LINQ to XML. Методы расширения и дополнительные возможности

Методы расширения LINQ to XML

Некоторые методы классов модели X-DOM можно рассматривать как *запросы*, специфические для интерфейса LINQ to XML, — это методы, возвращающие *последовательности* IEnumerable<T>. Приведем список этих методов, сгруппировав его по «иерархической природе» возвращаемых последовательностей (перед именем метода указывается тип объекта, в котором впервые определен этот метод и, после символа «→», — тип возвращаемой последовательности).

1) Компоненты одного уровня (узлы-братья и элементы-братья):

```
XNode → IEnumerable<XNode>
  NodesAfterSelf()
  NodesBeforeSelf()
XNode → IEnumerable<XElement>
  ElementsAfterSelf([XName name])
  ElementsBeforeSelf([XName name])
```

2) Элементы-предки:

```
XNode → IEnumerable<XElement> Ancestors([XName name])
XElement → IEnumerable<XElement>
  AncestorsAndSelf([XName name])
```

3) Дочерние компоненты (атрибуты, узлы и элементы):

```
XElement → IEnumerable<XAttribute>
  Attributes([XName name])
XContainer → IEnumerable<XNode> Nodes()
XContainer → IEnumerable<XElement>
  Elements([XName name])
```

4) Компоненты-потомки любого уровня вложенности (узлы и элементы):

```
XContainer → IEnumerable<XNode> DescendantNodes()
XElement → IEnumerable<XNode>
  DescendantNodesAndSelf()
XContainer → IEnumerable<XElements>
  Descendants([XName name])
XElement → IEnumerable<XElements>
  DescendantsAndSelf([XName name])
```

Для всех перечисленных методов, *кроме методов группы 1*, в интерфейсе LINQ to XML, определены одноименные *методы расширения*, которые вызываются не для отдельного объекта типа T, а для *последовательности* IEnumerable<T>. Все методы расширения являются статическими методами класса Extensions из пространства System.Xml.Linq. Благодаря методам расширения можно обрабатывать *наборы* узлов, атрибутов или элементов в одном запросе.

Опишем, в качестве примера, методы расширения группы 2:

```
IEnumerable<T> where T : XNode →
IEnumerable<XElement> Ancestors([XName name]):
IEnumerable<XElement> → IEnumerable<XElement>
  AncestorsAndSelf([XName name]):
```

В классе Extensions имеются еще два метода расширения:

```
IEnumerable<XNode> → IEnumerable<XNode> InDocumentOrder()
```

— изменяет порядок исходной последовательности узлов некоторого XML-документа, располагая их в порядке следования в этом документе.

```
IEnumerable<XAttribute> → void Remove()
IEnumerable<T> where T : XNode → void Remove()
```

— удаляет из документа атрибуты или узлы, содержащиеся в исходной последовательности.

Пример (удаление из элемента src всех комментариев):

```
src.DescendantNodes().OfType<XComment>().Remove();
```

Дополнительные возможности технологии LINQ to XML

Загрузка, разбор и сохранение элементов и документов XML

Операции, связанные с загрузкой, разбором и сохранением, предусмотрены как для всего дерева XML (класс XDocument), так и для отдельного элемента XML (класс XElement).

Загрузка: статический метод Load классов XDocument и XElement. Может иметь параметр (типа string — имя файла, типа TextReader — читающий поток, связанный с текстовым файлом или со строкой в памяти, и др.).

Сохранение: экземплярный метод Save этих же классов с аналогичными параметрами (имя файла или пишущий поток). При сохранении можно указать необязательный второй параметр SaveOptions.DisableFormatting, отменяющий автоматическое форматирование сохраняемого документа/элемента.

Разбор строки: статический метод Parse с параметром типа string, содержащим текст документа или элемента XML.

Преобразование в строку: метод ToString с необязательным параметром SaveOptions.DisableFormatting.

При загрузке или разборе документа/элемента XML выполняется проверка того, что он правильно сформирован, и в случае, если это не так, возбуждается исключение XmlException.

При загрузке или разборе документа можно указать дополнительные опции (в виде значений перечисления LoadOptions, объединенных операцией «|»). Например, значение LoadOptions.PreserveWhitespace сохраняет незначащие пробелы документа/элемента XML (по умолчанию все подобные пробелы удаляются). Чтобы после обработки документа незначащие пробелы были сохранены, необходимо использовать параметр SaveOptions.DisableFormatting.

Функциональное конструирование дерева XML

Механизм *функционального конструирования* позволяет построить все дерево XML-документа в одном выражении. Эта возможность обеспечивается с помощью методов, принимающих параметр типа params object[]. В качестве такого параметра можно передавать набор данных, разделяя их запятыми, что позволяет структурировать полученный программный код в соответствии с содержимым создаваемого XML-документа.

Кроме того, в качестве данного параметра можно передавать *последовательность* IEnumerable<T>, полученную в результате выполнения некоторого запроса (так называемое *проецирование последовательностей* в дерево XML).

Методы, принимающие параметр params object[] content:

```
XNode
void AddAfterSelf(params object[] content)
void AddBeforeSelf(params object[] content)
void ReplaceWith(params object[] content)
XContainer
void Add(params object[] content)
void AddFirst(params object[] content)
void ReplaceNodes(params object[] content)
XElement
void ReplaceAttributes(params object[] content)
void ReplaceAll(params object[] content)
конструктор XElement(XName name,
  params object[] content)
XDocument
конструктор XDocument(params object[] content)
конструктор XDocument(XDeclaration declaration,
  params object[] content)
```

В качестве элементов набора content могут выступать любые данные (за некоторыми указанными ниже исключениями).

Правила обработки элементов набора content:

- если элемент равен null, то он игнорируется;
- если элемент является наследником XObject (но не является объектом типа XAttribute), то он добавляется без изменений в соответствующую последовательность узлов дерева/элемента XML;
- если элемент имеет тип XAttribute, то для конструктора класса XElement и его методов ReplaceAttributes и ReplaceAll выполняется добавление этого элемента в последовательность атрибутов; для остальных методов, а также конструктора XDocument возбуждается исключение;
- если элемент имеет тип string, то он неявно преобразуется к объекту XText с указанным текстовым содержимым, после чего добавляется в соответствующую последовательность узлов дерева/элемента XML;
- если тип элемента отличен от string, но при этом реализует интерфейс IEnumerable или IEnumerable<T>, т. е. элемент представляет собой *последовательность*, то выполняется перебор всех элементов этой последовательности с учетом приведенных правил;
- если ни одно из перечисленных условий не выполняется, то элемент преобразуется к своему строковому представлению, которое обрабатывается как элемент типа string (преобразование выполняется по тем же правилам, что и аналогичное преобразование параметра value в методах SetValue, описываемое далее в пункте «Работа со значениями атрибутов и элементов»).

При добавлении к новому элементу существующего узла или атрибута, имеющего непустое свойство Parent (т. е. уже имеющего родителя), всегда выполняется *глубокое клонирование*, т. е. создаются новые объекты-копии как копируемого объекта, так и всех его потомков.

Работа со значениями атрибутов и элементов

Проблемы, связанные с заданием и получением значений атрибутов и элементов:

- изменяемое свойство Value имеет тип string, в то время как в качестве значений атрибутов/элементов часто приходится использовать данные числовых, логических типов или типов «дата/время» (DateTime и TimeSpan);
- при преобразовании данных некоторых типов к строковому представлению необходимо соблюдать требования спецификации XML; например, логические значения должны записываться в нижнем регистре ("true" и "false"), а в представлении дробных чисел с качестве десятичного разделителя должна использоваться *точка*;
- при обработке значения атрибута/элемента (в частности, при его преобразовании к типу данных, отличному от string), заранее может быть неизвестно, существует ли атрибут/элемент с требуемым именем.

Средства модели X-DOM, облегчающие **задание значений** различных типов и их корректное преобразование:

- метод SetValue с параметром value типа object, позволяющий передавать в качестве значения атрибута/элемента данные *любых типов*; при этом если параметр value имеет тип float, double, decimal, bool, DateTime, TimeSpan, то его значение преобразуется к специальному строковому представлению, удовлетворяющему спецификации языка XML (для остальных типов выполняется вызов их методов ToString, *за исключением* типов, унаследованных от класса XObject, для которых возбуждается исключение ArgumentException);
- методы класса XElement SetAttributeValue и SetElementValue с параметрами name типа XName и value типа object позволяют аналогичным образом настраивать свойства атрибутов и дочерних узлов self-элемента. Кроме того, с помощью этих методов можно быстро *добавлять* атрибуты и дочерние элементы (если атрибут или элемент с именем name не найден) и *удалять* их из списка атрибутов и дочерних элементов self-элемента (если в качестве параметра value указано значение null);
- возможность указания данных различных типов в качестве значения атрибута/элемента предусмотрена в *конструкторах* соответствующих классов, а также в методах AddBeforeSelf, AddAfterSelf, Add и AddFirst;

Средства X-DOM, облегчающие **получение значений**:

- *операция приведения к соответствующему типу*, применяемая непосредственно к объекту типа XAttribute или XElement. Для объектов типа XAttribute или XElement можно выполнять приведение ко всем стандартным числовым типам, типам bool, DateTime, TimeSpan, string;
- имеется возможность приведения к Nullable-вариантам всех перечисленных размерных типов; она используется, если заранее неизвестно, существует ли атрибут или дочерний элемент с требуемым именем.

Пример 1. Предположим, что некоторые дочерние элементы элемента src имеют атрибут mark со значением, приводимым к типу int, а некоторые — не имеют подобного атрибута. Требуется найти среднее значение всех атрибутов mark и записать его в качестве значения нового элемента res:

```
var res = new XElement("res", src.Elements()
  .Select(e => (int?)e.Attribute("mark")).Average());
```

Для записи результата в обычную переменную следует учесть, что метод Average, примененный к последовательности Nullable-элементов, также возвращает Nullable-результат:

```
double? avr = src.Elements()
  .Select(e => (int?)e.Attribute("mark")).Average();
```

Пример 2. Предположим теперь, что при обработке дочерних элементов требуется не игнорировать элементы с отсутствующим атрибутом mark, а считать, что для этих элементов атрибут mark имеет *значение по умолчанию*, равное 0:

```
var res = new XElement("res", src.Elements()
  .Select(e =>
    (int?)e.Attribute("mark") ?? 0).Average());
```

Результат метода Average уже не будет иметь Nullable-тип:

```
double avr = src.Elements().Select(e =>
  (int?)e.Attribute("mark") ?? 0).Average();
```

Пример 3 (приведение к типу string для корректной обработки отсутствующих данных). Предположим, что лишь некоторые дочерние элементы элемента src имеют атрибут info. Требуется преобразовать все дочерние элементы таким образом, чтобы значения их атрибутов были добавлены к их текстовому содержимому, а сами атрибуты удалены:

```
foreach (var e in src.Elements())
  e.SetValue(e.Value + (string)e.Attribute("info"));
(from e in src.Elements() select
  e.Attribute("info")).Remove();
```

Замечание 1. В цикле foreach нельзя вызывать метод e.Attribute("info").Remove() (для удаления только что обработанного атрибута), так как при отсутствии атрибута info метод Attribute вернет null. Вариант e.RemoveAttributes() использовать допустимо, однако только в том случае, если элемент e не содержит других атрибутов, поскольку в результате его выполнения будут удалены *все* атрибуты элемента e.

Замечание 2. Если не указать операцию приведения к типу string, то для атрибута, не равного null, будет вызван метод ToString, в результате чего к прежнему содержимому элемента будет добавлено не *значение* атрибута (например, abc), а *полный его текст* (например, info="abc").

Работа с пространствами имен XML-документа

Начало файла с расширением fb2:

```
<?xml version="1.0" encoding="windows-1251"?>
<FictionBook xmlns="http://www.gribuser.ru/xml/fictionbook/2.0"
  xmlns:l="http://www.w3.org/1999/xlink">
  <description>
    <title-info>
```

Если загрузить данный файл в переменную d типа XDocument и попытаться получить ее элемент с именем FictionBook (d.Element("FictionBook")) или дочерний элемент description корневого элемента (d.Root.Element("description")), мы получим в качестве результата значение null.

Причина: для корневого элемента нашего документа (а тем самым и для всех его элементов-потомков) определено *непустое пространство имен*, образующее *начальную часть имени* и этого элемента, и всех его потомков.

Пространство имен определяется с помощью атрибута XML с именем xmlns; для определения пространства имен достаточно обратиться к соответствующей части имени элемента:

```
XNamespace ns = d.Root.Name.Namespace;
Console.WriteLine(ns);
  // http://www.gribuser.ru/xml/fictionbook/2.0
```

На экран будет выведено *строковое представление* объекта ns типа XNamespace (благодаря неявному вызову для него функции ToString); это же представление может быть получено и с помощью его свойства NamespaceName (read-only):

```
Console.WriteLine(ns.NamespaceName);
  // http://www.gribuser.ru/xml/fictionbook/2.0
```

Так выглядит строковое представление полного имени:

```
Console.WriteLine(d.Root.Name);
// {http://www.gribuser.ru/xml/fictionbook/2.0}FictionBook
```

Для доступа по имени к корневому элементу нашего документа можно поступить следующим образом (в результате будет выведено полное содержимое этого элемента):

```
Console.WriteLine
  (d.Element("{" + ns + "}FictionBook"));
```

Однако можно поступить проще: *сложить* объект типа XNamespace и строку, представляющую локальное имя (поскольку в классе XNamespace переопределена операция «+» с операндами типа XNamespace и string, которая возвращает объект XName — полное имя XML-элемента):

```
Console.WriteLine(d.Element(ns + "FictionBook"));
```

Аналогичным образом можно получить доступ к дочерним элементам корневого элемента, например:

```
Console.WriteLine
  (d.Root.Element(ns + "description"));
```

В тексте XML-документа элемент-потомок description не содержит определения пространства имен, однако по правилам языка XML *пространство имен любого элемента XML распространяется на любые его элементы-потомки, если в них это пространство не переопределено*.

Пример программного переопределения имени элемента:

```
d.Root.Element(ns + "description").Name =
  "new-description"; // удалили пространство имен
d.Save("Demo.fb2");
```

Начало документа Demo.fb2 будет выглядеть так:

```
<?xml version="1.0" encoding="windows-1251"?>
<FictionBook
  xmlns:l="http://www.w3.org/1999/xlink"
  xmlns="http://www.gribuser.ru/xml/fictionbook/2.0">
  <new-description xmlns="">
   <title-info xmlns="http://www.gribuser.ru/xml/fictionbook/2.0">
```

Теперь с элементом new-description связывается *пустое* пространство имен, однако у потомка данного элемента автоматически восстанавливается пространство имен, определенное в корневом элементе. Следовательно, *программное изменение пространства имен элемента не приводит к автоматическому изменению пространства имен всех его потомков*. «Групповое» изменение пространств имен можно выполнить путем обработки *последовательности*, например:

```
foreach (var e in
  d.Root.Element("new-description").Descendants())
    e.Name = e.Name.LocalName;
```

Теперь пространство имен элемента title-info (и других потомков элемента new-description) не переопределяется.

```
<?xml version="1.0" encoding="windows-1251"?>
<FictionBook
  xmlns:l="http://www.w3.org/1999/xlink"
  xmlns="http://www.gribuser.ru/xml/fictionbook/2.0">
  <new-description xmlns="">
    <title-info>
```

Префиксы пространства имен

В стандарте XML предусмотрен еще один способ задания пространства имен — с помощью *префикса пространства имен*. Префикс для пространства имен задается с помощью атрибута следующего вида:

```
xmlns:имя_префикса="имя_пространства_имен"
```

В элементе FictionBook так определен префикс l, связываемый с пространством имен http://www.w3.org/1999/xlink.

Определение в некотором элементе префикса пространства имен не приводит к автоматическому связыванию этого элемента с данным пространством имен. Для того чтобы это связывание произошло, необходимо явно указать префикс перед именем элемента (как в открывающем, так и в закрывающем теге), отделив его от имени двоеточием. Однако даже в этом случае *указанное пространство имен не будет по умолчанию связано со всеми потомками данного элемента*.

Благодаря префиксам пространства имен могут быть заданы не только для элементов XML, но и для их *атрибутов*. В качестве примера можно привести само определение префикса, которое выполняется с помощью атрибута, снабженного префиксом xmlns. Префикс xmlns всегда связан со стандартным пространством имен http://www.w3.org/2000/xmlns/; для доступа к этому пространству имен можно использовать статическое свойство Xmlns класса XNamespace.

Пример. Получим имена всех пространств имен, для которых в корневом элементе документа d определены префиксы:

```
foreach (var e in d.Root.Attributes()
  .Where(a => a.Name.Namespace == XNamespace.Xmlns))
    Console.WriteLine(e.Value);
```

Для документов, рассмотренных в предыдущем пункте, будет выведено одно имя: http://www.w3.org/1999/xlink

Как изменится вид XML-документа, если явным образом связать один из его элементов с пространством имен, *для которого задан префикс*?

Пример. Изменим в последнем документе имя пространства имен для элемента new-description на http://www.w3.org/1999/xlink (в данный момент элемент new-description имеет пустое пространство имен:

```
XElement e = d.Root.Element("new-description");
e.Name = (XNamespace)"http://www.w3.org/1999/xlink"
  + e.Name.LocalName;
```

Сохранив измененный документ и открыв его для просмотра, мы увидим в начале документа следующий текст:

```
<?xml version="1.0" encoding="windows-1251"?>
<FictionBook
  xmlns:l="http://www.w3.org/1999/xlink"
  xmlns="http://www.gribuser.ru/xml/fictionbook/2.0">
  <l:new-description xmlns="">
    <title-info>
```

С элементом new-description теперь связано пространство имен, определяемое префиксом l. При этом в элементе new-description по-прежнему определено пустое пространство имен, которое действует *на всех потомков* данного элемента (но не на сам элемент, поскольку для него имя указано явно с помощью префикса). При необходимости можно было бы использовать префикс и для каких-либо потомков, поскольку *область видимости префикса распространяется на всех потомков элемента, в котором префикс определен*.

Если бы мы вывели измененный элемент e на экран оператором Console.WriteLine(e), то его начало выглядело бы так:

```
<l:new-description xmlns="" xmlns:l="http://www.w3.org/1999/xlink">
  <title-info>
```

В данном случае префикс l определяется в самом элементе, поскольку «выше» этого элемента ничего нет, и получить другим способом информацию о значении префикса l нельзя.

Для программного связывания элемента или атрибута с некоторым префиксом, уже определенным в документе, нет необходимости знать имя префикса; достаточно знать имя связанного с ним пространства имен. Имя префикса требуется указывать лишь при *формировании* данного префикса:

```
d.Root.Add(new XAttribute(XNamespace.Xmlns + "pref",
  "aa/bb/cc"));
```

В результате открывающий тег корневого элемента будет дополнен новым атрибутом:

```
xmlns:pref="aa/bb/cc"
```

В файлах fb2 пространство имен http://www.w3.org/1999/xlink используется в специальных атрибутах href, применяемых при организации различных ссылок (перекрестных ссылок, подстрочных примечаний, ссылок на иллюстрации). При этом для поиска атрибутов href несущественно, связан ли с ними префикс соответствующего пространства имен, — достаточно знать, какое именно пространство имен нам требуется.

Пример. Выведем все элементы с атрибутом href:

```
XNamespace ns1 = "http://www.w3.org/1999/xlink";
foreach (var a in d.Descendants().Where(e =>
  e.Attributes(ns1 + "href").Any()))
    Console.WriteLine(a);
```

Возможный результат:

```
<image l:href="#cover.jpg"
  xmlns:l="http://www.w3.org/1999/xlink" />
```

Замечание. В самом fb2-файле тег image не содержит атрибута xmlns, поскольку префикс l *уже определен* в корневом элементе XML-документа:

```
<image l:href="#cover.jpg" />
```

Хранение в XML-документе двоичных данных

Элемент image, рассмотренный в предыдущем пункте, имеет атрибут href со значением #cover.jpg. Естественно предположить, что данный элемент *ссылается на некоторое изображение*, соответствующее обложке книги. Символ # имеет тот же смысл, что и в языке HTML: он означает, что ссылка является *внутренней* и связана с тегом в этом же документе, имеющим *идентификатор* cover.jpg (идентификатор элемента задается специальным атрибутом id).

Каким образом в *текстовом* документе XML можно обеспечить хранение *двоичных* данных (в частности, двоичного изображения)? Для этого используется специальный формат *Base64*. В формате Base64 байты исходного двоичного набора объединяются в *тройки* (размера 24 бита), каждая из которых интерпретируется как *четыре* 6-разрядных двоичных числа. Затем каждое из таких двоичных чисел (находящихся в диапазоне от 0 до 63) переводится в соответствующий *алфавит* формата Base64, содержащий только латинские буквы (26 прописных и 26 строчных), цифры (10) и символы «*» и «/», — всего 64 символа.

Специальным образом обрабатывается ситуация, когда в конце двоичного набора данных имеются один или два отдельных байта, не образующих тройку. В этом случае к этим байтам добавляются нулевые биты, дополняющие их до целого количества 6-битных кодов: 4 бита в случае одиночного байта (в результате будут получены 12 бит, которые будут закодированы двумя символами) и 2 бита в случае двух байт (в результате будут получены 18 бит, которые будут закодированы тремя символами). Для того чтобы дополнить полученный «неполный» набор из двух или трех символов до полной четверки символов, к нему добавляется специальный *символ-заполнитель* «=». Этот 65-й символ, в отличие от перечисленных выше 64 основных символов формата Base64, может встретиться только в конце закодированных данных, причем количество его вхождений лежит в диапазоне от 0 до 2.

Преимущество описанного формата Base64 состоит в том, что он использует лишь 65 отображаемых символов из набора ASCII и, следовательно, будет корректно храниться и отображаться во всех распространенных текстовых кодировках.

Для преобразования массива байт в формат Base64 (и обратного преобразования) проще всего воспользоваться соответствующими статическими методами класса System.Convert:

```
string ToBase64String(byte[] inArray[, int offset,
  int length][, Base64FormattingOptions options]);
int ToBase64CharArray(byte[] inArray, int offsetIn,
  int length, char[] outArray[, int offsetOut][,
  Base64FormattingOptions options]);
byte[] FromBase64String(string s);
byte[] FromBase64CharArray(char[] inArray,
  int offset, int length);
```

Параметры:

- offset — индекс, начиная с которого будут обрабатываться элементы соответствующего массива,
- length — количество обрабатываемых элементов;
- options — перечислимый тип Base64FormattingOptions с двумя возможными значениями: None (результирующая строка не разбивается на части), InsertLineBreaks (результирующая строка разбивается на части по 76 символов, после каждой из которых вставляется символы перехода на новую строку "\r\n"). Отсутствие параметра options означает режим без разбиения на фрагменты.

Возвращаемое значение в методе ToBase64CharArray равно количеству заполненных элементов в результирующем символьном массиве outArray.

В fb2-файлах все двоичные данные (хранящиеся в формате Base64) размещаются в элементах с именем binary. Эти элементы содержат два атрибута:

- id (уникальный идентификатор элемента, позволяющий ссылаться на него с помощью атрибута href; обычно совпадает с именем исходного графического файла);
- content-type (строка, определяющая формат двоичного файла со значениями image/jpeg, image/png, image/bmp).

Пример. Выделим из файла demo.fb2 все иллюстрации и сохраним их в виде «обычных» графических файлов:

```
var d = XDocument.Load("demo.fb2");
XNamespace ns = d.Root.Name.Namespace;
foreach (var e in d.Descendants(ns + "binary"))
{
  byte[] ch = Convert.FromBase64String(e.Value);
  using (var fs =
    File.Create(e.Attribute("id").Value))
      fs.Write(ch, 0, ch.Length);
}
```