21. Използване на XML за структуриране, валидация, обработка и представяне на документно съдържание.

Добре структуриран XML

Основни концепции:

- XML таг – markup на документа, отделен от останалата част със символите < и >. Съдържа името на елемента
    - Start tag <elementname>
    - End tag </elementname>
- XML елемент – информацията между отварящ и затварящ таг включително. Информацията между таговете се нарича съдържание на елемента.
- XML Markup – състои се от таговете в документа.
- XML документ - съдържа текст под формата на един или повече елементи.
- XML атрибути – в допълнение към тагове и елементи, XML документите могат да съдържат и атрибути. Атрибутите са двойки име/стойност (стойността е задължителна дори да е празен стринг), асоциирани с елемент. Те се добавят към отварящия таг - <name nickname=”Smth”> и стойностите им задължително се поставят между ‘’ или „“.
- Processing instructions - <?....?> - представят информация, необходима на външни приложения.
- XML декларация - <?xml version=’1.0’ encoding=’UTF-16’ standalone=’yes’ ?> - указва версията на XML спецификацията, към която се придържа документа, енкодинга му и дали с него е асоцииран външен DTD.

XML Йерархии:

XML документите представляват йерархични дървета. Всеки документ може да бъде представен като дърво, в което отделните възели са XML елементи.
Root елемент – всеки XML документ има точно един елемент, който няма родител. Той е първият елемент в документа и съдържа всички други елементи.
Всеки XML документ съдържа следното:
- Prolog – stylesheet & document type декларации
- Instance – йерархията от елементи
- Additions – коментари <!—comment--> и processing instructions <?PITarget Status=”draft”?>

Синтакични правила

XML документите трябва да се придържат към определени правила, за да бъдат добре структурирани:

- Всеки отварящ таг трябва да има съвпадащ затварящ таг или да бъде self-closing таг.
- Таговете не могат да се застъпват, елементите трябва да са properly nested.
- XML документите могат да имат само един root елемент.
- Имената на елементите трябва да спазват XML конвециите.
- XML е case sensitive.
- XML запазва whitespace в PCDATA.
- В съдържанието на документите не се позволяват запазените от XML символи. Те се заменят с predefined entities или се поставят в <![CDATA[.....]]> секция.

XML пространства от имена

Namespace/пространство от имена – абстрактна нотация (категория) за група от имена. Едно име може да принадлежи само към една група. Създадени са с цел да се осигури уникалност между XML

елементите. За име на namespace в XML обикновено се използва URI. Във всеки документ може да се съдържа информация за множество пространства от имена.
Декларация на namespace – за да се асоциира определен елемент с namespace в деклаацията му се добавя атрибут xmlns - <pers:person xmlns:pers="http://www.wiley.com/pers"/>. Тази декларация важи за елемента, в който е поставена, както и за всички негови наследници. В един елемент могат да бъдат деклрирани няколко namespace-а.

XML валидация чрез Document Type Definitions (DTD)

Цели на валидирането

Синтактичните правила в XML се налагат от парсерите, но често е необходимо да има начини за разграничаване на различните XML-базирани езици(речници). Всеки документ обикновено се опитва да следва определен език/речник и са необходими методи за определяне на нивото на съответствие. За тези цели се използва валидация чрез DTD или XML Schema.
XML документ е валиден, ако съдържанието му отговаря на дефиницията на позволените елементи, атрибути и други части от документа.
Целта на един DTD е да декларира всички имената на всички елементи и атрибути, които се използват в XML документите, отговарящи на това DTD, типовете им, видовете елементи, как те се изолзват заедно и йерархията на документа. DTD-тата представляват шаблони за markup-а на документите и съдържат формалната дефиниция на съответния тип документ. DTD може да бъде механизъм за стандартизация и манипулация на документи/данни и обмяната им.

DTD структура

DTD се съхраняват във външни файлове, или в самия XML файл или и двете.
DTD се състоят от:

- Document Type Declaration (DOCTYPE) – информира парсера, че с документа има асоцииран DTD. Задължително трябва да е поставена в началото на документа
- Тяло на DTD – поставя се след DOCTYPE декларацията и съдържа декларациите на елементи, атрибути, entities и нотации.
- Затваряне на декларацията на DTD с "]>"

```
<!DOCTYPE name [                          // Document Type Declaration
        <!ELEMENT name (first,  last)>    // Body
        <!ELEMENT first (#PCDATA)>
        <!ELEMENT last (#PCDATA)>
]>                                        // End of DTD
```

DTD синтаксис

DTD-тата се състоят от декларации на елементи, атрибути към тях, entity-та и нотации. Форматът на декларациите е <! ...> или < ! ... [< ! ... >] >

Document Type Definition

<![DOCTYPE root_element SYSTEM file […] > и *<![DOCTYPE root_element PUBLIC fpi (system_id)[…] >* указва на парсера, че с XML документа е асоцииран DTD. Съдържа <![DOCTYPE, името на root елемента на документа, и идентификатор към файла с DTD-то. SYSTEM и име на файл указва, че DTDто, асоциирано с документа, се намира във file, а PUBLIC – указва публично DTD, достъпно на fpi и опционално задава локално DTD, което да се използва, в случай че публичното не е достъпно.

Декларация на елемент

<!ELEMENT first (#PCDATA)>

Състои се от три основни части

- ELEMENT декларация
- Име на елемент
- Модел на съдържание на елемента – дефинира допустимото съдържание на елемента. Има 4 вида модели на съдържание:
    - Елемент *<!ELEMENT name (surname) >*
    - Смесен - *<!ELEMENT name (#PCDATA)>*
    - Празен *<!ELEMENT br EMPTY>*
    - Any *<!ELEMENT description ANY>* - всеки елемент, деклриран в DTD може да бъде използван в description в какъвто и е да е ред, колкото и да е пъти.

Всеки елемент, който е посочен в модела на съдържание на елемента, трябва да има своя дефиниция в DTD.
Има 2 начина за определянето на елементите наследници:

- Последователности (sequences)- *<!ELEMENT contact (name, location, phone, knows, description)>* - указва, че елементът contact съдържа елементите name, location, phone, knows, description.
- Избор (choices) - *<!ELEMENT location (address|GPS)>* - указва, че елементът location съдържа елементът address или елементът GPS.

Кардналност на елемент (cardinality) – дефинира колко пъти елементът може да се появи в модела на съдържание. Всеки елемент може да има индикатор колко пъти се среща. Допустимите индикатори са [none], ?, +, *.

Декларация на атрибут

<!ATTLIST contacts source CDATA #IMPLIED>

Декларацията на атрибут се състои от следните 3 части:

- Ключовата дума ATTLIST
- Името на елемента, чийто атрибути се дефинират
- Списък с атрибути – състои се от:
    - Име на атрибута
    - Тип
    - Декларация на стойността

Типовете атрибути са CDATA, ELEMENT, ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS.
Атрибутът може и да се дефинира като списък със допустими стойности.
<!ATTLIST doc lang (en|bg|ge) #IMPLIED>
DTD позволява да се зададе, че атрибутът има:

- Стойност по подразбиране – default - *<!ATTLIST doc lang (en|bg|ge) en >*
- Фиксирана стойност - #FIXED - *<!ATTLIST doc ver CDATA '1.0'>*
- Е задължителен - #REQUIRED *<!ATTLIST person nationality CDATA #REQUIRED>*
- Е опционален - #IMPLIED - *<!ATTLIST doc lang (en|bg|ge) #IMPLIED>*

Декларация на символни данни (character data)

Понякога текстът трябва да съдъжа неинтерпетирани markup символи. Това става със CDATA декларация:
<![CDATA[Press <<<ENTER>>>]]>

Декларация на Entities

XML документите могат да бъдат разпределени в някоко файла. Всяка единица информация се нарича entity, което има идентифициращо го име. Entities се дефинират с entity декларации и се използват с entity reference.
Има 4 вида entities:

- Вградени entities - *&, <, ', "*

- Character entities – подобни са на вградените, но не се декларират в DTD. Могат да се използват в документа без декларации чрз референции например - *©*
- General entities – декларират се в DTD преди да могат да бъдат използвани в документа. Обикновено се използват за създаването на заменим текст - *<!ENTITY source-text "Alabala">* и се използват чрез референция - *&source-text;*
- Parameter entities – не могат да се използват в съдържанието – единствено в DTD. Могат да е използват за създаването на DTD отняколко файла. Декларират се по следния начин - *<!ENTITY DefaultPhoneKind "Home">*. Реферират се чрез *%DefaultPhoneKind*

Условни секции

Състоят се от:

- Include декларации - *<!INCLUDE[…..]]>* - Включва секция DTD
- Ignore декларации - *<!IGNORE[…]]>* - Изключва секция DTD

Декларация на нотация

Описва външно не-XML entity. Може да указва и helper приложение и документация. Например

<!NOTATION name SYSTEM "external_id">
<!ATTLIST element_name attr_name NOTATION default_value>

XML валидация чрез XML Schema

Спецификации

XML Schema Спецификацията се разделя на 3 части:

- Part 0: Primer - съдържа въведение в XML Schema
- Part 1: Structures - дефинира структурата, дефинициите на типове, атрибути, елементи, групи от атрибути (attribute group), модели от групи (model group), нотации и анотации
- Part 2: Data types - определя типовете данни за елементи и атрибути. Дефинира фасети и възможните стойности за типовете.

Типове данни

XML Schema Recommendation дефинира следните видове данни:

- Built-in datatypes
  - примитивни типове данни – string, boolean, float, double, ID и пр.
  - Derived – language, IDREF, long, int, short и др.
- User-defined datatypes

Дефиниране на потребителски типове данни

Става по 3 начина:

- дефиниране от съществуваши типове данни чрез определянето на 1 или повече фасети.

```
<datatype name="TelephoneNumber" source="string">
        <length value="8"/>
        <pattern value="\d{3}-\d{4}"/>
</datatype>
```

- Derived Types - може да се прилага форма на subclassing на дефинициите на типове. Има 2 начина за това:
    - derive by extension - разширяване на родителския тип с повече елементи –
      <complexType name="Book" source="cat:Publication" derivedBy="extension">
    - derive by restriction - ограничаване на родителския тип чрез ограничения върху възможните стойности или броя на occurrences.
      < complexType name= "SingleAuthorPublication" source="cat:Publication" derivedBy="restriction">
- <simpleType> декларации

Фасети

Фасетите контролират всички прости типове в XML Schemas. Фасет е качество или отличителна черта на <simpleType>. Има 12 ограничаващи фасети:

- minExclusive
- minInclusive
- maxExclusive
- maxInclusive
- totalDigits - брой цифри за числов тип
- fractionDigits - брой дробни цифри за числов тип
- length - брой елементи в списък или брой символи на стринг
- minLength
- maxLength
- enumeration
- whiteSpace - определя как ще се третира whitespace в типа
- pattern - позволява рестрикцията на символни типове чрез регулярни изрази
- precision
- scale
- encoding
- duration
- period

Др. вид фасети са фундаменталните (не могат да се променят):

- equal - всички типове осигуряват релация равенство
- order - някой типове осигуряват и order релация
- bounds - горна и долна граница
- cardinality
- numeric

Структури

Структурните типове се използват за описание на елементи, които имат асоциирани към тях елементи наследници или атрибути.

<element> декларации

При декларирането на елемент се задават името му и допустимото му съдържание:
<element
name=”name of the element”
type=”global type”
ref=”global element declaration”
form=”qualified or unqualified”
minOccurs=”non negative number”

maxOccurs=”non negative number or ‘unbounded’”
default=”default value”
fixed=”fixed value”>

Допустимото съдържание се определя от неговия тип. XML Schemas позволяват определянето на типа на елемента по 2 начина:

- създаване на локален тип - нямат <schema> като директен родител и могат да се ползват само в определен контекст.
- използване на глобален тип - глобалните декларации могат да се преизползват в XML Schema чрез добавянето на ref атрибут - <element ref=”target:first”/>. Задължително глобалният тип трябва да е именован.

<complexType> декларации

Чрез тях се създават сложни потребителски типове. В рамките на дефиницията може да се укаже допустимото съдържание на елемента.
<complexType mixed=”true or false” name=”Name of complexType”>

<simpleType> декларации

Винаги когато се декларира <simpleType> той трябва да се базира на съществуващ тип данни.

<simpleType name=”name of the simpleType” final=”#all or list or union or restriction”>

```
<simpleType name=“sku” base=“xsd:string>
    <pattern value=“\d{3}-[A-D]{4}”/>
</simpleType>
```

<group> декларации

Дефинират преизползваеми групи от елементи - чрез тях може лесно да се преизползват и комбинират цели модели на съдържание. Дефинициите на глобални групи трябва да са именовани - чрез добавянето на атрибут *name*. Могат да се дефинират групи единствено от елементи.

```
<group name=”NameGroup”>
    <!-- content model goes here -->
</group>
```

Към дефиницията на групата може да има атрибут order със следните допустими стойности:

- *choice* - инстанциите на групата съдържат някой от елементите
- *all* - инстанциите на групата съдържат всички елементи
- *seq* - - инстанциите на групата съдържат всички елементи в зададения ред

Модели на съдържание (Content Models)

В XML Schemas моделът на съдържание на елементите може да се дефинира чрез:

- <sequence> декларация - указва последователност от елементи, които се съдържат в реда на указването им.
  <sequence minOccurs=”non negative number” maxOccurs=”non negative number or unbounded”>
- А <choice> декларация - могат да се зададат множество декларации наследници. В документа обаче само 1 от декларациите може да бъде използвана.
  <choice minOccurs=”non negative number” maxOccurs=”non negative number or unbounded”>
- Референция към глобална <group> декларация -

<group ref="global group definition"
minOccurs="non negative number"
maxOccurs="non negative number or unbounded">

- An <all> декларация - позволява декларирането на елементи, които могат да се появят в какъвто и да е ред.
  <all minOccurs="0 or 1" maxOccurs="1">
- Вътрешни модели на съдържание
- Element декларация
- Element wildcards
- any Елемент - Позволява всеки well-formed XML. Може да бъде ограничен чрез атрибут Namespace, който да указва от кой namespace са допустимите елементи.
  <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>

<attribute> декларации

Винаги са последни след декларациите на всички елементи.
<attribute name="name of the attribute"
type="global type"
ref="global attribute declaration"
form="qualified or unqualified"
use="optional or prohibited or required"
default="default value"
fixed="fixed value">

Както при декларациите на елементи има 2 метода за деклариране на атрибути:

- създаване на локален тип
- използване на глобален тип

Декларациите на атрибути могат да са единствено прости типове (simple types). Може да се преизползват атрибути чрез реферирането на глобални декларации - <attribute ref="contacts:kind"/>.

<attributeGroup> декларации

Чрез тях могат да се дефинират преизползваеми множества от атрибути. Глобалните <attributeGroup> декларации трябва да са именовани с name атрибут.
<attributeGroup name="ContactsAttributes">
<!-- attribute declarations go here -->
</attributeGroup>

Сравнение с DTD

DTD предоставят фундаменентална граматика за дефиниране на XML документ под формата на метаданни, които описват съдържанието на документа. XML Schema предоставя това плюс детайлен начин за дефиниране на данните, които могат и не могат да се съдържат.
Разликите между XML Schema и DTD са:

- DTD не поддържат namespace за разлика от XML Schema, която пълно поддържа Namespace Recommendation.
- DTD осигуряват възможности за дефиниране на заменим текст, външно съдържание и условна обработка (ENTYTY декларациите). Това не е възможно в XSD.
- DTD описват целия документ, schema-ите могат да дефинират само части от него.
- XSD има дефинирана система от типове и позволява дефинирането на потребителски типове данни, както и наследяване, енкапсулация и субституция.
- XSD е по-богат език за описание на елементи и атрибути. DTD поддържа 10 типа, XSD - 37+.
- DTD може да бъде част от XML документа, докато XSD е задължително в отеделен файл.

- Официалната дефиниция на “валиден XML” изисква DTD.
- XSD може да дефинира много повече ограничения от DTD.
- XSD използва XML синтаксис, DTD има отделен синтаксис. XSD синтаксисът е по-експресивен от този на DTD
- XSD има повече поддържащи средства (DOM, browsers)
- XSD има по-добър модел на съдържанието – напр. Може точно да се задава ограничението за брой срещания на даден елемент
- XSD позволява лесно създаване на сложни и преизползваеми модели на съдържание.
- XSD е съвместима с други XML технологии като Web Services, XQuery, XSLT...
- XSD поддържа DOM и DOM манипулации
- XSD поддържа не само глобални, но и локални имена
- XSD има неограничена възможност за разширяване

Използване на XSLT (eXtensible StyleSheet Language Transformations) и XPath за алокиране, манипулиране и представяне на XML съдържание.

XSLT се използва съвместно с XPath за алокиране, манипулиране и представяне на XML съдържание. XSLT представлява език за трансформация на XML документи в какъвто и да е текстово-базиран формат. Езикът позволява дефинирането на шаблони, коио да се приложат към определена част от XML документ. Към коя част да се приложат се задава чрез XPath изрази.

XSLT спецификацията се разделя на две – XSL Formatting objects (или само XSL) и XSL Transformations. Първото е език за форматиране на XML данни за представянето им на екран, хартия или друг медиен носител. Второто е език за трансформиране на XML документи в други документи.

Асоцииране на XML документ с XSLT Style sheet се извършва чрез инструкция за обработка - *<?xml-stylesheet href="myStyles.xsl" type="text/xsl" ?>*

XPath

XPath – схема за локализиране и идентифициране на подструктури в документи; използва се от други XML технологии като XSLT, XPointer, XQL като средство за локализиране на определени части от XML съдържанието.

XPath модел на данните – при XPath повечето части на един сериализиран XML документ се представят като върхове на дърво. Коренът на дървото е самият XML документ – отбелязва се с ‘/’. За всеки елемент, атрибут, инструкция към процесора или коментар има съответен връх в дървото. Също така има текстови върхове, представящи текстовото съдържание на елементите.

XPath се възползва от последователния и йерархичен характер на XML документите, за да открие елементи според техния контекст (например къде в йерархията се намира даден елемент)

Контекстен възел – селектира се с .; това е текущата секция в XML документа, от където започва XPath изразът

XPath връх – може да бъде всяка част от XML документ – елемент, атрибут, инструкция към процесора или нещо друго

Корен на документа (document root) – не е първият елемент, а самият XML документ; означава се с ‘/’

XPath израз – текстов стринг, с който се избира даден елемент, атрибут, инструкция или текст; може да се намира в URL или като стойност на атрибут (http://abc.com/getQuery?/book/intro/title или <xsl:pattern match="chapter/title">…</xsl:pattern>). Изразите локализират нещата по тяхното място в XML йерархията

4 типа изрази – булеви, множества от върхове, числа, стрингове (това са типовете, които могат да бъдат върнати вследствие на оценката на XPath израза)

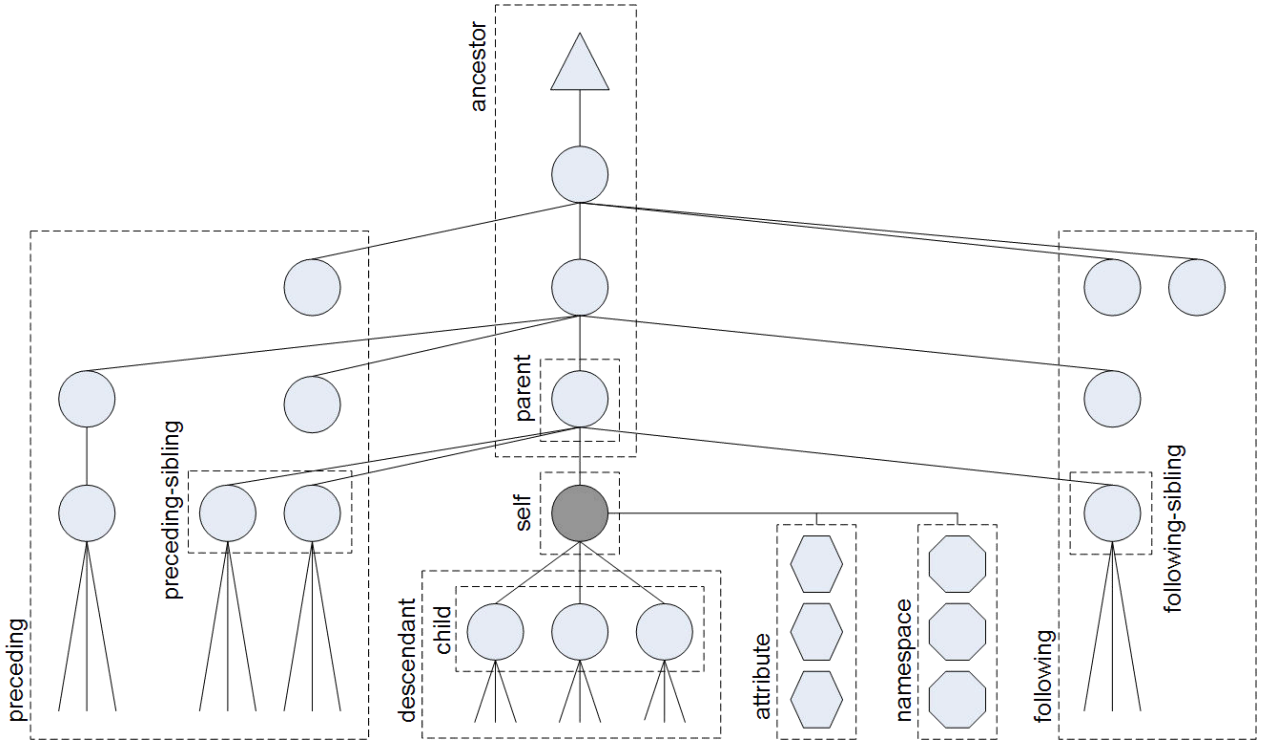
Location path – повечето XPath изрази връщат множество от върхове, тези изрази се наричат location paths. Пътищата, посочващи местонахождението (location paths), са съставени от стъпки. Всяка стъпка е съставена от ос, връх и предикат (опционален). Когато оста липсва, подразбира се child.

- ✓ Relative paths **–** започват от контекстния връх
- ✓ Absolute paths **–** започват от корена

Видове върхове

- ✓ Корен
- ✓ Възел на елемент
- ✓ Възел на атрибут – възелът на елемента, към който е този атрибут, се нарича възел родител. Всеки възел на атрибут има име и стойност.
- ✓ Текстов възел
- ✓ Възел за пространство от имена
- ✓ Възел за коментар
- ✓ Възел за инструкция към процесора

13 XPath оси – използват се така: име_на_оста::име_на_връх. Оста на контекстния връх се нарича self. Той може да се достъпи по два начина: '.' и self::node.



Заб: На картинката липсват descendant-or-self и ancestor-or-self. С тях не се селектират братя и сестри.

Рекурсивен оператор за достъп до всички потомци – '//'. Пример: chapter//para ще селектира всички para елементи, които са по-ниско в йерархията от chapter. Несъкратеният запис е: child::chapter/descendant-or-self::node()/child::para. Пример: За избиране на всички para елементи в текущия – .//para или descendant-or-self::node()/child::para.

С '..' се селектира родителят на контекстния връх, може да се достъпи и с parent оста.

С ‘@’ се селектира стойността на атрибут

Функции

- ✓ node()
- ✓ text()
- ✓ position()
- ✓ last()
- ✓ count()
- ✓ id()
- ✓ concat()

Предикати – задават се с начупени скоби ([]). Предикат може да се сложи след всяка стъпка в XPath израза, като на една стъпка могат да бъдат зададени и няколко предиката на стъпка.

Манипулиране на XML съдържание

Реализира се чрез дефиниране на правилата за трансформация чрез елемент ***<xsl:template>.*** В атрибутът му match се задава XPath израз, който определя върху кои възли ще бъдат приложени шаблоните.

```
<xsl:template match="exon">
        We have found the EXON tag!
</xsl:template>
```

Елементът ***apply-templates*** позволява изливкването на шаблони от текущо изпълнявания такъв. Това предизвиква рекурсивната обработка на всички наследници на обработвания елемент.

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

Извикване на темплейт се реализира чрез ***call-template*** елементът. XSLT позволява и предаването на параметри при извикването - <xsl:with-param name="prefix">new</xsl:with-param> - предава стойност на параметъра при извикването. Дефинирането на параметър става чрез ***param*** елемент.

```
<xsl:template match="title">
        <xsl:call-template name="CreateHeader" />
</xsl:template>
```

Изходните елементи могат да бъдат сортирани чрез ***xsl:sort*** елемент, който извършва сортировка по определено property.

```
<xsl:template match="list">
        <xsl:apply-templates>
                <xsl:sort select="@code"/>
        </xsl:apply-templates>
</xsl:template>
```

Итерация върху XML съдържание

<xsl:for-each> елементът може да селектира и итерира по всеки XML елемент от определено множество възли.

```
<xsl:for-each select="catalog/cd">
```

```
        .....................
</xsl:for-each>
```

Условна обработка на XML съдържание

Има 2 механизма за условна обработка:

- <xsl:if> елемент - съдържа шаблон, който да бъде приложен само ако зададеното условие е истина.

```
<xsl:if test="position() = 1">
        <xsl:attribute name="style">color: red</xsl:attribute>
</xsl:if>
```

- <xsl:choose> елемент заедно с <xsl:when> и <xsl:otherwise> - изразява множество условни тестове.

```
<xsl:choose>
        <xsl:when test="salary[number(.) &gt; 2000]">
                A big number
        </xsl:when>
        <xsl:when test="salary[number(.) &gt; 1000]">
                A medium number
        </xsl:when>
        <xsl:otherwise>A small number
        </xsl:otherwise>
</xsl:choose>
```

Деклариране на променливи

<xsl:variable> - могат да се декларират променливи; за да се достъпи стойност на променлива се ползва '$' (<xsl:value-of select="$colour"/>); също така стойността на променливата може да се ползва и в елементи от резултатното дърво (<ajr:glyph colour="{$colour}"/>). По този начин могат да се добавят прости константи (например да си дефинираме числото пи).

Алокиране на XML съдържание

XSLT позволява и ***създаването на елемент, атрибути и групи от атрибути*** в изходния документ. Това се реализира чрез xsl:element, xsl:attribute и xsl:attribute-set. Алокирането на динамични елементи и атрибути може да се реализира чрез задаването на XPath изрази за стойност или име.

```
<xsl:element namespace="html" name="lala">
<xsl:attribute name="style">purple</xsl:attribute>
```

Копиране на елементи

Има 2 възможности за копиране:

- Плитко – реализира се чрез copy елемент, който копира контекстния възел без наследниците и атрибутите му.

```
<xsl:template match="h1|h2|h3|h4|h5|h6|h7">
    <xsl:copy>
            Header: <xsl:apply-templates/>
    </xsl:copy>
```

</xsl:template>

- Дълбоко – елементът copy-of копира контекстния възел заедно с наследниците и атрибутите му.

  <xsl:copy-of select="//h1 | //h2" />

Извличането на информация от XML дървото се реализира чрез елемента <xsl:value-of>. Той взима информация от обработвания XML и я добавя към резултатното дърво. В атрибута select се указва чрез XPath израз какво да бъде селектирано, като се включват всички наследници. Този елемент свежда селектирания обек до символен низ.

```
<xsl:template match="b">
        <xsl:value-of select="." />
</xsl:template>
```

Представяне на XML съдържание

Тъй като XML данните обикновено не съдържат елементи, които определят как да се презентират данние, е необходимо да се използват други методи за представяне. Един от тях е XSL документи, трансформиращи XML. Представянето на данните се реализира чрез асоцииране на XSL документ към XML документа и обработката на XML съдържанието от XSLT процесора по зададените критерии.

Контрол върху изходния документ се получава чрез <xsl:output>, който позволява задаването на формата, версията, енкодинга и др.

Използване на DOM (Document Object Model) и SAX (Simple API for XML) за обработка на XML документи

Основни интерфейси на DOM и SAX и начини за използването им

Има три основни стъпки при използването на XML парсер:

1. Създаване на обект от типа на парсера
2. Подаване на XML документа към парсера
3. Обработка на резултатите

В общия случай генерирането на XML съдържание е извън възможностите на парсерите, но има и такива, които могат да поддържат и такава функционалност.

Двата главни програмни интерфейса (APIs) за парсване на XML са SAX и DOM.

DOM интерфейси

DOM дефинира няколко следните Java интерфейси:

- ***DOMImplementation*** - Интерфейс за създаване на началния Document възел на DOM дървото.
- ***Node Interface*** - Представя основния обект в DOM - единичен възел в DOM дървото. Той е базовият интерфейс за всички други в DOM и осигурява достъп до общи пропъртита на възли, методи за обхождане и модификация на дървото.
- ***Document::Node Interface*** - Представлява целия XML документ (корена на дървото). Освен това се използва за factory за други типове възли - елементи, текстови възли, коментари, CDATA секции, processing инструкции, атрибути, entity референции. Има методи за достъп до DOCTYPE, до информация за възможностите на DOM имплементацията, до корена на дървото и пр.

- ***DocumentType::Node interface*** - Предоставя достъп до Entity и Notation колекциите в документа, както и до някой аспекти на външните и вътрешни DTD.
- ***Element::Node Interface*** - Съдържа 2 категории методи:
  - общи методи за елементи - за получаване на името на таг, на елементите по таг, нормализиране
  - методи за манипулиране на атрибути - за взимане и промяна на атрибут, премахване на такъв и пр.
- ***Attr::Node Interface*** - моделира атрибутите в XML документ, осигурявайки достъп до различни properties на атрибута. Поддържа методи за достъп до име и стойност и промяната им. Въпреки че extend-ва Node интерфейсът, атрибутите не се считат за част от DOM дървото.
- ***Entity::Node Interface*** - представя вътрешно или външно entity в XML документ.
- ***DocumentFragment::Node Interface*** - е signature interface. Той съхранява части от Document, когато се добави към друг Node се премахва. Не е нобходимо да се съобразява с правилата за XML.

SAX Интерфейси

SAX API са разделени на следните 4 области:

- ***core interfaces*** - облекчават работата с core информацията в XML документ
- ***core classes*** - облекчават работата с core информацията в XML документ
- ***extended interfaces*** - моделират аспекти на документа, които не представляват интерес за повечето програмисти - DTD декларации, коментари и пр.
- ***helper classes*** - съдържат няколко класа за удобство, както и имплементациите по подразбиране на някои от core интерфейсите.

Attributes - моделира атрибутите на елементи. Те се предоставят чрез ненаредено множество от свойства, което може да бъде обхождано по име и позиция.

ContentHandler - основен SAX интерфейс. Той моделира core информацията за XML документ като подредена последователност от извиквания на методи и предоставя интерфейс за получаване на основни markup събития от парсера.

DTDHandler - предоставя callback методи за получаване на нотификации за DTD събития.

EntityResolver - интерфейс, позволяващ на имплементациите да осигуряват custom резолюция на външни entities.

ErrorHandler - моделира well-formed грешки, грешки при валидация и warrinig-и. Консуматорът на ContentHandler имплементацията използва този интерфейс да преустанови потокът от извиквания на методи, които довеждат до грешки. ErrorHandler дефинира 3 нива exceptions - error, fatalError, warning.

Locator - парсерът може да предостави Locator обект, който да бъде използван в event методите, за да се разбере къде точно в документа е текущия метод.

XMLFilter - подинтерфейс на XMLReader. Повечето SAX интерфейси позволяват pipeline обработка, където имплементацията на ContentHandler може да получава някой информационни елементи, които разпознава, но да предава неразпознатите елементи на следващият процесор във веригата. Това се реализира чрез XMLFilter интерфейс. Той работи върху събия вместо върху документа и се регистрира при предишния филтър

XMLReader - представлява интерфейс за четене на XML документ чрез използването на callback механизъм. Позволява на приложенията да задават и получават свойства в парсера, да се регистрират event handler-и за обработка на документ и да се инициира парсването на документа.

Сравнение между DOM и SAX

DOM създава дърво за целия документ в паметта, докато SAX парсва документа възел по възел. DOM е

подходящ, в случай че е необходимо да се обработва голяма част от елементите в документа и да се извършват операции върху XML. Подходящ е и при необхдимост за множество достъпи до XML. SAX е подходящ при големи XML документи, в които за обработката са необходими малко елементи, идеален е за извършването на прости операции върху XML файлове. SAX е разработен с цел да предостави по-ефективен анализ на големи документи. Той е подходящ за обработка по време на парсването на документа, тъй като единствено последния event е в паметта. Обработката със SAX изисква по-малко място и време, тъй като не се построява карта на целия документ в паметта, и е подходящ за сложни структури. Тъй като SAX е създаден единствено за четене на XML, но не и за писане при необходимост от това се използва DOM.
DOM е подходящ за приложения, който изискват структурни манипулации върху много XML tokens, а SAX - за приложения с ограничения в паметта.