

XML и основные технологии .NET Framework XML

БГУ, ММФ, кафедра веб-технологий и компьютерного
моделирования

Автор: Кравчук Анжелика Ивановна

Что такое XML

XML (eXtensible Markup Language, расширяемый язык разметки) – это способ описания структурированных данных, т.е. данных, которые обладают заданным набором семантических атрибутов и допускают иерархическое описание

XML-данные содержатся в *документе*, в роли которого может выступать файл, поток или другое хранилище информации, способное поддерживать текстовый формат.

XML это язык разметки, описывающий целый класс объектов данных, называемых XML-документами

Язык XML разработан рабочей группой XML консорциума World Wide Web Consortium (Working Group XML W3C) (1998)

Язык XML используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов, не содержит никаких тэгов, предназначенных для разметки, а просто определяет порядок их создания

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

Что такое XML

```
<flower>rose</flower>
```

```
<conservatory>  
    <flower>rose</flower>  
</conservatory>
```

```
<conservatory>  
    <flower>rose</flower>  
    <flower>tulip</flower>  
    <flower>cactus</flower>  
</conservatory>
```

Преимущества XML

Адаптация. XML вездесущ. Многие компании используют XML для хранения данных или намерены это делать. Всякий раз, когда возникает необходимость в разделении одних и тех же данных между приложениями, XML автоматически становится первым (и часто единственным) кандидатом на рассмотрение

Расширяемость и гибкость. XML не накладывает никаких ограничений на семантику данных. В результате XML подходит для любого типа данных, при этом он дешевле в реализации

Стандартизация и инструментарий. Широкий выбор инструментов (подобных анализаторам) и сопутствующие стандарты (такие как XML Schema, XPath и XSLT), помогающие в создании и обработке XML- документов. В результате программисты, работающие почти на любых языках, имеют в своем распоряжении готовые компоненты для чтения XML, проверки его соответствия наборам правил (называемым схемами), поиска в XML, а также трансформации одного формата XML в другой

Преимущества XML

XPath (XML Path Language) — язык запросов к элементам XML-документа. Разработан для организации доступа к частям документа XML в файлах трансформации XSLT и является стандартом консорциума W3C. XPath призван реализовать навигацию по DOM в XML

XQuery — язык запросов, разработанный для обработки данных в формате XML. XQuery использует XML как свою модель данных.

XSLT (*eXtensible Stylesheet Language Transformations*) — язык преобразования XML-документов. Спецификация XSLT входит в состав XSL и является рекомендацией W3C. При применении *таблицы стилей* XSLT, состоящей из набора *шаблонов*, к XML-документ (*исходное дерево*) преобразуется в *конечное дерево*, которое может быть сериализовано в виде XML-документа, XHTML-документа (только для XSLT 2.0), HTML-документа или простого текстового файла. Правила выбора (и, отчасти, преобразования) данных из исходного дерева пишутся на языке запросов XPath

Преимущества XML

XHTML (*Extensible Hypertext Markup Language* — расширяемый язык разметки гипертекста) — семейство языков разметки веб-страниц на основе XML, повторяющих и расширяющих возможности HTML 4

WSDL (*Web Services Description Language*) — язык описания веб-сервисов и доступа к ним, основанный на языке XML.

SVG (*Scalable Vector Graphics* — масштабируемая векторная графика) — язык разметки масштабируемой векторной графики, созданный Консорциумом Всемирной паутины (W3C) и входящий в подмножество расширяемого языка разметки XML, предназначен для описания двумерной векторной и смешанной векторно/растровой графики в формате XML

Как выглядит XML-документ?

```
<?xml version="1.0" encoding="UTF-8"?>
- <list_of_items>
  - <item id="1">
    <first/>
    Первый
  </item>
  - <item id="2">
    Второй
    <sub_item>подпункт 1</sub_item>
  </item>
  <item id="3">Третий</item>
  - <item id="4">
    <last/>
    Последний
  </item>
</list_of_items>
```

Простейший XML- документ

Тело документа XML состоит из элементов разметки (markup) и непосредственно содержимого документа - данных (content)

XML - тэги предназначены для определения элементов документа, их атрибутов и других конструкций языка

Конструкции языка XML

Содержимое XML- документа представляет собой набор элементов, секций CDATA, директив анализатора, комментариев, спецсимволов, текстовых данных

Единица информации – XML-элемент

Содержимое XML-элемента

`<period units="days">27.321582</period>`

Имя XML-элемента

`<empty/>`

Пустой элемент

Иерархия элементов

В качестве содержимого элементов могут выступать как просто какой-то текст, так и другие, вложенные, элементы документа, секции CDATA, инструкции по обработке, комментарии, т.е. практически любые части XML- документа

```
<planet>  
  <name>Mercury</name>  
</planet>
```

Набором всех элементов, содержащихся в документе, задается его структура и определяются все иерархические соотношения. Плоская модель данных превращается с использованием элементов в сложную иерархическую систему со множеством возможных связей между элементами

Конструкции языка XML

Корневой элемент

В XML-документе всегда должен быть единственный элемент, называемый *корневым*, с него программы-анализаторы начинают просмотр документа

```
- <planets>  
  - <planet>  
    <name>Mercury</name>  
  </planet>  
  - <planet>  
    <name>Venus</name>  
  </planet>  
  + <planet>  
  - <planet>  
    <name>Mars</name>  
    + <moon>  
    + <moon>  
  </planet>  
</planets>
```

Синтаксис имен элемента

- чувствительны к регистру
- могут содержать буквы, цифры, дефисы (-), символы подчеркивания (_), двоеточия (:) и точки (.)
- должны начинаться только с буквы или символа подчеркивания
- имена, начинающиеся с xml (вне зависимости от регистра), зарезервированы для нужд XML

```
- <planet>
  <name>Earth</name>
  - <moon>
    <name>Moon</name>
    <period units="days">27.321582</period>
  </moon>
</planet>
```

Специальные символы

Для того, чтобы включить в документ символ, используемый для определения каких-либо конструкций языка (например, символ угловой скобки) и не вызвать при этом ошибок в процессе разбора такого документа, нужно использовать его специальный символьный либо числовой идентификатор

`&` - символ `&`

`<` - символ `<`

`>` - символ `>`

`"` - символ `"`

`'` - символ `'`

`&#int;` - Unicode-символ с десятичным кодом *int*

`&#xhex;` - Unicode-символ с шестнадцатеричным кодом *hex*

Конструкции языка XML

Атрибуты элемента - параметры, уточняющие его характеристики

Любой XML-элемент может содержать один или несколько *атрибутов*, записываемых в открывающем теге

```
<elements-with-attributes>  
  <one attr = "value"/>  
  <several first = "1" second = "2" third = "3"/>  
  <apos_quote case1 = "John's" case2 = 'He said:"Hello, world!"' />  
</elements-with-attributes>
```

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

Особые части XML-документа

Кроме элементов, XML-документ может содержать

- XML-декларацию
- комментарии
- инструкции по обработке (директивы анализатора)
- секции CDATA

Инструкции по обработке (директивы анализатора)

Инструкции, предназначенные для анализаторов языка, описываются в XML документе при помощи специальных тэгов - `<? и ?>`. Программа клиента использует эти инструкции для управления процессом разбора документа

Конструкции языка XML

Любой XML- документ должен всегда начинаться с инструкции `<?xml?>`, внутри которой также можно задавать номер версии языка, номер кодовой страницы и другие параметры, необходимые программе-анализатору в процессе разбора документа

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
```

используемая
версия XML

кодировка XML-
документа

наличие внешних
зависимостей

Комментарии

Комментарии размещаются в любом месте документа и записываются в обрамлении `<!--` и `-->`

`<!--` Комментарии пропускаются анализатором и поэтому при разборе структуры документа в качестве значащей информации не рассматриваются `-->`

Секция CDATA

Чтобы задать область документа, которую при разборе анализатор будет рассматривать как простой текст, игнорируя любые инструкции и специальные символы, но, в отличие от комментариев, иметь возможность использовать их в приложении, необходимо использовать тэги `<![CDATA[` и `]]>`. Внутри этого блока можно помещать любую информацию, которая может понадобится программе- клиенту для выполнения каких-либо действий (в область CDATA, можно помещать, например, инструкции JavaScript).

Секция **CDATA** начинается со строки `<![CDATA[` и заканчивается строкой `]]>`

```
<example>  
  <![CDATA[ <aaa>bb&cc<<<]]>  
</example>
```

Конструкции языка XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- первые четыре планеты -->
- <planets>
  - <planet>
    <name>Mercury</name>
  </planet>
  - <planet>
    <name>Venus</name>
  </planet>
  - <planet>
    <name>Earth</name>
    - <moon>
      <name>Moon</name>
      <period units="days">27.321582</period>
    </moon>
  </planet>
  - <planet>
    <name>Mars</name>
    - <moon>
      <name>Phobos</name>
      <period units="days">0.318</period>
    </moon>
    - <moon>
      <name>Deimos</name>
      <period units="days">1.26244</period>
    </moon>
  </planet>
</planets>
```

Правильно построенный документ (well-formed document) - XML-документ оформленный по описанным выше правилам

Правила создания XML- документа

В общем случае XML- документы должны удовлетворять следующим требованиям:

- В заголовке документа помещается объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация
- Каждый открывающий тэг, определяющий некоторую область данных в документе обязательно должен иметь своего закрывающего "напарника", т.е., в отличие от HTML, нельзя опускать закрывающие тэги
- В XML учитывается регистр символов
- Все значения атрибутов, используемых в определении тэгов, должны быть заключены в кавычки
- Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов
- Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные и поэтому учитываются все символы форматирования (т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML)

Если XML- документ не нарушает приведенные правила, то он называется **формально-правильным (well-formed)** и все анализаторы, предназначенные для разбора XML- документов, смогут работать с ним корректно

Правила создания XML- документа

Процесс обработки XML-документа состоит в следующем:

- Текст XML-документа анализируется специальной программой, которая называется **XML-процессором** (анализатором, «парсером»), который ничего не знает о семантике данных в документе, он только производит синтаксический разбор текста документа и проверяет его правильность с точки зрения правил XML
- Если документ *правильно оформлен* (well-formed), то результаты разбора текста передаются XML-процессором прикладной программе, которая выполняет их содержательную (содержательную) обработку; если же документ оформлен неверно, т. е. содержит синтаксические ошибки, то XML-процессор должен сообщить о них пользователю

Пространства имен XML

Для устранения неоднозначности и обеспечения семантической уникальности элемента предназначены пространства имен XML

```
<!-- фрагмент документа с планетами -->
<planet>
  <name xmlns="http://astronomy.com/planet" >Earth</name>
  <moon>
    <name xmlns="http://astronomy.com/moon" >Moon</name>
    <period units="days">27.321582</period>
  </moon>
</planet>
```

Специальный атрибут (атрибут имени) для связывания элемента с пространством имен

Для идентификатора пространства имен используют унифицированный идентификатор ресурса (Uniform Resource Identifier, URI), чтобы уменьшить риск совпадения идентификаторов в разных документах

Пространство имен, заданное у элемента атрибутом xmlns, автоматически распространяется на все дочерние элементы

Пространства имен XML

При задании пространства имен можно определить **префикс**, который затем записывается перед именем требуемых элементов через двоеточие

Атрибуты также могут быть связаны с пространствами имен при помощи префиксов

```
<!-- пространства имен обычно определяют в начале документа -->
<planets xmlns="http://astronomy.com/planet"
         xmlns:m="http://astronomy.com/moon">
  <planet>
    <name>Earth</name>
    <m:moon>
      <m:name>Moon</m:name>
      <m:period m:units="days">27.321582</m:period>
    </m:moon>
  </planet>
</planets>
```

Просмотр XML - документов

В общем случае, программы - анализаторы можно разделить на две группы: верифицирующие и не верифицирующие

При создании собственного языка и описании его грамматики для анализа документов, написанных на этом языке, потребуется программа, проверяющая корректность составления (верификации) документа

В большинстве случаев после разбора документа предоставляется объектная модель, отображающая содержимое XML документа, и средства, необходимые для работы с ней (прохода по дереву элементов). При этом в некоторых анализаторах способ представления структуры документа основывается на спецификации DOM.

Верификация XML-документа

Два способа контроля правильности XML-документа

- DTD-определения (Document Type Definition)
- схемы данных (Semantic Schema)

Если XML-документ создается с использованием DTD-описаний или схем (Schemas), то он называется **валидным (valid)** или **действительным**

Действительный документ удовлетворяет некой семантической схеме, задающей его структуру и содержание

Documents Type Definitions (DTD)

В XML-документах DTD определяет набор действительных элементов, идентифицирует элементы, которые могут находиться в других элементах, и определяет действительные атрибуты для каждого из них

В XML использовать DTD не обязательно - документы, созданные без этих правил, будут правильно обрабатываться программой-анализатором, если они удовлетворяют основным требованиям синтаксиса XML

Documents Type Definitions (DTD)

Для того, чтобы использовать DTD в документе, можно или описать его во внешнем файле и при описании DTD просто указать ссылку на этот файл или же непосредственно внутри самого документа выделить область, в которой определить нужные правила. В первом случае в документе указывается имя файла, содержащего DTD- описания

```
<?xml version="1.0" standalone="yes" ?>  
<! DOCTYPE journal SYSTEM "journal.dtd">  
...
```

```
...  
<! DOCTYPE journal [  
<!ELEMENT journal (contacts, issues, authors)>  
...  
>  
...
```

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

Documents Type Definitions (DTD)

```
<!ELEMENT price (PCDATA)>
<!ATTLIST price data_currency CDATA #FIXED "CURRENCY">
<!ELEMENT rooms_num (PCDATA)>
<!ATTLIST rooms_num data_byte CDATA #FIXED "BYTE">
<!ELEMENT floor (PCDATA)>
<!ATTLIST floor data_byte CDATA #FIXED "INTEGER">
<!ELEMENT living_space (PCDATA)>
<!ATTLIST living_space data_float CDATA #FIXED "FLOAT">
<!ELEMENT counter (PCDATA)>
<!ATTLIST counter data_long CDATA #FIXED "LONG">
<!ELEMENT is_tel (PCDATA)>
<!ATTLIST is_tel data_bool CDATA #FIXED "BOOL">
<!ELEMENT house (rooms_num, floor, living_space, is_tel, counter, price)>
<!ATTLIST house id ID #REQUIRED>
...
<house id="0">
<rooms_num>5</rooms_num>
<floor>2</floor>
<living_space>32.5</living_space>
<is_tel>true</is_tel>
<counter>18346</counter>
<price>34 p. 28 κ.</price>
</house>
```

Схемы XML. XSD – XML Scheme Definition

Схема - формальный документ, устанавливающий правила конкретного языка разметки

- Правила не должны включать синтаксических деталей (например, требования применять угловые скобки или вложенные теги свойств)
- Правила перечисляют логические правила, относящиеся к типу данных

- **Словарь документа.** Определяет, какие имена элементов и атрибутов используются в XML-документах
- **Структура документа.** Определяет, где должны помещаться теги, а также может включать правила, регламентирующие то, какие теги должны быть помещены перед, после или внутри других. Можно также указывать, сколько раз может встречаться каждый элемент
- **Поддерживаемые типы данных.** Позволяют указать, должны ли данные быть обычным текстом, либо их следует интерпретировать как числовые данные, информацию о датах и т.д.
- **Допустимые диапазоны значений.** Позволяют установить ограничения на диапазоны допустимых значений числовых данных, ограничение длины текста, требование соответствия регулярным выражениям либо перечень специфических допустимых значений

Схемы XML. XSD – XML Schema Definition

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="planets">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="planet">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
              <xs:element minOccurs="0" maxOccurs="unbounded" name="moon">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="name" type="xs:string" />
                    <xs:element name="period">
                      <xs:complexType>
                        <xs:simpleContent>
                          <xs:extension base="xs:decimal">
                            <xs:attribute name="units" type="xs:string" use="required" />
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Стандарт схемы XML определяет правила, которых следует придерживаться при создании документа схемы

- Более мощное средство для определения сложных структур данных
- Более понятный способ описания грамматики языка
- Способны легко модернизироваться и расширяться
- Схемы данных позволяют описывать правила для XML-документа средствами самого же XML

Схемы XML. XSD – XML Schema Definition

```
<?xml version="1.0" encoding="utf-8"?>
<DvdList>
  <DVD ID="1" Category="Science Fiction">
    <Title>The Matrix</Title>
    <Director>Larry Wachowski</Director>
    <Price>18.74</Price>
    <Starring>
      <Star>Keanu Reeves</Star>
      <Star>Laurence Fishburne</Star>
    </Starring>
  </DVD>
  <DVD ID="2" Category="Drama">...</DVD>
  <DVD ID="3" Category="Horror">...</DVD>
  <DVD ID="4" Category="Mystery">...</DVD>
  <DVD ID="5" Category="Science Fiction">...</DVD>
</DvdList>
```

Схемы XML. XSD – XML Schema Definition

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
  elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DvdList">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="DVD">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Title" type="xs:string" />
              <xs:element name="Director" type="xs:string" />
              <xs:element name="Price" type="xs:decimal" />
              <xs:element name="Starring">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element maxOccurs="unbounded" name="Star" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="ID" type="xs:unsignedByte" use="required" />
            <xs:attribute name="Category" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


Объектная модель документа DOM

Одним из самых мощных интерфейсов доступа к содержимому XML документов является Document Object Model - DOM

Объектная модель XML документов является представлением его внутренней структуры в виде совокупности определенных объектов. Для удобства эти объекты организуются в некоторую древообразную структуру данных - каждый элемент документа может быть отнесен к отдельной ветви, а все его содержимое, в виде набора вложенных элементов, комментариев, секций CDATA и т.д. представляется в этой структуре поддеревьями. Т.к. в любом правильно составленном XML-документе обязательно определен главный элемент, то все содержимое можно рассматривать как поддеревья этого основного элемента, называемого в таком случае корнем дерева документа

<http://www.w3schools.com/dom/>

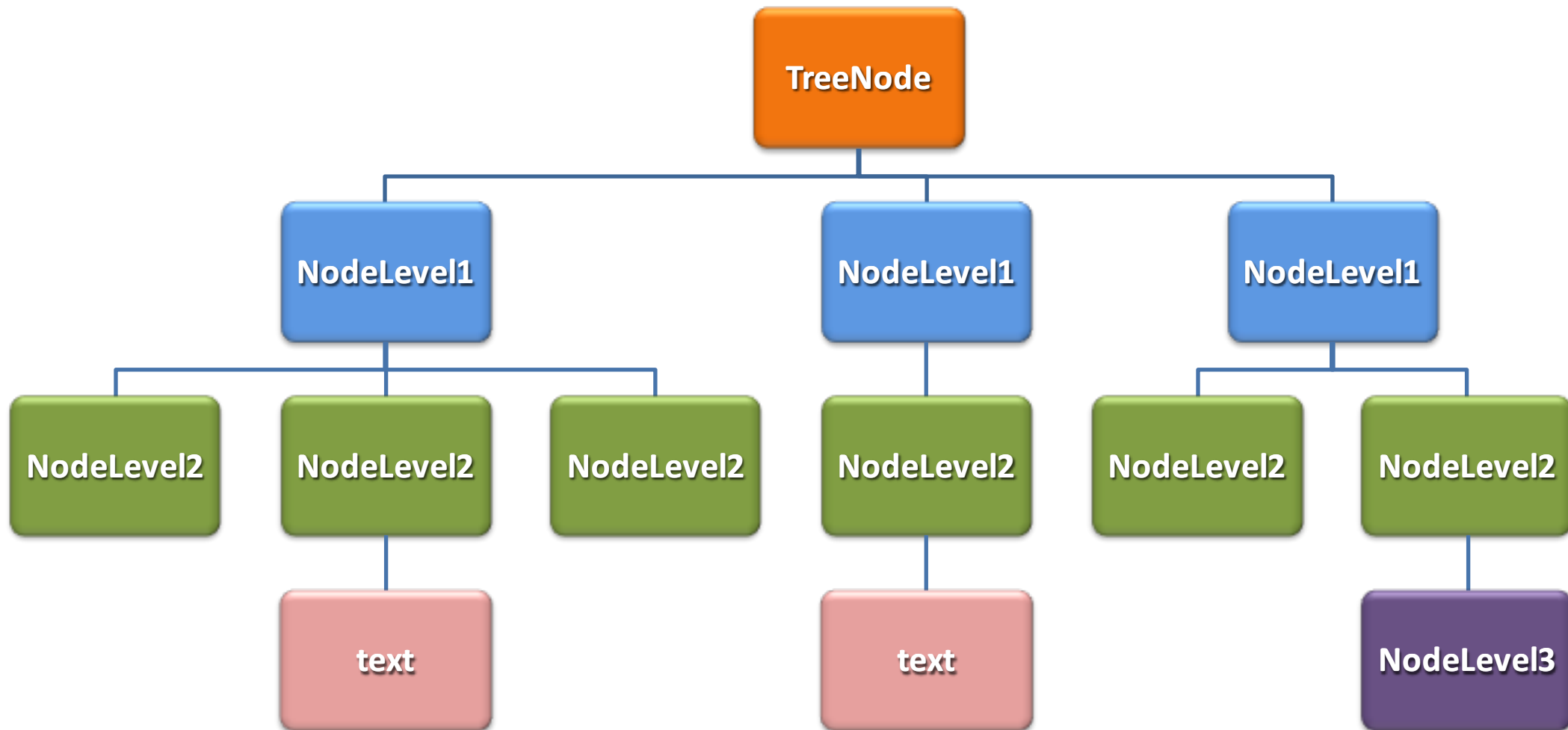
<http://www.w3.org/DOM/>

<http://javascript.ru/unsorted/w3c>

Объектная модель документа DOM

```
- <TreeNode>
  - <NodeLevel1>
    <NodeLevel2/>
    <NodeLevel2>text</NodeLevel2>
    <NodeLevel2/>
  </NodeLevel1>
  - <NodeLevel1>
    <NodeLevel2>text</NodeLevel2>
  </NodeLevel1>
  - <NodeLevel1>
    <NodeLevel2/>
    - <NodeLevel2>
      <NodeLevel3/>
    </NodeLevel2>
  </NodeLevel1>
</TreeNode>
```

Объектная модель документа DOM



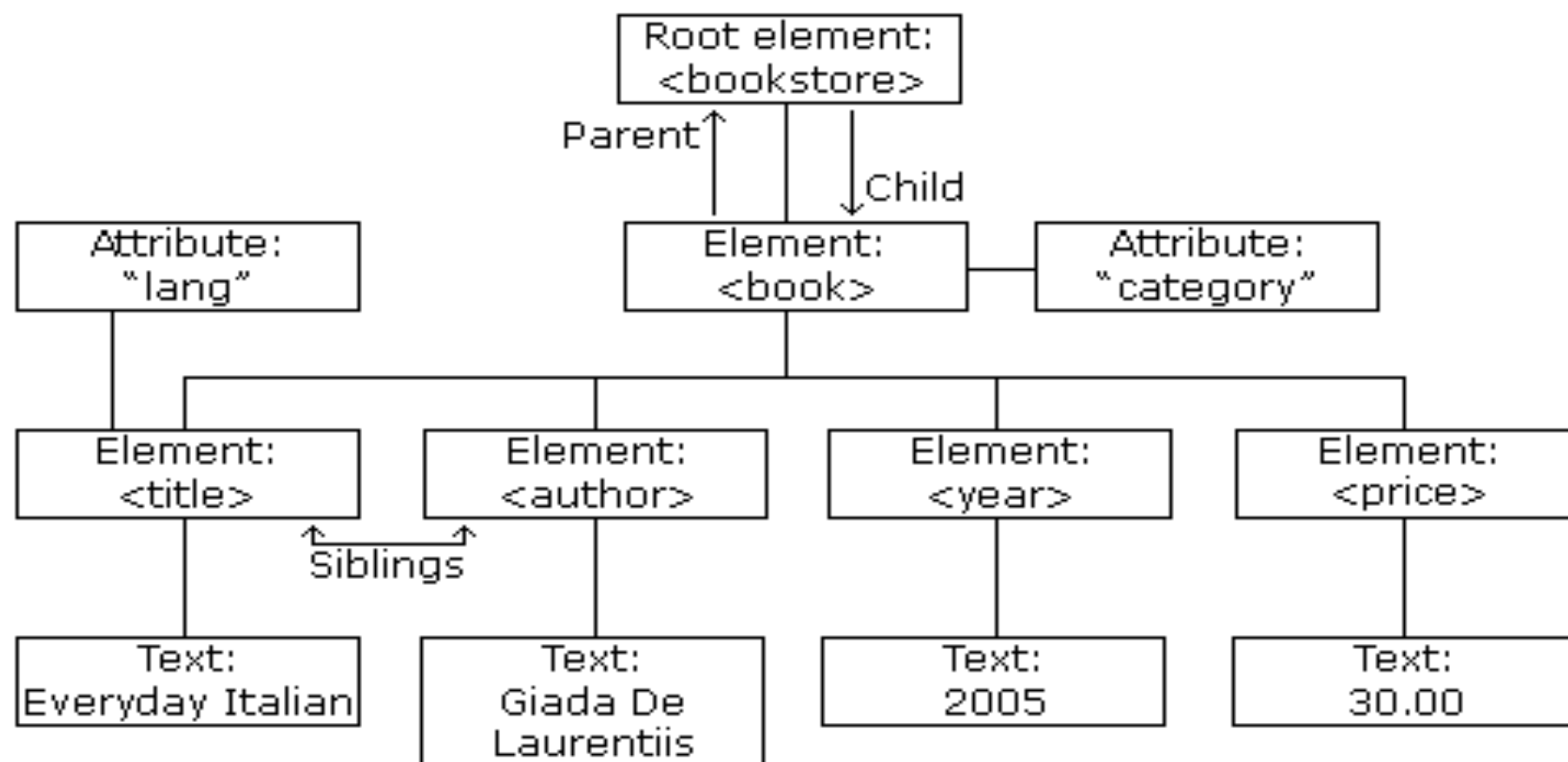
Объектная модель документа DOM

DOM - это спецификация универсального платформенно- и программно-независимого доступа к содержимому документов и является просто своеобразным API для их обработчиков. DOM является стандартным способом построения объектной модели любого HTML или XML документа, при помощи которой можно производить поиск нужных фрагментов, создавать, удалять и модифицировать его элементы.

При описании дерева XML-элементов используются следующие термины:

- Текущий элемент (self);
- Предок (ancestor) - любой элемент, содержащий текущий;
- Корень (root) - предок всех элементов;
- Родитель (parent) - непосредственный предок текущего элемента;
- Потомок (descendant) - любой элемент, вложенный в текущий;
- Ребенок (child) - непосредственный потомок текущего элемента;
- Сиблинг (sibling) - любой элемент, имеющий общего родителя с текущим элементом.

Объектная модель документа DOM



Просмотр XML - документов

XML никак не определяет способ отображения и использования описываемых с его помощью элементов документа, т.е. программе-анализатору предоставляется возможность самой выбирать нужное оформление

Принцип независимости определения внутренней структуры документа от способов представления этой информации

```
<flower>роза</flower>
```

Для того, чтобы использовать данные, определяемые элементами XML, например, отображать их на экране пользователя, необходимо написать программу-анализатор, которая бы выполняла эти действия

<http://www.w3schools.com/xml/simple.xml>

<http://www.w3schools.com/xml/simplexsl.xml>

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

Стилевые таблицы XSL

XSL (eXtensible Stylesheet Language, стиливыми таблицами или стиливыми листами) принято называть специальные инструкции, управляющие процессом отображения элемента в окне программы-клиента (например, в окне браузера)

- Стилевые таблицы XSL позволяют определять оформление элемента в зависимости от его месторасположения внутри документа, т.е. к двум элементам с одинаковым названием могут применяться различные правила форматирования.
- Языком, лежащем в основе XSL, является XML, а это означает, что XSL более гибок, универсален и у разработчиков появляется возможность использования средства для контроля за корректностью составления таких стиливых списков (используя DTD или схемы данных)
- Таблицы XSL не являются каскадными, подобно CSS, т.к. чрезвычайно сложно обеспечить "каскадируемость" стиливых описаний, или, другими словами, возможность объединения отдельных элементов форматирования путем вложенных описаний стиля, в ситуации, когда структура выходного документа заранее неизвестна и он создается в процессе самого разбора. Однако в XSL существует возможность задавать правила для стилей, при помощи которых можно изменять свойства стиливого оформления, что позволяет использовать довольно сложные приемы форматирования

Пространство имен **System.Xml** включает в себя следующие пространства имен и классы:

System.Xml.*

- **XmlReader** and **XmlWriter**. Высокопроизводительные однонаправленные курсоры для чтения и записи XML-потока
- **XmlDocument**. Представляет XML-документ в виде DOM-модели в стиле W3C

System.Xml.XPath Инфраструктура и API-интерфейс (XPathNavigator) для XPath – основанного на строках языка для написания XML

System.Xml.Schema Инфраструктура и API-интерфейс для XSD схем (W3C)

System.Xml.Xsl

Инфраструктура и API-интерфейс (XslCompiledTransform) для выполнения XSLT трансформаций XML (W3C)

System.Xml.Serialization Поддержка сериализации классов из и в XML

System.Xml.Linq Представляет XML-документ в виде DOM-модели, называемой X-DOM

Обработка файлов XML

Запись XML-файлов

Построение документа в памяти, используя класс **XmlDocument** или **XDocument**, и по завершении запись его в файл

- единственный способ записи XML-документа в нелинейном виде (позволяет вставлять новые узлы куда угодно)
- после создания над содержимым XML планируется выполнять другие операции, такие как поиск, трансформация или проверка достоверности

Запись документа непосредственно в поток с помощью **XmlWriter**, при этом данные выводятся по мере их записи, узел за узлом

- предоставляет более простую модель, которая лучше приспособлена для прямой записи в файл, поскольку не хранит весь документ целиком в памяти

Обработка файлов XML

Чтение XML-файлов

Чтение за один прием в память с использованием классов **XmlDocument**, **XPathNavigator** (только для чтения) или **XDocument**

- способ чтения XML-документа в нелинейном виде

Перемещение по содержимому, узел за узлом, используя **XmlReader** – объект чтения, основанный на потоке

- Сокращает накладные расходы памяти и обычно, но не всегда, более эффективен
- Наименее гибок

Обработка файлов XML. XmlReader. XmlWriter

Абстрактные классы **XmlReader** и **XmlWriter** - основа механизма последовательного чтения и записи XML-документов. Такой подход выгодно использовать, когда документ слишком велик, чтобы читать его в память целиком, или содержит ошибки в структуре.

System.Object

System.Xml.XmlReader

System.Xml.XmlDictionaryReader

System.Xml.XmlNodeReader

System.Xml.XmlTextReader

System.Xml.XmlValidatingReader

XmlTextReader (чтение на основе текстового потока)

XmlNodeReader (разбор XML из объектов XmlNode)

XmlValidatingReader (при чтении производится проверка схемы документа)

System.Object

System.Xml.XmlWriter

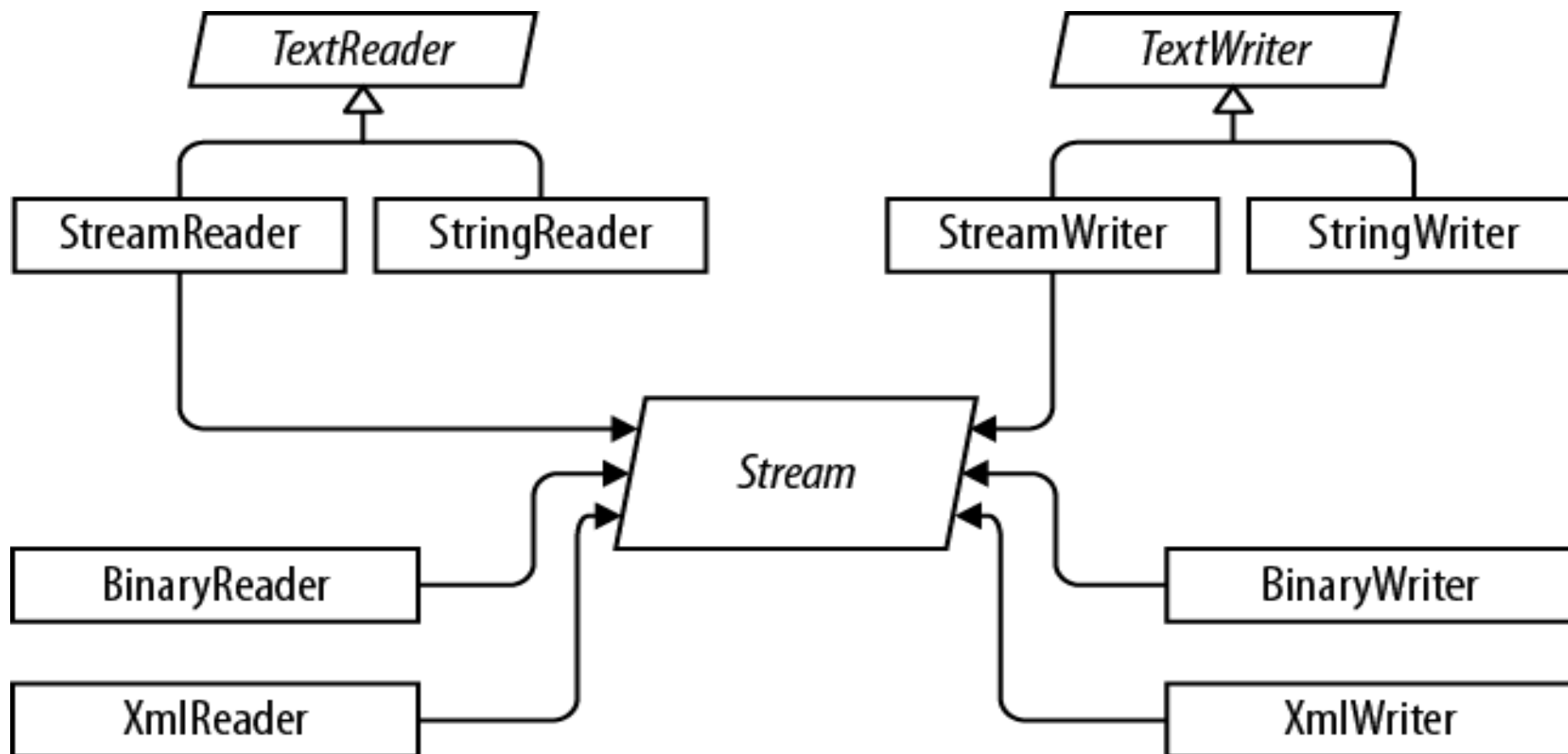
System.Xml.XmlDictionaryWriter

System.Xml.XmlTextWriter

System.Xml.Xsl.Runtime.XmlQueryOutput

XmlTextWriter (запись на основе текстового потока)

Обработка файлов XML. XmlReader. XmlWriter



Обработка файлов XML. XmlReader. XmlWriter

```
var xmlTextReader = new XmlTextReader("planets.xml");
while (xmlTextReader.Read())
{
    switch (xmlTextReader.NodeType)    // реагируем в зависимости от NodeType
    {
        case XmlNodeType.XmlDeclaration:
            Console.WriteLine("<?xml version='1.0'?>");
            break;
        case XmlNodeType.Element:
            Console.WriteLine("<{0}>", xmlTextReader.Name);
            break;
        case XmlNodeType.Text:
            Console.WriteLine(xmlTextReader.Value);
            break;
        case XmlNodeType.EndElement:
            Console.WriteLine("</{0}>", xmlTextReader.Name);
            break;
        case XmlNodeType.Comment:
            Console.WriteLine("<!--{0}-->", xmlTextReader.Value);
            break;
    }
}
```

```
<?xml version='1.0'?>
<!-- первые четыре планеты -->
<planets>
  <planet>
    <name>
      Mercury
    </name>
  </planet>
  <planet>
    <name>
      Venus
    </name>
  </planet>
  <planet>
    <name>
      Earth
    </name>
  </planet>
  <moon>
    <name>
      Moon
    </name>
    <period>
      27.321582
    </period>
  </moon>
</planets>
```

Обработка файлов XML. XmlReader. XmlWriter

Класс **XmlWriter** – это абстрактный класс для создания XML-данных: XML-данные всегда могут быть сформированы в виде простой строки и затем записаны в любой поток. Недостаток - возрастает вероятность неправильного формирования структуры XML из-за элементарных ошибок программиста

```
string message = "Hello, dude!";
var xmlTextWriter = new XmlTextWriter("greetings.xml", Encoding.UTF8)
{
    Formatting = Formatting.Indented,
    Indentation = 2
};

xmlTextWriter.WriteStartDocument();
xmlTextWriter.WriteStartElement("greeting");
xmlTextWriter.WriteString(message);
xmlTextWriter.WriteEndElement();
xmlTextWriter.WriteEndDocument();
xmlTextWriter.Flush();
```

```
<?xml version="1.0" encoding="UTF-8"?>
<greeting>Hello, dude!</greeting>
```

✓ XmlDocument

- Хранит XML- документ целиком в памяти
- Реализует полный интерфейс XML DOM Level 2 Core, как он определен W3C
- Наиболее стандартизованный интерфейс к данным XML, но временами также немного неуклюжий

✓ XPathNavigator

- Хранит XML- документ целиком в памяти
- Обеспечивает более быструю и прямолинейную модель, чем XML DOM, наряду с расширенными средствами поиска
- В отличие от XmlDocument в нем не предусмотрена возможность внесения и сохранения изменений

✓ XDocument

- Представляет собой часть LINQ to XML, но удобен и в том случае, когда запросы LINQ не конструируются
- Должен работать совместно с более старыми классами .NET XML для решения таких задач, как проверка достоверности

Обработка XML в памяти

XmlDocument

- ✓ XmlDocument хранит информацию в виде дерева узлов
- ✓ Узел (node) - это базовый элемент XML-файла, который может быть
 - элементом
 - атрибутом
 - комментарием
 - значением элемента
- ✓ Каждый узел представлен отдельным объектом XmlNode
- ✓ Класс XmlDocument является оболочкой для групп объектов XmlNode

System.Object

System.Xml.XmlNode

System.Xml.XmlAttribute

System.Xml.XmlDocument

System.Xml.XmlDocumentFragment

System.Xml.XmlEntity

System.Xml.XmlLinkedNode

System.Xml.XmlNotation

System.Object

System.Xml.XmlNodeList

Обработка XML в памяти

```
public static void OutputNode(XmlNode node)
{
    Console.WriteLine("Type= {0} \t Name= {1} \t Value= {2}",
        node.NodeType, node.Name, node.Value);
    if (node.Attributes != null)
    {
        foreach (XmlAttribute attr in node.Attributes)
        {
            Console.WriteLine("Type={0} \t Name={1} \t Value={2}",
                attr.NodeType, attr.Name, attr.Value);
        }
    }
    // если есть дочерние элементы, рекурсивно об
    if (node.HasChildNodes)
    {
        foreach (XmlNode child in node.ChildNodes)
        {
            OutputNode(child);
        }
    }
}
. . .
var doc = new XmlDocument();
doc.Load("planets.xml");
OutputNode(doc.DocumentElement);
```

| | | |
|-----------------|---------------|------------------|
| Type= Element | Name= planets | Value= |
| Type= Element | Name= planet | Value= |
| Type= Element | Name= name | Value= |
| Type= Text | Name= #text | Value= Mercury |
| Type= Element | Name= planet | Value= |
| Type= Element | Name= name | Value= |
| Type= Text | Name= #text | Value= Venus |
| Type= Element | Name= planet | Value= |
| Type= Element | Name= name | Value= |
| Type= Text | Name= #text | Value= Earth |
| Type= Element | Name= moon | Value= |
| Type= Element | Name= name | Value= |
| Type= Text | Name= #text | Value= Moon |
| Type= Element | Name= period | Value= |
| Type= Attribute | Name= units | Value= days |
| Type= Text | Name= #text | Value= 27.321582 |
| Type= Element | Name= planet | Value= |
| Type= Element | Name= name | Value= |
| Type= Text | Name= #text | Value= Mars |
| Type= Element | Name= moon | Value= |
| Type= Element | Name= name | Value= |
| Type= Text | Name= #text | Value= Phobos |
| Type= Element | Name= period | Value= |
| Type= Attribute | Name= units | Value= days |
| Type= Text | Name= #text | Value= 0.318 |
| Type= Element | Name= moon | Value= |
| Type= Element | Name= name | Value= |
| Type= Text | Name= #text | Value= Deimos |
| Type= Element | Name= period | Value= |
| Type= Attribute | Name= units | Value= days |
| Type= Text | Name= #text | Value= 1.26244 |

XPath

XPath – стандарт W3C для написания запросов к XML – посредством XPath можно запрашивать XmlDocument аналогично тому, как с помощью LINQ можно запрашивать Xdocument. XPath охватывает более широкую область применения, связанную с другими технологиями XML, такими как схемы XML, XSLT, XAML.

Записать запросы XPath можно следующими способами

1. Вызвать один из методов SelectXXX на XmlDocument или XmlNode
2. Создать XPathNavigator от классов XmlDocument или XPathDocument
3. Вызвать метод расширения XPathXXX на XmlNode

| Операция | Описание |
|----------|---------------------------------------|
| / | Дочерние узлы |
| // | Дочерние узлы рекурсивно |
| . | Текущий узел (обычно подразумевается) |
| .. | Родительский узел |
| * | Групповой узел |
| @ | Атрибут |
| [] | Фильтр |
| : | Разделитель пространства имен |

Запросы XPath выражаются в терминах XPath 2.0 Data Model – представление XML-документа в виде дерева


XPathNavigator

- ✓ Класс XPathNavigator (пространство имен System.Xml.XPath) позволяет только читать документ без возможности редактирования
- ✓ Загружает информацию в память, затем позволяет проходить по узлам
- ✓ Применяет подход на основе курсора, позволяющий использовать такие методы, как MoveToNext, для прохода по данным XML
- ✓ Может быть позиционирован только на одном узле одновременно

Интерфейс LINQ to XML API

API от Microsoft – неудобство, сложность или слабость

- конструирование деревьев XML
- центральная роль документа
- пространства имен и префиксы
- извлечение значений узлов



Решение – LINQ to XML

Интерфейс LINQ to XML API

Технология LINQ to XML предоставляет программный интерфейс для работы с XML-документами, описываемыми в виде дерева объектов (X-DOM) и обеспечивающий

- создание
- редактирование
- трансформацию XML
- возможность применение LINQ-подобного синтаксиса

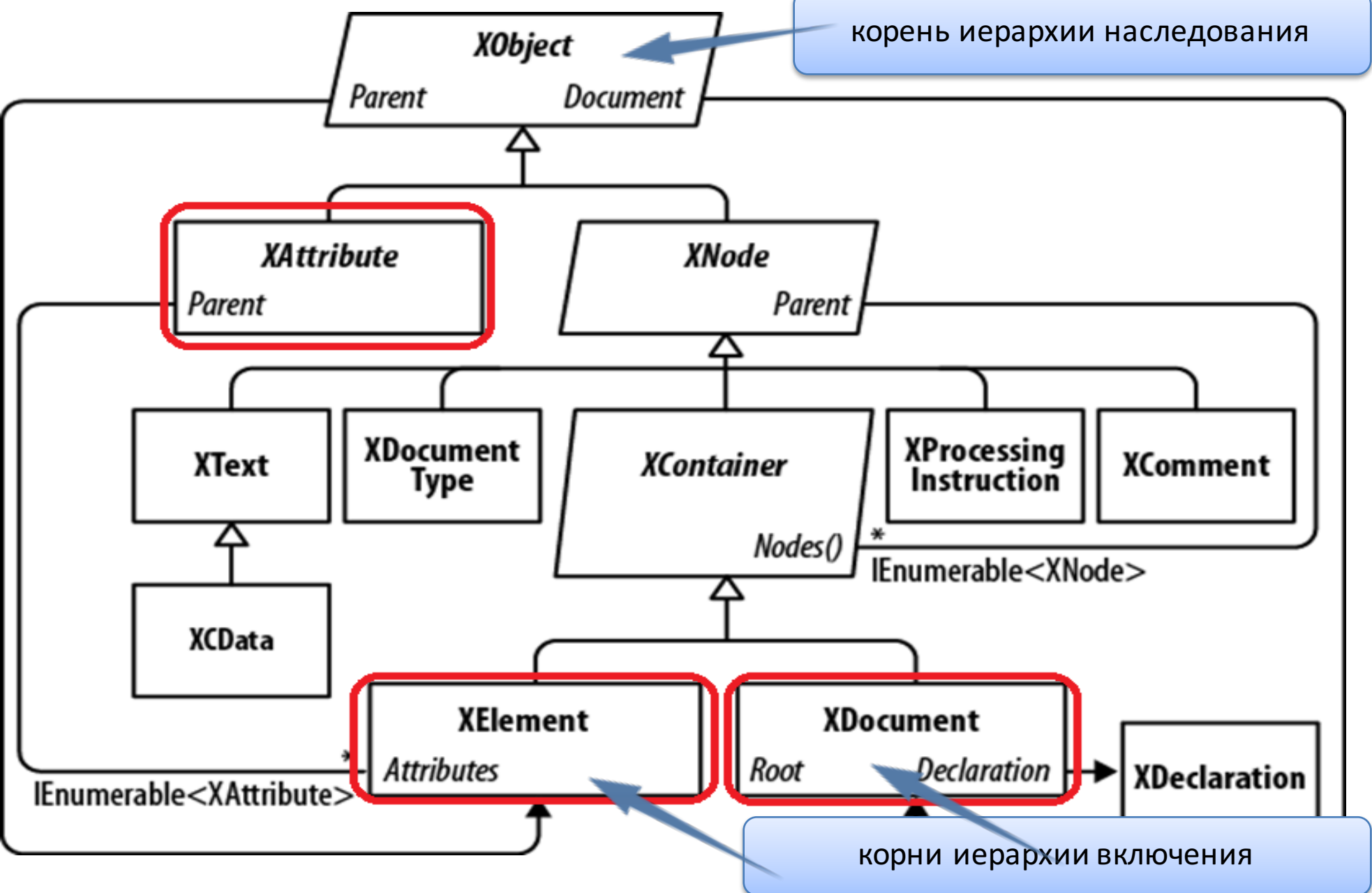
Модель LINQ to XML состоит из двух частей

- DOM-модель XML – X-DOM (XElement, XAttribute, XDocument)
- набор примерно 10 дополнительных операций запросов

Модель X-DOM дружелюбна к LINQ to XML

- методы, выдающие последовательности IEnumerable, которые можно запрашивать
- конструкторы спроектированы так, что можно строить X-DOM посредством проекций LINQ

Объектная модель LINQ to XML



Объектная модель LINQ to XML

- Абстрактные классы XObject, XElement и XContainer служат основой для иерархии классов, соответствующих различным объектам XML
- Классы XElement и XmlDocument представляют XML-элемент и XML-документ соответственно
- Класс XAttribute служит для описания XML-атрибута
- Класс XText представляет текст в XML-элементе
- Класс XName представляет имя атрибута или элемента
- Классы XDeclaration, XmlDocumentType, XProcessingInstruction, XComment описывают соответствующие части XML-документа
- Статический класс System.Xml.Linq.Extensions содержит методы расширения для выполнения запросов к XML-данным

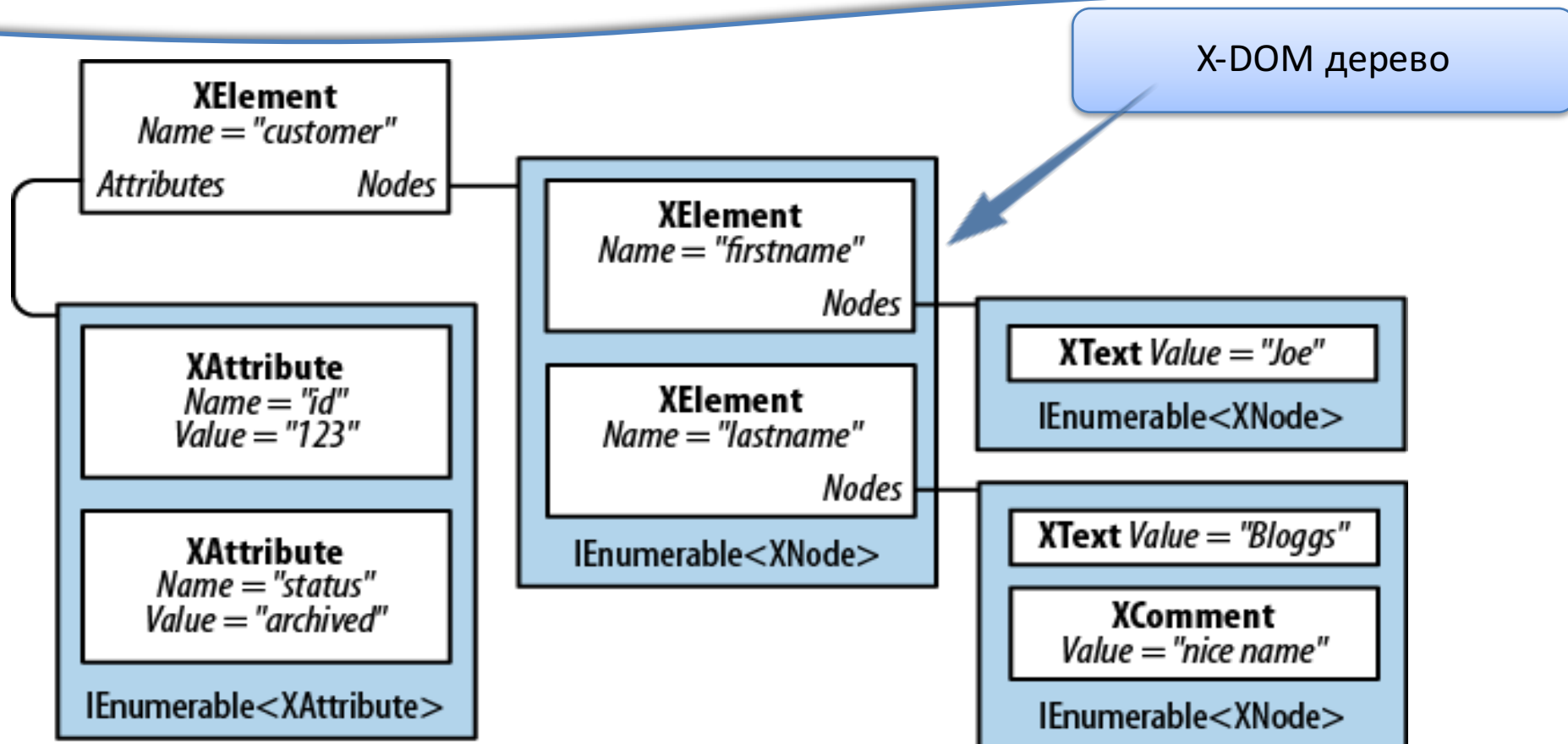
[https://msdn.microsoft.com/ru-ru/library/system.xml.linq.extensions_methods\(v=vs.100\).aspx](https://msdn.microsoft.com/ru-ru/library/system.xml.linq.extensions_methods(v=vs.100).aspx)

В LINQ to XML центральную роль играет элемент, в противоположность документу в W3C XML DOM

Объектная модель LINQ to XML

```
string xml = @"<customer id='123' status='archived'>
    <firstname>Joe</firstname>
    <lastname>Bloggs<!--nice name--></lastname>
</customer>";
```

```
XElement customer = XElement.Parse(xml);
```



Создание XML

Конструирование деревьев XML упрощено с помощью функционального построения

Функциональное построение позволяет схеме диктовать то, как конструируются объекты XML и инициализируются их значения, и все это — одновременно, в единственном операторе API-интерфейс достигает этого за счет предоставления конструкторов новых XML-объектов, которые принимают в качестве параметров как отдельные объекты, так и их множества, указывая их значения. Тип добавляемого объекта или объектов определяет то, где именно в схеме они располагаются. Общий шаблон выглядит следующим образом:

```
XMLObject obj =  
    new XMLObject (ObjectName,  
                   XMLObject1,  
                   XMLObject2,  
                   ...  
                   XMLObjectN);
```

Псевдокод

Некоторый концептуальный
абстрактный класс XML

Создание XML. Создание элементов с помощью XElement

```
XElement lastName = new XElement("lastname", "Bloggs");  
lastName.Add (new XComment("nice name"));
```

Императивная
конструкция

```
XElement customer = new XElement("customer");  
customer.Add (new XAttribute("id", 123));  
customer.Add (new XElement("firstname", "Joe"));  
customer.Add (lastName);
```

```
<customer id="123">  
  <firstname>Joe</firstname>  
  <lastname>Bloggs<!--nice name--></lastname>  
</customer>
```

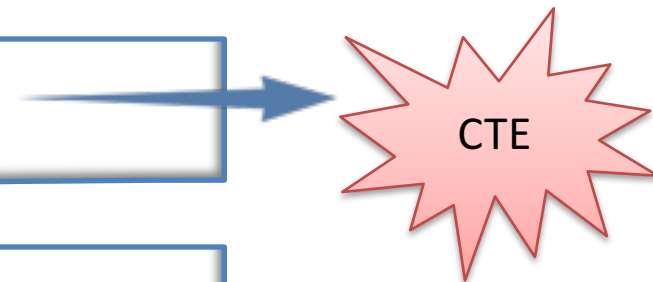
```
XElement customer = new XElement("customer",  
    new XAttribute("id", 123),  
    new XElement("firstname", "joe"),  
    new XElement("lastname", "bloggs",  
        new XComment("nice name")));
```

Функциональная
конструкция

Создание XML. Создание элементов с помощью XElement

Центральная роль элемента вместо документа

```
XmlElement xmlEmployee = new XmlElement("Employee");
```



```
XmlDocument xmlDoc = new XmlDocument ();  
XmlElement xmlEmployee = xmlDoc.CreateElement ("Employee");
```

```
XElement xmlEmployee = new XElement("Employee");
```

Ограничение W3C DOM – атрибуты, комментарии, разделы CData, инструкции обработки и ссылки на сущности — все это должно было создаваться из документа XML
LINQ to XML дает возможность непосредственно создавать каждый из этих объектов «на лету»

Создание XML. Создание элементов с помощью XElement

Для создания отдельного XML-элемента обычно используется один из конструкторов класса XElement

```
public XElement(XElement other);
public XElement(XName name);
public XElement(XStreamingElement other);
public XElement(XName name, object content);
public XElement(XName name, params object[] content);
```

| Тип или значение объекта содержимого | Способ обработки |
|---|---|
| string | Преобразуется в дочерний объект типа XText и добавляется как текстовое содержимое элемента |
| XText | Добавляет как дочерний объект - текстовое содержимое элемента |
| XElement | Добавляется как дочерний элемент |
| XAttribute | Добавляется как атрибут элемента |
| XProcessingInstruction, XComment | Добавляется как дочернее содержимое |
| IEnumerable | Объект перечисляется и обрабатывается рекурсивно. Коллекция строк добавляется в виде единого текста |
| null | Этот объект игнорируется |
| Любой прочий тип | Вызывается метод ToString, и результат трактуется как string |

Создание XML. Создание элементов с помощью XElement

```
var e1 = new XElement("name", "Earth");  
var e2 = new XElement("planet", e1);
```

<name>Earth</name>

<planet>
 <name>Earth</name>
</planet>

```
var e3 = new XElement("period", new XAttribute("units", "days"));
```

<period units="days" />

```
var e4 = new XElement("comment", new XComment("the comment"));
```

<comment>
 <!--the comment-->
</comment>

```
var e5 = new XElement("list", new List<object> { "text",  
                                                new XElement("name", "Mars") });
```

<list>
 text<name>Mars</name>
</list>

```
var e6 = new XElement("moon", null);  
var e7 = new XElement("date", DateTime.Now);
```

<moon />
<date>2010-01-19T11:04:54.625+02:00</date>

Создание XML. Создание элементов с помощью XElement

Имена, пространства имен и префиксы

Чтобы исключить путаницу, связанную с именами, пространствами имен и префиксами пространств имен, последние изъяты из API-интерфейса. С помощью LINQ to XML префиксы пространств имен разворачиваются на вводе и возвращаются в выводе. Внутри они не существуют!

В XML каждый элемент должен иметь имя

```
XElement xmlEmployee = new XElement("Employee");
```



Когда элемент создается, если его имя указано в конструкторе, оно неявно преобразуется из string в **объект XName**

Объект XName состоит из пространства имен — объекта XNamespace — и своего локального имени, того, которое было указано

Создание XML. Создание элементов с помощью XElement

```
XNamespace nameSpace = "http://www.bsu.by";  
XElement xEmployees = new XElement(nameSpace + "Employees");
```

```
XElement xEmployees = new XElement("{http:// www.bsu.by}" + "Employees");
```

При установке пространства имен простого указания URI компании или домена организации может быть недостаточно для того, чтобы гарантировать уникальность. Это гарантирует лишь отсутствие коллизий с любой другой существующей организацией, которая соблюдает установленные соглашения названий пространств имен. Однако внутри организации могут случиться коллизии с любым другим подразделением, если в пространстве имен не будет указано ничего помимо URI организации. И здесь весьма пригодится знание организационной структуры предприятия — его подразделений, департаментов и т.д. Лучше всего, если пространство имен будет раскрывать весь путь до определенного уровня, находящегося под четким контролем

Создание XML. Создание элементов с помощью XElement

Извлечение значения узла

```
XElement name = new XElement("Name", "Alex");  
Console.WriteLine(name.ToString());
```

Метод ToString элемента выводит саму строку XML, а не тип объекта, как это делается в W3C DOM API

<Name>Alex</Name>

Создание XML. Создание элементов с помощью XElement

```
XElement name = new XElement("Person",  
    new XElement("FirstName", "Alex"),  
    new XElement("LastName", "Erohin"));  
  
Console.WriteLine("XML: \n\n" + name +  
    "\n\nИзвлекаем значение: " + (string)name);
```

Если привести узел к типу данных, к которому может быть преобразовано его значение, то будет выведено само значение

XML :

```
<Person>  
<FirstName>Alex</FirstName>  
<LastName>Erohin</LastName>  
</Person>
```

Извлекаем значение: AlexErohin

Существуют операции приведения для string, int, int?, uint, uint?, long, long?, ulong, ulong?, bool, bool?, float, float?, double, double?, decimal, decimal?, TimeSpan, TimeSpan?, DateTime, DateTime?, GUID и GUID?

Создание XML. Создание атрибутов с помощью XAttribute

Атрибут (атрибуты не наследуются от узлов!) реализованный в LINQ to XML с помощью класса XAttribute, является парой "имя-значение", хранящейся в коллекции объектов XAttribute, которые относятся к объекту XElement. Используя функциональное конструирование, можно создать атрибут и "на лету" добавить его к элементу

```
XElement xEmployee = new XElement("Employee",  
    new XAttribute("type", "Programmer"));
```

```
<Employee type="Programmer" />
```

Иногда не удастся создать атрибут одновременно с конструированием его элемента. В таком случае должен быть создан его экземпляр и затем добавлен к элементу

```
XElement xEmployee = new XElement("Employee");  
XAttribute xAttr = new XAttribute("type", "Programmer");  
xEmployee.Add(xAttr);
```

```
<Employee type="Programmer" />
```

Создание XML. Создание комментариев с помощью XComment

С использованием функционального конструирования можно создать комментарий и добавить его к элементу "на лету"

```
XElement xEmployee = new XElement("Employee",  
    new XComment("Добавление нового сотрудника"));
```

```
<Employee>  
<!--Добавление нового сотрудника-->  
</Employee>
```

```
XElement xEmployee = new XElement("Employee");  
XComment xCom = new XComment("Добавление нового сотрудника");  
xEmployee.Add(xCom);
```

Создание XML. Создание контейнеров с помощью XContainer

Поскольку **XContainer** — абстрактный класс, создавать его экземпляры нельзя. Вместо этого понадобится создавать экземпляр одного из его подклассов — **XDocument** или **XElement**. Концептуально **XContainer** — это класс, унаследованный от **XNode**, который может содержать другие классы-наследники **XNode**

Создание XML. Создание объявлений с помощью XDeclaration

В отличие от большинства других классов LINQ to XML, объявления должны добавляться к XML-документу, а не к элементу

```
XDocument xDoc = new XDocument(new XDeclaration("1.0", "UTF-8", "yes"),  
    new XElement("Employee"));
```

```
<Employee />
```

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

```
XDocument xDoc = new XDocument(new XElement("Employee"));  
XDeclaration xDeclaration = new XDeclaration("1.0", "UTF-8", "yes");  
xDoc.Declaration = xDeclaration;
```

Создание XML. Создание типов документов с помощью XmlDocumentType

API-интерфейс LINQ to XML делает операцию создания типов документов совершенно безболезненной. Типы XML-документов реализованы LINQ to XML с помощью класса **XmlDocumentType**

В отличие от большинства других классов в LINQ to XML, типы документов предназначены для добавления к XML-документам, а не к элементам

```
XmlDocument xDoc = new XmlDocument(new XmlDocumentType("Employees", null, "Employees.dtd", null), new XElement("Employee"));
```

```
<!DOCTYPE Employees SYSTEM "Employees.dtd">  
<Employee />
```

```
XmlDocument xDoc = new XmlDocument();  
XmlDocumentType docType = new XmlDocumentType("Employees", null, "Employees.dtd", null);  
xDoc.Add(docType, new XElement("Employee"));
```

Если добавить тип документа после добавления любого элемента, сгенерируется исключение

Создание XML. Создание документов с помощью XmlDocument

Документы XML реализованы в LINQ to XML в виде класса XmlDocument

```
XmlDocument xDocument = new XmlDocument();
```

```
xDocument = new XmlDocument(  
    new XDeclaration("1.0", "UTF-8", "yes"),  
    new XmlDocumentType("Employees", null, "Employees.dtd", null),  
    new XProcessingInstruction("EmployeeCataloger", "out-of-print"),  
    new XElement("Employees"));
```

```
<!DOCTYPE Employees SYSTEM "Employees.dtd">  
<?EmployeeCataloger out-of-print?>  
<Employees />
```


Создание XML. Создание имен с помощью XName

В LINQ to XML нет необходимости непосредственно создавать имена с объектами XName. Фактически класс XName не имеет общедоступных конструкторов, так что нет способа создавать его экземпляры. Объект XName может быть создан из строки с необязательным пространством имен автоматически при создании объекта XElement. Объект XName состоит из LocalName — строки — и пространства имен, которое представлено в XNamespace

```
XNamespace ns = "http://www.bsu.by/training EPAM/ASP.NET/LINQ2XML";  
XElement xEmps = new XElement(ns + "Employee");
```

```
<Employee xmlns="http://www.bsu.by/training EPAM/ASP.NET/LINQ2XML" />
```

Создание XML. Создание инструкций обработки с помощью XProcessingInstruction

Инструкции обработки никогда ранее не было так легко создавать, как в API-интерфейсе LINQ to XML. Здесь они реализуются классом **XProcessingInstruction**.

Инструкции обработки можно создавать на уровне документа или элемента, создавая их "на лету" в обоих случаях с помощью функционального конструирования

```
XDocument xDoc = new XDocument(  
    new XProcessingInstruction("EployeCataloger", "out-of-print"),  
    new XElement("Employees",  
        new XElement("Employee",  
            new XProcessingInstruction("EmployeeDeleter", "delete"),  
            new XElement("FirstName", "Alex"),  
            new XElement("LastName", "Erohin"))));
```

```
<?EployeCataloger out-of-print?>  
<Employees>  
  <Employee>  
    <?EmployeeDeleter delete?>  
    <FirstName>Alex</FirstName>  
    <LastName>Erohin</LastName>  
  </Employee>  
</Employees>
```

Создание XML. Создание потоковых элементов с помощью XElement

Многие из стандартных операций запросов в действительности откладывают свою работу до момента, пока не начнется перечисление возвращаемых ими данных. Если вызывается операция, которая фактически откладывает свое выполнение, и нужно спроектировать вывод запроса в виде XML, возникает дилемма. С одной стороны, есть желание воспользоваться преимуществом отложенной природы операции и выполнить работу, только когда в ней возникнет необходимость. Но с другой стороны, вызов API-интерфейса LINQ to XML заставит запрос выполняться немедленно

```
string[] names = { "John", "Paul", "George", "Pete" };  
XElement xNames = new XElement("Beatles",  
    from n in names  
    select new XElement("Name", n));  
  
names[3] = "Ringo";
```

```
<Beatles>  
  <Name>John</Name>  
  <Name>Paul</Name>  
  <Name>George</Name>  
  <Name>Pete</Name>  
</Beatles>
```

Создание XML. Создание инструкций обработки с помощью XProcessingInstruction

Чтобы конструирование дерева XML было отложено, необходим какой-то другой способ, и именно для этого предназначены *потокосы* (streaming) элементы. В LINQ to XML потокосые элементы реализованы классом XStreamingElement

```
string[] names = { "John", "Paul", "George", "Pete" };  
XStreamingElement xNames = new XStreamingElement("Beatles",  
    from n in names  
    select new XElement("Name", n));  
  
names[3] = "Ringo";
```

```
<Beatles>  
  <Name>John</Name>  
  <Name>Paul</Name>  
  <Name>George</Name>  
  <Name>Ringo</Name>  
</Beatles>
```

Сохранение XML. Сохранение с помощью XDocument.Save

```
void XDocument.Save(string filename);  
void XDocument.Save(TextWriter textWriter);  
void XDocument.Save(XmlWriter writer);  
void XDocument.Save(string filename, SaveOptions options);  
void XDocument.Save(TextWriter textWriter, SaveOptions options);
```

Методы Save являются методами экземпляра

```
XDocument xDoc = new XDocument(  
    new XElement("Employees",  
        new XElement("Employee",  
            new XAttribute("type", "Programmer"),  
            new XAttribute("language", "Russian"),  
            new XElement("FirstName", "Alex"),  
            new XElement("LastName", "Erohin"))));  
xDoc.Save("employees.xml");
```

Сохранение XML-документа в файле,
находящемся в папке проекта

Сохранение XML. Сохранение с помощью XElement.Save

```
void XElement.Save(string filename);  
void XElement.Save(TextWriter textWriter);  
void XElement.Save(XmlWriter writer);  
void XElement.Save(string filename, SaveOptions options);  
void XElement.Save(TextWriter textWriter, SaveOptions options);
```

```
XElement xElement = new XElement("Employees",  
    new XElement("Employee",  
        new XAttribute("type", "Programmer"),  
        new XAttribute("language", "Russian"),  
        new XElement("FirstName", "Alex"),  
        new XElement("LastName", "Erohin"))));  
xElement.Save("employees.xml", SaveOptions.None);
```

XML-документ не создается!

Загрузка XML из файла. Загрузка с помощью XDocument.Load

```
static XDocument XDocument.Load(string uri);  
static XDocument XDocument.Load(TextReader textReader);  
static XDocument XDocument.Load(XmlReader reader);  
static XDocument XDocument.Load(string uri, LoadOptions options);  
static XDocument XDocument.Load(TextReader textReader, LoadOptions options);  
static XDocument XDocument.Load(XmlReader reader, LoadOptions options);
```

Метод Load — статический

Методы Load могут принимать в параметре string строку URI

LoadOptions.None Для указания, что никакие опции загрузки не используются

LoadOptions.PreserveWhitespace Для того, чтобы предохранить пробелы и пустые строки в исходном XML

LoadOptions.SetLineInfo Для того, чтобы иметь возможность получать строку и позицию любого объекта, унаследованного от XObject, посредством интерфейса IXmlLineInfo

LoadOptions.SetBaseUri Для того, чтобы получать базовый URI любого объекта-наследника XObject

Загрузка XML из файла. Загрузка с помощью XmlDocument.Load()

```
XmlDocument xDoc = XmlDocument.Load("employees.xml",  
                                     LoadOptions.SetBaseUri | LoadOptions.SetLineInfo);  
  
Console.WriteLine(xDoc);  
  
XmlElement firstName = xDoc.Descendants("FirstName").First();  
  
Console.WriteLine("firstName Строка {0} - Позиция {1}",  
                  ((IXmlLineInfo)firstName).LineNumber,  
                  ((IXmlLineInfo)firstName).LinePosition);  
  
Console.WriteLine("Базовый URI: {0}", firstName.BaseUri);
```

```
<Employees>  
  <Employee type="Programmer" language="Russian">  
    <FirstName>Alex</FirstName>  
    <LastName>Erohin</LastName>  
  </Employee>  
</Employees>  
firstName Строка 4 - Позиция 6  
Базовый URI: file:///C:/install/LINQPad/employees.xml
```


Загрузка XML из файла. Загрузка с помощью XElement.Load

```
static XElement XElement.Load(string uri);  
static XElement XElement.Load(TextReader textReader);  
static XElement XElement.Load(XmlReader reader);  
static XElement XElement.Load(string uri, LoadOptions options);  
static XElement XElement.Load(TextReader textReader, LoadOptions options);  
static XElement XElement.Load(XmlReader reader, LoadOptions options);
```

Загрузка XML из файла. Разбор содержимого с помощью методов XmlDocument.Parse или XElement.Parse

Метод Parse – получение данных из переменной типа string в переменную типа XML-документа

```
XDocument xDoc = new XDocument(  
    new XElement("Employees",  
        new XElement("Employee",  
            new XAttribute("type", "Programmer"),  
            new XAttribute("language", "Russian"),  
            new XElement("FirstName", "Alex"),  
            new XElement("LastName", "Erohin"))));  
  
xDoc.Save("employees.xml", SaveOptions.DisableFormatting);  
string xml = XDocument.Load("employees.xml").ToString();  
XElement xElement = XElement.Parse(xml);
```

```
<Employees>  
  <Employee type="Programmer" language="Russian">  
    <FirstName>Alex</FirstName>  
    <LastName>Erohin</LastName>  
  </Employee>  
</Employees>
```

Чтение и обход XML

Методы классов `XNode` и `XContainer` позволяют получить у элемента наборы предков и дочерних узлов (элементов). При этом возможно указание фильтра – имени элемента. Большинство методов возвращают коллекции, реализующие `IEnumerable`

// методы выборки у `XNode`

```
public IEnumerable<XElement> Ancestors();           // + XName name
public IEnumerable<XElement> ElementsAfterSelf();   // + XName name
public IEnumerable<XElement> ElementsBeforeSelf();  // + XName name
public bool IsAfter(XNode node);
public bool IsBefore(XNode node);
public IEnumerable<XNode> NodesAfterSelf();
public IEnumerable<XNode> NodesBeforeSelf();
```

// методы выборки у `XContainer`

```
public IEnumerable<XNode> DescendantNodes();
public IEnumerable<XElement> Descendants();           // + XName name
public XElement Element(XName name);
public IEnumerable<XElement> Elements();             // + XName name
```

Методы расширения System.Xml.Linq.Extensions

Статический класс Extensions определяет несколько методов расширения, работающих с коллекциями элементов или узлов XML

```
public static class Extensions
{
    // Methods
    public static IEnumerable<XElement> Ancestors<T>(this IEnumerable<T> source) where T: XNode;
    public static IEnumerable<XElement> Ancestors<T>(this IEnumerable<T> source, XName name) where T: XNode;
    public static IEnumerable<XElement> AncestorsAndSelf(this IEnumerable<XElement> source);
    public static IEnumerable<XElement> AncestorsAndSelf(this IEnumerable<XElement> source, XName name);
    public static IEnumerable<XAttribute> Attributes(this IEnumerable<XElement> source);
    public static IEnumerable<XAttribute> Attributes(this IEnumerable<XElement> source, XName name);
    public static IEnumerable<XNode> DescendantNodes<T>(this IEnumerable<T> source) where T: XContainer;
    public static IEnumerable<XNode> DescendantNodesAndSelf(this IEnumerable<XElement> source);
    public static IEnumerable<XElement> Descendants<T>(this IEnumerable<T> source) where T: XContainer;
    public static IEnumerable<XElement> Descendants<T>(this IEnumerable<T> source, XName name) where T: XContainer;
    public static IEnumerable<XElement> DescendantsAndSelf(this IEnumerable<XElement> source);
    public static IEnumerable<XElement> DescendantsAndSelf(this IEnumerable<XElement> source, XName name);
    public static IEnumerable<XElement> Elements<T>(this IEnumerable<T> source) where T: XContainer;
    public static IEnumerable<XElement> Elements<T>(this IEnumerable<T> source, XName name) where T: XContainer;
    public static IEnumerable<T> InDocumentOrder<T>(this IEnumerable<T> source) where T: XNode;
    public static IEnumerable<XNode> Nodes<T>(this IEnumerable<T> source) where T: XContainer;
    public static void Remove(this IEnumerable<XAttribute> source);
    public static void Remove<T>(this IEnumerable<T> source) where T: XNode;
}
```

Expand Methods

Трансформации XML (XSLT)

С помощью LINQ to XML можно выполнять трансформации XML, используя два совершенно разных подхода

- применение языка XSLT (eXtensible Stylesheet Language Transformations) через классы-мосты — XmlReader и XmlWriter
- использование для трансформаций самого API-интерфейса LINQ to XML за счет функционального конструирования целевого документа XML и встраивания запроса LINQ to XML в некоторый документ XML

XSD и проверка достоверности схемы

Содержимое отдельного XML-документа почти всегда является специфичным для предметной области, такой как документ Microsoft Word, документ с конфигурацией приложения или веб-служба – для каждой области XML-файл соответствует какому-то шаблону

Стандарты описания таких шаблонов, предназначенные для унификации и автоматизации процедур обработки и проверки достоверности XML-документов – XSD (XML Scheme Definition) и DTD (Document Type Definition) (System.Xml)

Выполнение достоверности схемы

Перед чтением или обработкой файл либо документ XML можно проверить на предмет соответствия одной или нескольким схемам

- уменьшение количества проверок на ошибки и обработок исключений
- позволяет обнаружить ошибки, которые в противном случае остались незамеченными
- сообщения об ошибках являются подробными и информационными

Для выполнения проверки необходимо подключить схему к XmlReader, XmlDocument или объекту X-DOM и затем читать или загружать документ или файл XML обычным способом – проверка происходит автоматически по мере чтения содержимого

```
XmlReaderSettings settings = new XmlReaderSettings();  
settings.ValidationType = ValidationType.Schema;  
settings.Schemas.Add (null, "customers.xsd");  
using (XmlReader r = XmlReader.Create ("customers.xml", settings))  
    try { while(r.Read()); }  
    catch (XmlSchemaValidationException ex)  
    { . . . }
```

Выполнение достоверности схемы

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.ValidationType = ValidationType.Schema;
settings.Schemas.Add (null, "customers.xsd");

XDocument doc;
using (XmlReader r = XmlReader.Create ("customers.xml", settings))
    try { doc = XDocument.Load (r); }
    catch (XmlSchemaValidationException ex) { ... }

XmlDocument xmlDoc = new XmlDocument();
using (XmlReader r = XmlReader.Create ("customers.xml", settings))
    try { xmlDoc.Load (r); }
    catch (XmlSchemaValidationException ex) { ... }
```

С помощью методов расширения класса `System.Xml.Schema.Extensions` можно выполнить проверку достоверности объекта `XDocument` или `Xelement`, уже находящегося в памяти

Источники

- <http://www.codenet.ru/webmast/xml/>
- <http://www.wisdomweb.ru/HDOM/hdom-first.php>
- <http://www.w3.org/>
- http://professorweb.ru/my/LINQ/ling_xml/level5/ling_to_xml_index.php

Спасибо за внимание

БГУ, ММФ, кафедра веб-технологий и компьютерного
моделирования

Автор: к. ф.-м. н., доцент, Кравчук Анжелика Ивановна

e-mail: anzhelika.kravchuk@gmail.com