XML

Es un lenguaje de marcas personalizadas. Nació para compartir información entre aplicaciones de manera simple.

Su funcionalidad le ha permitido ser utilizada como base de datos, como presentación de datos en web, como formato de exportación e importación.

Es una estructura de carácter jerárquico.

Tiene un nodo principal.

<Datos> //para abrir el nodo

</Datos> // Para cerrar slash (/) y el nombre del nodo es para cerrar el nodo

Cada nodo puede tener uno o varios hijos.

<Datos>

<Empleado\_Administrativo>

<Nombre>Juan</Nombre>

<Apellidos>Perez</Apellidos>

<Cedula>101.313.313</Cedula>

</Empleado\_Administrativo >

<Empleado>

<Nombre>Andres</Nombre>

<Apellidos>Gomez</Apellidos>

<Cedula>100.434.354</Cedula>

<Telefono>312 415 15 85</ Telefono >

</Empleado>

</Datos>

**Los nodos pueden tener atributos**

<Datos>

<Empleado>

<Nombre>Juan</Nombre>

<Apellidos>Perez</Apellidos>

<Cedula FechaExpedicion=”2010/03/16”>>101.313.313</Cedula>

<Empleado>

<Empleado>

<Nombre>Andres</Nombre>

<Apellidos>Comez</Apellidos>

<Cedula FechaExpedicion=”1999/01/25”>100.434.354</Cedula>

<Empleado>

</Datos>

Proviene del ingles, no debe manejar tildes ni “ñ”

<Datos>

<Empleado Nombre=”Mario” Apellidos=”Jaramillo” Telefono=”3233435”>

</Empleado>

<Empleado Nombre=”Ana” Apellidos=”Builes” Telefono=”3434”>

</Empleado>

<Empleado Nombre=”Luis” Apellidos=”Linares” Telefono=”79879798”>

</Empleado>

</Datos>

En C# se utiliza la librería System.Xml;

La clase principal: XmlDocument Es la clase que me permite cargar un archivo XML para procesarlo.

La clase XmlNode permite seleccionar un nodo del documento.

La clase XmlNodes permite seleccionar múltiples nodos

Para consultar información del xml se utiliza **Xpath**: es un lenguaje de consultas estándar para XML.

º<https://msdn.microsoft.com/en-us/library/d271ytdx(v=vs.110).aspx>

https://msdn.microsoft.com/en-us/library/ms256095(v=vs.110).aspx

## **XPath Operators**

Below is a list of the operators that can be used in XPath expressions:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| | | Computes two node-sets | //book | //cd |
| + | Addition | 6 + 4 |
| - | Subtraction | 6 - 4 |
| \* | Multiplication | 6 \* 4 |
| div | Division | 8 div 4 |
| = | Equal | price=9.80 |
| != | Not equal | price!=9.80 |
| < | Less than | price<9.80 |
| <= | Less than or equal to | price<=9.80 |
| > | Greater than | price>9.80 |
| >= | Greater than or equal to | price>=9.80 |
| or | or | price=9.80 or price=9.70 |
| and | and | price>9.00 and price<9.90 |
| mod | Modulus (division remainder) | 5 mod 2 |

using System;

using System.Text;

using System.Xml;

namespace ParsingXml

{

class Program

{

static void Main(string[] args)

{

XmlDocument xmlDoc = new XmlDocument();

xmlDoc.Load("http://rss.cnn.com/rss/edition\_world.rss");

XmlNode titleNode = xmlDoc.SelectSingleNode("//rss/channel/title");

if(titleNode != null)

Console.WriteLine(titleNode.InnerText);

Console.ReadKey();

}

}

}

using System;

using System.Text;

using System.Xml;

namespace ParsingXml

{

class Program

{

static void Main(string[] args)

{

XmlDocument xmlDoc = new XmlDocument();

xmlDoc.Load("http://rss.cnn.com/rss/edition\_world.rss");

XmlNodeList itemNodes = xmlDoc.SelectNodes("//rss/channel/item");

foreach(XmlNode itemNode in itemNodes)

{

XmlNode titleNode = itemNode.SelectSingleNode("title");

XmlNode dateNode = itemNode.SelectSingleNode("pubDate");

if((titleNode != null) && (dateNode != null))

Console.WriteLine(dateNode.InnerText + ": " + titleNode.InnerText);

}

Console.ReadKey();

}

}

}

|  |  |
| --- | --- |
| /Datos | selects the root element |
| /catalog/cd | selects all the cd elements of the catalog element |
| /catalog/cd/price | selects all the price elements of all the cd elements of the catalogelement |
| /catalog/cd[price>10.0] | selects all the cd elements with price greater than 10.0 |
| starts with a slash(/) | represents an absolute path to an element |
| starts with two slashes(//) | selects all elements that satisfy the criteria |
| //cd | selects all cd elements in the document |
| /catalog/cd/title | /catalog/cd/artist | selects all the title and artist elements of the cd elements ofcatalog |
| //title | //artist | selects all the title and artist elements in the document |
| /catalog/cd/\* | selects all the child elements of all cd elements of the catalog element |
| /catalog/\*/price | selects all the price elements that are grandchildren of catalog |
| /\*/\*/price | selects all price elements which have two ancestors |
| //\* | selects all elements in the document |
| /catalog/cd[1] | selects the first cd child of catalog |
| /catalog/cd[last()] | selects the last cd child of catalog |
| /catalog/cd[price] | selects all the cd elements that have price |
| /catalog/cd[price=10.90] | selects cd elements with the price of 10.90 |
| /catalog/cd[Price>10.90]/price | selects all price elements with the price greater than 10.90 |
| //@country | selects all "country" attributes |
| //cd[@country] | selects cd elements which have a "country" attribute |
| //cd[@\*] | selects cd elements which have any attribute |
| //cd[@country='UK'] | selects cd elements with "country" attribute equal to 'UK' |

using System;

using System.Data;

using System.Xml;

using System.IO;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

XmlDocument xmlDoc = new XmlDocument();

xmlDoc.Load("d:\\product.xml");

XmlNodeList nodeList = xmlDoc.DocumentElement.SelectNodes("/Table/Product");

string proID = "", proName = "", price="";

foreach (XmlNode node in nodeList)

{

proID = node.SelectSingleNode("Product\_id").InnerText;

proName = node.SelectSingleNode("Product\_name").InnerText;

price = node.SelectSingleNode("Product\_price").InnerText;

MessageBox.Show(proID + " " + proName + " " + price);

}

}

}

}

|  |  |
| --- | --- |
| **Expression** | **Refers to** |
| ./author | All <author> elements within the current context. Note that this is equivalent to the expression in the next row. |
| author | All <author> elements within the current context. |
| first.name | All <first.name> elements within the current context. |
| /bookstore | The document element (<bookstore>) of this document. |
| //author | All <author> elements in the document. |
| book[/bookstore/@specialty=@style] | All <book> elements whose style attribute value is equal to the specialty attribute value of the <bookstore>element at the root of the document. |
| author/first-name | All <first-name> elements that are children of an <author> element. |
| bookstore//title | All <title> elements one or more levels deep in the <bookstore> element (arbitrary descendants). Note that this is different from the expression in the next row. |
| bookstore/\*/title | All <title> elements that are grandchildren of <bookstore> elements. |
| bookstore//book/excerpt//emph | All <emph> elements anywhere inside <excerpt> children of <book> elements, anywhere inside the <bookstore>element. |
| .//title | All <title> elements one or more levels deep in the current context. Note that this situation is essentially the only one in which the period notation is required. |
| author/\* | All elements that are the children of <author> elements. |
| book/\*/last-name | All <last-name> elements that are grandchildren of <book> elements. |
| \*/\* | All grandchildren elements of the current context. |
| \*[@specialty] | All elements with the specialty attribute. |
| @style | The style attribute of the current context. |
| price/@exchange | The exchange attribute on <price> elements within the current context. |
| price/@exchange/total | Returns an empty node set, because attributes do not contain element children. This expression is allowed by the XML Path Language (XPath) grammar, but is not strictly valid. |
| book[@style] | All <book> elements with style attributes, of the current context. |
| book/@style | The style attribute for all <book> elements of the current context. |
| @\* | All attributes of the current element context. |
| ./first-name | All <first-name> elements in the current context node. Note that this is equivalent to the expression in the next row. |
| first-name | All <first-name> elements in the current context node. |
| author[1] | The first <author> element in the current context node. |
| author[first-name][3] | The third <author> element that has a <first-name> child. |
| my:book | The <book> element from the my namespace. |
| my:\* | All elements from the my namespace. |
| @my:\* | All attributes from the my namespace (this does not include unqualified attributes on elements from the mynamespace). |

Note that indexes are relative to the parent. Consider the following data:

<x>

<y/>

<y/>

</x>

<x>

<y/>

<y/>

</x>

|  |  |
| --- | --- |
| **Expression** | **Refers to** |
| x/y[1] | The first <y> child of each <x>. This is equivalent to the expression in the next row. |
| x/y[position() = 1] | The first <y> child of each <x>. |
| (x/y)[1] | The first <y> from the entire set of <y> children of <x> elements. |
| x[1]/y[2] | The second <y> child of the first <x>. |

The remaining examples refer to the Sample XML file for XPath.

|  |  |
| --- | --- |
| **Expression** | **Refers to** |
| book[last()] | The last <book> element of the current context node. |
| book/author[last()] | The last <author> child of each <book> element of the current context node. |
| (book/author)[last()] | The last <author> element from the entire set of <author> children of <book> elements of the current context node. |
| book[excerpt] | All <book> elements that contain at least one <excerpt> element child. |
| book[excerpt]/title | All <title> elements that are children of <book> elements that also contain at least one <excerpt>element child. |
| book[excerpt]/author[degree] | All <author> elements that contain at least one <degree> element child, and that are children of <book>elements that also contain at least one <excerpt> element. |
| book[author/degree] | All <book> elements that contain <author> children that in turn contain at least one <degree> child. |
| author[degree][award] | All <author> elements that contain at least one <degree> element child and at least one <award> element child. |
| author[degree and award] | All <author> elements that contain at least one <degree> element child and at least one <award> element child. |
| author[(degree or award) and publication] | All <author> elements that contain at least one <degree> or <award> and at least one <publication> as the children |
| author[degree and not(publication)] | All <author> elements that contain at least one <degree> element child and that contain no<publication> element children. |
| author[not(degree or award) and publication] | All <author> elements that contain at least one <publication> element child and contain neither<degree> nor <award> element children. |
| author[last-name = "Bob"] | All <author> elements that contain at least one <last-name> element child with the value Bob. |
| author[last-name[1] = "Bob"] | All <author> elements where the first <last-name> child element has the value Bob. Note that this is equivalent to the expression in the next row. |
| author[last-name [position()=1]= "Bob"] | All <author> elements where the first <last-name> child element has the value Bob. |
| degree[@from != "Harvard"] | All <degree> elements where the from attribute is not equal to "Harvard". |
| author[. = "Matthew Bob"] | All <author> elements whose value is Matthew Bob. |
| author[last-name = "Bob" and ../price > 50] | All <author> elements that contain a <last-name> child element whose value is Bob, and a <price>sibling element whose value is greater than 50. |
| book[position()<= 3] | The first three books (1, 2, 3). |
| author[not(last-name = "Bob")] | All <author> elements that do no contain <last-name> child elements with the value Bob. |
| author[first-name = "Bob"] | All <author> elements that have at least one <first-name> child with the value Bob. |
| author[\* = "Bob"] | all author elements containing any child element whose value is Bob. |
| author[last-name = "Bob" and first-name = "Joe"] | All <author> elements that has a <last-name> child element with the value Bob and a <first-name>child element with the value Joe. |
| price[@intl = "Canada"] | All <price> elements in the context node which have an intl attribute equal to "Canada". |
| degree[position() < 3] | The first two <degree> elements that are children of the context node. |
| p/text()[2] | The second text node in each <p> element in the context node. |
| ancestor::book[1] | The nearest <book> ancestor of the context node. |
| ancestor::book[author][1] | The nearest <book> ancestor of the context node and this <book> element has an <author> element as its child. |
| ancestor::author[parent::book][1] | The nearest <author> ancestor in the current context and this <author> element is a child of a <book>element. |