本文说明 LINQ to XML 和当前主导 XML 编程 API（W3C 文档对象模型 (DOM)）之间的主要区别。

**构造 XML 树的新方式**

在 W3C DOM 中，应当从下至上生成 XML 树；即先创建文档，然后创建元素，再将元素添加到文档。

例如，以下示例使用 DOM 的 Microsoft 实现 [XmlDocument](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.xmldocument) 创建 XML 树的典型方式。

C#复制

XmlDocument doc = new XmlDocument();

XmlElement name = doc.CreateElement("Name");

name.InnerText = "Patrick Hines";

XmlElement phone1 = doc.CreateElement("Phone");

phone1.SetAttribute("Type", "Home");

phone1.InnerText = "206-555-0144";

XmlElement phone2 = doc.CreateElement("Phone");

phone2.SetAttribute("Type", "Work");

phone2.InnerText = "425-555-0145";

XmlElement street1 = doc.CreateElement("Street1");

street1.InnerText = "123 Main St";

XmlElement city = doc.CreateElement("City");

city.InnerText = "Mercer Island";

XmlElement state = doc.CreateElement("State");

state.InnerText = "WA";

XmlElement postal = doc.CreateElement("Postal");

postal.InnerText = "68042";

XmlElement address = doc.CreateElement("Address");

address.AppendChild(street1);

address.AppendChild(city);

address.AppendChild(state);

address.AppendChild(postal);

XmlElement contact = doc.CreateElement("Contact");

contact.AppendChild(name);

contact.AppendChild(phone1);

contact.AppendChild(phone2);

contact.AppendChild(address);

XmlElement contacts = doc.CreateElement("Contacts");

contacts.AppendChild(contact);

doc.AppendChild(contacts);

这种编码风格隐藏了 XML 树的结构。 LINQ to XML 还支持另一种方法，即功能构造，它可以更好地显示结构。 这种方法可以使用 [XElement](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.linq.xelement) 和 [XAttribute](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.linq.xattribute) 构造函数来完成。 在 Visual Basic 中，也可以使用 XML 文本来完成。 此示例演示了使用函数构造来构造相同的 XML 树：

C#复制

XElement contacts =

new XElement("Contacts",

new XElement("Contact",

new XElement("Name", "Patrick Hines"),

new XElement("Phone", "206-555-0144",

new XAttribute("Type", "Home")),

new XElement("phone", "425-555-0145",

new XAttribute("Type", "Work")),

new XElement("Address",

new XElement("Street1", "123 Main St"),

new XElement("City", "Mercer Island"),

new XElement("State", "WA"),

new XElement("Postal", "68042")

)

)

);

请注意，缩进用于构造 XML 树的代码可显示基础 XML 的结构。 Visual Basic 版本使用 XML 文本。

有关详细信息，请参见 [XML 树](https://learn.microsoft.com/zh-cn/dotnet/standard/linq/functional-construction)。

**直接使用 XML 元素**

在使用 XML 编程时，主要关注的通常是 XML 元素，也可能关注属性。 在 LINQ to XML 中，可以直接使用 XML 元素和属性。 例如，可以执行以下操作：

* 创建 XML 元素而根本不使用文档对象。 当必须使用 XML 树的片段时，这可简化编程。
* 直接从 XML 文件加载 T:System.Xml.Linq.XElement 对象。
* 将 T:System.Xml.Linq.XElement 对象序列化为文件或流。

比较而言，W3C DOM 中的 XML 文档用作 XML 树的逻辑容器。 在 DOM 中，必须在 XML 文档的上下文中创建 XML 节点，包括元素和属性。 下面是在 DOM 中创建一个 name 元素的代码片段：

C#复制

XmlDocument doc = new XmlDocument();

XmlElement name = doc.CreateElement("Name");

name.InnerText = "Patrick Hines";

doc.AppendChild(name);

如果要跨多个文档使用某个元素，则必须跨文档导入节点。 LINQ to XML 避免了这一复杂操作。

使用 LINQ to XML 时，仅在文档的根级别添加注释或处理说明时，才需使用 [XDocument](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.linq.xdocument) 类。

**名称和命名空间的简化处理**

处理名称、命名空间和命名空间前缀通常是 XML 编程的复杂部分。 LINQ to XML 完全不需要处理命名空间前缀，从而简化了名称和命名空间。 可以轻松控制命名空间前缀。 但如果您决定不显式控制命名空间前缀，LINQ to XML 将会在序列化过程中分配命名空间前缀（如果需要）或使用默认命名空间进行序列化（如果不需要）。 如果使用默认命名空间，则生成的文档中将没有命名空间前缀。 有关详细信息，请参阅[命名空间概述](https://learn.microsoft.com/zh-cn/dotnet/standard/linq/namespaces-overview)。

DOM 的另一个问题是它不允许您更改节点的名称。 您必须创建新节点并将所有子节点复制到此节点，从而会失去原始节点标识。 LINQ to XML 允许对节点设置 [XName](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.linq.xname) 属性，因此可避免此问题。

**对加载 XML 的静态方法支持**

LINQ to XML 允许您通过使用静态方法而不是实例方法来加载 XML。 这简化了加载和分析。 有关详细信息，请参阅[如何从文件加载 XML](https://learn.microsoft.com/zh-cn/dotnet/standard/linq/load-xml-file)。

**移除对 DTD 构造的支持**

通过移除对实体和实体引用的支持，LINQ to XML 进一步简化了 XML 编程。 实体因管理复杂而很少使用。 移除对它们的支持可提高性能并简化编程接口。 在填充 LINQ to XML 树时，会展开所有 DTD 实体。

**对片段的支持**

LINQ to XML 没有为 XmlDocumentFragment 类提供等效项。 但在很多情况下，XmlDocumentFragment 概念都可以通过执行类型化为[IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) 的 [XNode](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.linq.xnode) 或 [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) 的 [XElement](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.linq.xelement) 的查询来进行处理。

**对 XPathNavigator 的支持**

LINQ to XML 通过 [System.Xml.XPath](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.xpath) 命名空间中的扩展方法提供对 [XPathNavigator](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.xpath.xpathnavigator) 的支持。 有关详细信息，请参阅 [System.Xml.XPath.Extensions](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.xpath.extensions)。

**对空白和缩进的支持**

LINQ to XML 处理空白的方式比 DOM 更简单。

一种常见方案是，读取缩进式 XML，在内存中创建不含任何空白文本节点（即不保留空白）的 XML 树，对 XML 执行一些操作，再保存带缩进的 XML。 在序列化带格式的 XML 时，只保留 XML 树中有意义的空白。 这是 LINQ to XML 的默认行为。

另一个常见的情况是读取和修改已经有意缩进的 XML。 您可能不想以任何方式更改这种缩进。 在 LINQ to XML 中，你可以通过以下方式执行此操作：

* 在加载或解析 XML 时保留空白。
* 序列化 XML 时禁用格式设置。

LINQ to XML 将空白存储为 [XText](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.linq.xtext) 节点，而不像 DOM 那样具有专门的 [Whitespace](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.xmlnodetype#system-xml-xmlnodetype-whitespace) 节点类型。

**对批注的支持**

LINQ to XML 元素支持可扩展的批注集。 这对于跟踪有关元素的杂项信息（如架构信息、关于元素是否绑定到 UI 的信息或应用程序特定的任何其他信息）很有用。 有关详细信息，请参阅 [LINQ to XML 批注](https://learn.microsoft.com/zh-cn/dotnet/standard/linq/linq-xml-annotations)。

**对架构信息的支持**

LINQ to XML 通过 [System.Xml.Schema](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.schema) 命名空间中的扩展方法提供对 XSD 验证的支持。 你可以验证 XML 树是否符合 XSD。 你可以用架构验证后信息集 (PSVI) 填充 XML 树。 有关详细信息，请参阅[如何使用 XSD 进行验证](https://learn.microsoft.com/zh-cn/dotnet/standard/linq/validate-xsd)和 [Extensions](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.schema.extensions)。