<XML/>

<Ивана_Димитрова />

Съдържание

- Какво е XML?
- Основни градивни единици
- Namespaces
- XML парсери - DOM / SAX
- XPath и XSLT
- XML и .NET Framework
- Схеми и валидация

Какво е XML?

- Extensible Markup Language
- XML е технология за описване и структуриране на данни
- Метаезик без семантика и фиксирано множество тагове
- Световно утвърден ISO стандарт
- Платформено-независим

В началото бе SGML...

- Standard Generalized Markup Language – един от първите опити да се комбинира универсален формат с богати възможности
- Оказва се твърде сложен
- Най-известните му наследници са XML и HTML

XML и HTML : ябълки и червени вкусни ябълки

- Подобни на външен вид, но много различни

Прилики	Разлики
• Текстово-базирани • Използват тагове и атрибути • Произлизат от SGML	• Различно предназначение • HTML – специализирано приложение на SGML, а XML – функционално подмножество • Елементите и атрибутите в HTML са строго определени, а в XML – не • HTML описва как да представи информация, а XML – самата информация

А как изглежда?

```
<?xml version="1.0" encoding="utf-8" ?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>
      An in-depth look at creating applications
      with XML.
    </description>
  </book>
</catalog>
```

Кой го разбира??? (XML парсерите)

- XML парсер – библиотека, която чете XML фрагмента и извлича таговете и информацията в тях
- Предоставя данните на програмата ни
- Стандартизиран интерфейс за създаване и промяна на XML документи
- MSXML – вграден в Internet Explorer

Градивни елементи

- Тагове
  - Думи между “<” и “>”
  - Отварящ: <element>
  - Затварящ: </element>
  - <element>content</element>

    съдържание
  - Съкратен синтаксис: <element />

Градивни елементи (2)

- XML декларация
  - Обозначава XML документа като XML
  - Носи информация за парсера

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

  - Задължително указва версията на спецификацията, която поддържа
  - В началото на файла

Градивни елементи (3)

- Елементи
  - Root element
  - Произволно ниво на вграждане
  - Имената им са case-sensitive!
  - Не разчитайте на пълен/съкратен синтаксис – парсерът може да избира!
  - 4 типа: празни, съдържащи само текст, съдържащи само поделементи, съдържащи и текст, и поделементи

Градивни елементи (4)

```
<elements>
    <emptyOne></emptyOne>
    <emptyTwo />
    <textOnly>This is the inner text</textOnly>
    <childrenOnly>
        <child>Some inner text</child>
    </childrenOnly>
    <mixedElement>This element contains both text and child element
        <childElement>This element contains text
        </childElement>
    </mixedElement>
</elements>
```

Градивни елементи (5)

- Атрибути
  - Двойки име-стойност, асоциирани с елемент и даващи информация за него
  - Стойностите – в кавички (‘ или “)
  - Трябва да са с уникални имена

```
<person nickName=“John’s nick name” id=‘12345’>
        <firstName>John</firstName>
        <lastName>Doe</lastName>
</person>
```

Градивни елементи (6)

- Коментари

"Не забравяйте... този, който ще дойде след вас може да сте самите вие!"

- Започват с "<!--", завършват с "-->"
- Не могат да се слагат в таг
- Низът "--" не се използва в коментара
- Не разчитайте, че парсерът ще ги подаде на приложението ви!

```
<!-- Here is a place for my comment -->
```

Градивни елементи (7)

- Инструкции за обработка (processing instructions – PI)
  - Вградено в информацията указание за приложението как да обработи данните
  - Започват с “<?”, завършват с “?>”
  - Указва се PITarget – име на приложението и инструкции за него
  - XML декларацията не е PI!

```
<?xml-stylesheet type=“text/xsl” href=“style.xsl” ?>
```

Добре дефинирани документи

- Well formed documents
- Правила от спецификацията:
  - Всеки отварящ таг трябва да има съответен затварящ
  - Таговете не могат да се припокриват
  - Има само един главен/основен елемент
  - Имената на елементите и атрибутите трябва да отговарят на правилата

Правила за именоване

- Няма запазени думи
- Името започва с буква(вкл. не-латинска) или с “_”, но не и с цифра или друг пукнтуационен знак
- След първата буква може да има числа, “-” или ”.”

Правила за именоване(2)

- Не може да има “:” – спецификацията го определя като запазен за namespaces
- Имената не започват с “xml” и разновидности
- Не може да има интервал след “<“, но може да има преди “>”

Специални символи

```
<comparison> 6 < 7 & 7 > 6 </comparison>
```

- Как се разрешава:
  - Промяна интерпретацията на символите:

Символ	Заменя се с
<	&lt;
>	&gt;
&	&amp;
‘	&apos;
“	&quot;

```
<comparison> 6 &lt; 7 &amp; 7 &gt; 6 </comparison>
```

Специални символи (2)

- Знаците могат да се заменят със знакови референции:

  Unicode номер: &#nnn

  16-тичен Unicode: &#xnnn

  – Независимо от кодировката, знаците с променена интерпретация са Unicode!
  – Пример:

  © : &#169

  &#XA9

Специални символи (3)

- CDATA секции (character data)
  - Указват на парсера да не обработва текста до края на секцията
  - По-четим вид
  - Можем да вградим в тях цели скриптове и функции (JavaScript)
  - CDATA секции не могат да се влагат
  - Започват с “![CDATA[“ и завършват с “]]”

Специални символи (4)

- CDATA секция:

```
<script>
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
  return 1;
else
  return 0;
}
]]>
</script>
```

CDATA или PCDATA?

- PCDATA (parsed character data)
  - Нормално, всичко в XML документа се парсва, тъй като вътре в един елемент може да има друг
  - PCDATA е текстът, който нормално се парсва
  - CDATA е “обратното”

Споменахме namespaces

- Поради същността на XML всеки може да описва света със свои термини
- С пространства от имена можем да разграничаваме елементи с еднакви имена и да разделяме на логически групи
- По-лесно разпознаване от парсера

Namespaces (2)

- Дефинират се за елементи и нямат отношение към атрибутите
- Qualified name :

  име на пространство(префикс)+локално име
- Могат да се дефинират пространства с различни префикси и такива по подразбиране
- Идентификаторът не се анализира от парсера

Namespaces (3)

- Пространството от имена има идентификатор URI (Uniform Resource Identificator) – низ от знакове, идентифициращ ресурс (URL / URN)
  - URL (Uniform Resource Locator)

```
http://dotnet.graduate-bg.com/news.aspx
mailto:someone@example.com
```

  - URN (Uniform Resource Name)

```
urn:Canadian-Citizen:123-456-789
```

Namespaces (4)

- Дефиниране: **xmlns:<prefix>="<URI>"**
  - Namespace с префикс “xsi”:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

  - Namespace по подразбиране:

```
xmlns="http://schemas.microsoft.com/office/infopath"
```

  - Отмяна на namespace по подразбиране:

```
xmlns=""
```

  - Дефинират се в началния или в друг произволен елемент

Namespaces - пример

```
<?xml version="1.0" encoding="utf-8" ?>
<faculty:student
    xmlns:faculty="urn:fmi"
    xmlns="urn:foo“
    id="235329">
        <name>Ivan Ivanov</name>
        <language xmlns="">C#</language>
        <rating>6.00</rating>
</faculty:student>
```

Namespaces - пример

Пълни имена на елементите:

– urn:fmi:student
– urn:foo:name
– urn:foo:rating
– language

DOM и SAX, или още нещо за парсерите

- Помните какво е парсер нали?
- А как обработва той документа:
  - като го моделира като дървовидна структура

  или
  - като го разглежда като поток от данни

DOM

- Document Object Model –програмен интерфейс за достъп и манипулация съдържанието на XML документ
- Описва интерфейси, а не конкретни класове – за работа с него ни трябва DOM парсер

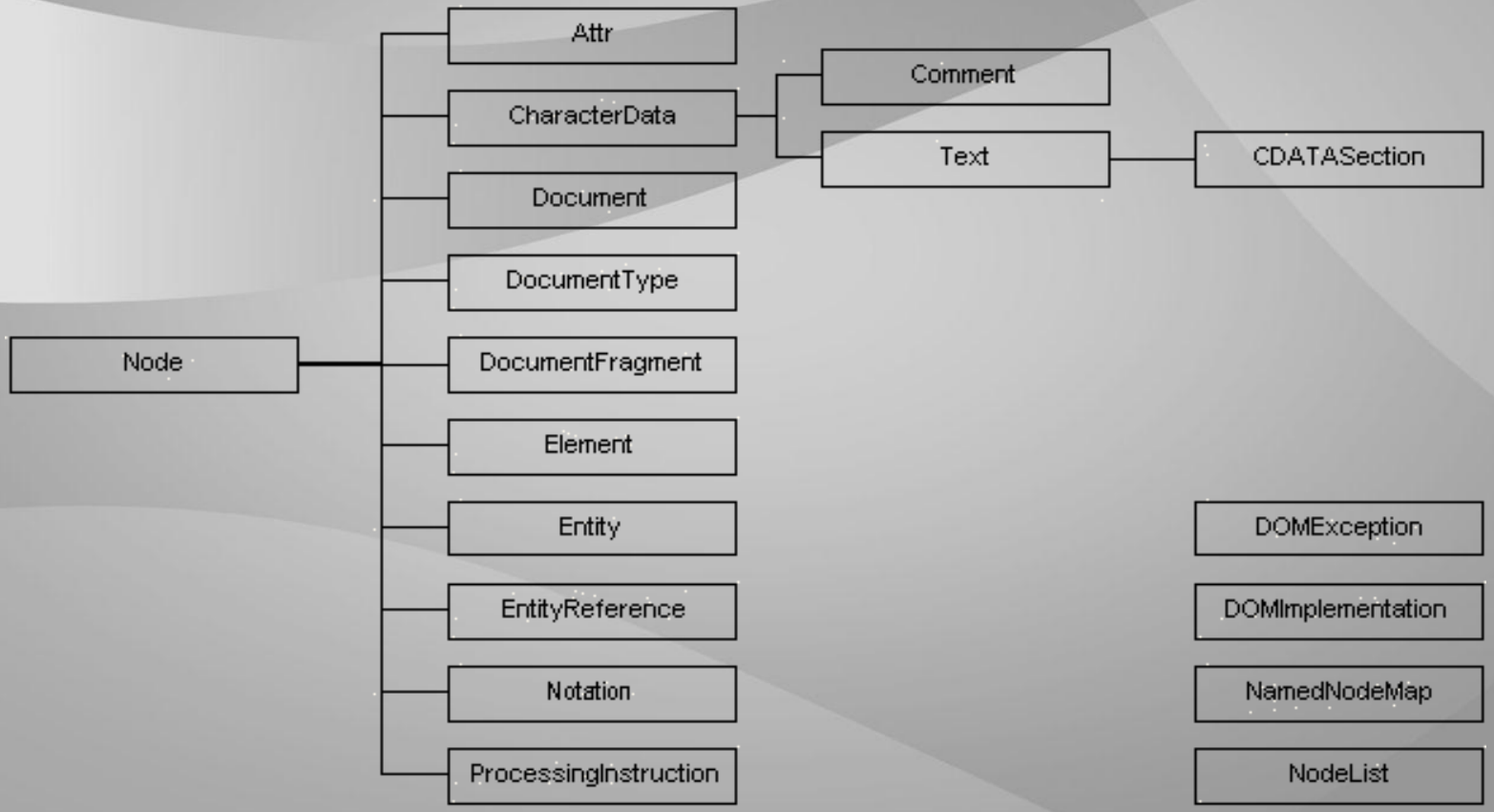
DOM (2)

- Представя XML документа като дърво, като зарежда целия в паметта преди да го подаде на приложението

- Позволява произволен достъп, навигиране и промяна на елементи и атрибути

DOM (3)

- Типове възли:



SAX

- Simple API for XML Processing
- Базиран на събития програмен интерфейс

  (Срещане на различни компоненти напр.)
- Чете последователно данни от поток
- Обработката на вече прочетени данни не чака целия документ

Хмм...DOM или SAX?

- Използвайте DOM за:
  - По-малки по обем документи
  - За произволен достъп
  - Ако ще променяте данните
- Използвайте SAX за:
  - Големи документи
  - Потоков достъп до данните
  - Read-only

И все пак...къде се използва?

XPath

- Технология за навигиране и селектиране на части от XML документи
- Утвърден от W3C език за адресиране на възли от документа
- Не е XML-базиран
- За адресиране се използват XPath изрази, подобни на пътищата във файловата система

XPath изрази

- Описания на пътища до възли и критерии, на които тези възли трябва да отговарят
- Контекст на израза – контекстен възел или множество от възли
  - Началната точка се определя от приложението
  - На всяка стъпка приема ст-та на текущия

XPath стъпка

- XPath пътят се състои от една или повече локализационни стъпки
  - Разделят се с “/”
  - Компоненти: [ос], тест на възли, [предикат]
  - Само тестът на възли е задължителен

XPath стъпка -Ос

- Роля – подобна на drive letter-а във файловата система – отправна точка за навигация
- Възможни оси:
  - Self (.)
  - Child / Parent (..) / Ancestor
  - Attribute (@)
  - Following / Preceding-Sibling
  - Descendant-or-self (//)

XPath стъпка - Тест

- Позволява проверка за изпълнение на условие от даден възел
- За селектиране на всички възли : *
- Възможни са тестове на:
  - Всички възли с дадено име
  - Всички processing instructions
  - Всички текстови елементи под дадена ос

XPath стъпка - Предикат

- Булев израз за допълнително филтриране на текущото множество възли
- На една стъпка може да има повече от един предикат
- Предикатът може да използва функции за стрингове (startswith(..), concat(..)), логически (not(), true()) и др.

XPath стъпка - Пример

- Общ синтаксис:

```
ос::тест-на-възли[предикат]
```

- Пример:

```
following::employee[@employeeid='2']
```

(наследници на текущия: employee и атрибут employeeid със ст-т 2)

```
/books/book[price<'10']/author
```

(всички /books/book/author, чиято цена на книгата е по-малка от 10)

XPath – Пример (2)

```
/
                                        <!--root tag -->
/root_tag
            <!--root tag only if named “root_tag”-->
//element_a
            <!—all element_a tags wherever in doc-->
text()
                <!—text contents of current node -->
@name
                 <!—name atribute of current node-->
..
                    <!—parent of the current node-->
```

XPath – Пример (3)

```
/doc/chapter[5]/section[2]
                <!–2nd section of 5th chapter of doc-->
body/p[last()]
                  <!–the last “p” tag in “body” tag-->

/html/body/p[@class=“a”]
       <!–all “p” tags in the body under the html tag
         that have a class attribute with value “a”-->

//p[@class and @style]
                <!–every “p” tag in doc that has both
                          class and style attribute-->
```

XSLT

(или как XML става достъпен за всеки)

- Extensible Stylesheet Language Transformation – език, позволяващ трансформация на XML документ от едно представяне в друго
- XML става HTML, PDF, обикновен текстов файл или каквото друго е удобно само с помощта на XML-базиран шаблон

XSLT шаблонът

- С какво е по-добър от CSS?
  - Може да преподреди данните
  - Може да представи калкулирани данни
  - Може да комбинира няколко документа
  - Може да използва условна логика
- XSLT използва XPath заявки за избиране на възли, върху които да се извършат операциите

XSLT конструкциите

- Специални тагове
- Примери:
  - Замества частта от док. с тялото на конструкцията

```
<xsl:template match="XPath-израз">…</xsl:template>
```

  - Извлича стойността

```
<xsl:value-of select="XPath-израз" />
```

  - Замества всеки намерен израз от този тип

```
<xsl:for-each select="XPath-израз">…</xsl:for-each>
```

  - При положителен резултат замества

```
<xsl:if test="XPath-израз">…</xsl:if>
```

XSLT - пример

```
<xsl:stylesheet version="1.0"  xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
<xsl:template match="/">
   <html>
    <body>
          <!– define html elements here -->
          <xsl:for-each select="/catalog/cd">
        <tr><td><xsl:value-of select="title"/></td>
          <td>
          <xsl:choose>
            <xsl:when test="price &gt; 20">
              <xsl:value-of select="price * (1.0 - $discount)"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="price"/>
            </xsl:otherwise>
          </xsl:choose>
           </td></tr>
```

XML и .NET Framework

- Силно интегрирана XML поддръжка, а не просто добавка
- .NET Framework Base Class Library включва няколко основни XML класа
- Поддържа DOM парсване и дава функционалност за SAX-базирано такова

XML и DOM в .NET

- Пълен набор от класове за работа с XML според DOM W3C спецификацията
- Документът първо се зарежда целия в обект от тип XmlDocument в паметта и след това се обработва
- Дадени са средства за навигация и трансформации

Класове за работа с DOM

- XmlNode – абстрактен базов клас
- XmlDocument - корена на DOM дървото
  - Заглавна част
  - Документен елемент
- XmlElement
- XmlAttribute
- XmlNodeList
- XmlAttributeCollection

Класът XmlNode

- Абстрактен
- Представя възел в XML документа
- Наследява се от всички специализирани класове за DOM възли (XmlElement, XmlAttribute, XmlDocument)
- Позволява навигация в дървото
- Има достъп до характеристиките на всеки възел (тип, родител, съседи,...)

Класът XmlNode (2)

- Properties:
  - ParentNode – възел-родител или null
  - PreviousSibling / NextSibling
  - FirstChild / LastChild
  - Item (индексатор) – наследник по име
  - Attributes
  - InnerXml / OuterXml / InnerText
  - NodeType
  - Name / Value

Класът XmlNode (3)

- Methods:
  - AppendChild(…) / PrependChild(…)
  - InsertBefore(…) / InsertAfter(…)
  - RemoveChild(…) / ReplaceChild(…)
  - RemoveAll()

Класът XmlDocument

- Наследява абстрактния XmlNode
- Дава методи за зареждане на документа в паметта и запазване на модификациите
- Основен за работа с XML данни
- Допълва методите и свойствата на XmlNode

XML и SAX в .NET

- Няма пълна чиста имплементация на SAX парсер
- Класът XmlReader
  - Подобна функционалност
  - Абстрактен
  - Дава бърз еднопосочен достъп замо за четене до XML данни в поток
  - Базиран на събития, извличани от парсера
  - За оптимизация : XmlNameTable

Класът XmlReader

- Properties:
  - NodeType
  - Name
  - Value
  - HasValue
  - AttributeCount
  - IsEmptyElement
  - Prefix

Класът XmlReader (2)

- Methods:
  - Read() – чете следващ възел или връща false ако няма; чрез него XmlReader се информира за настъпили събития
  - ReadElementString()
  - GetAttribute(…)
  - MoveToAttribute() / MoveToNextAttribute()
  - MoveToElement()

Класът XmlWriter

- Бърз, еднопосочен, поточно-ориентиран способ за запис на XML данни
- Генерира добре дефинирани документи само при коректни данни
- Не проверява за:
  - Дублирани имена на атрибути
  - Невалидни символи в имената
- Не валидира по схема

Класът XmlWriter (2)

- Methods:
  - WriteStartDocument()
  - WriteStartElement(…)
  - WriteEndElement()
  - WriteAttributeString(…)
  - WriteElementString(…)
  - WriteEndDocument()

Класът XmlTextWriter

- Единствен наследник на абстрактния XmlWriter
- Поддържа запис на XML данни в поток, файл или TextWriter
- В конструктора му се задава кодираща схема
  - По подразбиране използва UTF-8

Схеми и валидация

- Формално описание на формата на XML документ
- Гарантира, че валиден документ (проверен срещу схемата) изпълнява определени изисквания
- Дефинира допустими:
  - тагове и атрибути за тях
  - стойности на елементи и атрибути
  - ред на елементите

DTD схеми

- Document Type Definition (DTD) – от XML спецификацията
- Текстово-базиран, разработен преди появата на XML, с SGML синтаксис
- Дефинира модел на съдържание (content model) за всеки елемент – данни, наредба, брой, задължителност
- Декларира множество позволени атрибути за всеки елемент

DTD схеми (2)

- Свързва се с XML документа със специална конструкция, разположена веднага след декларацията и преди документния елемент

```
<DOCTYPE documentElementName source location >
```

- Пример:

```
<DOCTYPE restaurantMenu PUBLIC
                "file:///C:\Schemas\DTDSchema.dtd" >
```

DTD схеми - пример

```
<!ELEMENT restaurantMenu (breakfastMenu, lunchMenu,
    dinnerMenu)>
<!ELEMENT breakfastMenu (foodItem*, drinkItem*)>
<!ELEMENT foodItem (name, price, description, calories)>
<!ATTLIST foodItem id #PCDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT calories (#PCDATA)>
<!ELEMENT drinkItem (name, price, description)>
<!ATTLIST drinkItem id #PCDATA #REQUIRED>
<!ELEMENT lunchMenu #PCDATA>
<!ELEMENT dinnerMenu #PCDATA>
```

XSD схеми

- XML Schema Definition Language
- По-нов и мощен от DTD
- XML-базиран
- Осигурява силно типизирана система за XML обработка
- За разлика от DTD, поддържа пространства от имена!

XSD схеми (2)

- Вградени типове данни – описани в:

```
http://www.w3c.org/2001/XMLSchema
```

- Потребителски типове
  - Прости типове (xs:simpleType) – не задават структура, а само стойностно поле; само за текстови елементи без наследници и атрибути
  - Комплексни типове (xs:complexType) – за елементи с допълнителна структура

XSD схеми - пример

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="restaurantMenu">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="breakfastMenu">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="foodItem">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="name" type="xs:string" />
                    <xs:element name="price" type="xs:decimal" />
                    <xs:element name="description" type="xs:string" />
                    <xs:element name="calories" type="xs:unsignedShort" />
                  </xs:sequence>
                  <xs:attribute name="id" type="xs:unsignedByte" use="required" />
                </xs:complexType>
              </xs:element>
….
```

XDR схеми

- XML-Data Reduced
- XML-базиран език
- Въведен от Microsoft преди стандартизираните от W3C XSD схеми
- Могат да описват съответствия между XML структура и релационни бази данни
- Поддържат типове от данни и namespaces

<Въпроси? />

Ресурси

- XML Tutorial:

http://www.w3schools.com/xml/default.asp

- W3C XML page:

http://www.w3.org/XML/

- MSDN XML Page & Technical Articles:

http://msdn.microsoft.com/en-us/data/bb190600.aspx

http://msdn.microsoft.com/bg-bg/library/aa286548(en-us).aspx

- MSDN System.Xml namespace reference:

http://msdn.microsoft.com/en-us/library/system.xml.aspx