Ввод и обработка цифровой информации

XSL

Пучинин Сергей Александрович

XSL

XSL (eXtensible Stylesheet Language) — семейство рекомендаций W3C, описывающее языки преобразования и отображения XML-документов.

XSLT — язык преобразований XML-документов.

XPath — язык запросов к элементам XML-документа

XPath

XPath — определяет синтаксис выражений, позволяющих выделить из XML-документа, некоторый набор узлов.

Синтаксис XPath-выражения

Выражение состоит из некоторого множества шагов, разделённых символом / .

Каждый шаг имеет вид:

Ось::Проверка узлов[предикат]

Оси XPath

ancestor:: — Возвращает множество предков.

ancestor-or-self:: — Возвращает множество предков и текущий элемент.

attribute:: — Возвращает множество атрибутов текущего элемента. Это обращение можно заменить на «@»

child:: — Возвращает множество потомков на один уровень ниже. Это название сокращается полностью, то есть его можно вовсе опускать.

descendant:: — Возвращает полное множество потомков (то есть, как ближайших потомков, так и всех их потомков).

descendant-or-self:: — Возвращает полное множество потомков и текущий элемент. Выражение «/descendant-or-self::node()/» можно сокращать до «//».

Оси XPath

following:: — Возвращает необработанное множество, ниже текущего элемента.

following-sibling:: — Возвращает множество элементов на том же уровне, следующих за текущим.

namespace:: — Возвращает множество, имеющее пространство имён (то есть присутствует атрибут xmlns).

parent:: — Возвращает предка на один уровень назад. Это обращение можно заменить на «..»

preceding:: — Возвращает множество обработанных элементов исключая множество предков.

preceding-sibling:: — Возвращает множество элементов на том же уровне, предшествующих текущему.

self:: — Возвращает текущий элемент. Это обращение можно заменить на «.»

Дополнительные символы

* — обозначает любое имя или набор символов по указанной оси, например: * — любой дочерний узел; @* — любой атрибут.

$name — обращение к переменной, где name — имя переменной или параметра.

{} — если применяется внутри тега другого языка (например HTML), то XSLT процессор рассматривает содержимое фигурных скобок как XPath.

| — объединяет результат. То есть, можно написать несколько путей разбора через знак | и в результат такого выражения войдёт всё, что будет найдено любым из этих путей.

Системны функции XPath

`node-set node()`

Возвращает все узлы.

`string text()`

Возвращает набор текстовых узлов.

`node-set current()`

Возвращает текущий элемент.

`number position()`

Возвращает позицию элемента.

`number last()`

Возвращает номер последнего элемента.

`number count(node-set)`

Возвращает количество элементов в node-set.

`string name(node-set)`

Возвращает полное имя первого тега в множестве.

`string generate-id(node-set)`

Возвращает строку, являющуюся уникальным идентификатором.

`node-set id(object)`

Находит элемент с уникальным идентификатором

Строковые функции

`string string(object?)`

Возвращает текстовое содержимое элемента.

`string concat(string, string, string*)`

Объединяет две или более строк

`number string-length(string)`

Возвращает длину строки.

`boolean contains(string, string)`

Возвращает истину, если первая строка содержит вторую.

`string substring(string, number, number?)`

Возвращает строку вырезанную из строки с указанного номера, второй номер — количество символов.

`string normalize-space(string?)`

Убирает лишние и повторные пробелы, а также управляющие символы, заменяя их пробелами.

Числовые функции

+ — сложение

- — вычитание

* — умножение

`div` — обычное деление (не деление нацело!)

`mod` — остаток от деления

`number number(object?)`

Переводит объект в число.

`number sum(node-set)`

Вернёт сумму множества, каждый тег множества будет преобразован в строку и из него получено число.

`number round(number)`

Округляет число.

```
<?xml version="1.0" encoding="UTF-8"?>
<корень>
  <предок/>
  <предок>
      <потомок/>
      <потомок имя="Ваня"/>
      <потомок имя="Петя">
            <потомок имя="Ваня"/>
      <потомок/>
  </предок>
</корень>
```

Примеры XPath

//предок — вернёт обоих предков

//предок/потомок — вернёт 3-х потомков

//потомок — вернёт 4(!) потомков

//потомок/потомок — вернёт внутреннего потомка

//потомок[@имя="Ваня"] — вернёт 2-х потомков

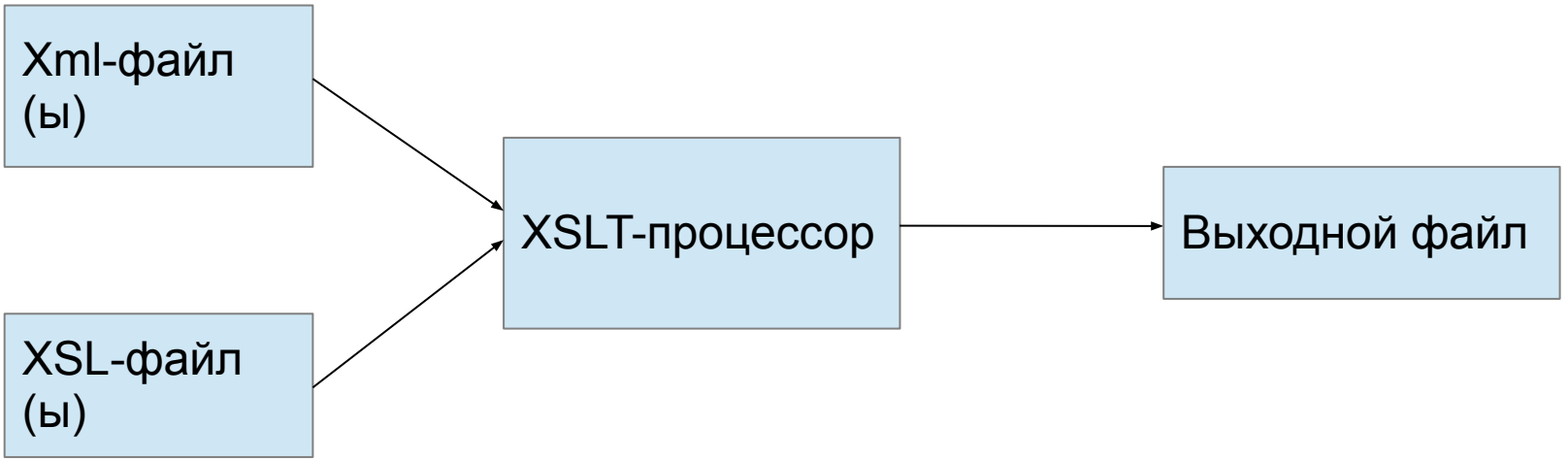
//потомок[2] — вернёт потомка Ваня

//потомок[1] — вернёт двух потомков

XSLT

eXtensible Stylesheet Language Transform

Xml-подобный декларативный язык позволяющий преобразовать Xml-файл в другой текстовый формат.



Ссылка на XSL в XML

В xml-файл после заголовка помещается строка:

<?xml-stylesheet type="text/xsl" href="trans.xsl"?>

href — путь к xsl-файлу

Если браузер имеет встроенный xslt-парсер — то браузер автоматически отобразит xml, как html

Заголовок XSL

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transf
   orm" version="1.0">

    <xsl:output method="html" indent="yes"/>
…
</xsl:stylesheet>
```

Шаблоны

Элемент xsl:template задаёт шаблон по которому будет обрабатываться определённая часть документа.

Проводя аналогию с другими языками программирования можно сказать, что это процедура.

Есть два способа вызвать обработку по шаблону.

1) <xsl:apply-templates select = "Expression" /> вызывает шаблоны соответствующие элементам выбранным с помощью выражения

2) <xsl:call-template name = "TemplateName" /> вызывает шаблон по имени

Создание шаблона

<xsl:template
  match = pattern
  name = qname >

pattern - XPath выражение для которого будет применяться шаблон.

qname — имя шаблона по которому его можно вызывать.

Передача параметров

Параметры в шаблоне задаются с помощью элемента

<xsl:param name="ParamName"/>

Для передачи параметров в шаблон используется элемент

<xsl:with-param name="ParamName">

Пример шаблона для корня

```
<xsl:template match="/">
  <HTML>
    <BODY>
      <!-- Применять остальные шаблоны -->
      <xsl:apply-templates/>
    </BODY>
  </HTML>
</xsl:template>
```

Передача параметров

Параметры в шаблоне задаются с помощью элемента

<xsl:param name="ParamName"/>

Для передачи параметров в шаблон используется элемент

<xsl:with-param name="ParamName">

Пример передачи параметров

```
<xsl:template name="msg23" match="msg23">
  <xsl:call-template name="subTemplate">
    <xsl:with-param name="code">msg23</xsl:with-
param>
  </xsl:call-template>
</xsl:template>

<xsl:template name="subTemplate">
  <xsl:param name="code"/>
  <!-- Do something. -->
</xsl:template>
```

Пример передачи параметров

```
<xsl:template name="msg23" match="msg23">
  <xsl:call-template name="subTemplate">
    <xsl:with-param name="code">msg23</xsl:with-
param>
  </xsl:call-template>
</xsl:template>

<xsl:template name="subTemplate">
  <xsl:param name="code"/>
  <!-- Do something. -->
</xsl:template>
```

Переменные

```
<xsl:variable name = "Имя_переменной"
  select = "Значение" />

<xsl:variable name = "Имя_переменной">
  "Значение"
</xsl:variable>
```

Вывод значения

```
<xsl:value-of
  select = Expression
/>
```

Expression - XPath выражение, которое будет преобразовано в текстовый вид.

Условия

```
<xsl:if
    test = "выражение">
    <!-- шаблон содержимого -->
</xsl:if>
```

Шаблон содержимого будет выполнятся только тогда, когда результат *выражения,* приведённый к логическому типу будет истинной

Ветвление

```
<xsl:choose>
    <xsl:when test="$level = 1">
        <!--Шаблон для 1 -->
    </xsl:when>
    <xsl:when test="$level = 2">
        <!--Шаблон для 2 -->
    </xsl:when>
    <xsl:otherwise>
        <!--Шаблон для остальных случаев -->
    </xsl:otherwise>
</xsl:choose>
```

Циклы

```
<xsl:for-each
    select = "выражение">
    <!-- Содержимое: (xsl:sort*, шаблон) -->
</xsl:for-each>
```

выражение — XPath-выражение, возвращающее набор узлов.

Сортировка

```
<xsl:sort
    select = "string-expression"
    lang = "nmtoken" //"en" | "en-us" | "ru"
    data-type = "text | number | qname-but-not-ncname"
    order = "ascending | descending"
    case-order = "upper-first | lower-first" />
```

Этот элемент может содержаться внутри xsl:for-each и xsl:apply-templates для изменения порядка обхода узлов. Возможно использование нескольких xsl:sort для сортировки по нескольким ключам.

Создание элементов

```
<xsl:element
  name = "Имя" >
  <!-- Определение содержимого -->
</xsl:element>
```

Создание атрибутов

```
<xsl:attribute
  name = "Имя" >
  <!-- Определение содержимого -->
</xsl:attribute>
```

Пример: Замена имени элемента именем атрибута

```
<xsl:template match="*">
 <xsl:element name="{@*}">
  <xsl:attribute name="{name(@*)}">
   <xsl:value-of select="name()"/>
  </xsl:attribute>
 </xsl:element>
</xsl:template>
```

<fire on="babylon"/> → <babylon on="fire"/>