2010

LINQ to XML 编程基础

Via C#

System.Xml.Linq 命名空间及其相应的类赋予了 LINQ to XML 强大的功能。这些类提供了轻松处理 XML 的能力，而不需要借助于复杂的、有时会很繁琐的技术，如 DOM 和 XQuery。

Sunny D.D
www.sunnycoder.cn
www.cnblogs.com/sunnycoder
sunny19788989@gmail.com
2010-01-30

目录

一、LINQ to XML 编程基础

1、LINQ to XML 类

System.Xml.Linq 命名空间含有 19 个类，下表列出了它们的名称及其描述：

类	描述
XAttribute	表示一个 XML 属性
XCData	表示一个 CDATA 文本节点
XComment	表示一个 XML 注释
XContainer	适用于可能具有子节点的所有节点的抽象基类
XDeclaration	表示一个 XML 声明
XDocument	表示一个 XML 文档
XDocumentType	表示一个 XML 文档类型定义 (DTD)
XElement	表示一个 XML 元素
XName	表示一个 XML 元素或属性的名称
XNamespace	表示一个 XML 的命名空间
XNode	一个抽象类，它表示 XML 树的节点
XNodeDocumentOrderComparer	提供用于比较节点的文档顺序的功能
XNodeEqualityComparer	提供用于比较节点的值是否相等的功能
XObject	XNode 和 XAttribute 的抽象基类
XObjectChange	XObject 引发事件时的事件类型
XObjectChangeEventArgs	为 Changing 和 Changed 事件提供数据
XProcessingInstruction	表示一个 XML 处理指令
XText	表示一个文本节点

以下的代码演示了如何使用 LINQ to XML 来快速创建一个 xml：

```
public static void CreateDocument()
{
    XDocument xdoc = new XDocument
    (
        new XDeclaration("1.0", "utf-8", "yes"),
        new XElement("Root","root")
    );
    xdoc.Save(path);
}
```

运行该示例将会得到一个 xml 文件，其内容为：

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

可以看出微软在 LINQ 上投入了很大的精力，使我们在编程时感觉到很舒服。下面将详细介绍处理 XML 时使用最多的三个类：XElement、XAttribute 和 XDocument。如果掌握了这些类，使用 LINQ to XML 时将会感到很顺手。

2、XElement 类

XElement 类是 LINQ to XML 中的基础类之一。它表示一个 XML 元素。可以使用该类创建元素；更

改元素内容；添加、更改或删除子元素；向元素中添加属性；或以文本格式序列化元素内容。 还可以与 System.Xml 中的其他类（例如 XmlReader、XmlWriter 和 XslCompiledTransform）进行互操作。

使用 LINQ to XML 创建 xml 文档有很多种方式，具体使用哪种方法要根据实际需要。而创建 xml 文档最简单、最常见的方式是使用 XElement 类。以下的代码演示了如何使用 XElement 类创建一个 xml 文档：

```
public static void CreateCategories()
{
    XElement root = new XElement("Categories",
        new XElement("Category",
            new XElement("CategoryID", Guid.NewGuid()),
            new XElement("CategoryName", "Beverages")
            ),
        new XElement("Category",
            new XElement("CategoryID", Guid.NewGuid()),
            new XElement("CategoryName", "Condiments")
            ),
        new XElement("Category",
            new XElement("CategoryID", Guid.NewGuid()),
            new XElement("CategoryName", "Confections")
            )
       );
    root.Save(path);
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories>
  <Category>
    <CategoryID>57485174-46fc-4e8c-8d98-d25b53d504a1</CategoryID>
    <CategoryName>Beverages</CategoryName>
  </Category>
  <Category>
    <CategoryID>1474dde1-8014-48f7-b093-b47ca5d5b770</CategoryID>
    <CategoryName>Condiments</CategoryName>
  </Category>
  <Category>
    <CategoryID>364224e0-e002-4939-90fc-0fd93e0cf35b</CategoryID>
    <CategoryName>Confections</CategoryName>
  </Category>
</Categories>
```

LINQ to XML 的强大之处还在于它可以使用 LINQ to SQL 或者 LINQ to Object 获取数据源，然后填充到 xml 树。以下的示例从 Northwind 数据库中读取 Categories、Products 表中的数据来创建包含产品类别，以及每个类别下所有产品的 xml 文件：

```
public static void CreateCategoriesFromDatabase()
{
    using (NorthwindDataContext db = new NorthwindDataContext())
    {
```

```
        XElement root = new XElement("Categories",
                db.Categories
                .Select
                (
                    c => new XElement
                    (
                        "Category"
                        , new XElement("CategoryID", c.CategoryID)
                        , new XElement("CategoryName", c.CategoryName)
                        , new XElement
                            (
                                "Products"
                                , c.Products
                                .Select
                                (
                                    p => new XElement
                                    (
                                        "Product"
                                        , new XElement("ProductName", p.ProductName)
                                    )
                                )
                                .Take(2)
                            )
                    )
                )
                .Take(3)
            );
        root.Save(path);
    }
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories>
  <Category>
    <CategoryID>1</CategoryID>
    <CategoryName>Beverages</CategoryName>
    <Products>
      <Product>
        <ProductName>Chai</ProductName>
      </Product>
      <Product>
        <ProductName>Chang</ProductName>
      </Product>
    </Products>
  </Category>
```

```
<Category>
  <CategoryID>2</CategoryID>
  <CategoryName>Condiments</CategoryName>
  <Products>
    <Product>
      <ProductName>Aniseed Syrup</ProductName>
    </Product>
    <Product>
      <ProductName>Chef Anton's Cajun Seasoning</ProductName>
    </Product>
  </Products>
</Category>
<Category>
  <CategoryID>3</CategoryID>
  <CategoryName>Confections</CategoryName>
  <Products>
    <Product>
      <ProductName>Pavlova</ProductName>
    </Product>
    <Product>
      <ProductName>Teatime Chocolate Biscuits</ProductName>
    </Product>
  </Products>
</Category>
</Categories>
```

XElement 类包含了许多方法，这些方法使得处理 xml 变得轻而易举。有关这些方法请参照 MSDN。其中，Save、CreateReader、ToString 和 WriteTo 方法是比较常用的三个方法：

方法	参数	返回值	描述
CreateReader	无	System.Xml.XmlReader	创建此节点的 XmlReader
Save	System.String	void	将此元素序列化为文件
	System.IO.TextWriter	void	将此元素序列化为 TextWriter
	System.Xml.XmlWriter	void	将此元素序列化为 XmlWriter
	System.String, System.Xml.Linq.SaveOptions	void	将此元素序列化为文件，并可以选择禁用格式设置
	System.IO.TextWriter, System.Xml.Linq.SaveOptions	void	将此元素序列化为 TextWriter，并可以选择禁用格式设置
WriteTo	System.Xml.XmlWriter	void	将此元素写入 XmlWriter
ToString	无	System.String	返回此节点的缩进 XML
	System.Xml.Linq.SaveOptions	System.String	返回此节点的 XML，并可以选择禁用格式设置

现在有很多使用 XmlReader 作为数据源的应用程序，使用 XElement 可以很方便地提供支持。

3、XAttribute 类

XAttribute 类用来处理元素的属性，属性是与元素相关联的“名称-值”对，每个元素中不能有名称重复的属性。使用 XAttribute 类与使用 XElement 类的操作十分相似，下面的示例演示了如何在创建 xml 树时

为其添加一个属性：

```
public static XElement CreateCategoriesByXAttribute()
{
    XElement root = new XElement("Categories",
        new XElement("Category",
            new XAttribute("CategoryID", Guid.NewGuid()),
            new XElement("CategoryName", "Beverages")
            ),
        new XElement("Category",
            new XAttribute("CategoryID", Guid.NewGuid()),
            new XElement("CategoryName", "Condiments")
            ),
        new XElement("Category",
            new XAttribute("CategoryID", Guid.NewGuid()),
            new XElement("CategoryName", "Confections")
            )
       );
    root.Save(path);
    return root;
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories>
  <Category CategoryID="a6d5ef04-3f83-4e00-aeaf-52444add7570">
    <CategoryName>Beverages</CategoryName>
  </Category>
  <Category CategoryID="67a168d5-6b22-4d82-9bd4-67bec88c2ccb">
    <CategoryName>Condiments</CategoryName>
  </Category>
  <Category CategoryID="17398f4e-5ef1-48da-8a72-1c54371b8e76">
    <CategoryName>Confections</CategoryName>
  </Category>
</Categories>
```

XAttribute 类的方法比较少，常用的三个是：

方法	描述
AddAnnotation	为该属性添加注解
Remove	删除该属性
SetValue	设定该属性的值

以下的示例使用 Remove 来删除第一个元素的 CategoryID 属性：

```
public static void RemoveAttribute()
{
    XElement xdoc = CreateCategoriesByXAttribute();
    XAttribute xattr = xdoc.Element("Category").Attribute("CategoryID");
    xattr.Remove();
    xdoc.Save(path);
```

```
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories>
  <Category>
    <CategoryName>Beverages</CategoryName>
  </Category>
  <Category CategoryID="5c311c1e-ede5-41e5-93f7-5d8b1d7a0346">
    <CategoryName>Condiments</CategoryName>
  </Category>
  <Category CategoryID="bfde8db5-df84-4415-b297-cd04d8db9712">
    <CategoryName>Confections</CategoryName>
  </Category>
</Categories>
```

作为尝试，试一试以下删除属性的方法：

```
public static void RemoveAttributeByDoc()
{
    XElement xdoc = CreateCategoriesByXAttribute();
    XAttribute xattr = xdoc.Attribute("CategoryID");
    xattr.Remove();
    xdoc.Save(path);
}
```

运行该示例将会抛出一个空引用异常，因为元素 Categories 没有一个叫做 CategoryID 的属性。

4、XDocument 类

XDocument 类提供了处理 xml 文档的方法，包括声明、注释和处理指令。一个 XDocument 对象可以包含以下内容：

对象	个数	说明
XDeclaration	一个	用于指定 xml 声明中的重要组成部分，如文档编码和版本等
XElement	一个	指定文档的根元素
XDocumentType	一个	表示一个 xml DTD
XComment	多个	Xml 注释。它不能是第一个参数，因为一个有效的 xml 文档不能以注释作为开始
XProcessingInstruction	多个	为处理 xml 的应用程序指定任何所需信息

下面的示例创建了一个简单的 xml 文档，它包含几个元素和一个属性，以及一个处理指令和一些注释：

```
public static void CreateXDocument()
{
    XDocument xdoc = new XDocument(
            new XProcessingInstruction("xml-stylesheet", "title='EmpInfo'"),
            new XComment("some comments"),
            new XElement("Root",
                    new XElement("Employees",
                            new XElement("Employee",
                                    new XAttribute("id", "1"),
```

```
                                new XElement("Name", "Scott Klein"),
                                new XElement("Title", "Geek"),
                                new XElement("HireDate", "02/05/2007"),
                                new XElement("Gender", "M")
                            )
                        )
                ),
            new XComment("more comments")
        );
    xdoc.Save(path);
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet title='EmpInfo'?>
<!--some comments-->
<Root>
  <Employees>
    <Employee id="1">
      <Name>Scott Klein</Name>
      <Title>Geek</Title>
      <HireDate>02/05/2007</HireDate>
      <Gender>M</Gender>
    </Employee>
  </Employees>
</Root>
<!--more comments-->
```

XDocument 类包含多个与 XElement 类相同的方法，具体内容可以参阅 MSDN。需要注意的是，处理节点和元素的大部分功能都可以通过 XElement 获得，只有当绝对需要文档层次的处理能力，以及需要访问注释、处理指令和声明时，才有使用 XDocument 类的必要。

创建了 xml 文档后，可以使用 NodesAfterSelf 方法返回指定的 XElement 元素之后的所有同级元素。需要注意的是，此方法只包括返回集合中的同级元素，而不包括子代。此方法使用延迟执行。以下代码演示了这一过程：

```
public static void NodesAfterSelf()
{
    XElement root = new XElement("Categories",
        new XElement("Category",
                new XElement("CategoryID", Guid.NewGuid()),
                new XElement("CategoryName", "食品"),
                new XElement("Description", "可以吃的东西")
            )
        );
    foreach (var item in root.Element("Category").Element("CategoryID").NodesAfterSelf())
    {
        Console.WriteLine((item as XElement).Value);
    }
```

```
}
```

执行的结果如下：

使用 LINQ to XML 中的类来处理 xml 十分简单和高效，包括创建、查询和操纵 xml。

二、LINQ to XML 编程概念

本节将介绍 LINQ to XML 编程的相关概念，例如如何加载 xml、创建全新 xml、操纵 xml 的信息以及遍历 xml 文档。

1、加载已有的 xml

使用 LINQ to XML 加载 xml 可以从多种数据源获得，例如字符串、XmlReader、TextReader 或文件。

下面的示例演示了如何从文件中加载 xml：

```
public static void LoadFromFile()
{
    XElement root = XElement.Load(path);
    Console.WriteLine(root.ToString());
}
```

也可以使用 Parse 方法从一个字符串加载 xml：

```
public static void LoadFromString()
{
    XElement root = XElement.Parse(@"
        <Categories>
          <Category>
            <CategoryID>1</CategoryID>
            <CategoryName>Beverages</CategoryName>
            <Description>Soft drinks, coffees, teas, beers, and ales</Description>
          </Category>
        </Categories>
    ");
    Console.WriteLine(root.ToString());
}
```

2、保存 xml

在前面的示例中曾多次调用 XElement 对象的 Save 方法来保存 xml 文档，在这里就不冗述了。

3、创建 xml

在前面的示例中曾多次调用 XElement 对象的构造函数来创建 xml 文档，在这里就不冗述了。需要说明的是，在使用 LINQ to XML 创建 xml 文档时，会有代码缩进，这使代码的可读性大大加强。

4、遍历 xml

使用 LINQ to XML 在 xml 树中遍历 xml 是相当简单的。只需要使用 XElement 和 XAttribute 类中所提供

的方法。Elements 和 Element 方法提供了定位到某个或某些元素的方式。下面的示例演示了如何遍历 xml 树，并获取指定元素的方式：

```
public static void Enum()
{
    XElement root = new XElement("Categories");
    using (NorthwindDataContext db = new NorthwindDataContext())
    {
        root.Add(
                db.Categories
                .Select
                (
                    c => new XElement
                    (
                        "Category"
                        , new XElement("CategoryName", c.CategoryName)
                    )
                )
            );
    }
    foreach (var item in root.Elements("Category"))
    {
        Console.WriteLine(item.Element("CategoryName").Value);
    }
}
}
```

上述代码运行的结果为：

是不是很简单呢？Nodes()、Elements()、Element(name)和 Elements(name)方法为 xml 树的导航提供了基本功能。

5、操纵 xml

LINQ to XML 一个重要的特性是能够方便地修改 xml 树，如添加、删除、更新和复制 xml 文档的内容。

I.插入

使用 XNode 类的插入方法可以方便地向 xml 树添加内容：

方法	说明
AddAfterSelf	紧跟在此节点之后添加指定的内容
AddBeforeSelf	紧邻此节点之前添加指定的内容

在下面的示例中，使用 AddAfterSelf 方法向现有 xml 中添加一个新节点：

```
public static void AddAfterSelf()
```

```
{
    XElement root = XElement.Parse(@"
        <Categories>
          <Category>
            <CategoryID>1</CategoryID>
            <CategoryName>Beverages</CategoryName>
            <Description>Soft drinks, coffees, teas, beers, and ales</Description>
          </Category>
        </Categories>
    ");
    XElement xele = root.Element("Category").Element("CategoryName");
    xele.AddAfterSelf(new XElement("AddDate", DateTime.Now));
    root.Save(path);
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories>
  <Category>
    <CategoryID>1</CategoryID>
    <CategoryName>Beverages</CategoryName>
    <AddDate>2010-01-31T03:08:51.813736+08:00</AddDate>
    <Description>Soft drinks, coffees, teas, beers, and ales</Description>
  </Category>
</Categories>
```

当需要添加一个元素到指定节点之前时，可以使用 AddBeforeSelf 方法。

II.更新

在 LINQ to XML 中更新 xml 内容可以使用以下几种方法：

方法	说明
ReplaceWith	用指定的内容来取代当前元素的内容
ReplaceAll	用指定的内容来取代当前元素的子节点及相关的属性
ReplaceNodes	用指定的内容来取代文档或当前元素的子节点
SetAttributeValue	设置属性的值、添加属性或移除属性
SetElementValue	设置子元素的值、添加子元素或移除子元素

在下面的示例中使用了 ReplaceWith 与 SetElementValue 方法对 xml 进行了更新操作：

```
public static void Update()
{
    XElement root = XElement.Parse(@"
        <Categories>
          <Category>
            <CategoryID>1</CategoryID>
            <CategoryName>Beverages</CategoryName>
            <Description>Soft drinks, coffees, teas, beers, and ales</Description>
          </Category>
        </Categories>
    ");
```

```
    root.Element("Category").Element("CategoryID").ReplaceWith(new XElement("ID", "2"));
    root.Element("Category").SetElementValue("CategoryName", "test data");
    root.Save(path);
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories>
  <Category>
    <ID>2</ID>
    <CategoryName>test data</CategoryName>
    <Description>Soft drinks, coffees, teas, beers, and ales</Description>
  </Category>
</Categories>
```

III.删除

可以使用 Remove(XElement)与 RemoveAll 方法来删除 xml。

在下面的示例中，使用了 RemoveAll 方法：

```
public static void Remove()
{
    XElement root = XElement.Parse(@"
        <Categories>
          <Category>
            <CategoryID>1</CategoryID>
            <CategoryName>Beverages</CategoryName>
            <Description>Soft drinks, coffees, teas, beers, and ales</Description>
          </Category>
        </Categories>
    ");
    root.RemoveAll();
    root.Save(path);
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories />
```

在下面的示例中，使用了 Remove 方法删除了 xml 的 Description 元素：

```
public static void Remove()
{
    XElement root = XElement.Parse(@"
        <Categories>
          <Category>
            <CategoryID>1</CategoryID>
            <CategoryName>Beverages</CategoryName>
            <Description>Soft drinks, coffees, teas, beers, and ales</Description>
          </Category>
        </Categories>
    ");
```

```
    root.Element("Category").Element("Description").Remove();
    root.Save(path);
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories>
  <Category>
    <CategoryID>1</CategoryID>
    <CategoryName>Beverages</CategoryName>
  </Category>
</Categories>
```

6、处理属性

I.添加

LINQ to XML 添加属性与添加元素师类似的，可以使用构造函数或者 Add 方法来添加属性：

```
public static void AddAttribute()
{
    XElement root = new XElement("Categories",
        new XElement("Category",
            new XAttribute("CategoryID","1"),
            new XElement("CategoryName","Beverages"),
            new XElement("Description", "Soft drinks, coffees, teas, beers, and ales")
        )
    );
    root.Element("Category").Add(new XAttribute("AddDate", DateTime.Now.ToShortDateString()));
    root.Save(path);
}
```

运行该示例将会得到一个 xml 文件，其内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<Categories>
  <Category CategoryID="1" AddDate="2010-01-31">
    <CategoryName>Beverages</CategoryName>
    <Description>Soft drinks, coffees, teas, beers, and ales</Description>
  </Category>
</Categories>
```

II.检索

检索属性可以使用 Attribute(name)方法：

```
public static void SelectAttribute()
{
    XElement root = new XElement("Categories",
        new XElement("Category",
            new XAttribute("CategoryID", "1"),
            new XElement("CategoryName", "Beverages"),
            new XElement("Description", "Soft drinks, coffees, teas, beers, and ales")
        )
```

```
    );
    XAttribute xattr = root.Element("Category").Attribute("CategoryID");
    Console.WriteLine(xattr.Name);
    Console.WriteLine(xattr.Value);
}
```

上述代码的运行结果为：

```
CategoryID
1
```

III.删除

删除属性的操作是调用 XAttribute 对象的 Remove 方法来完成的。

本文总结

本文介绍了 LINQ to XML 的编程基础，即 System.Xml.Linq 命名空间中的多个 LINQ to XML 类，这些类都是 LINQ to XML 的支持类，它们使得处理 xml 比使用其他的 xml 工具容易得多。在本文中，着重介绍的是 XElement、XAttribute 和 XDocument。