```

Rückgabewert mit **return**, sonst **undefined**

Funktionen sind *higher order*. Können überall dort auftreten, wo normale Werte auch auftreten können.

# Variablen

Variablen müssen deklariert werden. Name ist case-sensitive.

```
var name [= expr];
```

- Überall sichtbar, wenn auf Toplevel deklariert
- Sonst im momentanen Scope (Damit auch in lokalen Funktionsdefinitionen)

⇒ Lexikalische Bindung und Closures

# Kontrollstrukturen

Übliche Kontrollstrukturen einer imperativen Sprache:

- `if (<test>) <statement>`
- `if (<test>) <statement> else <statement>`
- `while (<test>) <statement>`
- `switch (<expr>) {  
    case <label>: <statement> break;  
    ...  
    default: <statement> }`
- `do <statement> while (<expr>)`
- `for (<expr>; <expr>; <expr>) <statement>`

# Objekt-Orientierung

JavaScript ist objekt-basiert:

- Keine Klassen, nur Konstruktoren für Objekte
- Wichtiger Unterschied zu Java
- Konstruktoren sind normale JavaScript-Funktionen

Aufruf eines Konstruktors:

```
new constructor(name, ...);
```

Der Konstruktor kann Instanzvariablen hinzufügen, indem er sie an Object mit `this` und der `.`-Notation anfügt

## Objekt definieren und instanziiieren

```
function Text(contents){  
    this.contents = contents;  
}
```

```
bla = new Text("Brabbel");
```

```
document.write(bla.contents);
```

# Methoden hinzufügen

Methoden sind Instanzvariablen, die eine Funktion als Wert haben

```
function Farbigertext(farbe, text){
    this.text = text;
    this.farbe = farbe;
}

roter_text = new Farbigertext("#FF0000", bla);

function write_colored(){
    document.write("<font color=" + this.farbe + ">"
        + this.text.contents
        + "</font>")
}

roter_text.write = write_colored;

roter_text.write();
```

# Arrays

Arrays werden mit vordefiniertem Objekt erzeugt.

- Erzeuge mit `new Array(length)`
- Oder: `new Array(element-1, element-2)`

Referenzieren und Modifizieren mit `[ ]`

Es gibt keine Bereichsüberprüfung:

- Referenzieren außerhalb liefert `undefined`
- Modifizieren außerhalb fügt Element hinzu

Index kann ein beliebiges Objekt sein

## Beispiel: Arrays

```
myArray = new Array("eins","zwei","drei");
document.write("An Index 0 steht " + myArray[0] + " ");

a2 = new Array(2);
document.write("Länge: " + a2.length);
a2[3] = 4;
document.write("An Index 0 steht " + a2[0] + " ");
document.write("An Index 3 steht " + a2[3] + " ");

a2["sdf"] = 6;
document.write("An Index sdf steht " + a2["sdf"] + " ");

a2[-1] = "minus eins";
document.write("An Index -1 steht " + a2[-1] + " ");
document.write("Länge: " + a2.length);
```