ESQUEMA PARA LA PRESENTACIÓN DE PFC

RESUMEN

(2)

Explicaremos qué es la serialización, qué es el programa, cómo funciona, cómo son los serializadores que genera, etc.

La serialización es un proceso que codifica un objeto en una cadena (de texto o de bytes) para almacenarlo o transmitirlo por la red con el fin de que esa cadena pueda ser usada para crear un nuevo objeto idéntico

La serialización es un proceso muy utilizado, sobre todo en entornos distribuidos donde para cada conexión entre los componentes del sistema distribuido suele llevar consigo el intercambio de parámetros o elementos serializados. Es tan importante en este contexto que el tiempo que requiere su ejecución y el transporte del código que generen, puede resultar significativamente grande respecto al resto del proceso.

Es en este contexto donde una mejora significativa en la serialización más se notaría.

(3)

Hay muchos serializadores, con sus ventajas y desventajas, válidos para determinados tipos de objetos y no para otros, que generan código serializado para ser almacenado y/o transportado en diversos formatos y contextos.

(4)

Con este proyecto hemos conseguido, tras analizar una amplia variedad de serializadores, obtener el *HiperSerializador*, un generador de serializadores particulares para cualquier clase.

EL serializador creado por nuestra aplicación es un serializador versátil que sirva para cualquier tipo de objeto, generando cualquier tipo de código serializado transportable y almacenable en cualquier contexto,

(5)

capaz de reconocer los atributos estándar o cualquier atributo particular que se desee generar, ampliable mediante plug-ins y por supuesto, con los mejores resultados en cuanto a rendimiento.

(6)

Esta aplicación es un **generador de serializadores particulares de alto rendimiento**. O sea, nuestra aplicación analiza una determinada clase (tipo) y fabrica el serializador óptimo para esa determinada clase.

(7)

SERIALIZACIÓN Y SERIALIZADORES

**Serialización** es el proceso de convertir un objeto en un stream de datos que contiene toda la información del mismo con el fin de almacenarla, compartirla o distribuirla.

**Deserialización** es el proceso inverso, a partir de un stream de datos se recompone el estado original del objeto, creando una copia idéntica del objeto original.

Ambos procesos son complementarios

Factores a tener en cuenta

Hemos analizar múltiples serializadores ya existentes, con el fin de identificar su funcionamiento, sus características comunes, y aquellos factores en los que se podría mejorar su funcionamiento o características. Finalmente, hemos llegado a la conclusión de que para mejorar el mecanismo de serialización, lo podemos hacer a través de tres factores que son fundamentales en el uso de serializadores.

(8)

- VELOCIDAD Tiempo que dedica el sistema a generar código serializable, y a regenerar un objeto a partir de ese código.

(9)

- LONGITUD DE LA CADENA Principalmente cuando se piensa en la distribución de los objetos por una red, interesa que el tamaño del stream a distribuir sea lo más liviano posible. Ya que el transporte es un factor clave en el rendimiento de cualquier sistema, sobre todo en los distribuidos, reducir el tamaño de los datos a transmitir es reducir el cuello de botella del proceso.

(10)

- APLICABILIDAD DEL SERIALIZADOR Llamamos aplicabilidad a la disponibilidad que tiene el serializador para funcionar correctamente con cualquier elemento dentro del objeto a serializar. Algunos serializadores, por ejemplo tienen problemas para serializar arrays multidimensionales, tipos genéricos, o ciertas estructuras. Un buen serializador debería poder trabajar con cualquier tipo de elementos.

(11)

En esta diapositiva se enumeran los serializadores que hemos estudiado y analizado. Hay hasta 4 de .Net, y aunque en realidad hay bastantes más, muchos son variantes de éstos. También hemos tomado en consideración otros serializadores open-source, como SharpSerializer, NetSerializer y el proyecto Protobuf para .Net de Google.

Añadimos a la lista el nuestro, enmarcado en verde para distinguirlo en las siguientes diapositivas.

(12)

Comparativa de características

En esta tabla se describen los formatos de código serializado que pueden generar cada uno de los serializadores analizados.

Como se puede apreciar, cada serializador tiene su formato "preferido", aunque varios pueden generar el código serializado en varios formatos. Con nuestra aplicación, el serializador que se genere se puede configurar para aceptar cualquier formato de serialización, lo que es una ventaja porque en cada contexto se podrá utilizar el que se desee, sin perder las prestaciones del serializador.

(13)

Hablábamos de la aplicabilidad del serializador, la capacidad para procesar cualquier tipo de elementos que contenga el objeto a serializar. En esta tabla se aprecian algunos handicaps que poseen otros serializadores, y que el nuestro no tiene.

Así, los arrays anidados son elementos que en muchos de los serializadores no son capaces de serializar correctamente. Lo mismo ocurre con las listas genéricas, y alguonos tipos de estruturas o objetos con herencia.

Nuestro serializador es capaz de serializar cualquier elemento dentro de la clase con la que se trabaje.

(14)

Con toda esta información hemos conseguido el objetivo del proyecto, que consiste en conseguir el serializador idóneo para cada tipo de objeto, que genera el stream con la serialización con el formato que el proramador espere, y al que además se pueden añadir opciones avanzadas que pueden hacer de él algo más que un simple serializador.

(15)

HIPERSERIALIZADOR

Qué es HiperSerializer

Es un pequeño programa que tiene una gran potencia. Su función es generar un serializador para cualquier clase cuyo tipo se le indique en el constructor. Este serializador realiza la tarea de serializar y deserializar cualquier objeto de esa clase de manera óptima.

Arquitectura de HiperSerializer

El proyecto se compone básicamente de una única clase HiperSerializer. La aplicación HiperSerializer.dll se integra en cualquier aplicación .net simplemente cargándose como un DLL más.

A la hora de instanciar, al constructor hay que pasarle como parámetro un System.Type y opcionalmente un String que indique el formato de salida del código serializado.

(16)

A partir de ese momento, se utiliza instanciando la clase HiperSerializer y llamando al método getSerializer.

En el momento de invocar al método getSerializer, se obtendrá el serializador particular para ese tipo en una variable de tipo dinámico, ya que el objeto devuelto será de una clase particular no disponible en tiempo de compilación.

(17)

Cómo funciona HiperSerializer

El programa recibe como entrada un tipo de dato de .Net. Mediante Reflection, se averiguan todos los elementos serializables del tipo.

Reflection consiste en una serie de utilidades dentro del lenguaje que permiten examinar, modificar y trabajar con los tipos de datos de un programa EN TIEMPO DE EJECUCIÓN. Es muy útil en muy distintos contextos, desde simplemente identificar las características de una variable según su tipo, hasta la ejecución de métodos privados de una clase instanciada.

(18)

A partir de ese momento se va generando el código que tendrá el programa serializador para ese tipo. A continuación, compila al vuelo ese código, se instancia un objeto de esa clase y eso es lo que devuelve como salida.

Este código generado posee dos acciones a realizar: la serialización y la deserialización. Para cada elemento se van generando a la vez el código para los dos métodos, rellenando sendas cadenas llamadas strEncode y strDecode.

Para ello, de cada elemento obtenemos sus características, también mediante Reflection. Una vez analizadas nos permite averiguar qué tipo de elemento es. Así, sabremos si es un array, si es una estructura, una lista, etc., y generamos el código que lo serializa y deserializa adecuadamente

El programa generado realiza la tarea exclusiva de serializar y deserializar objetos de ese tipo. Además, lo hace de una manera muy eficiente y rápida, como se verá en las pruebas realizadas. Es lógico, ya que la única acción que realiza es invocar al elemento para su serialización, o recorrer el fichero y volcando los valores en los elementos correspondientes, para la deserialización

3.4. Cómo genera cada serializador

El código para serializar básicamente consiste en ir montando un String que acumule la información de cada elemento en el formato pertinente.

El código para la deserialización básicamente consiste en ir recogiendo a partir de

Una vez que se invoca el método Generador.getSerializer, el programa ya sabe para qué tipo tiene que generar el código de su serializador particular. Para ayudar en esta tarea se utiliza el espacio de nombres System.Reflection.

Reflection es una característica de .Net que permite trabajar con los elementos que componen un tipo, campos, propiedades, métodos, eventos, etc. Nos permite obtener su nombre, tipo, y muchas más cualidades (si es público o privado, sus atributos, etc.).

Se usa Reflection para capturar todos los elementos del tipo susceptibles de ser serializados. Para cada uno de ellos, se realiza la tarea de generar el código que permite su serialización y deserialización e ir metiéndolo en un stream.

Para cada elemento, se generará el código que permitirá colocar en un string la información de su valor (serialización) o capturar de dicho string su valor para colocarlo en el elemento correspondiente (deserialización) según el formato que se desee para la salida de la serialización. Ambos procesos se hacen de manera sucesiva para cada elemento, por lo que el código de la serialización se genera a la vez que el código de la deserialización.

Por ejemplo: Clase01Basica

public class Clase01Basica

{

public string v1 = "1";

public int v2 = 2;

}

El código del serializador para esta clase se conformará así:

public class Clase01BasicaCodec{

public static string encode(Fase02.Clase01Basica obj){  
 StringBuilder texto = new StringBuilder();  
 texto.Append(obj.v1.ToString() + ",");  
 texto.Append(obj.v2.ToString() + ",");  
 return texto.ToString();  
 }

public static void decode(ref String codigo, ref Fase02.Clase01Basica obj){  
 Queue<string> elementos = new Queue<string>(codigo.Split(','));  
 obj.v1 = elementos.Dequeue();  
 obj.v2 = Int32.Parse(elementos.Dequeue());  
 }

}

Para cada elemento recogido con Reflection con getMembers() se añade la línea texto.Append(obj.v1.ToString() + “,”)

3.5. Arquitectura de los serializadores generados

La clase serializadora se llama con el nombre del tipo, seguido de Codec. En su interior contiene dos métodos estáticos y dos métodos de instancia. Los métodos de instancia son aliases de los métodos estáticos, ya que hemos analizado que la manera más rápida era usar métodos estáticos.

4. COMPARATIVA DE RENDIMIENTOS

5. PROBLEMAS ENCONTRADOS

6. MEJORAS

7. CONCLUSIONES