XSLT

Докладчик:
Пакудин Николай

Академический физико-технологический университет РАН
Санкт-Петербург
2010

Проблема

- Есть XML документ.
- В принципе, в нем есть вся нужная нам информация.
- Но нам она нужна в виде XML другой схемы/в виде HTML/в виде текста.

Пути решения

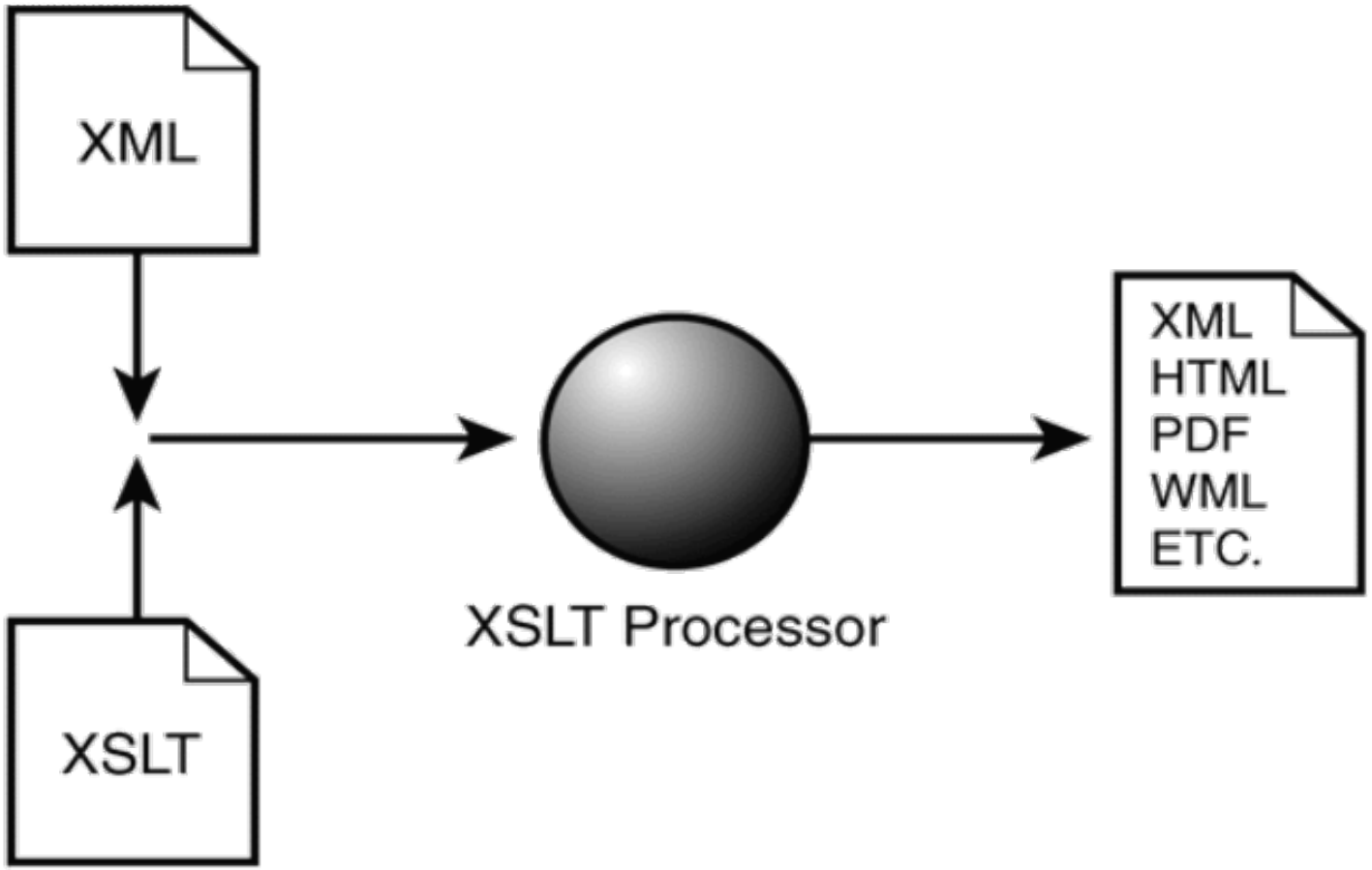
- Можно написать программу на C++/Java/Python/..., которая будет преобразовывать именно этот документ.

  Минус: много кода, занимающегося разбором xml, а не преобразованием из нашей схемы.

  Отлаживать довольно трудно.

- А можно использовать XSLT.

Принцип работы



Исходный XML

```
<?xml version="1.0" encoding="utf-8"?>
<university name="АФТУ">
<student gender="male">
<surname>Иваницкий</surname>
<name>Андрей</name>
</student>
<student gender="female">
<surname>Егорова</surname>
<name>Светлана</name>
</student>
<student gender="male">
<surname>Калегин</surname>
<name>Андрей</name>
</student>
</university>
```

Пример 1: результат

```
<html>
<body>
<h1>АФТУ</h1>
<ul>
<li>Иваницкий</li>
<li>Егорова</li>
<li>Калегин</li>
</ul>
</body>
</html>
```

Пример 1: XSLT преобразование

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html" indent="yes"/>
<xsl:strip-space elements="*"/>

<xsl:template match="/">
<html><body>
<xsl:apply-templates/>
</body></html>
</xsl:template>

<xsl:template match="university">
<h1><xsl:value-of select="@name"/></h1>
<ul><xsl:apply-templates/></ul>
</xsl:template>

<xsl:template match="student">
<li><xsl:value-of select="surname"/></li>
</xsl:template>
</xsl:stylesheet>
```

Пример 1: как это работает

- **<?xml version="1.0" encoding="utf-8"?>** - это валидный xml.
- **<xsl:stylesheet ...>** - это преобразование xml.
- **<xsl:output method="html" indent="yes"/>** - на выходе будет html, и там надо красиво расставить отступы.
- **<xsl:strip-space elements="*"/>** - надо удалить лишние пробелы, чтобы не нарушалось форматирование.

Пример 1: как это работает

- **<xsl:template match="/">** говорит: взять корневой элемент исходного xml файла и применить к нему этот шаблон.
- Когда xslt процессор дойдет до корневого элемента в исходном xml, он подставит вместо него то, что написано внутри этого шаблона.
- Перед подстановкой специальные теги (обычно начинаются с **xsl:**) будут раскрыты.
- **<xsl:apply-templates/>** говорит: для вложенных элементов искать шаблоны и проводить преобразование, результат поставить сюда.
- **<xsl:template match="university">** говорит: если внутри текущего тега есть тег **university**, то применить для него это правило.
- **<xsl:value-of select="@name"/>** говорит: у текущего тега взять значение атрибута **name** и подставить его сюда.
- **<xsl:value-of select="surname"/>** говорит: у текущего тега взять ребенка по имени **surname** и подставить его содержимое сюда.

Схема преобразования

Важно: XSLT — это набор декларативных правил, а не последовательные инструкции, как в C++.

- Парсим файл преобразования и строим XML дерево входного файла.
- Ищем шаблон, который лучше всего подходит для корневого узла и вычисляем содержимое найденного шаблона.
- Инструкции в каждом шаблоне могут:
  - говорить XSLT процессору "создай здесь такой-то тег";
  - говорить XSLT процессору "обработай другие узлы по тому же правилу, что и корневой узел".

Обработка одного узла

- Ищем подходящее правило (чтобы то, что написано в **match**, подходило под текущий элемент).
- Если подходит несколько правил, то у тега **xsl:template** есть атрибут **priority**, и выбирается правило с наибольшим приоритетом.
- Если приоритеты тоже совпадают, срабатывает какой-то хитрый алгоритм.
- **<xsl:apply-templates>** применяет правила для текущего узла или его детей.
- Если узел — комментарий, то по умолчанию он удаляется.
- Если узел — текст, то по умолчанию он копируется без преобразования.

Набор правил по умолчанию

```
<xsl:template match="* | /">
<xsl:apply-templates />
</xsl:template>

<xsl:template match="text() | @*">
<xsl:value-of select="." />
</xsl:template>

<xsl:template match="processing-instruction() | comment()" />
```

Пример 2: результат

```
<html>
<body>
<h1>АФТУ</h1>
<ul>
<li>Иваницкий Андрей</li>
<li>Калегин Андрей</li>
</ul>
</body>
</html>
```

Пример 2: XSLT преобразование

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"/>
<xsl:strip-space elements="*"/>

<xsl:template match="/">
<html><body>
<xsl:for-each select="university/student">
<xsl:if test="contains(surname, 'и')">
<li>
<xsl:value-of select="surname"/>
<xsl:text> </xsl:text>
<xsl:value-of select="name"/>
</li>
</xsl:if>
</xsl:for-each>
</body></html>
</xsl:template>

<xsl:template match="student">
<li><xsl:value-of select="surname"/></li>
</xsl:template>
</xsl:stylesheet>
```

Пример 2: как это работает

- **<xsl:for-each select="...">** - идет последовательно по элементам, которые указаны в **select**.
- **select="university/student"** - выбирает в текущем элементе все подэлементы university, а внутри них - элементы **student**. Выдает только **student**.
- **<xsl:if test="...">** - если выполняется условие, то выполняем то, что внутри.
- **contains(surname, 'и')** - true, если содержимое surname содержит строку 'и'.
- **<xsl:text> </xsl:text>** - содержимое этого тега будет выведено без изменений (иначе из-за **<xsl:strip-space elements="*"/>** все пробелы между тегами будут удалены, и имя и фамилия будут идти слитно).
- **<xsl:template match="student">** - правило есть, но оно ни на что не влияет, т.к. нигде нет вызова **<xsl:apply-templates/>.**

XPath: отношения между узлами в XML

- Parent
- Children
- Siblings — братья.
- Ancestors — все предки.
- Descendants — все потомки.

XPath: пути

- **nodename** — все дети текущего узла с именем nodename.
- **nodename/subnodename** — см. выше.
- **nodename//subnodename** — между nodename и subnodename может быть любое количество «папок».
- **/nodename/subnodename** — путь от корня.
- **.** — текущий узел.
- **..** — родитель.
- **@attribute-name** — атрибут с именем **attribute-name.**

- Пути пишем в:
  - **<xsl:template match="...">**
  - **<xsl:apply-templates select="..." />**
  - **<xsl:for-each select="...">**
  - **...**

XPath: предикаты

- **nodename/subnodename[1]** — 1-й узел в последовательности **subnodename**.
- **nodename[last() - 1]** — предпоследний узел.
- **nodename/subnodename[position() < 3]** — первые два узла.
- **nodename[@lang='ru']** — все узлы, имеющие атрибут **lang** со значением **ru**.

XPath: wildcards

- * - любой узел.
- @* - любой атрибут.
- **node()** - любой узел или атрибут.

XPath: альтернативы

- **nodename1 | nodename2** — узлы с именем **nodename1** или с именем **nodename2**.
- **//nodename1 | nodename2[@gender] | nodename3//subnode** — можно писать сколько угодно альтернатив.

XPath: оси (axes)

- **ancestor** — все предки.
- **ancestor-or-self** — все предки или сам.
- **following-sibling** — все последуюшие братья.
- **preceding-sibling** — все предыдущие братья.
- **child** — все дети.
- **attribute** — все атрибуты.

XPath: полное имя элемента

- **axisname::nodetest[predicate]** — общий вид.
- **ancestor-or-self::nodename[subnode < 20]** — все предки или сам с именем **nodename**, где значение подузла **subnode** меньше 20.
- **attribute::*** - все атрибуты.
- **child::*/child::nodename** — все внуки с именем **nodename**.

Пример 3 - функции: результат

```
<html>
<body>
<li>Калегин
(АФТУ)
</li>
<li>Иваницкий
(АФТУ)
</li>
</body>
</html>
```

Пример 3 - функции: XSLT преобразование

```
<xsl:template match="/">
<html><body>
<xsl:for-each
select="university/child::student[@gender='male']">
<xsl:sort select="name"/>
<xsl:sort select="surname" order="descending"/>
<xsl:call-template name="print-student">
<xsl:with-param name="university-name" select="../@name"/>
</xsl:call-template>
</xsl:for-each>
</body></html>
</xsl:template>

<xsl:template name="print-student">
<xsl:param name="university-name" />
<li>
<xsl:value-of select="surname"/>
(<xsl:value-of select="$university-name"/>)
</li>
</xsl:template>
```

Пример 3 - функции: как это работает

- **<xsl:sort select="name"/>** - сортирует все элементы перед попаданием их в цикл по узлу name.
- **<xsl:sort select="surname" order="descending"/>** - если name совпадают, то сортирует по surname в обратном порядке.
- **<xsl:template name="print-student">**

<xsl:param name="university-name" />

определяет функцию **print-student** с параметром **university-name.**

- **<xsl:value-of select="$university-name"/>** - выводит значение этого параметра.
- **<xsl:call-template name="print-student">**

<xsl:with-param name="university-name" select="../@name"/>

вызывает функцию **print-student** со значением **../@name.**

Еще несколько полезных XSLT тегов

- Switch

```
<xsl:choose>
<xsl:when test="...">
</xsl:when>
<xsl:otherwise>
</xsl:otherwise>
</xsl:choose>
```

- Вставить код из файла, расположенного по адресу

```
<xsl:include href="..."/>
```

- Скопировать текущий узел без потомков

```
<xsl:copy>
some templates
</xsl:copy>
```

- Скопировать указанный узел со всеми потомками

```
<xsl:copy-of select="."/>
```

Пример 4 - группировка: результат

```
<html>
<body>
<ul>
<li>male<ul>
<li>Иваницкий</li>
<li>Калегин</li>
</ul>
</li>
<li>female<ul>
<li>Егорова</li>
</ul>
</li>
</ul>
</body>
</html>
```

Пример 4 - группировка: XSLT преобразование

```
<xsl:key name="key-gender" match="student" use="@gender"/>
<xsl:template match="university">
<html><body><ul>
<xsl:apply-templates select="student" />
</ul></body></html>
</xsl:template>

<xsl:template
match="student[not(preceding-sibling::student/@gender=@gender)]">
<li><xsl:value-of select="@gender"/><ul>
<xsl:apply-templates select="key('key-gender', @gender)"
mode="list-students"/>
</ul></li>
</xsl:template>

<xsl:template match="student" mode="list-students">
<li><xsl:value-of select="surname"/></li>
</xsl:template>

<xsl:template match="*">
</xsl:template>
```

Пример 4 - группировка: как это работает

- **<xsl:key name="key-gender" match="student" use="@gender"/>**
  - создает список из элементов student;
  - сопоставляет каждому значение @gender;
  - называет этот список key-gender.
- **<xsl:apply-templates select="student" /> -** применяет шаблоны только к элементам students.
- **student[not(preceding-sibling::**student/@gender=@gender**)]**
  - **preceding-sibling::**student/@gender=@gender **-** возвращает true, когда в списке предшествующих братьев есть элемент, у которого @gender равен @gender текущего.
  - **not(...) -** отрицание, т.е. true, когда в списке предшествующих братьев нет текущего значения @gender.
  - т.е. все выражение говорит: взять текущий элемент, когда в предыдущих нет текущего значения @gender.
  - Получается список уникальных значений @gender.

Пример 4 - группировка: как это работает

- **<xsl:apply-templates**

select="key('key-gender', @gender)"

mode="list-students"/>

- взять список **key-gender**
- выбрать только те элементы, значения у которых равны значению @gender у текущего элемента
- применить ко всем выбранным элементам шаблоны, у которых указан **mode="list-students"**
- **<xsl:template match="student" mode="list-students"> -** шаблон для student, у которого указан **mode="list-students"**
- **<xsl:template match="*">**

</xsl:template>

нужно, чтобы избавиться от артефактов прохода для группировки

Еще несколько замечаний

- Над корневым элементом создается еще один «суперкорень», которому соответствует путь /
- **<xsl:output method="xml" indent="yes"/>** - method может принимать значения html, text, xml.

Это означает, что можно из xml сделать:

- pdf;
- svg (формат масштабируемой векторной графики).
- Когда выводим в xml или html, то чтобы установить значение атрибута, пишем **<a href="{@my-address}">.**
- С помощью XSLT можно генерировать программный код.
- Связанные технологии:
  - XPath;
  - XSL-FO.

XSL- FO

- Документ XLS-FO — это XML файл, в котором хранятся данные для печати или вывода на экран (например, просто текст). Эти данные находятся внтури тегов <fo:block>, <fo:table>, <fo:simple-page-master> и др., где указаны отступы, переводы строк и т.д.
- Общая идея использования XSL-FO состоит в том, что пользователь создаёт документ, не в FO, но в виде XML. Это может быть, например, XHTML, DocBook или FictionBook. Затем пользователь применяет XSLT-преобразование либо написав его самостоятельно, либо взяв готовое, подходящее к этому типу документа. Этот XSLT преобразует XML в XSL-FO.
- После того как документ на XSL-FO получен, он передаётся приложению, которое носит название FO-процессор. Эта программа конвертирует XSL-FO-документ в какой-либо читаемый и/или печатаемый формат. Наиболее часто используется преобразование в PDF либо PS.

Ссылки

- http://www.w3schools.com/xml/xml_namespaces.asp - подробнее про XML namespaces.
- http://www.w3schools.com/xsl/default.asp - XSLT tutorial.
- http://www.w3schools.com/xpath/default.asp - XPath tutorial.
- http://www.w3schools.com/xsl/xsl_examples.asp - примеры XSLT. Здесь же можно без установки дополнительного софта попробовать XSLT.
- http://www.microsoft.com/express/downloads/#2008-All - Visual Studio Express — там есть XSLT редактор для Windows.

Задание

Исходный файл

```
<?xml version="1.0" encoding="utf-8"?>

<persons>
<person person-name="vasya">
<city city-name="spb"/>
<city city-name="moscow"/>
<city city-name="helsinki"/>
</person>
<person person-name="petya">
<city city-name="helsinki"/>
<city city-name="spb"/>
</person>
<person person-name="zhenya">
<city city-name="helsinki"/>
<city city-name="moscow"/>
</person>
<person person-name="masha">
<city city-name="helsinki"/>
</person>
</persons>
```

Результат

```
<?xml version="1.0" encoding="utf-8"?>


<cities>
<city city-name="spb">
<person person-name="vasya" />
<person person-name="petya" />
</city>
<city city-name="moscow">
<person person-name="vasya" />
<person person-name="zhenya" />
</city>
<city city-name="helsinki">
<person person-name="vasya" />
<person person-name="petya" />
<person person-name="zhenya" />
<person person-name="masha" />
</city>
</cities>
```

Задание - уточнения

- Есть список людей, и у каждого указано, в каких городах он бывал.
- Внутри person все **city** имеют уникальное имя.
- Надо создать список городов, и в каждом городе указать, кто в нем бывал.
- Желательно обойтись без конструкций **xsl:if** и **xsl:for-each**.

Спасибо за внимание!