

# Основы XML



eXtensible Markup Language

## XML (eXtensible Markup Language, расширяемый язык разметки)

Это универсальный 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.

### Основное назначение

Язык XML предназначен для хранения и передачи данных.

**Теги XML не predeterminedены**, что дает возможность самостоятельно определять необходимые теги т.е. **XML - самоопределяемый**.

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

## Плюсами языка являются :

- **Независимость** от платформы, операционной системы и программ обработки
- Представление **произвольных данных** и дополнительной метаинформации
- XML позволяет хранить и упорядочивать информацию почти **любого рода** в формате, приспособленном к потребностям пользователя .
- Используя **Unicode** в качестве стандартного набора символов, XML поддерживает внушительное число различных систем письма и символов, от скандинавских рунических символов до китайских иероглифов Хань.
- XML предоставляет несколько способов **проверки качества документа** путем применения синтаксических правил, внутренней проверки ссылок, сравнения с моделями документов и типов данных .
- Благодаря простому и понятному синтаксису, а также однозначной структуре, **XML легко читается и анализируется**, как человеком, так и программами.

## Общее представление об XML

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

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

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

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

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

## Дерево XML

XML документы формируют **древовидную структуру**, которая начинается с «**корневого**» элемента и разветвляется на «**дочерние**» элементы.

```
<bookstore>
```

```
  <book category="PROGRAMMING">
    <title lang="ru" >Язык C# 4.0</title>
    <author>Г. Шилдт</author>
    <year>2014</year>
    <price>50.00</price>
```

```
  </book>
```

```
  <book category="PROGRAMMING ">
    <title lang="ru" >.NET</title>
    <author>Э. Троелсен</author>
    <year>2013</year>
    <price>60.99</price>
```

```
  </book>
```

```
  <book category="WEB">
    <title lang="en">Learning
```

```
XML</title>
```

```
    <author>Erik T. Ray</author>
    <year>2010</year>
    <price>39.95</price>
```

```
  </book>
```

```
</bookstore>
```

Корневой элемент  
**bookstore**

Элемент  
**<book>**

Атрибут  
**category**

Атрибут  
**lang**

Элемент  
**<title>**

текст  
**Язык C# 4.0**

Элемент  
**<author>**

текст  
**Г. Шилдт**

Элемент  
**<year>**

текст  
**2014**

Элемент  
**<price>**

текст  
**50.00**

1. У XML документа должен быть корневой элемент
2. Все XML элементы должны иметь закрывающий тег

*<p>Это параграф.</p>*

3. Теги XML регистрозависимы

*<Message>Это неправильно</message>*  
*<message>Это правильно</message>*

4. XML элементы должны соблюдать корректную вложенность

*<b><i>Это жирный и курсивный текст</i></b>*

5. Значения XML атрибутов должны заключаться в кавычки

*<note date=12/11/2007>*  
*< note date="12/11/2007">*

6. Комментарии в XML

*<!-- Это комментарий -->*

## **Синтаксически верный XML документ**

Если XML документ составлен в соответствии с приведенными синтаксическими правилами, то говорят, что это "синтаксически верный" XML документ.

# Иерархия элементов

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

```
<bookstore>  
  <book category="PROGRAMMING">  
    <title lang="ru" >Язык C# 4.0</title>  
    <author>Г. Шилдт</author>  
    <year>2014</year>  
    <price>50.00</price>  
  </book>  
  <book category="PROGRAMMING">  
  </book>  
</bookstore>
```

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

# Корневой элемент

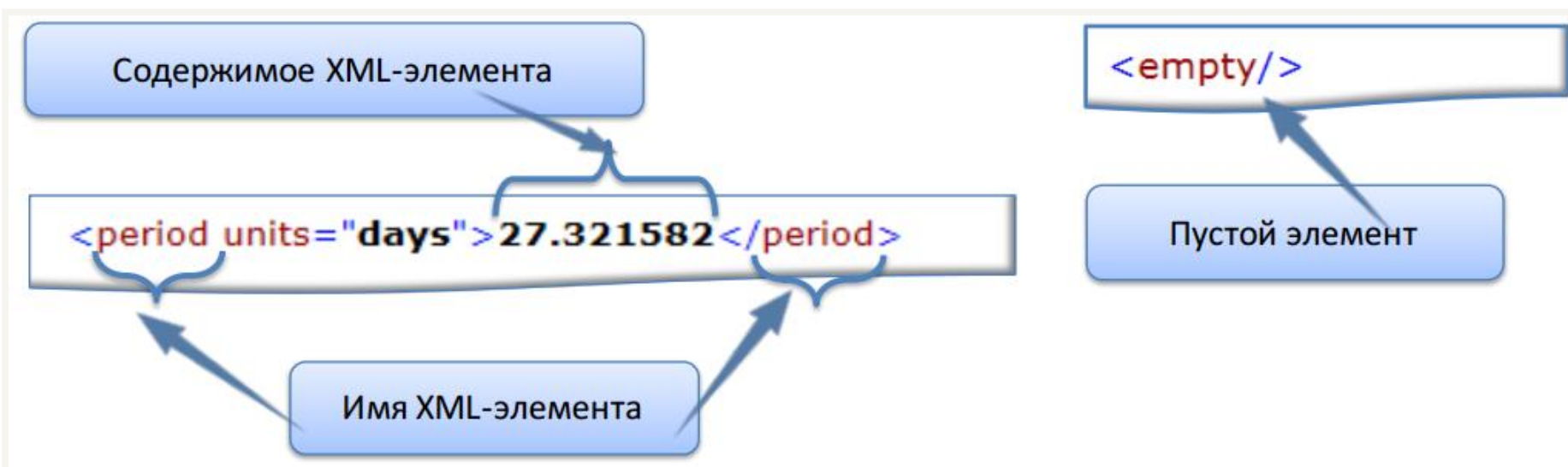
В XML-документе всегда должен быть единственный элемент, называемый корневым, с него программы-анализаторы начинают просмотр документа

```
<bookstore>
  <book category="PROGRAMMING">
    <title lang="ru" >Язык C# 4.0</title>
    <author>Г. Шилдт</author>
    <year>2014</year>
    <price>50.00</price>
  </book>
  <book category="PROGRAMMING ">
    <title lang="ru" > .NET</title>
    <author>Э. Троелсен</author>
    <year>2013</year>
    <price>60.99</price>
  </book>
</bookstore>
```



# XML элемент

## Единица информации – XML-элемент



- 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

# XML элементы

XML документ состоит из XML элементов.

## Что такое XML элемент?

**XML элемент** – это все от начального тега элемента до конечного тега элемента.

Элемент может содержать:

- другие элементы
- текст
- атрибуты
- или набор из всего выше названного

## Пустые XML элементы

`<book></book>`

`<book/>` – такой синтаксис элемента называется самозакрывающийся.

## Правила написания имен XML

XML элементы должны следовать следующим правилам написания имен:

- Имена могут содержать буквы, числа и другие символы
- Имена **не** могут начинаться с цифры или символа пунктуации
- Имена **не** могут начинаться с сочетания "xml" (или XML, или Xml и т. п.)
- Имена **не** могут содержать пробельные символы.

**В качестве имен можно использовать любые слова. Нет зарезервированных слов.**

## XML атрибуты

Атрибуты элемента - параметры, уточняющие его характеристики.

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

Значение атрибута всегда должно заключаться в кавычки. Это могут быть либо двойные, либо одинарные кавычки.

```
<book category="PROGRAMMING">  
  <title lang="ru" >Язык C# 4.0</title>  
  <author>Г. Шилдт</author>  
  <year>2014</year>  
  <price>50.00</price>  
</book>
```

## Пролог

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

# Семантика&Грамматика

```
<?xml version="1.0" encoding="windows-1251"
```

```
<!-- Пример разметки -->
```

```
<notebook>
```

```
<person sex="man">
```

```
<name>Ivan</name>
```

```
<surname>Ivanov</surname>
```

```
<phone>22-22-22</phone>
```

```
</person>
```

```
<person sex="wooman">
```

```
<name>Petr</name>
```

```
<surname>Petrov</surname>
```

```
<phone>33-33-33</phone>
```

```
</person>
```

```
</notebook>
```

# Encoding (кодировка)

На сегодняшний день используются многие кодировки, но основные из них:

- *ASCII* – 8 битная, но охватывающая лишь 128 символов. Включает латинский алфавит, цифры и основные знаки пунктуации. Важность ее в том, что все остальные кодовые страницы совместимы с ней.
- *Windows-1251* - кириллица. Стандартная кодировка для Windows.
- *Windows-1252* - западноевропейские языки. Для англоязычных символов.
- *KOI8-R* - кириллица, но для Unix операционных систем.
- *KOI8-U* - для украинской языка - для Unix операционных систем.
- *MacCyrillic* - кириллица для Apple.
- *UTF-8* (Unicode Transfer Format) - вмещает все символы ASCII (128 символов), при этом каждый символ кодируется одним байтом (8 бит). Кодировка по умолчанию.
- *UTF-16* - по 2 байта на символ. Используется для символов некоторых национальных языков и содержит 65 тыс. символов.
- *UTF-32* - по 4 байта на символ. Используется для менее распространенных символов

**Пример:**

```
<?xml version="1.0" encoding="windows-1251"?>
```

## Способы программного анализа документа :

- Для программного анализа используются парсеры XML. *Парсер – синтаксический анализатор документа*
- DOM – document object model



## Необходимость проверки грамматики документа

- Документ может быть не предназначен для Вашей системы
- Документ может содержать неправильные данные
- Документ может содержать ошибки в структуре

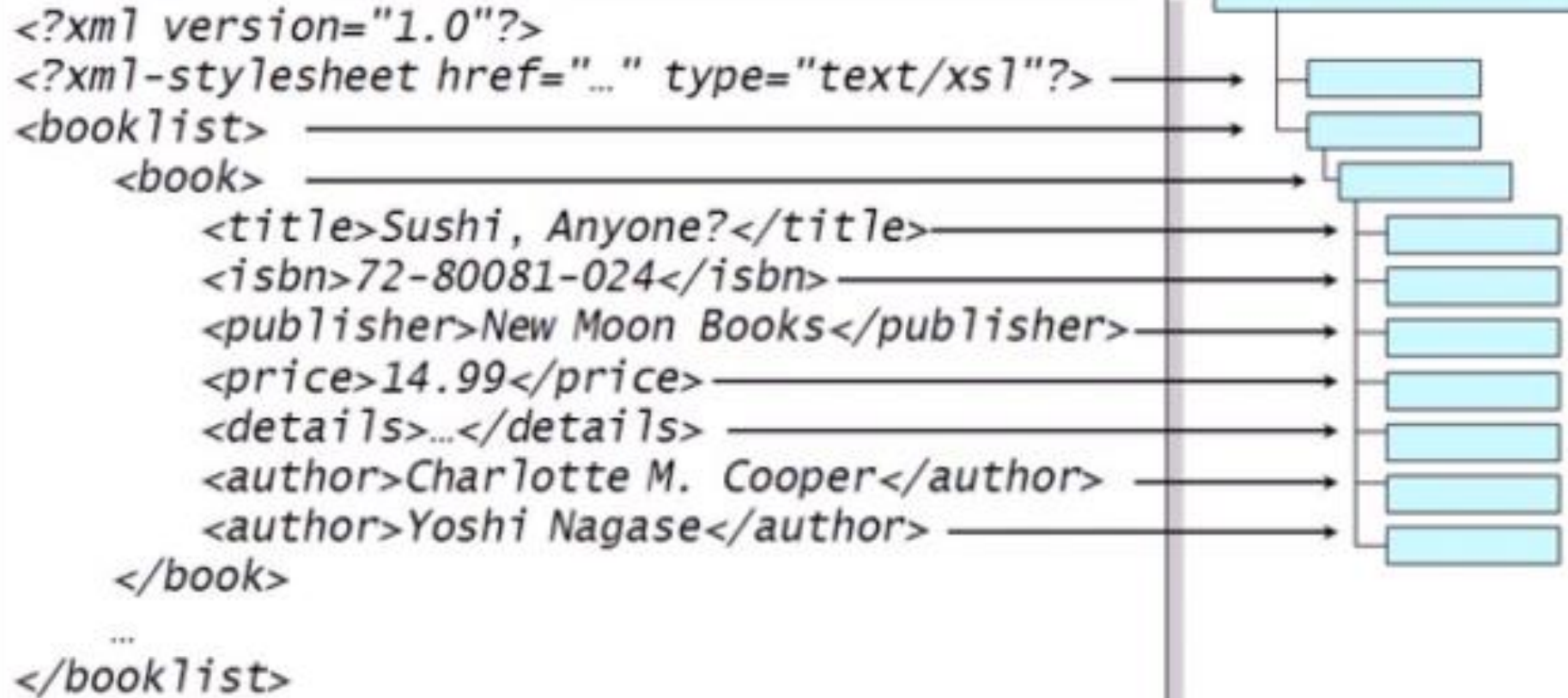
**DOM (от англ. Document Object Model — «объектная модель документа»)** — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

### **Модель DOM:**

- не накладывает ограничений на структуру документа;
- любой документ **известной** структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой объект: элемент, атрибут, текстовый, графический или любой другой объект;
- узлы связаны между собой отношениями «родительский-дочерний».

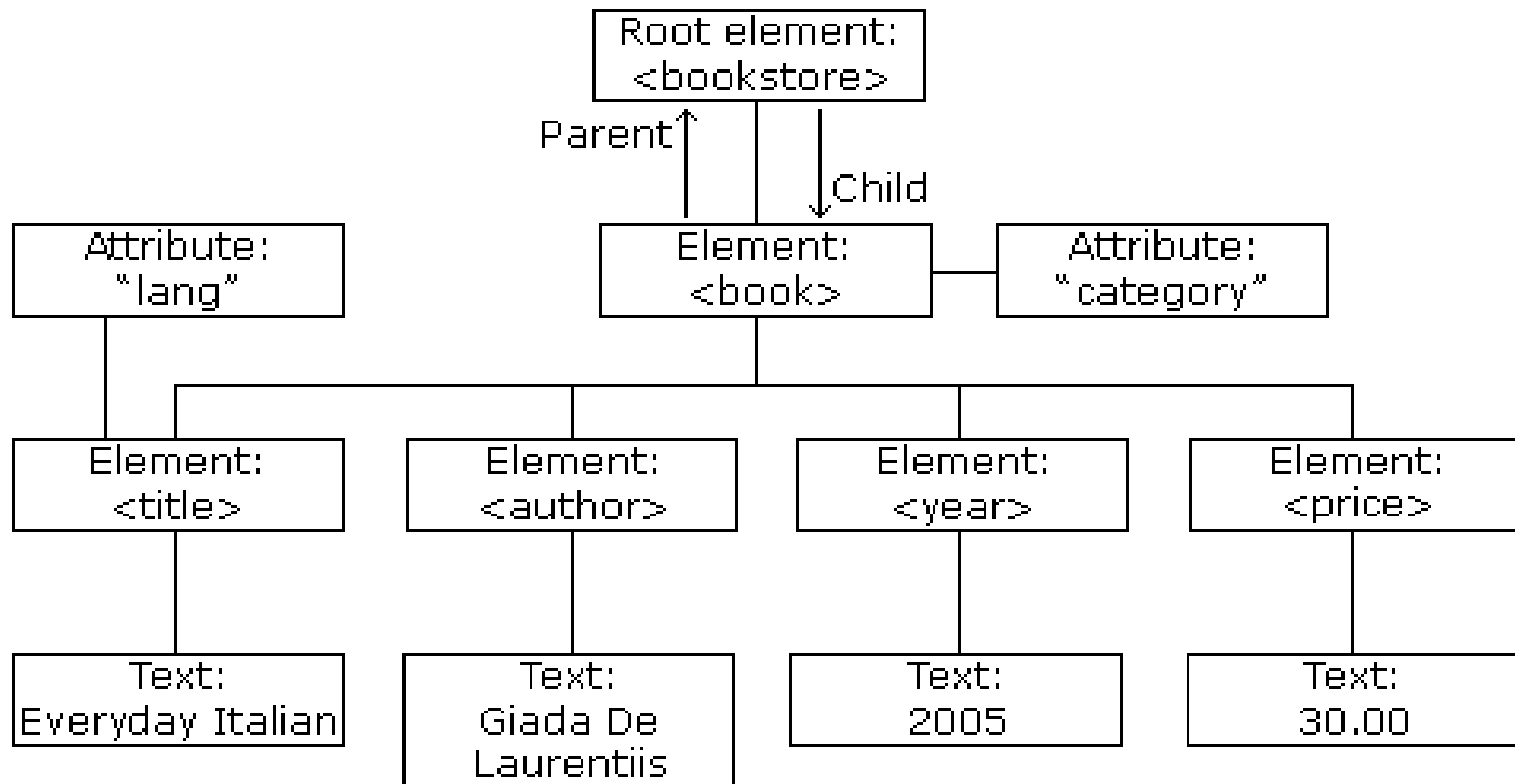


# DOM (data object model) – представление объектов



# DOM (data object model) – представление объектов

Объекты DOM –это узлы связанные друг с другом



**DTD** (англ. **Document Type Definition**  
**определение типа документа**) - язык описания  
структуры документа

Описывает:

- Какие элементы могут присутствовать в документе
- Повторения элементов
- Какие атрибуты могут быть у элементов
- Какие атрибуты обязательны
- Какие сущности могут применяться

## Пример схемы DTD

```
<?xml version="1.0" encoding="UTF-8"?>  
<!ELEMENT notebook(book+)>  
<!ELEMENT book (title, author, price, description?)>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT author (#PCDATA)>  
<!ELEMENT price (#PCDATA)>  
<!ELEMENT description (#PCDATA)>  
<!ATTLIST price currency CDATA #IMPLIED>
```

# Объявление DTD

`<!DOCTYPE корневой_элемент.....>`

Ключевое слово DOCTYPE указывает на то, что это DTD декларация. После него идет имя документа, которого касаются описание правила. Не следует забывать о том, что именем документа является название его корневого элемента.

**Например:**

`<!DOCTYPE notebook>`

**Первый способ** заключается в том, что определение правил задается в самом XML-документе, то есть внутреннее объявления.

`<!DOCTYPE notebook [ ..... ]>`

**Второй способ** определения DTD-декларации используется при описании правил во внешнем файле

`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`

**Третий способ** смешанный.

# Объявление DTD

- *ELEMENT*—определение элемента
- *ATTLIST*—определение атрибута

## Модификаторы

\* - 0 или много

? - 0 или 1

+ - один или много

, - перечисление

| - выбор одного элемента из возможных

() - группировка

1. **EMPTY** - пустой элемент, т.е. элемент может иметь атрибуты, но не может содержать текст или дочерние элементы.
2. **ANY** - что-либо в смеси, т.е. элемент может иметь произвольный текст
3. **(# PCDATA)** - элемент может содержать только обычный текст.
4. **(модель содержимого)** - элемент может содержать только дочерние элементы, через запятую перечисляются в скобках.

## Недостатки и проблемы DTD

- Нет проверки типов данных
- Отличный от XML синтаксис языка
- Нельзя поставить документу в соответствие 2 и более DTD описаний (актуально для составных документов)
- Проблемы с кодировками
- Проблемы с пространством имен

**XML схемы (XSD) – унифицированный способ описания структуры**

- <http://www.w3.org/XML/Schema>
- промышленный стандарт описания XML документа

Описывает:

1. словарь (название элементов и атрибутов)
2. модель содержания (отношения между элементами и атрибутами и их структурами)
3. типы данных (базовые и пользовательские)



# XML схемы – унифицированный способ описания структуры

```
<xs:schema xmlns:xs=
  "http://www.w3.org/2001/XMLSchema">
  <xs:element name="страна"
    type="Страна"/>
  <xs:complexType name="Страна">
    <xs:sequence>
      <xs:element name="название"
        type="xs:string"/>
      <xs:element name="население"
        type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```
<страна xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation = "country.xsd">
  <название>Франция</название>
  <население>59.7</население>
</страна>
```

## Объявление XML-схем

Язык описания схем XSD (*XML Schema Definition Language*) создан как реализация XML.

**Схема XML** записывается в виде документа XML за исключением того, что ее элементы называются компонентами.

Корневой элемент схемы носит имя **schema**, а все остальные компоненты описывают элементы XML документа и их правила использования.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
.....  
</xs:schema>
```

# Пример схемы XSD

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string" />
        <xs:element name="myAge" type="xs:integer" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# Средства обработки XML документа

# XmlDocument

Предоставляет XML-документ.

Название	Описание
<b>Attributes</b>	Возвращает коллекцию атрибутов данного узла.
<b>ChildNodes</b>	Возвращает все дочерние узлы данного узла.
<b>DocumentElement</b>	Возвращает корневой элемент для документа.
<b>FirstChild</b>	Возвращает первый дочерний узел данного узла.
<b>HasChildNodes</b>	Возвращает значение, показывающее наличие дочерних узлов у текущего узла
<b>LastChild</b>	Возвращает последний дочерний узел данного узла.
<b>Name</b>	Возвращает проверенное имя узла.
<b>NodeType</b>	Получает тип текущего узла.
<b>ParentNode</b>	Получает родительский узел данного узла (для узлов, которые могут иметь родительские узлы).
<b>Value</b>	Возвращает или задает значение узла.

Таблица Основные методы

Название	Описание
<b>AppendChild</b>	Добавляет указанный узел в конец списка дочерних узлов данного узла.
<b>CreateAttribute(String)</b>	Создает объект XmlAttribute с указанным свойством Name.
<b>CreateComment</b>	Создает XmlComment, содержащий указанные данные.
<b>CreateElement(String)</b>	Создает элемент с указанным именем.
<b>CreateTextNode</b>	Создает XmlText с указанным текстом.
<b>GetElementById</b>	Возвращает элемент XmlElement с указанным идентификатором.
<b>GetElementsByTagName(String)</b>	Возвращает XmlNodeList, содержащий список всех элементов-потомков, соответствующих указанному свойству Name.
<b>InsertAfter</b>	Вставляет заданный узел сразу после указанного узла-ссылки.
<b>InsertBefore</b>	Вставляет заданный узел сразу перед указанным узлом-ссылкой.
<b>Load(Stream)</b>	Загружает XML-документ из указанного потока.
<b>Load(String)</b>	Загружает XML-документ из указанного URL-адреса.
<b>LoadXml</b>	Загружает XML-документ из указанной строки.
<b>PrependChild</b>	Добавляет указанный узел в начало списка дочерних узлов данного узла
<b>RemoveChild</b>	Удаляет указанный дочерний узел. (Унаследовано от XmlNode.)
<b>Save(Stream)</b>	Сохраняет XML-документ в указанном потоке.
<b>Save(String)</b>	Сохраняет XML-документ в указанном файле.

```
<bookstore>
  <book category="PROGRAMMING">
    <title lang="ru" >Язык C# 4.0</title>
    <author>Г. Шилдт</author>
    <year>2014</year>
    <price>50.00</price>
  </book>
  <book category="PROGRAMMING ">
    <title lang="ru" > .NET</title>
    <author>Э. Троелсен</author>
    <year>2013</year>
    <price>60.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2010</year>
    <price>39.95</price>
  </book>
</bookstore>
```

## Пример использования класса XmlDocument

```
XmlDocument doc = new XmlDocument();  
// загрузка XML документа  
doc.Load("books.xml");  
  
// Отображение содержимого XML.  
Console.WriteLine(doc.InnerText);  
  
Console.WriteLine(new string('-', 20));  
  
// Отображение кода XML документа.  
Console.WriteLine(doc.InnerXml);  
  
Console.WriteLine(new string('-', 20));  
Console.WriteLine(new string('-', 20));  
  
// получение узлов документа по имени book  
XmlNodeList nodes = doc.GetElementsByTagName("book");  
// проходим по каждому узлу  
foreach (XmlNode node in nodes)  
{  
    // вывод значений узлов title и author  
    Console.WriteLine("{0} - {1}",  
        node["title"].InnerText, node["author"].InnerText);  
}
```



file:///G:/\_Работа (нов)/\_ITstep/\_Уч. материал/1. C#/Day19 XML ч1/\_Example(XML)/ExampleSimpleRead

Язык C# 4.0Г. Шиндт201450.00Платформа .NETЭ. Троенсен  
. Ray201039.95

-----  
<?xml version="1.0"?><bookstore><book category="C#"  
ык C# 4.0</title><author>Г. Шиндт</author><year>2014</year></book><book category="PROGRAMMING "><title lang="ru">Язык C# 4.0</title><author>Э. Троенсен</author><year>2013</year><price>50.00</price></book><book category="WEB"><title lang="en">Learning XML</title><author>Эрик Т. Рай</author><year>2010</year><price>39.95</price></book></bookstore>  
-----

Язык C# 4.0 - Г. Шиндт  
Платформа .NET - Э. Троенсен  
Learning XML - Erik T. Ray

```

static void Main(string[] args)
{
    XmlDocument doc = new XmlDocument();
    // разбирает XML и строит его представление в памяти
    doc.Load("books.xml");
    // DocumentElement- корневой элемент XML
    OutputNode(doc.DocumentElement);
}

static void OutputNode(XmlNode node)
{
    // вывод элементов
    Console.WriteLine("Type={0}\tName={1}\tValue={2}",
                      node.NodeType, node.Name, node.Value);

    // есть ли у узла атрибуты?
    if (node.Attributes != null)
    { // перебор атрибутов узла!
        foreach (XmlAttribute attr in node.Attributes)
        { // вывод типа, имени и значения атрибута
            Console.WriteLine("Type= {0}\tName={1}\tValue={2}",
                              attr.NodeType, attr.Name, attr.Value);
        }
    }

    // есть ли у узла потомки?
    if (node.HasChildNodes)
    { // получение набора узлов - потомков
        XmlNodeList children = node.ChildNodes;
        // перебор потомков-узлов
        foreach (XmlNode child in children)
        { OutputNode(child); }
    }
}

```

Type=Element	Name=bookstore	Value=
Type=Element	Name=book	Value=
Type=Attribute	Name=category	Value=PROGRAMMING
Type=Element	Name=title	Value=
Type=Attribute	Name=lang	Value=ru
Type=Text	Name=#text	Value=Язык C# 4.0
Type=Element	Name=author	Value=
Type=Text	Name=#text	Value=Г. Шилдт
Type=Element	Name=year	Value=
Type=Text	Name=#text	Value=2014
Type=Element	Name=price	Value=
Type=Text	Name=#text	Value=50.00
Type=Element	Name=book	Value=
Type=Attribute	Name=category	Value=PROGRAMMING
Type=Element	Name=title	Value=
Type=Attribute	Name=lang	Value=ru
Type=Text	Name=#text	Value=Платформа .NET
Type=Element	Name=author	Value=
Type=Text	Name=#text	Value=Э. Троелсен
Type=Element	Name=year	Value=
Type=Text	Name=#text	Value=2013
Type=Element	Name=price	Value=
Type=Text	Name=#text	Value=60.99
Type=Element	Name=book	Value=
Type=Attribute	Name=category	Value=WEB
Type=Element	Name=title	Value=
Type=Attribute	Name=lang	Value=en
Type=Text	Name=#text	Value=Learning XML
Type=Element	Name=author	Value=
Type=Text	Name=#text	Value=Erik T. Ray
Type=Element	Name=year	Value=
Type=Text	Name=#text	Value=2010
Type=Element	Name=price	Value=
Type=Text	Name=#text	Value=39.95

# XmlNode

Предоставляет отдельный узел в XML-документе.

Название	Описание
<b>AppendChild</b>	Добавляет указанный узел в конец списка дочерних узлов данного узла.
<b>InsertAfter</b>	Вставляет заданный узел сразу после указанного узла-ссылки.
<b>InsertBefore</b>	Вставляет заданный узел сразу перед указанным узлом-ссылкой.
<b>PrependChild</b>	Добавляет указанный узел в начало списка дочерних узлов данного узла.
<b>RemoveAll</b>	Удаляет все дочерние узлы и (или) атрибуты текущего узла.
<b>RemoveChild</b>	Удаляет указанный дочерний узел.
<b>ReplaceChild</b>	Заменяет дочерний узел oldChild на узел newChild.
<b>Supports</b>	Проверяет, присутствует ли определенная функция в реализации DOM

## Пример использования класса XmlNode

```
XmlDocument doc = new XmlDocument();  
// разбирает XML и строит его представление в памяти  
doc.Load("books.xml");  
// получение корневого элемента  
XmlNode root = doc.DocumentElement;  
// удаление дочернего элемента (1-го дочернего)  
root.RemoveChild(root.FirstChild);  
// создание нового элемента  
XmlNode newPaper = doc.CreateElement("NewPaper");  
XmlNode elem1 = doc.CreateElement("title");  
XmlNode elem2 = doc.CreateElement("autor");  
XmlNode elem3 = doc.CreateElement("pages");  
XmlNode elem4 = doc.CreateElement("price");  
  
XmlNode text1 = doc.CreateTextNode("C# 6.0");  
XmlNode text2 = doc.CreateTextNode("Ivanov I.I.");  
XmlNode text3 = doc.CreateTextNode("10");  
XmlNode text4 = doc.CreateTextNode("5.00");  
// Добавление текстовых полей  
elem1.AppendChild(text1); elem2.AppendChild(text2);  
elem3.AppendChild(text3); elem4.AppendChild(text4);  
// Добавление новых элементов  
newPaper.AppendChild(elem1); newPaper.AppendChild(elem2);  
newPaper.AppendChild(elem3); newPaper.AppendChild(elem4);  
// Добавить новый узел в корневой элемент  
root.AppendChild(newPaper); doc.Save("newBooks.xml");
```

## XmlTextReader

Предоставляет средство чтения, обеспечивающее быстрый прямой доступ к данным XML.

Важно! Данный класс эффективнее XmlDocument

- **читает весь документ целиком;**
- **обеспечивает более простое сканирование по документу в поисках элементов, атрибутов**

## Пример использования класса XmlTextReader

```
// открытие файлового потока на файл
using (FileStream stream = new FileStream("books.xml", FileMode.Open))
{
    XmlTextReader xmlReader = new XmlTextReader(stream);
    // чтение информации из xml файла
    while (xmlReader.Read())
    {
        Console.WriteLine("{0,-10} {1,-10} {2,-10}",
                           xmlReader.NodeType,
                           xmlReader.Name,
                           xmlReader.Value);
        Console.WriteLine("-----");
    }
}
```

```
<?xml version="1.0"?>
<bookstore>
  <book category="PROGRAMMING">
    <title lang="ru" >Язык C# 4.0</title>
    <author>Г. Шилдт</author>
    <year>2014</year>
    <price>50.00</price>
  </book>
```

XmlDeclaration	xml	version="1.0"
Whitespace		
Element	bookstore	
Whitespace		
Element	book	
Whitespace		
Element	title	
Text		Язык C# 4.0
EndElement	title	
Whitespace		
Element	author	
Text		Г. Шилдт
EndElement	author	
Whitespace		

## Пример использования класса XmlTextReader

```
XmlTextReader reader = new XmlTextReader("books.xml");  
// Поэлементное чтение XML файла.  
while (reader.Read())  
{ if (reader.NodeType == XmlNodeType.Element)  
    { reader.Read(); // Читаем содержимое узла.  
      Console.WriteLine("{0}:{1}", reader.NodeType, reader.Value);  
    }  
  else  
  { Console.WriteLine("{0}", reader.NodeType);  
  }  
}
```

```
<?xml version="1.0"?>  
<bookstore>  
  <book category="PROGRAMMING">  
    <title lang="ru" >Язык C# 4.0</title>  
    <author>Г. Шилдт</author>  
    <year>2014</year>  
    <price>50.00</price>  
  </book>
```

```
XmlDeclaration  
Whitespace  
Whitespace:  
  
Whitespace:  
  
Text:Язык C# 4.0  
EndElement  
Whitespace  
Text:Г. Шилдт  
EndElement  
Whitespace  
Text:2014  
EndElement  
Whitespace  
Text:50.00  
EndElement  
Whitespace  
EndElement  
Whitespace  
Whitespace:
```



## Пример использования класса XmlTextReader

```
XmlTextReader reader = new XmlTextReader("books.xml");  
// чтение атрибутов!  
while (reader.Read())  
{  
    if (reader.NodeType == XmlNodeType.Element)  
    {  
        // Проверка на тип узла необходима, иначе будут найдены  
        // не только открывающие элементы (XmlNodeType.Element),  
        // но и закрывающие (XmlNodeType.EndElement).  
        if (reader.Name.Equals("title"))  
        {  
            Console.WriteLine("<{0}>", reader.GetAttribute("lang"));  
        }  
    }  
}
```

```
<?xml version="1.0"?>  
<bookstore>  
  <book category="PROGRAMMING">  
    <title lang="ru" >Язык C# 4.0</title>  
    <author>Г. Шилдт</author>  
    <year>2014</year>  
    <price>50.00</price>  
  </book>
```



## Пример использования класса XmlTextReader

```
XmlTextReader reader = new XmlTextReader("books.xml");  
// чтение всех атрибутов  
while (reader.Read())  
{  
    if (reader.NodeType == XmlNodeType.Element)  
    {  
        if (reader.HasAttributes)  
        {  
            while (reader.MoveToNextAttribute())  
            {  
                Console.WriteLine("{0} = {1}", reader.Name, reader.Value);  
            }  
        }  
    }  
}
```

```
<?xml version="1.0"?>  
<bookstore>  
  <book category="PROGRAMMING">  
    <title lang="ru" >Язык C# 4.0</title>  
    <author>Г. Шилдт</author>  
    <year>2014</year>  
    <price>50.00</price>  
  </book>
```

```
category = PROGRAMMING  
lang = ru  
category = PROGRAMMING  
lang = ru  
category = WEB  
lang = en
```

## Пример использования класса XmlTextWriter

// формирование xml-файла:

```
XmlTextWriter xmlWriter =
```

```
    new XmlTextWriter("books.xml", Encoding.Default);
```

```
xmlWriter.WriteStartDocument(); // Заголовок XML - <?xml version="1.0"?>
```

```
xmlWriter.WriteStartElement("Books"); // Корневой элемент - <Books>
```

```
xmlWriter.WriteStartElement("Book");
```

```
// Книга 1 - <Book
```

```
xmlWriter.WriteStartAttribute("Rating");
```

```
// Атрибут - Rating
```

```
xmlWriter.WriteString("10+");
```

```
xmlWriter.WriteEndAttribute();
```

```
// >
```

```
xmlWriter.WriteString("Платформа .NET");
```

```
// Платформа .NET
```

```
xmlWriter.WriteEndElement();
```

```
// </Book>
```

```
xmlWriter.WriteStartElement("Book");
```

```
// <Book>
```

```
xmlWriter.WriteString("Язык C#");
```

```
// Язык C#
```

```
xmlWriter.WriteEndElement();
```

```
// </Book>
```

```
xmlWriter.WriteStartElement("Book");
```

```
// <Book>
```

```
xmlWriter.WriteString("ASP.NET");
```

```
// ASP.NET
```

```
xmlWriter.WriteEndElement();
```

```
// </Book>
```

```
xmlWriter.WriteComment("Строка комментария.");
```

```
xmlWriter.WriteEndElement();
```

```
// </Books>
```

```
xmlWriter.Close();
```

```
<?xml version="1.0" encoding="WINDOWS-1251"?>  
- <Books>  
    <Book Rating="10+">Платформа .NET</Book>  
    <Book>Язык C#</Book>  
    <Book>ASP.NET</Book>  
    <!--Строка комментария.-->  
</Books>
```

# LINQ to XML

```
<?xml version="1.0" encoding="utf-8" ?>
<Employees>
  <Employee>
    <EmpId>1</EmpId>
    <Name>Sam</Name>
    <Sex>Male</Sex>
    <Phone Type="Home">423-555-0124</Phone>
    <Phone Type="Work">424-555-0545</Phone>
    <Address>
      <Street>7A Cox Street</Street>
      <City>Acampo</City>
      <State>CA</State>
      <Zip>95220</Zip>
      <Country>USA</Country>
    </Address>
  </Employee>
  <Employee>
```

// Подключение пространства имен!

using System.Xml.Linq;

```
XDocument xdocument = XDocument.Load("../..\\Employees.xml");  
IEnumerable<XElement> employees = xdocument.Elements();  
// получение структуры всего документа  
foreach (XElement employee in employees)  
{  
    Console.WriteLine(employee);  
}
```

```
<Employees>  
  <Employee>  
    <EmpId>1</EmpId>  
    <Name>Sam</Name>  
    <Sex>Male</Sex>  
    <Phone Type="Home">423-555-0124</Phone>  
    <Phone Type="Work">424-555-0545</Phone>  
    <Address>  
      <Street>7A Cox Street</Street>  
      <City>Acampo</City>  
      <State>CA</State>  
      <Zip>95220</Zip>  
      <Country>USA</Country>  
    </Address>  
  </Employee>  
</Employees>
```

```
XDocument doc = new XDocument();
XElement catalog = new XElement("catalog");
doc.Add(catalog);
XElement book = new XElement("book");
book.Add(new XAttribute("id", "bk101")); //добавляем необходимые атрибуты

XElement author = new XElement("author"); //создаем элемент "author"
author.Value = "Gambardella, Matthew";
book.Add(author);

XElement title = new XElement("title"); //создаем элемент "title"
title.Value = "XML Developer's Guide";
book.Add(title);

XElement genre = new XElement("genre"); //создаем элемент "genre"
genre.Value = "Computer";
book.Add(genre);

XElement price = new XElement("price"); //создаем элемент "price"
price.Value = "44.95";
book.Add(price);

doc.Root.Add(book);
//сохраняем наш документ
doc.Save("books.xml");
```

```
// Получение корневого элемента
XElement rootEmployee = xdocument.Element("Employees");
foreach (XElement itemEmpl in rootEmployee.Elements())
{
    Console.WriteLine("Имя: " + itemEmpl.Element("Name").Value);
    Console.WriteLine("Телефон:" + itemEmpl.Element("Phone").Value);
    if (itemEmpl.Element("Phone").HasAttributes)
    {
        Console.WriteLine(itemEmpl.Element("Phone")
            .Attribute("Type").Value);
    }
    Console.WriteLine(itemEmpl.Element("Address")
        .Element("City").Value);

    Console.WriteLine(itemEmpl.Element("Address")
        .Element("Country").Value);}

```

```
<Employees>
  <Employee>
    <EmpId>1</EmpId>
    <Name>Sam</Name>
    <Sex>Male</Sex>
    <Phone Type="Home">423-555-0124</Phone>
    <Phone Type="Work">424-555-0545</Phone>
    <Address>
      <Street>7A Cox Street</Street>
      <City>Acampo</City>
      <State>CA</State>
      <Zip>95220</Zip>
      <Country>USA</Country>
    </Address>
  </Employee>
  <Employee>
    <EmpId>2</EmpId>
    <Name>Lucy</Name>
    <Sex>Female</Sex>
    <Phone Type="Home">143-555-0763</Phone>
    <Phone Type="Work">144-555-0763</Phone>
    <Address>
      <Street>Alta</Street>
      <City>Acampo</City>
      <State>CA</State>
      <Zip>95220</Zip>
      <Country>USA</Country>
    </Address>
  </Employee>
  <Employee>
    <EmpId>3</EmpId>
    <Name>Kate</Name>
    <Sex>Female</Sex>
    <Phone Type="Home">166-555-0231</Phone>
    <Phone Type="Work">167-555-0231</Phone>
    <Address>
      <Street>Milford</Street>
      <City>Acampo</City>
      <State>CA</State>
      <Zip>95220</Zip>
      <Country>USA</Country>
    </Address>
  </Employee>
  <Employee>
    <EmpId>4</EmpId>
    <Name>Chris</Name>
    <Sex>Male</Sex>
    <Phone Type="Home">564-555-0122</Phone>
    <Phone Type="Work">565-555-0122</Phone>
    <Address>
      <Street>Montara</Street>
      <City>Acampo</City>
      <State>CA</State>
      <Zip>95220</Zip>
      <Country>USA</Country>
    </Address>
  </Employee>
</Employees>

```

```
Имя: Sam
Телефон: 423-555-0124
Home
Acampo
USA
Имя: Lucy
Телефон: 143-555-0763
Home
Alta
USA
Имя: Kate
Телефон: 166-555-0231
Home
Milford
USA
Имя: Chris
Телефон: 564-555-0122
Home
Montara
USA

```



# XPath

**Язык XPath** не является реализацией XML - это язык, который основан на наборе синтаксических правил для выделения различных частей XML документов.

- Основу данного языка составляют выражения различных типов, в состав которых входят логический, числовой и строковый типы.
- В выражениях записываются константы, переменные и функции, которые входят в состав XPath.
- В результате вычисления выражения получаем ссылку на один или несколько участков документа.

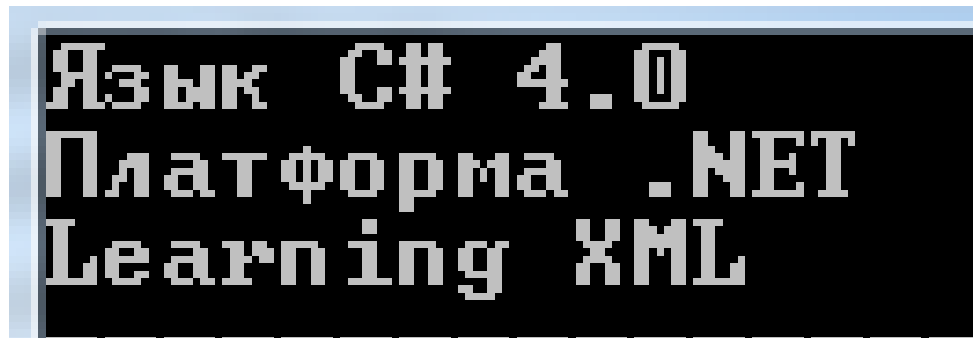
# XPath

// Создание XPath документа.

```
XPathDocument document = new XPathDocument("books.xml");
XPathNavigator navigator = document.CreateNavigator();

XPathNodeIterator iterator1 =
    navigator.Select("bookstore/book/title");
while (iterator1.MoveNext())
{
    Console.WriteLine(iterator1.Current);
}
Console.WriteLine(new string('-', 20));
```

```
<bookstore>
  <book category="PROGRAMMING">
    <title lang="ru" >Язык C# 4.0</title>
    <author>Г. Шилдт</author>
    <year>2014</year>
    <price>50.00</price>
  </book>
  <book category="PROGRAMMING ">
    <title lang="ru" > .NET</title>
    <author>Э. Троелсен</author>
    <year>2013</year>
    <price>60.99</price>
  </book>
</bookstore>
```



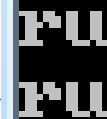
# XPath

```
XPathNodeIterator iterator2 =  
    navigator.Select("bookstore/book[2]/price");  
while (iterator2.MoveNext())  
{  
    Console.WriteLine(iterator2.Current);  
}
```



60.99

```
// Запрос на получение значения атрибута  
XPathNodeIterator iterator3 =  
    navigator.Select("bookstore/book/title/@lang");  
while (iterator3.MoveNext())  
{  
    Console.WriteLine(iterator3.Current);  
}
```



ru  
ru

```
<bookstore>  
  <book category="PROGRAMMING">  
    <title lang="ru" >Язык C# 4.0</title>  
    <author>Г. Шилдт</author>  
    <year>2014</year>  
    <price>50.00</price>  
  </book>  
  <book category="PROGRAMMING ">  
    <title lang="ru" >.NET</title>  
    <author>Э. Троелсен</author>  
    <year>2013</year>  
    <price>60.99</price>  
  </book>  
</bookstore>
```