Serialización con Clases Complejas

.NET Framework proporciona una solución orientada a aspectos para la serialización XML que ahorra una gran cantidad de escritura y depuración en comparación con la implementación manual. El diseño incorrecto de la clase puede descalificarlo para la serialización XML. El siguiente ejemplo se refiere a una clase llamada Persona que, como se puede esperar, representa un ser humano reducido a solo un par de características:

- Nombre - por simplicidad, contiene el nombre completo de la Persona.
- Fecha de nacimiento
- Género - enumeración: hombre / mujer

Ejemplo de clase simple

Aquí viene la implementación simple de la clase Persona que demuestra las capacidades básicas de serialización XML.

```
using System;
using System.Xml.Serialization;
using System.IO;

namespace Ejemplo_Serializacion
{


    public enum Genero
{
    Hombre,
    Mujer
}

public class Persona
{
    public Persona() // Se define el constructor por defecto
    {
    }

    public Persona(string nombre, DateTime dob, Genero genero)
    {
        _nombre = nombre;
        _fechacumpleanios = dob;
        _genero = genero;
    }
    [XmlElement("Nombre")]
    public string Nombre
    {
        get { return _nombre; }
        set { _nombre = value; }
    }
    [XmlElement("Fechacumpleanios")]
```

```
    public DateTime Fechacumpleanios
    {
        get { return _fechacumpleanios; }
        set { _fechacumpleanios = value; }
    }
    [XmlElement("Genero")]
    public Genero Genero
    {
        get { return _genero; }
        set { _genero = value; }
    }
    private string _nombre;
    private DateTime _fechacumpleanios;
    private Genero _genero;
}
class Program
{
    static void Main(string[] args)
    {
        Persona Francisco = new Persona("Francisco", new DateTime(1970, 5, 12),
Genero.Hombre);
        Persona Maria = new Persona("Maria", new DateTime(1972, 3, 6),
Genero.Mujer);
        XmlSerializer serializer = new XmlSerializer(typeof(Persona));
        using (Stream salida = Console.OpenStandardOutput())
        {
            serializer.Serialize(salida, Francisco);
            serializer.Serialize(salida, Maria);
        }
            Console.ReadKey();
    }
}

}
```

La función principal simplemente crea una instancia de dos personas, **Francisco** y **Maria**, y luego las serializa a XML.

Salida XML

```
<?xml version="1.0"?>

<Persona xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <Nombre>Francisco</Nombre>

  <Fechacumpleanios>1970-05-12T00:00:00</Fechacumpleanios>

  <Genero>Hombre</Genero>

</Persona><?xml version="1.0"?>
```

```
<Persona xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <Nombre>Maria</Nombre>

  <Fechacumpleanios>1972-03-06T00:00:00</Fechacumpleanios>

  <Genero>Mujer</Genero>

</Persona>
```

Agregar funcionalidad

Ahora supongamos que queremos algo más. Por ejemplo, podemos decidir agregar propiedades llamadas **Madre** y **Padre**, que representan otras instancias de la clase Persona que en el sistema modelado desempeñan los roles de los padres de la Persona.

Además, agregaremos un método simple **EsHermanoOHermana**, que acepta la referencia de otra Persona y comprueba si tanto la madre como el padre son iguales para ambos objetos. Aquí está el código agregado a la clase Persona:

```
        [XmlElement("Madre")]
        public Persona Madre
        {
            get { return _madre; }
            set { _madre = value; }
        }
        [XmlElement("Padre")]
        public Persona Padre
        {
            get { return _padre; }
            set { _padre = value; }
        }
        public bool EsHermanoOHermana(Persona p)
        {
            return
        object.ReferenceEquals(_madre, p._madre) &&
        object.ReferenceEquals(_padre, p._padre);
        }

        private Persona _madre;

        private Persona _padre;
```

Ahora podemos realizar una pequeña prueba diferente en esta clase extendida. La prueba requerirá que tanto Francisco como **Maria** comprueben que son hermanos antes y después de la deserialización. Para ello, primero serializaremos a Francisco y **Maria** en el flujo de memoria, y luego deserializaremos el flujo en otros objetos que representan a **Francisco** y **Maria** (llamados **Francisco1** y **Maria1**). Aquí está el código:

```
      static void Main(string[] args)
       {
           Persona Francisco = new Persona("Francisco", new DateTime(1970, 5,
12), Genero.Hombre);
           Persona maria = new Persona("Maria", new DateTime(1972, 3, 6),
Genero.Mujer);
           Persona Rosa = new Persona("Rosa", new DateTime(1941, 2, 14),
Genero.Mujer);
           Persona Pedro = new Persona("Pedro", new DateTime(1938, 3, 18),
Genero.Hombre);
           Francisco.Madre = maria.Madre = Rosa;
           Francisco.Padre = maria.Padre = Pedro;

           bool related = Francisco.EsHermanoOHermana(maria);
           Console.WriteLine("{0} y {1} {2} están emparentados.",
       Francisco.Nombre, maria.Nombre, related ? "" : "NO");
           XmlSerializer serializar = new XmlSerializer(typeof(Persona));
           using (MemoryStream ms = new MemoryStream())
           {

               serializar.Serialize(ms, Francisco);
               ms.Position = 0;
               Persona Francisco1 = serializar.Deserialize(ms) as Persona;
               long PosicionInicial = ms.Position;
               serializar.Serialize(ms, maria);
               ms.Position = PosicionInicial;
               Persona Maria1 = serializar.Deserialize(ms) as Persona;

               related = Francisco1.EsHermanoOHermana(Maria1);
               Console.WriteLine("{0} y {1} {2} están emparentados.",
       Francisco1.Nombre, Maria1.Nombre, related ? "" : "NO");

               Console.ReadKey();
           }

       }
```

Resultado

```
Francisco y Maria  están emparentados.

Francisco y Maria NO están emparentados.
```

Ahora puede observar que la serialización y la deserialización han causado una pérdida total de información. Antes de la serialización, tanto **Francisco** como **Maria** habían hecho referencia a los mismos objetos que su madre y su padre. Pero después de la deserialización, esta información simplemente se perdió, dos objetos distintos reemplazaron a **Rosa** y lo mismo le sucedió a **Pedro**. Desde el punto de vista de **Francisco** y **Maria**, sus padres ya no son los mismos objetos y, en consecuencia, ya no se los considera hermanos en nuestro modelo de objetos.

Adaptar el modelo

Este caso simple muestra cómo funciona realmente la serialización. Serializar las propiedades a medida que se dan, y cierta información con respecto a la semántica de nuestros objetos se puede perder en la serialización y no se puede recuperar en la deserialización, lo que se pierde es sobre las referencias: la serialización no conserva las referencias.

La solución a este problema es planificar tales situaciones por adelantado, para ello debemos tener en cuenta que la clase se diseña para ser serializada a XML.

Mostraremos una de esas decisiones usando la clase **Persona** como ejemplo.

A lo que nos enfrentamos con la clase **Persona** es que varios objetos de esta clase se serializan como un todo, no solo un conjunto de objetos no relacionados. Para ese propósito, crearemos otra clase, llamada **SerializablePersonaSet**. Esa clase no será un simple conjunto de objetos Persona, porque para ese propósito podemos usar cualquier colección disponible, al igual que para mantener información sobre los padres de las personas. La clase tiene la palabra **Serializable** precedida para nombrar intencionadamente, para dejar en claro que esta clase no está hecha para actuar como un conjunto funcional de personas, sino solo para permitir la serialización y deserialización de varias personas como una clase de utilidad.

Para diseñar esta clase tenemos que tener en cuenta que las madres y los padres no deben crearse varias veces, sino solo una vez, y sus hijos deben hacer referencia a los mismos objetos después de la deserialización. Lo que vamos a ver en este ejemplo es una técnica a veces llamada deshidratación / rehidratación. Al serializar, reemplazaremos la información que se perdería porque es invisible para el serializador con información más explícita que normalmente es visible para el serializador.

Por ejemplo, si construimos una familia con **Francisco** y **Maria** como niños y **Rosa** y **Pedro** como su madre y su padre, entonces podemos agregar información de que la madre de **Francisco** es la **Persona 2**, basada en cero, y el padre es la **Persona 3**. La misma información de 2 y 3 se adjuntará a **Maria**. Si una Persona no tiene un padre o madre conocidos, los índices correspondientes tendrán un valor inválido, es decir, -1, que indica la situación. Con esta modificación, podemos preservar la información sobre quién está relacionado con quién en una variedad de personas y dejar que las referencias se olviden en el camino. Este es el proceso de deshidratación: los datos insondables se reemplazan con sus equivalentes simples.

Una vez hecho esto, la serie de personas serializadas contendrá a las personas sin madres ni padres y dos series de números enteros que indicarán los índices de las madres y los padres en la matriz original. Esto es algo que la serialización XML no puede realizar por sí sola, pero esos datos adicionales deben proporcionarse específicamente al serializador.

La deserialización se realiza primero deserializando personas, sin madres ni padres. Los índices de los objetos madre/padre se utilizan para establecer las propiedades de la madre y el padre en referencias reales. Este proceso se denomina rehidratación: la

información, los enteros simples que indican las posiciones en la matriz, se reemplazan por un formulario más avanzado: las referencias a los objetos.

Para hacer esto posible, también tenemos que hacer cambios en la clase **Persona**. En primer lugar, **Madre** y **Padre** ya no se serializarán: los atributos de **XmlElement** se reemplazarán con **XmlIgnore**. A continuación, el valor entero se agregará a la clase **Persona** para indicar la posición del objeto en una matriz. Este valor no es de uso general y estará representado por una propiedad interna, por lo que las clases fuera del espacio de nombres no pueden modificarlo y esta no se serializará. Además, las propiedades para el índice de la madre y el padre se agregarán a la clase **Persona**.

Aquí están las modificaciones hechas a la clase **Persona**:

```
        [XmlIgnore()]
        public Persona Madre
        {
            get { return _madre; }
            set { _madre = value; }
        }
        [XmlIgnore()]
        public Persona Padre
        {
            get { return _padre; }
            set { _padre = value; }

        }

        internal int Id
        {
            get { return _id; }
            set { _id = value; }
        }
        internal int PadreId
        {
            get { return _padre == null ? -1 : _padre.Id; }
        }
        internal int MadreId
        {
            get { return _madre == null ? -1 : _madre.Id; }
        }

        private int _id;
```

Teniendo estas propiedades disponibles, podemos diseñar la clase **SerializablePersonaSet** de la siguiente manera.

```
    public class SerializablePersonaSet
    {
        public SerializablePersonaSet()
        {
            _personas = new List<Persona>();
        }
        public void Limpiar()
        {
            _personas.Clear();
```

```
        }
        [XmlArray(ElementName = "Personas", Order = 1)]
        [XmlArrayItem("Persona")]
        public Persona[] Personas
        {
            get { return _personas.ToArray(); }
            set
            {
                _personas.Clear();
                _personas.AddRange(value);
                for (int i = 0; i < value.Length; i++)
                {
                    value[i].Id = i;
                }
            }
        }
        [XmlArray(ElementName = "Madres", Order = 2)]
        [XmlArrayItem("MadreIndex")]
        public int[] MadreIndices
        {
            get
            {
                int[] indices = new int[_personas.Count];
                for (int i = 0; i < _personas.Count; i++)
                    indices[i] = _personas[i].MadreId;
                return indices;
            }
            set
            {
                for (int i = 0; i < value.Length; i++)
                    if (value[i] >= 0)
                        _personas[i].Madre = _personas[value[i]];
            }
        }
        [XmlArray(ElementName = "Padres", Order = 3)]
        [XmlArrayItem("PadreIndex")]
        public int[] PadreIndices
        {
            get
            {
                int[] indices = new int[_personas.Count];
                for (int i = 0; i < _personas.Count; i++)
                    indices[i] = _personas[i].PadreId;
                return indices;
            }
            set
            {
                for (int i = 0; i < value.Length; i++)
                    if (value[i] >= 0)
                        _personas[i].Padre = _personas[value[i]];
            }
        }
        private List<Persona> _personas;


    }
```

Esta clase expone tres propiedades para la serialización. La primera propiedad es un conjunto de personas, y los otros son índices de madres e índices de padres en el conjunto de personas. El orden de serialización se especifica estrictamente usando el atributo **Order**, esto es significativo porque todas las personas deben estar disponibles

en el objeto antes de que se establezcan las madres y los padres, porque tanto la madre como el padre de cada objeto se seleccionan de la matriz común de objetos. Por lo tanto, las personas deben ser serializadas y deserializadas primero, y solo entonces pueden seguir los índices de las madres y los padres.

Primero veamos cómo se ve usar **SerializablePersonaSet** para serializar nuestra familia:

```
      static void Main(string[] args)
       {
            Persona Francisco = new Persona("Francisco", new DateTime(1970, 5,
12), Genero.Hombre);
            Persona maria = new Persona("Maria", new DateTime(1972, 3, 6),
Genero.Mujer);
            Persona Rosa = new Persona("Rosa", new DateTime(1941, 2, 14),
Genero.Mujer);
            Persona Pedro = new Persona("Pedro", new DateTime(1938, 3, 18),
Genero.Hombre);

            Francisco.Madre = maria.Madre = Rosa;
            Francisco.Padre = maria.Padre = Pedro;

            Persona[] familia = new Persona[] { Francisco, maria, Rosa, Pedro };
            XmlSerializer serializador = new
XmlSerializer(typeof(SerializablePersonaSet));
            SerializablePersonaSet establecer = new SerializablePersonaSet();

            establecer.Personas = familia;

            Stream salida = Console.OpenStandardOutput();
            serializador.Serialize(salida, establecer);

            Console.ReadKey();

       }
```

Salida XML

```
<?xml version="1.0"?>

<SerializablePersonaSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <Personas>

  <Persona>

   <Nombre>Francisco</Nombre>

   <Fechacumpleanios>1970-05-12T00:00:00</Fechacumpleanios>

   <Genero>Hombre</Genero>

  </Persona>
```

```
  <Persona>
    <Nombre>Maria</Nombre>
    <Fechacumpleanios>1972-03-06T00:00:00</Fechacumpleanios>
    <Genero>Mujer</Genero>
  </Persona>
  <Persona>
    <Nombre>Rosa</Nombre>
    <Fechacumpleanios>1941-02-14T00:00:00</Fechacumpleanios>
    <Genero>Mujer</Genero>
  </Persona>
  <Persona>
    <Nombre>Pedro</Nombre>
    <Fechacumpleanios>1938-03-18T00:00:00</Fechacumpleanios>
    <Genero>Hombre</Genero>
  </Persona>
</Personas>
<Madres>
  <MadreIndex>2</MadreIndex>
  <MadreIndex>2</MadreIndex>
  <MadreIndex>-1</MadreIndex>
  <MadreIndex>-1</MadreIndex>
</Madres>
<Padres>
  <PadreIndex>3</PadreIndex>
  <PadreIndex>3</PadreIndex>
```

```
  <PadreIndex>-1</PadreIndex>

  <PadreIndex>-1</PadreIndex>

 </Padres>

</SerializablePersonaSet>
```

Ahora la información sobre madres y padres se almacena en matrices de padres y madres como se espera. No se ha producido ninguna pérdida de información real durante el proceso de serialización, aunque las referencias no se serializaron.

Ahora la deserialización es simple, donde tendremos que probar la relación de **Francisco** y **Maria** como la prueba máxima de que todos los datos se conservaron, después de la deserialización, deben señalar los mismos objetos que representan a **Rosa** y **Pedro**:

```
       static void Main(string[] args)
        {
            Persona Francisco = new Persona("Francisco", new DateTime(1970, 5,
12), Genero.Hombre);
            Persona maria = new Persona("Maria", new DateTime(1972, 3, 6),
Genero.Mujer);
            Persona Rosa = new Persona("Rosa", new DateTime(1941, 2, 14),
Genero.Mujer);
            Persona Pedro = new Persona("Pedro", new DateTime(1938, 3, 18),
Genero.Hombre);
            Francisco.Madre = maria.Madre = Rosa;
            Francisco.Padre = maria.Padre = Pedro;
            bool relacion = Francisco.EsHermanoOHermana(maria);
            Console.WriteLine("{0} y {1} {2} están emparentados.",
        Francisco.Nombre, maria.Nombre, relacion ? "" : "NO ");
            Persona[] familia = new Persona[] { Francisco, maria, Rosa, Pedro };
            SerializablePersonaSet establecer = new SerializablePersonaSet();
            establecer.Personas = familia;

            using (MemoryStream ms = new MemoryStream())
            {
                XmlSerializer serializador = new
XmlSerializer(typeof(SerializablePersonaSet));
                serializador.Serialize(ms, establecer);
                ms.Position = 0;
                SerializablePersonaSet establecer1 = serializador.Deserialize(ms)
as SerializablePersonaSet;
                Persona[] familia1 = establecer1.Personas;
                Persona Francisco1 = familia1[0];
                Persona maria1 = familia1[1];
                relacion = Francisco1.EsHermanoOHermana(maria1);
                Console.WriteLine("{0} y {1} {2} están emparentados.",
        Francisco1.Nombre, maria1.Nombre, relacion ? "" : "NO ");

            }

            Console.ReadKey();

        }
```

Resultado

```
Francisco y Maria  están emparentados.

Francisco y Maria  están emparentados.
```

Esto prueba que la deserialización ha creado relaciones correctas entre objetos.

Serializar clases complejas, esencialmente aquellas que contienen referencias a otras clases, requiere un cuidado especial. La forma más efectiva de preservar las referencias es crear otra clase que represente una contraparte serializable de la clase original, en vez de hacer serializable la clase original. Esto puede requerir además que las clases referenciadas también sean reemplazadas por sus propias versiones serializables, lo que puede llevar a una traducción relativamente compleja entre las clases originales y serializables.

El resultado de este proceso de diseño es doble:

- Todas las clases traducidas a sus versiones serializables pueden ser serializadas a XML y luego deserializadas y traducidas a su forma original sin pérdida de información relevante.
- Las clases originales no tienen que exponer propiedades públicas que de otro modo podrían poner en peligro la encapsulación y otros objetivos de diseño.

Ejemplo Completo

```
using System;
using System.Xml.Serialization;
using System.IO;
using System.Collections.Generic;

namespace Ejemplo_Serializacion
{
    public enum Genero
    {
        Hombre,
        Mujer
    }
    public class SerializablePersonaSet
    {
        public SerializablePersonaSet()
        {
            _personas = new List<Persona>();
        }
        public void Clear()
        {
            _personas.Clear();
        }
        [XmlArray(ElementName = "Personas", Order = 1)]
        [XmlArrayItem("Persona")]
        public Persona[] Personas
        {
            get { return _personas.ToArray(); }
            set
            {
                _personas.Clear();
                _personas.AddRange(value);
                for (int i = 0; i < value.Length; i++)
                {
                    value[i].Id = i;
                }
            }
        }
        [XmlArray(ElementName = "Madres", Order = 2)]
        [XmlArrayItem("MadreIndex")]
        public int[] MadreIndices
        {
            get
            {
                int[] indices = new int[_personas.Count];
                for (int i = 0; i < _personas.Count; i++)
                    indices[i] = _personas[i].MadreId;
                return indices;
            }
            set
            {
                for (int i = 0; i < value.Length; i++)
                    if (value[i] >= 0)
                        _personas[i].Madre = _personas[value[i]];
            }
        }
        [XmlArray(ElementName = "Padres", Order = 3)]
        [XmlArrayItem("PadreIndex")]
```

```
        public int[] PadreIndices
        {
            get
            {
                int[] indices = new int[_personas.Count];
                for (int i = 0; i < _personas.Count; i++)
                    indices[i] = _personas[i].PadreId;
                return indices;
            }
            set
            {
                for (int i = 0; i < value.Length; i++)
                    if (value[i] >= 0)
                        _personas[i].Padre = _personas[value[i]];
            }
        }
        private List<Persona> _personas;

    }

    public class Persona
    {
        internal int Id
        {
            get { return _id; }
            set { _id = value; }
        }
        internal int PadreId
        {
            get { return _padre == null ? -1 : _padre.Id; }
        }
        internal int MadreId
        {
            get { return _madre == null ? -1 : _madre.Id; }
        }
        private int _id;



        [XmlIgnore()]
        public Persona Madre
        {
            get { return _madre; }
            set { _madre = value; }
        }
        [XmlIgnore()]
        public Persona Padre
        {
            get { return _padre; }
            set { _padre = value; }
        }
        public bool EsHermanoOHermana(Persona p)
        {
            return
        object.ReferenceEquals(_madre, p._madre) &&
        object.ReferenceEquals(_padre, p._padre);
        }

        private Persona _madre;
        private Persona _padre;

        public Persona() // Se define el constructor por defecto
```

```
        {
        }

        public Persona(string nombre, DateTime dob, Genero genero)
        {
            _nombre = nombre;
            _fechacumpleanios = dob;
            _genero = genero;
        }
        [XmlElement("Nombre")]
        public string Nombre
        {
            get { return _nombre; }
            set { _nombre = value; }
        }
        [XmlElement("Fechacumpleanios")]
        public DateTime Fechacumpleanios
        {
            get { return _fechacumpleanios; }
            set { _fechacumpleanios = value; }
        }
        [XmlElement("Genero")]
        public Genero Genero
        {
            get { return _genero; }
            set { _genero = value; }
        }
        private string _nombre;
        private DateTime _fechacumpleanios;
        private Genero _genero;
    }
    class Program
    {
        static void Main(string[] args)
        {
            Persona Francisco = new Persona("Francisco", new DateTime(1970, 5,
12), Genero.Hombre);
            Persona maria = new Persona("Maria", new DateTime(1972, 3, 6),
Genero.Mujer);
            Persona Rosa = new Persona("Rosa", new DateTime(1941, 2, 14),
Genero.Mujer);
            Persona Pedro = new Persona("Pedro", new DateTime(1938, 3, 18),
Genero.Hombre);
            Francisco.Madre = maria.Madre = Rosa;
            Francisco.Padre = maria.Padre = Pedro;
            bool relacion = Francisco.EsHermanoOHermana(maria);
            Console.WriteLine("{0} y {1} {2} están emparentados.",
        Francisco.Nombre, maria.Nombre, relacion ? "" : "NO ");
            Persona[] familia = new Persona[] { Francisco, maria, Rosa, Pedro };
            SerializablePersonaSet establecer = new SerializablePersonaSet();
            establecer.Personas = familia;

            using (MemoryStream ms = new MemoryStream())
            {
                XmlSerializer serializador = new
XmlSerializer(typeof(SerializablePersonaSet));
                serializador.Serialize(ms, establecer);
                ms.Position = 0;
                SerializablePersonaSet establecer1 = serializador.Deserialize(ms)
as SerializablePersonaSet;
                Persona[] familia1 = establecer1.Personas;
                Persona Francisco1 = familia1[0];
```

```
                Persona maria1 = familia1[1];
                relacion = Francisco1.EsHermanoOHermana(maria1);
                Console.WriteLine("{0} y {1} {2} están emparentados.",
        Francisco1.Nombre, maria1.Nombre, relacion ? "" : "NO ");

            }

            Console.ReadKey();
        }
        }

    }
```

FIN