LINQ-запросы.
Работа с XML-файлами.

Доцент Евдошенко О.И.

XML. Основные понятия

На сегодняшний день XML является одним из распространенных стандартов документов, который позволяет в удобной форме сохранять сложные по структуре данные. Поэтому разработчики платформы .NET включили в фреймворк широкие возможности для работы с XML.

XML (англ. *e**X**tensible **M**arkup **L**anguage)* **—** расширяемый язык разметки.

Рекомендованный Консорциумом Всемирной паутины (W3C) язык разметки.

Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка

XML. Основные понятия

Корневой элемент может включать (а может не включать) вложенные в него ***элементы*** и *символьные данные*, а также комментарии. Вложенные в корневой элемент элементы, в свою очередь, могут включать вложенные в них элементы, символьные данные и комментарии, и так далее.

Элементы документа должны быть *правильно вложены*: любой элемент, начинающийся внутри другого элемента (то есть любой элемент документа, кроме корневого), должен заканчиваться внутри элемента, в котором он начался.

С физической точки зрения, символы, составляющие документ, делятся на ***разметку*** (англ. *markup*) и ***символьные данные*** (англ. *character data*).

Структура XML-документа

```
<?xml version="1.0" encoding="utf-8"?>
<books>
    <book>
      <Author>Пушкин</Author>
      <Title>Евгений Онегин</Title>
      <Description>123</Description>
    </book>
    <book>
      <Author>Булгаков</Author>
      <Title>Роковые яйца</Title>
      <Description>456</Description>
    </book>
    <book>
      <Author>Толстой</Author>
      <Title>Война и мир</Title>
      <Description>789</Description>
    </book>
</books>
```

Объявление XML-документа

Корневой тег

Секция

Поле с данными внутри секции

Корневой тег

Работа с XML с помощью классов System.Xml

Классы, которые позволяют манипулировать xml-документом:

XmlNode: представляет узел xml. В качестве узла может использоваться весь документ, так и отдельный элемент

XmlDocument: представляет весь xml-документ

XmlElement: представляет отдельный элемент. Наследуется от класса XmlNode

XmlAttribute: представляет атрибут элемента

XmlText: представляет значение элемента в виде текста, то есть тот текст, который находится в элементе между его открывающим и закрывающим тегами

XmlComment: представляет комментарий в xml

XmlNodeList: используется для работы со списком узлов

Ключевым классом, который позволяет манипулировать содержимым xml, является XmlNode, поэтому рассмотрим некоторые его основные методы и свойства:

Свойство **Attributes** возвращает объект XmlAttributeCollection, который представляет коллекцию атрибутов

Свойство **ChildNodes** возвращает коллекцию дочерних узлов для данного узла

Свойство **HasChildNodes** возвращает true, если текущий узел имеет дочерние узлы

Свойство **FirstChild** возвращает первый дочерний узел

Свойство **LastChild** возвращает последний дочерний узел

Свойство **InnerText** возвращает текстовое значение узла

Свойство **InnerXml** возвращает всю внутреннюю разметку xml узла

Свойство **Name** возвращает название узла. Например, <user> - значение свойства Name равно "user"

Свойство **ParentNode** возвращает родительский узел у текущего узла

Работа с XML с помощью классов System.Xml

```xml
<?xml version="1.0" encoding="utf-8" ?>
<users>
  <user name="Попов">
    <company>1C</company>
    <age>48</age>
  </user>
  <user name="Иванов">
    <company>МИРЭА</company>
    <age>42</age>
  </user>
</users>
```

```
XmlDocument xDoc = new XmlDocument();
xDoc.Load("D://users.xml");
// получим корневой элемент
XmlElement xRoot = xDoc.DocumentElement;
```

```
// обход всех узлов в корневом элементе
        foreach (XmlNode xnode in xRoot)
        {
            // получаем атрибут name
            if (xnode.Attributes.Count > 0)
            {
                XmlNode attr = xnode.Attributes.GetNamedItem("name");
                if (attr != null) Console.WriteLine(attr.Value);
            }
            // обходим все дочерние узлы элемента user
            foreach (XmlNode childnode in xnode.ChildNodes)
            {
                // если узел - company
                if (childnode.Name == "company")
                {
                    Console.WriteLine($"Компания: {childnode.InnerText}");
                }
                // если узел age
                if (childnode.Name == "age")
                {
                    Console.WriteLine($"Возраст: {childnode.InnerText}");
                }}
            Console.WriteLine();
        }
```

Изменение XML-документа

Для редактирования xml-документа (изменения, добавления, удаления элементов) мы можем воспользоваться методами класса **XmlNode**:

AppendChild: добавляет в конец текущего узла новый дочерний узел

InsertAfter: добавляет новый узел после определенного узла

InsertBefore: добавляет новый узел до определенного узла

RemoveAll: удаляет все дочерние узлы текущего узла

RemoveChild: удаляет у текущего узла один дочерний узел и возвращает его

Класс XmlElement, унаследованный от XmlNode, добавляет еще ряд методов, которые позволяют создавать новые узлы:

CreateNode: создает узел любого типа

CreateElement: создает узел типа XmlDocument

CreateAttribute: создает узел типа XmlAttribute

CreateTextNode: создает узел типа XmlTextNode

CreateComment: создает комментарий

Изменение XML-документа

```
XmlDocument xDoc = new XmlDocument();
xDoc.Load("D://users.xml");
XmlElement xRoot = xDoc.DocumentElement;
// создаем новый элемент user
XmlElement userElem = xDoc.CreateElement("user");
// создаем атрибут name
XmlAttribute nameAttr = xDoc.CreateAttribute("name");
// создаем элементы company и age
XmlElement companyElem = xDoc.CreateElement("company");
XmlElement ageElem = xDoc.CreateElement("age");
// создаем текстовые значения для элементов и атрибута
XmlText nameText = xDoc.CreateTextNode("Иванов Иван Иванович");
XmlText companyText = xDoc.CreateTextNode("ВК");
XmlText ageText = xDoc.CreateTextNode("30");
```

```
nameAttr.AppendChild(nameText);
companyElem.AppendChild(companyText);
 ageElem.AppendChild(ageText);
 userElem.Attributes.Append(nameAttr);
 userElem.AppendChild(companyElem);
 userElem.AppendChild(ageElem);
 xRoot.AppendChild(userElem);
 xDoc.Save("D://users.xml");
```

XPath

XPath представляет язык запросов в XML. Он позволяет выбирать элементы, соответствующие определенному селектору.

Рассмотрим некоторые наиболее распространенные селекторы:

. выбор текущего узла

.. выбор родительского узла

***** выбор всех дочерних узлов текущего узла

user выбор всех узлов с определенным именем, в данном случае с именем "user"

@name выбор атрибута текущего узла, после знака @ указывается название атрибута (в данном случае "name")

@+ выбор всех атрибутов текущего узла

element[3] выбор определенного дочернего узла по индексу, в данном случае третьего узла

//user выбор в документе всех узлов с именем "user"

user[@name=‘Кукушкин’] выбор элементов с определенным значением атрибута. В данном случае выбираются все элементы "user" с атрибутом name=‘Кукушкин'

user[company='Microsoft’] выбор элементов с определенным значением вложенного элемента. В данном случае выбираются все элементы "user", у которых дочерний элемент "company" имеет значение 'Microsoft'

//user/company выбор в документе всех узлов с именем "company", которые находятся в элементах "user"

XPath

Действие запросов XPath основано на применении двух методов класса XmlElement:

SelectSingleNode(): выбор единственного узла из выборки. Если выборка по запросу содержит несколько узлов, то выбирается первый

SelectNodes(): выборка по запросу коллекции узлов в виде объекта XmlNodeList

```
// выбор всех дочерних узлов
XmlNodeList childnodes = xRoot.SelectNodes("*");
foreach (XmlNode n in childnodes)
   Console.WriteLine(n.OuterXml);
```

```
<user name="Попов"><company>1C</company><age>48</age></user>
<user name="Иванов"><company>АГУ</company><age>42</age></user>
```

XmlNodeList childnodes = xRoot.SelectNodes("user") - все узлы <user>

```
foreach (XmlNode n in childnodes)
        Console.WriteLine(n.SelectSingleNode("@name").Value) – вывод значений атрибутов name у элементов user
```

```
Попов
Иванов
```

```
XmlNode childnode = xRoot.SelectSingleNode("user[@name='Попов']");
      if (childnode != null)
         Console.WriteLine(childnode.OuterXml) – выбор узла, у которого атрибут name имеет значение "Попов"
```

```
<user name="Попов"><company>1C</company><age>48</age></user>
```

XmlNode childnode = xRoot.SelectSingleNode("user[company='1C’]») – выбор узла, у которого вложенный элемент "company" имеет значение "1C"

```
<user name="Попов"><company>1C</company><age>48</age></user>
```

```
XmlNodeList childnodes = xRoot.SelectNodes("//user/company");
      foreach (XmlNode n in childnodes)
         Console.WriteLine(n.InnerText) - получить только компании
```

Linq to Xml. Создание документа XML

Еще один подход к работе с Xml представляет технология LINQ to XML. Вся функциональность LINQ to XML содержится в пространстве имен System.Xml.Linq. Рассмотрим основные классы этого пространства имен:

XAttribute: представляет атрибут xml-элемента

XComment: представляет комментарий

XDocument: представляет весь xml-документ

XElement: представляет отдельный xml-элемент

Ключевым классом является XElement, который позволяет получать вложенные элементы и управлять ими. Среди его методов можно отметить следующие:

Add(): добавляет новый атрибут или элемент

Attributes(): возвращает коллекцию атрибутов для данного элемента

Elements(): возвращает все дочерние элементы данного элемента

Remove(): удаляет данный элемент из родительского объекта

RemoveAll(): удаляет все дочерние элементы и атрибуты у данного элемента

Linq to Xml. Создание документа XML

```
XDocument doc = new XDocument(
                    new XElement("books",   //корневой тег
                        new XElement("book",
                            new XElement("Author", "Пушкин"),
                            new XElement("Title", "Евгений Онегин"),
                            new XElement("Description", "123")
                                    ),
                         new XElement("book",
                            new XElement("Author", "Булгаков"), //
поле
                            new XElement("Title", "Роковые яйца"),
                            new XElement("Description", "456")
                                    ),
                         new XElement("book",
                            new XElement("Author", "Толстой"),
                            new XElement("Title", "Война и мир"),
                            new XElement("Description", "789")
                                    )
                            ));

                doc.Save("\\books.xml");
```

секция

Добавление данных в XML-документа

```
XDocument doc;
doc = XDocument.Load("\\books.xml");

doc.Element("books").Add(new XElement("book",
new XElement("Author", Author),
new XElement("Title", Title),
new XElement("Description", Description)));
```

Добавление новой секции

Element – получает первый дочерний элемент

Elements – возвращает фильтрованную коллекцию дочерних элементов

Добавление данных в XML-документа

Для добавления узла в начало списка дочерних узлов служит метод **AddFirst.**

```
XDocument doc;

doc = XDocument.Load("\\books.xml");

doc.Element("books").AddFirst(
                    new XElement("book",
                    new XElement("Author", "Author"),
                    new XElement("Title", "Title"),
                    new XElement("Description", "Description")));
```

Добавление данных в XML-документа

Чтобы вставить узел в определенное место внутри списка дочерних узлов, необходимо получить ссылку либо на предшествующий узел, либо на узел, непосредственно следующий за местом вставки, и вызвать метод **AddBeforeSelf** или **AddAfterSelf**.

```
doc.Element("books")
                .Elements("book")
                .Where(e => ((string)e.Element("Author")) == "Булгаков")
                .Single<XElement>()
                .AddBeforeSelf(
                   new XElement("book",
                       new XElement("Author", "Зощенко М.М."),
                       new XElement("Title", "Нервные люди "),
                       new XElement("Description", "В серию вошли
произведения, включенные в школьную программу и рекомендованные к
внеклассному чтению")));
```

Изменение данных в XML-документе

Изменение значения одного поля в секции:

```
 doc.Element("books").Elements("book").First(e => e.Elements("Author").
Any(n => n.Value.Contains("Булгаков"))).Element("Author").Value="Булгаков
M.A.";
```

Изменение значения нескольких полей в секции:

```
doc.Element("books").Elements("book").First(e => e.Elements("Author").
Any(n => n.Value.Contains("Булгаков"))).ReplaceAll(
          new XElement("Author", "Булгаков Михаил Афанасьевич"),
          new XElement("Title", "Собачье сердце"));
```

Поля, для которых не указано значение будут удалены из секции.

Изменение данных в XML-документе

Метод XElement.SetElementValue()

Обновление значения поля в секции

```
doc.Element("books").Elements("book").First(
e => ((string)e.Element("Author")) ==
"Булгаков").
SetElementValue("Author", "Булгаков М.А.");
```

Добавление нового поля в секцию

```
doc.Elements("books").Elements("book").First(
e => ((string)e.Element("Author")) == "Булгаков").
SetElementValue("Year", "1968");
```

Удаление поля из секции

```
doc.Elements("books").Elements("book").First(
e => ((string)e.Element("Author")) == "Булгаков").
SetElementValue("Description", null);
```

LINQ-запрос к полям XML-документа

```
 var Записи =
             from x in doc.Element("books").Elements("book")
             where (string)x.Element("Author") == "Толстой"
             select x.Element("Title").Value;

 foreach (var x in Записи)
                Console.WriteLine("{0}",
x);
var Записи =
             from x in doc.Element("books").Elements("book")
             where (string)x.Element("Author") == "Толстой"
             select x;


foreach (var x in Записи)
              Console.WriteLine("{0}", x.Element("Title").Value);
```

Удаление узлов из XML-документа

Удаление определенного поля в каждой секции:

```
doc.Descendants().Where(e => e.Name == "Title").Remove();
```

Метод **Descendants** возвращает фильтрованную коллекцию элементов потомков для данного элемента в порядке следования документа. Метод осуществляется рекурсивный обход вниз всего дерева XML с возвратом только элементов с именем **FirstName**, для чего используется операция **Where**. Затем на результирующей последовательности вызывается метод **Remove**.

Удаление секции:

```
doc.Element("books").Elements("book").First(e => e.Elements("Author").
Any(n => n.Value.Contains("Булгаков"))).Remove();
```

LINQ-запрос к полям XML-документа

```
IEnumerable<XElement> elements = doc.Descendants("books");

foreach (var element in elements.Elements())
            {
                Console.WriteLine("{0}", element.Element("Title").Value);
            }
```

```
IEnumerable<XElement> elements =
doc.Descendants("book").Where(e=>e.Element("Author").Value=="Булгаков");

foreach (var element in elements)
            {
                Console.WriteLine("{0}", element.Element("Title").Value);
            }
```

```
var elements = doc.Descendants("book").GroupBy(e => e.Element("Author").Value);

foreach (var element in elements)
{
    Console.WriteLine("{0} \r\n", element.Key);
    foreach (var title in element)
    {
        Console.WriteLine("{0}", title.Element("Title").Value);
    }
}
```

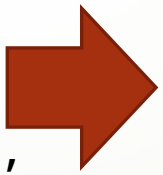
Атрибуты XML-документа

Атрибут реализованный в LINQ to XML с помощью класса **XAttribute**, является парой "имя-значение", хранящейся в коллекции объектов XAttribute, которые относятся к объекту XElement.

```
XElement xBook = new XElement("Author",
    new XAttribute("BirthDay", "1980"));
```

```
XElement xBook = new XElement("Author");
XAttribute xAttr = new XAttribute("BirthDay", "1980");
xBook.Add(xAttr);
```

```
new XElement("books",
    new XElement("book",
    new XElement("Author", new
XAttribute("BirthDay", "1799"), "Пушкин"),
    new XElement("Title", "Евгений
Онегин"),
    new XElement("Description", "123")
```



```
<book>
  <Author BirthDay="1799">Пушкин</Author>
  <Title>Евгений Онегин</Title>
  <Description>123</Description>
 </book>
```

Выборка данных в XML-документ

```
var res = from xe in  doc.Element("books").Elements("book")
                        where (int)xe.Element("Author").Attribute("BirthDay")
== 1799
                        select new
                        {
                            Author = xe.Element("Author").Value,
                            Title = xe.Element("Title").Value
                        };


IEnumerable<XElement> elements =
                from e in doc.Descendants("book")
                where ((int)e.Element("Author").Attribute("BirthDay")) != 1799
                orderby ((string)e.Element("Author"))
                select e;
```

Структура XML-документа

```
<aero>
        <Number>122</Number>
        <From>Санкт-Петербург</From>
          <To>Москва</To>
        <Flights>
              <Flight>
                    <Date>01.10.2017</Date>
                    <Time>11.00</Time>
               </Flight >
               <Flight>
                    <Date>02.10.2017</Date>
                    <Time>11.30</Time>
                </Flight >
        </Flights>
        </aero>
```

Добавление данных в XML-документ

```
doc = XDocument.Load(".\\aero.xml");

            var aero = new XElement("aero",
                new XElement("Number", "122"),
                 new XElement("From", "Санкт-Петербург
                 new XElement("To", "Москва"),
                 new XElement("Flights")
                                        );

            doc.Root.Add(aero);
            doc.Save(".\\aero.xml");
```

Добавление информации о самолёте

```
<aero>
  <Number>122</Number>
  <From>Санкт-Петербург</From>
  <To>Москва</To>
  <Flights />
 </aero>
```

Добавление данных в XML-документ

```
var Flight = new XElement("Flight",
                new XElement("Date", "01.10.2017"),
                new XElement("Time", "11:00")
             );

doc
.Element("aeros")
.Elements("aero")
.First(x =>  (int)x.Element("Number")==122)
.Element("Flights")
.Add(Flight);

             doc.Save(".\\aero.xml");
```

Добавление информации о рейсах

```
<aero>
  <Number>122</Number>
  <From>Санкт-Петербург</From>
  <To>Москва</To>
  <Flights>
   <Flight>
    <Date>01.10.2017</Date>
    <Time>11:00</Time>
   </Flight>
  </Flights>
 </aero>
```

Выборка данных в XML-документ

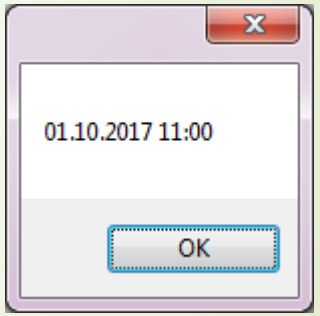
```
 var Flights = doc.Root
.Elements()
.First(x => (int)x.Element("Number")==122)
.Element("Flights");


foreach (var flight in Flights.Elements())
            {

MessageBox.Show(flight.Element("Date").Value + " "
+ flight.Element("Time").Value);

            }
```

Поиск информации о рейсах



Выборка данных в XML-документ

```
 var items = from xe in doc.Element("aeros").Elements("aero")
                          where (int)xe.Element("Number") == 122
                          select new
                          {
                              From = xe.Element("From").Value,
                              To = xe.Element("To").Value,
                              Flights =
xe.Element("Flights").Elements("Flight").Select(p=>p)
                          };

  foreach (var aero in items)
              {
                  Console.WriteLine($"{aero.From} - {aero.To}");

                  foreach (var flights in aero.Flights)
                  Console.WriteLine($"{flights.Element("Date").Value}");
              }
```

Работа с JSON

```
{
 "orderID": 12345,
 "shopperName": "Покупатель",
 "shopperEmail": "ivanov@example.com",
 "contents": [
    {
     "productID": 34,
     "productName": "Компьютер",
    },
    {
     "productID": 56,
     "productName": "Электрический чайник",
     }
  ]
 }
```

Запись — это неупорядоченное множество пар **ключ**:**значение**, заключённое в фигурные скобки *«{ }»*. Ключ описывается **строкой**, между ним и значением стоит символ *«:»*. Пары *ключ-значение* отделяются друг от друга запятыми.

"orderID": 12345

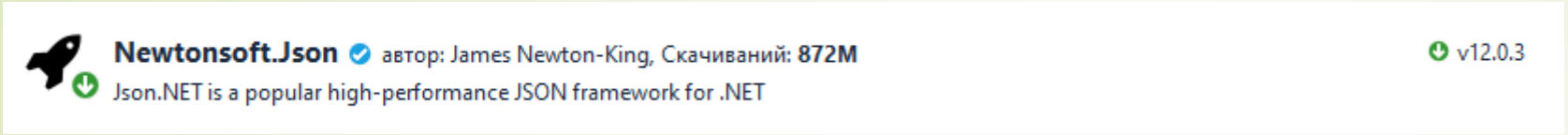
Свойство с именем "orderId" и целочисленным значением 12345

Массив (одномерный) — это упорядоченное множество **значений**. Массив заключается в квадратные скобки *«[ ]»*. Значения разделяются запятыми. Массив может быть пустым, то есть не содержать ни одного значения.

"contents": [...]

Свойство с именем "contents", значение которого является массивом

Работа с JSON




```
using (var sr = new StreamReader("d:\\2.json"))
        {
  var reader = new JsonTextReader(sr);
  var jObject = JObject.Load(reader);
  var orderid= jObject["orderID"].Value<string>();
   textBox1.Text = orderid;
   var contents = jObject["contents"].Select(p => p);
    foreach(var content in contents)
          textBox1.Text += content["productID"].ToString()+ content["productName"].ToString()+"\r\n";

        }
                    var contents =
                        jObject["contents"].Where(p=>p["productID"].Value<int>()==56).Select(p => p);
```

Работа с JSON

```
{
 "MIREA":
 {
        "projectOffices1":
        {
        "Name": "Проектный офис \"Искусственный интеллект\"",
        "structure":
                [
                        {"Name": "Отдел машинного обучения",
                         "Tel":555555  },
                        {"Name": "Отдел разработки и внедения ИИ",
                         "Tel":555556 }
                ]
        },
        "projectOffices2":
        {
        "Name": "Инженерный проектный офис",
        "structure":
                [
                        {"Name": "Инжиниринговый центря"},
                        {"Name": "Каспийская высшая инженерная школа"}
                ]
        }
 }
}
```

```
var name= jObject["MIREA"]["projectOffices1"]["Name"].Value<string>();
textBox1.Text = name;
var contents = jObject["MIREA "]["projectOffices1"]["structure"].Select(p => p);
foreach (var content in contents)
textBox1.Text += content["Name"].ToString()+ content["Tel"].ToString()+"\r\n";
```