**LINQ to XML**

Реферат по дисциплине «Современные технологии программирования»

Выполнил: Студент группы ПМИ-072 Шулаков Егор Валерьевич

Новокузнецкий филиал-институт ГОУ ВПО «Кемеровский государственный университет»

Новокузнецк 2011

**Введение**

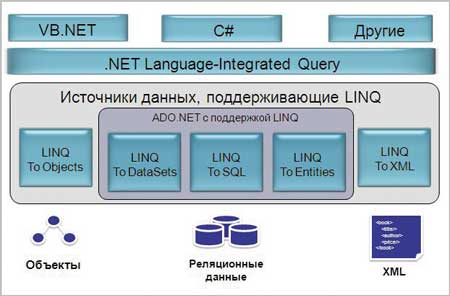
По мере взросления платформы Microsoft .NET и поддерживаемых ею языков С# и VB. стало ясно, что одной из наиболее проблемных областей для разработчиков остает­ся доступ к данным из разных источников. В частности, доступ к базе данных и мани­пуляции XML часто в лучшем случае запутаны, а в худшем — проблематичны.

Проблемы баз данных многочисленны. Во-первых, сложность представляет то, что вы не можете программно взаимодействовать с базой данных на уровне естественного языка. Это приводит к синтаксическим ошибкам, которые не обнаруживаются вплоть до момента запуска. Неправильные ссылки на поля базы данных также не обнаруживаются. Это может пагубно отразиться на программе, особенно если произойдет во время выпол­нения кода обработки ошибок. Нет ничего хуже, чем крах механизма обработки ошибок из-за синтаксически неверного кода, который никогда не тестировался. Иногда это неиз­бежно из-за непредсказуемого поведения ошибки. Наличие кода базы данных, который не проверяется во время компиляции, определенно может привести к этой проблеме.

Вторая проблема связана с неудобством, которое вызвано различными типами дан­ных, используемыми определенным доменом данных, таким как разница между типами базы данных или типами данных XML и естественным языком, на котором написана программа. В частности, серьезно докучать могут времена и даты.

Анализ, итерация и манипуляция XML могут быть достаточно утомительными. Часто фрагмент XML— это все, что вам нужно, но из-за W3C DOM XML API объект XmlDocument должен быть создан только для того, чтобы выполнять различные опера­ции над фрагментом XML.

Вместо того чтобы просто добавить больше классов и методов для постепенного восполнения этих недостатков, команда разработчиков Microsoft решила пойти на один шаг дальше в абстрагировании основ запросов данных из этих конкретных доменов данных. В результате появился LINQ. LINQ — это технология Microsoft, предназначенная для обеспечения механизма поддержки уровня языка для опроса данных всех типов. Эти типы включают массивы и коллекции в памяти, базы данных, документы XML и многое другое. Благодаря LINQ запрос теперь является одним из основных структурных элементов языка, подобно классам, методам, событиям и т. д.



Для разработчиков, создающих запросы, наиболее очевидная "встроенная в язык" часть LINQ — это выражение запросов. Выражения запросов составляются в соответствии с декларативным синтаксисом запроса, который был впервые предложен в C# 3.0. Синтаксис запроса позволяет выполнять достаточно сложную фильтрацию, упорядочение и операции группирования при работе с источниками данных, используя минимум программного кода. Можно использовать одинаковые базовые шаблоны запросов для запроса и преобразования данных в базах данных SQL, наборах данных ADO.NET, документах и потоках XML, а также в коллекциях .NET.

**Основные операции запросов LINQ**

Получение источника данных

В первую очередь в запросе LINQ нужно указать источник данных. В C#, как и в большинстве языков программирования, переменная должна быть объявлена до ее использования. В запросе LINQ первым идет предложение from для указания источника данных (customers) и переменная диапазона (cust).

//queryAllCustomers is an IEnumerable<Customer>

var queryAllCustomers = from cust in customers

select cust;

Переменная диапазона схожа с переменной итерации в цикле foreach за исключением того, что в выражении запроса не происходит фактической итерации. При выполнении запроса переменная диапазона будет использоваться как ссылка на каждый последующий элемент в customers. Поскольку компилятор может определить тип cust, нет необходимости указывать его в явном виде. Дополнительные переменные диапазона могут быть введены предложением let. Дополнительные сведения см. в разделе Предложение let (справочник по C#).

Фильтрация

Возможно, наиболее распространенной операцией запроса является применение фильтра в виде логического выражения. Фильтр приводит к возвращению запросом только тех элементов, для которых выражение является истинным. Результат создается с помощью предложения where. Фильтр фактически указывает элементы для исключения из исходной последовательности. В следующем примере возвращаются только customers, находящиеся в Лондоне.

var queryLondonCustomers = from cust in customers

where cust.City == "London"

select cust;

Для применения нужного числа выражений фильтра в предложении where можно использовать знакомые логические операторы C# AND и OR. Например, для получения только заказчиков из Лондона AND с именем Devon следует написать следующий код.

where cust.City=="London" && cust.Name == "Devon"

Для получения заказчиков из Лондона или Парижа следует написать следующий код.

where cust.City == "London" || cust.City == "Paris"

Порядок

Часто целесообразно отсортировать возвращенные данные. Предложение orderby сортирует элементы возвращаемой последовательности в зависимости от компаратора по умолчанию для сортируемого типа. Например, следующий запрос может быть расширен для сортировки результатов на основе свойства Name. Поскольку Name является строкой, сравнение по умолчанию выполняется в алфавитном порядке от А до Я.

var queryLondonCustomers3 =

from cust in customers

where cust.City == "London"

orderby cust.Name ascending

select cust;

Группировка

Предложение group позволяет группировать результаты на основе указанного ключа. Например, можно указать, что результаты должны быть сгруппированы по City так, чтобы все заказчики из Лондона или Парижа оказались в отдельных группах. В этом случае ключом является cust.City.

// queryCustomersByCity is an IEnumerable<IGrouping<string, Customer>>

var queryCustomersByCity =

from cust in customers

group cust by cust.City;

// customerGroup is an IGrouping<string, Customer>

foreach (var customerGroup in queryCustomersByCity)

{

Console.WriteLine(customerGroup.Key);

foreach (Customer customer in customerGroup)

{

Console.WriteLine(" {0}", customer.Name);

}

}

Когда запрос завершается предложением group, результаты представляются в виде списка из списков. Каждый элемент в списке является объектом, имеющим член Key и список элементов, сгруппированных по этому ключу. При итерации запроса, создающего последовательность групп, необходимо использовать вложенный цикл foreach. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл — итерацию членов каждой группы.

Если необходимо ссылаться на результаты операции группировки, можно использовать ключевое слово into для создания идентификатора, который можно будет запрашивать. Следующий запрос возвращает только те группы, которые содержат более двух заказчиков.

// custQuery is an IEnumerable<IGrouping<string, Customer>>

var custQuery =

from cust in customers

group cust by cust.City into custGroup

where custGroup.Count() > 2

orderby custGroup.Key

select custGroup;

Соединение

Операции соединения создают связи между последовательностями, неявно смоделированными в источниках данных. Например, можно выполнить соединение для поиска всех клиентов и распространителей, которые находятся в одном месте. В LINQ предложение join всегда работает с коллекциями объектов, а не непосредственно с таблицами базы данных.

var innerJoinQuery =

from cust in customers

join dist in distributors on cust.City equals dist.City

select new { CustomerName = cust.Name, DistributorName = dist.Name };

В LINQ нет необходимости использовать join так часто, как в SQL, поскольку внешние ключи в LINQ представлены в объектной модели свойствами, содержащими коллекцию элементов. Например, объект Customer содержит коллекцию объектов Order. Вместо выполнения соединения, доступ к заказам можно получить с помощью точечной нотации.

from order in Customer.Orders...

Выбор ( Проецирование )

Предложение select создает результаты запроса и задает форму или тип каждого возвращаемого элемента. Например, можно указать, будут ли результаты состоять из полных объектов Customer, только из одного члена, подмножества членов или некоторых совершенно других типов, на основе вычислений или создания новых объектов. Когда предложение select создает что-либо отличное от копии исходного элемента, операция называется проекцией. Использование проекций для преобразования данных является мощной возможностью выражений запросов LINQ

XML

XML (Extensible Markup Language) - это новый SGML-производный язык разметки документов, позволяющий структурировать информацию разного типа, используя для этого произвольный набор инструкций. Разработчиков Интернет приложений, пытающихся на практике использовать новую технологию, могут заинтересовать конкретные вопросы, связанные с проблемами создания, обработки XML-документов, их отображения на стороне клиента.

Сегодня XML может использоваться в любых приложениях, которым нужна структурированная информация - от сложных геоинформационных систем, с гигантскими объемами передаваемой информации до обычных "однокомпьютерных" программ, использующих этот язык для описания служебной информации. При внимательном взгляде на окружающий нас информационный мир можно выделить множество задач, связанных с созданием и обработкой структурированной информации, для решения которых может использоваться XML:

В первую очередь, эта технология может оказаться полезной для разработчиков сложных информационных систем, с большим количеством приложений, связанных потоками информации самой различной структурой. В этом случае XML - документы выполняют роль универсального формата для обмена информацией между отдельными компонентами большой программы.

XML является базовым стандартом для нового языка описания ресурсов, RDF, позволяющего упростить многие проблемы в Web, связанные с поиском нужной информации, обеспечением контроля за содержимым сетевых ресурсов, создания электронных библиотек и т.д.

Язык XML позволяет описывать данные произвольного типа и используется для представления специализированной информации, например химических, математических, физических формул, медицинских рецептов, нотных записей, и т.д. Это означает, что XML может служить мощным дополнением к HTML для распространения в Web "нестандартной" информации. Возможно, в самом ближайшем будущем XML полностью заменит собой HTML, по крайней мере, первые попытки интеграции этих двух языков уже делаются (спецификация XHTML).

XML-документы могут использоваться в качестве промежуточного формата данных в трехзвенных системах. Обычно схема взаимодействия между серверами приложений и баз данных зависит от конкретной СУБД и диалекта SQL, используемого для доступа к данным. Если же результаты запроса будут представлены в некотором универсальном текстовом формате, то звено СУБД, как таковое, станет "прозрачным" для приложения. Кроме того, сегодня на рассмотрение W3C предложена спецификация нового языка запросов к базам данных XQL, который в будущем может стать альтернативой SQL.

Информация, содержащаяся в XML-документах, может изменяться, передаваться на машину клиента и обновляться по частям. Разрабатываемые спецификации XLink и Xpointer позволят ссылаться на отдельные элементы документа, c учетом их вложенности и значений атрибутов.

Использование стилевых таблиц (XSL) позволяет обеспечить независимое от конкретного устройства вывода отображение XML- документов.

XML может использоваться в обычных приложениях для хранения и обработки структурированных данных в едином формате.

XML-документ представляет собой обычный текстовый файл, в котором при помощи специальных маркеров создаются элементы данных, последовательность и вложенность которых определяет структуру документа и его содержание. Основным достоинством XML документов является то, что при относительно простом способе создания и обработки (обычный текст может редактироваться любым тестовым процессором и обрабатываться стандартными XML анализаторами), они позволяют создавать структурированную информацию, которую хорошо "понимают" компьютеры.

Пример XML-документа:

<?xml version="1.0" encoding="koi-8"?>

<notepad>

<note id="1" date="12/04/99" time="13:40">

<subject>Важная деловая встреча</subject>

<importance/>

<text>

Надо встретиться с <person id="1625">Иваном Ивановичем</person>,

предварительно позвонив ему по телефону <tel>123-12-12</tel>

</text>

</note>

...

<note id="2" date="12/04/99" time="13:58">

<subject>Позвонить домой</subject>

<text>

<tel>124-13-13</tel>

</text>

</note>

</notepad>

LINQ to XML

Microsoft могла бы ограничиться тем, что предоставила бы нам интерфейс LINQ XML API, который позволял бы лишь выполнять запросы LINQ, и все. К счастью для разработчиков XML, они прошли немного дальше. В дополнение к предоставлению поддержки XML запросов LINQ, Microsoft компенсировала многие недостатки стандартного DOM XML API. После нескольких лет мучений с W3C DOM XML API большинству разработчиков стало ясно, что многие задачи оказывается, не так просты, как должны были быть. Имея дело с небольшими фрагментами XML, используя W3C DOM, всегда приходится создавать целый документ XML, даже когда нужно создать всего несколько элементов. Случалось ли вам строить строки, выглядящие как XML вместо применения DOM API, просто потому что это было проще? Уверен, что да.

Несколько ключевых недостатков W3C DOM XML API были преодолены. Была создана новая модель документов. И в результате появился намного более простой и элегантный метод создания деревьев XML. Непомерно раздутый код с появлением LINQ стал достоянием прошлого. Создание полного дерева XML с помощью единственного оператора стало реальностью, благодаря функциональному конструированию. Функциональное конструирование - термин, используемый для описания возможности создания полной иерархии XML в единственном операторе. Уже одно это заставляет ценить LINQ to XML на вес золота. Конечно, это не стало бы частью LINQ, если бы новый XML API не поддерживал запросы LINQ. Именно для этого было добавлено несколько специфичных для XML операций запросов, реализованных в виде расширяющих методов. Комбинация этих новых XML-специфичных операций со стандартными операциями запросов LINQ to Objects создает мощное элегантное решение для нахождения любых нужных данных в дереве XML. LINQ не только поддерживает все это, но комбинируя запрос с функциональным конструированием, вы можете получить трансформации XML. LINQ to XML чрезвычайно гибок.

Конструирование деревьев XML

При чтении кода первого примера сразу становится ясно, что, глядя на этот код создания дерева XML, очень сложно определить схему XML. После создания документа XML мы должны создать некоторого типа узел XML, такой как элемент, установить его значение и добавить к родительскому элементу. Однако каждый из этих трех шагов должен быть выполнен отдельно с использованием W3C DOM API. Это приводит к неясной схеме и объемному коду. Этот API не предусматривает поддержки создания элемента или любого типа узла в определенном месте дерева XML по отношению к его родителю и его инициализации за одну операцию.

LINQ to XML API не только предоставляет ту же возможность создания дерева XML, что и W3C DOM API, но также предлагает новую технику, называемую функциональным конструированием, для создания дерева XML. Функциональное конструирование позволяет схеме диктовать то, как конструируются объекты XML и инициализируются их значения, и все это - одновременно, в единственном операторе. API достигает этого за счет предоставления конструкторов новых XML-объектов, которые принимают в качестве параметров как отдельные объекты, так и их множества, специфицируя их значения. Тип добавляемого объекта или объектов определяет то, где именно в схеме они располагаются. Общий шаблон выглядит так:

XMLOBJECT о =

new XMLOBJECT(OBJECTNAME,

XML0BJECT1,

XML0BJECT2,

XMLOBJECTN);

На заметку! Приведенный фрагмент является просто псевдокодом, предназначенным для иллюстрации шаблона. Ни один из классов, присутствующих в нем, не существует на самом деле; они лишь представляют некоторый концептуальный абстрактный класс XML.

Если вы добавляете к элементу, реализованному классом XElement, атрибут XML, который реализован классом XAttribute из LINQ to XML, этот атрибут становится атрибутом этого элемента. Например, если в предыдущем псевдокоде XML0BJECT1 добавляется ко вновь созданному XMLOBJECT по имени о, и о является XElement, a XMLOBJECT 1 есть XAttribute, то XMLOBJECT 1 становится атрибутом XElement о.

Если вы добавляете XElement к XElement, то добавляемый XElement становится дочерним элементом того, к которому он добавлен. Поэтому, например, если XMLOBJECT1 - элемент и о — элемент, то XMLOBJECT1 становится дочерним элементом о.

При создании экземпляра объекта XMLOBJECT, как показано в приведенном псевдокоде, мы можем специфицировать его содержимое, указав от 1 до N объектов XMLOBJECT. Как вы узнаете далее в разделе "Создание текста с помощью XText", можно даже специфицировать его содержимое, включающее строку, поскольку строка будет автоматически преобразована для вас в XMLOBJECT.

Все это совершенно логично и составляет суть функционального конструирования

XElement xBookParticipant =

new XElement("BookParticipant",

new XElement("FirstName", "Joe"),

new XElement("LastName", "Rattz"));

Console.WriteLine(xBookParticipant.ToString() ) ;

Обратите внимание, что при конструировании элемента по имени BookParticipant я передал два объекта XElement в качестве его значения, каждый из которых стал его дочерним элементом. Также обратите внимание, что при конструировании элементов FirstName и LastName вместо специфицирования дочерних объектов, как это было сделано при конструировании элемента BookParticipant, я указал просто текстовые значения элементов. И вот результат работы этого кода:

<BookParticipant>

<FirstName>Joe</FirstName>

<LastName>Rattz</LastName> </BookParticipant>

Обратите внимание, насколько проще теперь инициализировать в коде схему XML. Также отметьте, насколько менее многословным стал код, чем код первого примера.

XElement xBookParticipants =

new XElement("BookParticipants",

new XElement ("BookParticipant", "

new XAttribute ("type", "Author'V) ,

new XElement("FirstName", "Joe"),

new XElement("LastName", "Rattz")),

new XElement("BookParticipant",

new XAttribute("type", "Editor"),

new XElement("FirstName", "Ewan"),

new XElement("LastName", "Buckingham")));

Console.WriteLine(xBookParticipants.ToString());

Как видите, здесь намного меньше кода пришлось создать, и намного меньше впоследствии сопровождать. К тому же схему намного легче понять, просто прочитав этот код.

И вот вывод:

<BookParticipants>

<BookParticipant type="Author">

<FirstName>Joe</FirstName>

<LastName>Rattz</LastName>

</BookParticipant>

<BookParticipant type="Editor">

<FirstName>Ewan</FirstName>

<LastName>Buckingham</LastName>

</BookParticipant>

</BookParticipants>

Есть еще одно преимущество нового API, которое наглядно проявляется в результате этого примера. Обратите внимание, что вывод сформатирован так, что он выглядит как дерево XML. Если я выведу дерево XML, оно будет выглядеть следующим образом:

<BookParticipants><BookParticipant type="Author"XFirstName>Joe</FirstName>...

Имена XML

Имена XML часто становятся источником сложности при программировании на XML. Имя XML состоит из пространства имен XML (которое также называется URI-кодом пространства имен XML) и локального имени. Пространство имен XML аналогично пространству имен в программах на основе .NET Framework. Позволяет уникально квалифицировать имена элементов и атрибутов. Это помогает избежать конфликтов имен в разных частях XML-документа. При задании пространства имен XML можно выбрать локальное имя, которое должно быть уникальным только по значению пространства имен.

Другим аспектом имен XML являются префиксы пространств имен XML. Именно префиксы создают основную сложность в работе с именами XML. Эти префиксы позволяют создавать ярлык пространства имен XML, что делает XML-документ более организованным и понятным. Однако, чтобы префиксы XML несли значение, необходимо, чтобы они были соотнесены с определенным контекстом, а это вносит дополнительную сложность. Например, префикс XML aw можно ассоциировать с одним пространством имен XML в одной части XML-дерева и с другим пространством имен XML в другой его части.

Одним из преимуществ использования LINQ to XML в C# является отсутствие необходимости использования префиксов XML. Когда LINQ to XML выполняет загрузку или синтаксический анализ XML-документа, каждый префикс XML соотносится с соответствующим пространством имен XML. После этого при работе с документом, в котором используются пространства имен, почти всегда можно оценить пространство имен при помощи URI-кода пространства имен, а не по префиксу. Когда разработчики работают с именами XML на LINQ to XML, они всегда работают с полностью выраженными именами XML (т. е. пространство имен XML и локальное имя). Однако при необходимости LINQ to XML позволяет управлять префиксами пространства имен и обрабатывать их.

При использовании LINQ to XML в сочетании с литералами Visual Basic XML необходимо использовать префиксы пространства имен при работе с документами в пространствах имен.

В LINQ to XML следующий класс представляет имена XML: XName. Имена XML часто появляются в API LINQ to XML, и, когда требуется использовать имя XML, обнаруживается параметр XName. Однако напрямую работать с XName приходится редко. XName содержит неявное преобразование строки.

Чтобы создать элемент или атрибут, находящийся в пространстве имен, необходимо сначала объявить и инициализировать объект XNamespace. Затем следует использовать перегруженный оператор сложения для объединения пространства имен с локальным именем, выраженным строкой.

В следующем примере создается документ с одним пространством имен. По умолчанию LINQ to XML сериализует документ с использованием пространства имен по умолчанию.

// Create an XML tree in a namespace.

XNamespace aw = "http://www.adventure-works.com";

XElement root = new XElement(aw + "Root",

new XElement(aw + "Child", "child content")

);

Console.WriteLine(root);

Этот пример выводит следующие данные.

xmlLang

<Root xmlns="http://www.adventure-works.com">

<Child>child content</Child>

</Root>

В следующем примере создается документ с одним пространством имен. Он также создает атрибут, который объявляет пространство имен с префиксом пространства имен. Создать атрибут, объявляющий пространство имен с префиксом, можно, указав имя атрибута в качестве префикса пространства имен и поместив его в пространство имен Xmlns. Значение этого атрибута представляет собой URI пространства имен.

// Create an XML tree in a namespace, with a specified prefix

XNamespace aw = "http://www.adventure-works.com";

XElement root = new XElement(aw + "Root",

new XAttribute(XNamespace.Xmlns + "aw", "http://www.adventure-works.com"),

new XElement(aw + "Child", "child content")

);

Console.WriteLine(root);

Этот пример выводит следующие данные.

xmlLang

<aw:Root xmlns:aw="http://www.adventure-works.com">

<aw:Child>child content</aw:Child>

</aw:Root>

Следующий пример иллюстрирует создание документа, содержащего два пространства имен. Одно из них — пространство имен по умолчанию. Другое — пространство имен с префиксом.

При включении атрибутов пространств имен в корневой элемент пространства имен сериализуются, так что http://www.adventure-works.com становится пространством имен по умолчанию, а www.fourthcoffee.com сериализуется с префиксом «fc». Чтобы создать атрибут, объявляющий применяемое по умолчанию пространство имен, необходимо создать атрибут с именем «xmlns» без пространства имен. Значение этого атрибута является используемым по умолчанию идентификатором URI пространства имен.

// The http://www.adventure-works.com namespace is forced to be the default namespace.

XNamespace aw = "http://www.adventure-works.com";

XNamespace fc = "www.fourthcoffee.com";

XElement root = new XElement(aw + "Root",

new XAttribute("xmlns", "http://www.adventure-works.com"),

new XAttribute(XNamespace.Xmlns + "fc", "www.fourthcoffee.com"),

new XElement(fc + "Child",

new XElement(aw + "DifferentChild", "other content")

),

new XElement(aw + "Child2", "c2 content"),

new XElement(fc + "Child3", "c3 content")

);

Console.WriteLine(root);

Этот пример выводит следующие данные.

xmlLang

<Root xmlns="http://www.adventure-works.com" xmlns:fc="www.fourthcoffee.com">

<fc:Child>

<DifferentChild>other content</DifferentChild>

</fc:Child>

<Child2>c2 content</Child2>

<fc:Child3>c3 content</fc:Child3>

</Root>

В этом примере создается документ, который содержит два пространства имен с префиксами.

XNamespace aw = "http://www.adventure-works.com";

XNamespace fc = "www.fourthcoffee.com";

XElement root = new XElement(aw + "Root",

new XAttribute(XNamespace.Xmlns + "aw", aw.NamespaceName),

new XAttribute(XNamespace.Xmlns + "fc", fc.NamespaceName),

new XElement(fc + "Child",

new XElement(aw + "DifferentChild", "other content")

),

new XElement(aw + "Child2", "c2 content"),

new XElement(fc + "Child3", "c3 content")

);

Console.WriteLine(root);

Этот пример выводит следующие данные.

xmlLang

<aw:Root xmlns:aw="http://www.adventure-works.com" xmlns:fc="www.fourthcoffee.com">

<fc:Child>

<aw:DifferentChild>other content</aw:DifferentChild>

</fc:Child>

<aw:Child2>c2 content</aw:Child2>

<fc:Child3>c3 content</fc:Child3>

</aw:Root>

Другой метод получения того же результата состоит в использовании развернутых имен вместо объявления и создания объекта XNamespace.

Этот подход влияет на производительность. Всякий раз при передаче LINQ to XML строки, содержащей развернутое имя, система должна проанализировать это имя, обнаружить атомизированное пространство имен и атомарное имя. Этот процесс требует затрат процессорного времени. Если производительность является важным фактором, целесообразнее объявить и использовать объект XNamespace явным образом.

Дополнительные сведения, касающиеся повышения производительности, см. в разделе Предварительная атомизация объектов XName (LINQ to XML).

// Create an XML tree in a namespace, with a specified prefix

XElement root = new XElement("{http://www.adventure-works.com}Root",

new XAttribute(XNamespace.Xmlns + "aw", "http://www.adventure-works.com"),

new XElement("{http://www.adventure-works.com}Child", "child content")

);

Console.WriteLine(root);

Этот пример выводит следующие данные.

xmlLang

<aw:Root xmlns:aw="http://www.adventure-works.com">

<aw:Child>child content</aw:Child>

</aw:Root>

Извлечения значения элемента

В этом разделе показано получение значений элементов. Это можно сделать двумя основными способами. Первый способ состоит в приведении XElement или XAttribute к желаемому типу. Оператор явного преобразования, который преобразует содержимое элемента или атрибута в указанный тип и присваивает полученное значение указанной переменной. Иначе можно использовать свойство XElement.Value или XAttribute.Value.

Однако при использовании C# приведение, как правило, является лучшим подходом. В частности, становится проще написание кода, обеспечивающего получение значения элемента или атрибута, который может существовать или не существовать, после приведения элемента (или атрибута) к типу, допускающему значение NULL. Это демонстрирует последний пример из данного раздела. Однако нельзя так задать содержимое элемента через приведение, как через свойство XElement.Value.

При использовании Visual Basic лучшим подходом является использование свойства XElement.Value.

Пример

Для получения значения элемента нужно просто привести объект XElement к желаемому типу. Всегда можно привести элемент к строке следующим образом.

XElement e = new XElement("StringElement", "abcde");

Console.WriteLine(e);

Console.WriteLine("Value of e:" + (string)e);

Этот пример выводит следующие данные.

<StringElement>abcde</StringElement>

Value of e:abcde

Также можно приводить элементы к типам, отличным от строковых. Например, если имеется элемент, содержащий целое число, его можно привести к типу int, как показано в следующем коде.

XElement e = new XElement("Age", "44");

Console.WriteLine(e);

Console.WriteLine("Value of e:" + (int)e);

Этот пример выводит следующие данные.

<Age>44</Age>

Value of e:44

LINQ to XML предусматривает операторы явного приведения для следующих типов данных: string, bool, bool?, int, int?, uint, uint?, long, long?, ulong, ulong?, float, float?, double, double?, decimal, decimal?, DateTime, DateTime?, TimeSpan, TimeSpan?, GUID и GUID?.

LINQ to XMLпредоставляет аналогичные операторы приведения для объектов XAttribute.

Свойство Value может использоваться для получения содержимого элемента.

XElement e = new XElement("StringElement", "abcde");

Console.WriteLine(e);

Console.WriteLine("Value of e:" + e.Value);

Этот пример выводит следующие данные:

<StringElement>abcde</StringElement>

Value of e:abcde

Список литературы

1. Статья «Язык XML - Описание технологии». Взято с www.codenet.ru

2. Pro LINQ: Language Integrated Query in C# 2008 Joseph C. Rattz, Jr.

3. Статьи с http://msdn.microsoft.com

4. C# 2010: ускоренный курс для профессионалов, Трей Нэш

5. Статьи с www.wikipedia.org