Reflection

La API Reflection de .NET Framework le permite obtener información de tipo (ensamblaje) en tiempo de ejecución mediante programación. También podemos lograr un enlace tardío utilizando .NET Reflection. En el tiempo de ejecución, el mecanismo de Reflexión usa el archivo PE para leer información sobre el ensamblaje. Reflection nos permite usar código que no está disponible en tiempo de compilación. .NET Reflection permite a una aplicación recopilar información sobre sí misma y también manipularla. Se puede usar de manera efectiva para encontrar todos los tipos en un ensamblaje y/o invocar dinámicamente métodos en un ensamblaje. Esto incluye información sobre el tipo, las propiedades, los métodos y los eventos de un objeto. Con Reflection, podemos crear dinámicamente una instancia de un tipo, vincular el tipo a un objeto existente u obtener el tipo de un objeto existente e invocar sus métodos o acceder a sus campos y propiedades.

Con Reflection, puede obtener cualquier tipo de información que pueda ver en un visor de clases; por ejemplo, información sobre los métodos, propiedades, campos y eventos de un objeto.

El espacio de nombres **System.Reflection** y la clase **System.Type** juegan un papel muy importante en la reflexión de .NET. Estos dos trabajan juntos y le permiten reflexionar sobre muchos otros aspectos de un tipo.

El espacio de nombres **System.Reflection** contiene las clases e interfaces que proporcionan una vista administrada de los tipos, métodos y campos cargados, con la capacidad de crear e invocar dinámicamente tipos. A continuación, podemos ver los más usados:

Clase	Descripción
`Assembly`	Representa un ensamblaje, que es un bloque de construcción reutilizable, versionable y autodescriptivo de una aplicación Common Language Runtime. Esta clase contiene una serie de métodos que le permiten Cargar, investigar y manipular un ensamblaje.
`Module`	Realiza la reflexión sobre un módulo. Esta clase le permite acceder a un módulo dado dentro de un ensamblaje de varios archivos.
`AssemblyName`	Esta clase le permite descubrir numerosos detalles detrás de la identidad de un conjunto. La identidad de un ensamblado consiste en lo siguiente: • Nombre simple • Número de versión • Par de claves criptográficas • Cultura
`EventInfo`	Esta clase contiene información para un evento dado. Utilizamos la clase `EventInfo` para inspeccionar eventos y enlazar con los controladores de eventos.
`FieldInfo`	Esta clase contiene información para un campo dado. Los campos son

	variables definidas en la clase. `FieldInfo` proporciona acceso a los metadatos para un campo dentro de una clase, y proporciona un conjunto dinámico y funcionalidad para obtener el campo. La clase no se carga en la memoria hasta se llama a `Invoke` o `get` en el objeto.
`MemberInfo`	La clase `MemberInfo` es la clase base abstracta para las clases utilizadas para obtener información sobre todos los miembros de una clase (constructores, eventos, campos, métodos y propiedades).
`MethodInfo`	Esta clase contiene información para un método dado.
`ParameterInfo`	Esta clase contiene información para un parámetro dado.
`PropertyInfo`	Esta clase contiene información para una propiedad dada.

El espacio de nombres **System.Reflection.Emit** contiene las clases para emitir metadatos.

La clase de **Type** representa declaraciones de tipo para tipos de clase, tipos de interfaz, tipos de enumeración, tipos de matriz, tipos de valor, etc. Se encuentra en el espacio de nombres del sistema y hereda la clase **System.Reflection.MemberInfo**.

A continuación, se proporciona una lista de propiedades importantes de la clase **Type**:

Propiedad	Descripción
Assembly	Obtiene el ensamblado para este tipo.
AssemblyQualifiedName	Obtiene el nombre calificado de ensamblado para este tipo.
Attributes	Obtiene los atributos asociados con el tipo.
BaseType	Obtiene el tipo base o padre.
FullName	Obtiene el nombre completo del tipo.
IsAbstract	Se utiliza para comprobar si el tipo es abstracto.
IsArray	Se utiliza para comprobar si el tipo es Array.
IsClass	Se usa para verificar si el tipo es una clase.
IsEnum	Se utiliza para comprobar si el tipo es Enum.
IsInterface	Se utiliza para comprobar si el tipo es Interface.
IsNested	Se utiliza para comprobar si el tipo está anidado.
IsPrimitive	Se utiliza para comprobar si el tipo es primitivo.
IsPointer	Se utiliza para comprobar si el tipo es puntero.
IsNotPublic	Se utiliza para comprobar si el tipo no es público.
IsPublic	Se utiliza para comprobar si el tipo es público.
IsSealed	Se utiliza para comprobar si el tipo está sellado.
IsSerializable	Se utiliza para comprobar si el tipo es serializable.
MemberType	Se utiliza para verificar si el tipo es tipo de un miembro anidado de un tipo.
Module	Obtiene el módulo del tipo.

Name	Obtiene el nombre del tipo.
Namespace	Obtiene el espacio de nombres del tipo.

A continuación, vamos a ver una lista de los métodos importantes de la clase **Type**:

Método	Descripción
GetConstructors ()	Devuelve todos los constructores públicos para el Type.
GetConstructors (BindingFlags)	Devuelve todos los constructores para el Type con los BindingFlags especificados.
GetFields ()	Devuelve todos los campos públicos para el Type.
GetFields (BindingFlags)	Devuelve todos los constructores públicos para el Type con los BindingFlags especificados.
GetMembers ()	Devuelve todos los miembros públicos para el Type.
GetMembers (BindingFlags)	Devuelve todos los miembros para el Type con los BindingFlags especificados.
GetMethods ()	Devuelve todos los métodos públicos para el Type.
GetMethods (BindingFlags)	Devuelve todos los métodos para el Type con los BindingFlags especificados.
GetProperties ()	Devuelve todas las propiedades públicas para el Type.
GetProperties (BindingFlags)	Devuelve todas las propiedades para el Type los con BindingFlags especificados.
GetType ()	Obtiene el Type actual.
GetType (String)	Obtiene el Type para el nombre dado.

Ejemplos Básicos

Obtener nombre de métodos

Si queremos obtener los nombres de métodos de un tipo dado en C #, podemos usar el método **Type.GetMethods**. Este método nos devuelve una matriz de objetos I**nformacionMetodo**. **InformacionMetodo** contiene muchas informaciones sobre el método y, por supuesto, una función para conocer el nombre de método llamado **InformacionMetodo.Name**.

Para filtrar los métodos devueltos podemos usar el parámetro **BindingFlags** cuando llamamos al método **GetMethods**. Se requieren al menos dos indicadores, uno de **Public/Non Public** y otro de **Instance/Static**. También tenemos otras banderas como **DeclaredOnly** y

FlattenHierarchy. La enumeración **BindingFlags** y la clase **InformacionMetodo** se declaran en el espacio de nombres **System.Reflection**.

Este ejemplo se obtienen los nombres de métodos estáticos públicos de MiClase, los clasifica por nombre y los escribe en la consola.

```
using System;
using System.Reflection;

public class EjemploReflection
{
    public class MiClase
    {
        public int nombre { get; set; }
        public string apellidos { get; set; }

    }

    public static void Main()
    {
        // Obtener todos los métodos estáticos públicos del tipo MiClase
        MethodInfo[] InformacionMetodos =
typeof(MiClase).GetMethods(BindingFlags.Public | BindingFlags.Instance);

        // Ordenamos los métodos por nombre
        Array.Sort(InformacionMetodos, delegate (MethodInfo InformacionMetodo1,
MethodInfo InformacionMetodo2)
        {
            return InformacionMetodo1.Name.CompareTo(InformacionMetodo2.Name);
        });

        // Muestra por pantalla el nombre de los métodos
        foreach (MethodInfo InformacionMetodo in InformacionMetodos)
        {
            Console.WriteLine(InformacionMetodo.Name);
        }

        Console.ReadKey();
    }

}
```

Resultado:

```
Equals

get_apellidos

get_nombre

GetHashCode

GetType

set_apellidos
```

```
set_nombre

ToString
```

Obtener nombres de propiedades

Para obtener los nombres de propiedades para un tipo específico podemos usar el método **Type.GetProperties**. El método devuelve la matriz de objetos **PropertyInfo** y los nombres de las propiedades están disponibles a través de la propiedad **PropertyInfo.Name**. Si queremos obtener un subconjunto de todas las propiedades podemos usar las **BindingFlags** cuando llamamos al método **GetProperties**, para ello debemos especificar al menos dos indicadores, uno de **Instance/Static** y uno de **Instance/Static**. Cuando usamos **GetProperties** sin un parámetro **BindingFlags**, los indicadores predeterminados son **Public** + **NonPublic** + **Instance**.

En este método, se pueden usar **BindingFlags**, que es un enum, que controla el enlace y la forma en que se utiliza la exploración de los miembros y tipos. Esto proporciona un mayor nivel de control sobre lo que estamos buscando exactamente, por ejemplo, miembros solo static, miembros solo public, etc.).

Algunas banderas vinculantes son:

- **BindingFlags.Public**: Especifica que los miembros **public** deben ser incluidos en la búsqueda.
- **BindingFlags.Instance**: Especifica que los miembros de la **instancia** deben ser incluidos en la búsqueda.
- **BindingFlags.NonPublic**: Especifica que los miembros **no public** deben ser incluidos en la búsqueda.

El siguiente ejemplo muestra cómo obtener propiedades estáticas públicas.

```
using System;
using System.Reflection;

public class EjemploReflection
{
    public class MiClase
    {
        public int nombre { get; set; }
        public string apellidos { get; set; }

    }

    public static void Main()
    {
        PropertyInfo[] InformacionPropiedades;
        InformacionPropiedades =
typeof(MiClase).GetProperties(BindingFlags.Public | BindingFlags.Instance);

        // Ordenar las propiedades por nombre
        Array.Sort(InformacionPropiedades, delegate (PropertyInfo
InformacionPropiedades1, PropertyInfo InformacionPropiedades2)
        {
```

```
            return
InformacionPropiedades1.Name.CompareTo(InformacionPropiedades2.Name);
        });

        // Mostrar por pantalla los nombres de las propiedades
        foreach (PropertyInfo InformacionPropiedad in InformacionPropiedades)
        {
            Console.WriteLine(InformacionPropiedad.Name);
        }

        Console.ReadKey();
    }

}
```

Resultado:

```
apellidos

nombre
```

Obtener el nombre de la llamada a un método

Para obtener el nombre de la llamada a un método usamos del método **StackTrace.GetFrame**, para ello creamos una nueva instancia de **StackTrace** y llamamos al método **GetFrame**. El parámetro es el índice de la llamada al método en la pila de llamadas. El índice de la primera llamada al método, el más cercano, es "1", por lo que nos devuelve un **StackFrame** del método de llamada, que es el método que llamó directamente al método actual. Para obtener el nombre del método utilizamos el método **StackFrame.GetMethod** para obtener el método base y luego acaba de obtener el valor de la propiedad **Name**.

En el siguiente ejemplo podemos ver cómo obtener el nombre del método de llamada:

```
using System;
using System.Diagnostics;

public class EjemploReflection
{
    public static void Main()
    {
        // Obtenemos la Pila de llamada
        StackTrace stacktrace = new StackTrace();

        // Obtenemos el nombre del método de llamada
        Console.WriteLine(stacktrace.GetFrame(1).GetMethod().Name);
        Console.ReadKey();
    }
}
```

Resultado:

```
_nExecuteAssembly
```

Obtener Pila de Llamadas

Este ejemplo muestra cómo obtener la pila de llamadas en .NET. La pila de llamadas está representada por la clase **StackTrace** y las llamadas de métodos están representadas por la clase **StackFrame**. Podemos obtener los **frames** utilizando el método **StackTrace.GetFrames** que nos devuelve la matriz de **frames**. El **frame** con índice 0 es el método que se está ejecutando actualmente, el **frame** 1 está llamando al método, el siguiente **frame** es su llamador, etc.

El siguiente ejemplo escribe los nombres de los métodos de la pila de llamadas. Crea una instancia de **StackTrace** (pila de llamadas), obtiene todos los **frames** (llamadas de método) y escribe los nombres de los métodos.

```
using System;
using System.Diagnostics;
[STAThread]

    public static void Main()
    {
        //Obtenemos la llamada a la pila
        StackTrace stackTrace = new StackTrace();

        //Obtenemos el método de llamadas
        StackFrame[] stackFrames = stackTrace.GetFrames();


        // recorremos los nombres de los métodos de la pila de llamadas
        foreach (StackFrame stackFrame in stackFrames)
        {

            // mostramos el nombre del método
            Console.WriteLine(stackFrame.GetMethod().Name);
        }

    }
```

Resultado:

```
Main

_nExecuteAssembly

ExecuteAssembly

RunUsersAssembly

RunInternal

Run

Run
```

```
ThreadStart
```

Obtener el tipo

```
using System;

public class EjemploReflection
{
    public static void Main()
    {
        int numero = 65;
        Type tipo = numero.GetType();
        Console.WriteLine(tipo);
        Console.ReadKey();
    }

}
```

Resultado:

```
System.Int32
```

Obtener Assembly

```
using System;

public class EjemploReflection
{
    public static void Main()
    {
        Type tipo = typeof(System.String);
        Console.WriteLine(tipo.Assembly);
        Console.ReadKey();
    }

}
```

Resultado:

```
mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

Mostrar por pantalla la información del Tipo

```
using System;

public class EjemploReflection
{
    public static void Main()
    {
        Type tipo = typeof(System.String);
        Console.WriteLine(tipo.FullName);
        Console.WriteLine(tipo.BaseType);
        Console.WriteLine(tipo.IsClass);
        Console.WriteLine(tipo.IsEnum);
        Console.WriteLine(tipo.IsInterface);
        Console.ReadKey();
    }

}
```

Resultado:

```
System.String

System.Object

true

false

false
```

Mostrar por pantalla los Constructores

```
using System;
using System.Reflection;

public class EjemploReflection
{
    public static void Main()
    {
        Type tipo = typeof(System.String);

        Console.WriteLine("Los Constructores del tipo {0} son:", tipo);
        ConstructorInfo[] constructor = tipo.GetConstructors(BindingFlags.Public
| BindingFlags.Instance);
        foreach (ConstructorInfo c in constructor)
        {
            Console.WriteLine(c);
        }
        Console.ReadKey();
    }

}
```

Resultado:

```
Los constructores del tipo System.String son:

Void .ctor(Char*)

Void .ctor(Char*, Int32, Int32)

Void .ctor(SByte*)

Void .ctor(SByte*, Int32, Int32)

Void .ctor(SByte*, Int32, Int32, System.Text.Encoding)

Void .ctor(Char[], Int32, Int32)

Void .ctor(Char[])

Void .ctor(Char, Int32)
```

Mostrar por Pantalla los métodos

```
using System;
using System.Reflection;

public class EjemploReflection
{
    public static void Main()
    {
        Type t = typeof(System.String);

        Console.WriteLine("Los métodos del tipo {0} son:", t);
        MethodInfo[] metodos = t.GetMethods(BindingFlags.Public |
BindingFlags.Instance);
        foreach (MethodInfo metodo in metodos)
        {
            Console.WriteLine(metodo);
        }
        Console.ReadKey();
    }

}
```

Resultado:

```
Los métodos del tipo System.String son:

Boolean Equals(System.Object)

Boolean Equals(System.String)
```

```
Boolean Equals(System.String, System.StringComparison)

Char get_Chars(Int32)

Void copyTo(Int32, char[], Int32, Int32)

Char[] ToCharArray()

....
```

Obtener tipo de referencia

Podemos obtener una instancia de clase **Type** de varias maneras.

Utilizando System.Object.GetType ()

System.Object define un método llamado **GetType()**, que devuelve una instancia de la clase **Type** que representa los metadatos para el objeto actual.

```
// Obtiene la información del tipo usando una instancia de Persona
Persona persona = new Persona();
Type tipo = persona.GetType();
```

Este enfoque funcionará solo si tenemos conocimiento en tiempo de compilación del tipo que queremos explorar y actualmente tenemos una instancia del tipo en la memoria.

Utilizando typeof ()

La otra forma es obtener metadatos de tipo usando el operador **typeof**:

```
// Obtiene el tipo usando typeof
Type tipo = typeof(Persona);
```

Este enfoque no requiere crear una instancia de objeto para escribir información.

Utilizando System.Type.GetType ()

Type.GetType() proporciona información de tipo de una manera más flexible. Usando este método, no necesitamos tener conocimiento del tiempo de compilación del tipo, ya que especificamos el string con el nombre completo del tipo que se va a explorar.

El método **Type.GetType()** se ha sobrecargado para especificar dos parámetros booleanos, uno de los cuales controla si se debe lanzar una excepción si no se puede encontrar el tipo, y el otro comprueba la sensibilidad de las mayúsculas y minúsculas del string.

```
// Obtiene el tipo de información usando el método Type.GetType()
Type tipo = Type.GetType("MiEjemploReflection.Persona", false, true);
```

En este ejemplo, se supone que **Type** está definido dentro del ensamblaje que se está ejecutando actualmente. Cuando queremos obtener metadatos para un tipo dentro de un ensamblaje externo, el parámetro string se formatea con el nombre completo del tipo, seguido de una coma, seguido del nombre descriptivo del ensamblado que contiene tipo:

```
// Obtiene la información de tipo para un tipo dentro de un assembly externo
Type tipo = Type.GetType("MiEjemploReflection.Persona, PersonaLib")
```

En el método **Type.GetType()**, también podemos especificar un símbolo más **(+)** para indicar un tipo anidado. Supongamos que queremos obtener información del tipo para una clase llamada **Persona** que está anidada dentro de una clase llamada **_Persona**.

```
// Obtiene la información del tipo para una clase anidada dentro del assembly
actual
Type tipo = Type.GetType("MiEjemploReflection.Persona+PersonaOpcion");
```

Obtención de información de métodos

```
// Imprime todos los métodos con el retorno de la información de tipo y parámetro
        static void ListaMetodosConParametros(Type tipo)
        {
            Console.WriteLine("\n***** Métodos *****\n");
            //obtiene todos los métodos del tipo actual
            MethodInfo[] infoMetodo = tipo.GetMethods();
            StringBuilder infoParametro = new StringBuilder();
            foreach (MethodInfo metodo in infoMetodo)
            {
                // Obtener el tipo de retorno.
                string valorRetorno = metodo.ReturnType.FullName;
                infoParametro.Append("( ");
                // Obtener parámetros.
                foreach (ParameterInfo parametro in metodo.GetParameters())
                {
                    infoParametro.Append(string.Format("{0} {1} ",
parametro.ParameterType, parametro.Name));
                }
                infoParametro.Append(" )");
                // Muestra la firma del método básico.
                Console.WriteLine(String.Format("{0} {1} {2}\n", valorRetorno,
metodo.Name, infoParametro.ToString()));
                infoParametro.Clear();
            }
            Console.ReadKey();

        }
```

En este método, se pasa un parámetro de tipo **Type**. Para obtener todos los métodos del tipo se llama a **GetMethod()** que nos devuelve una instancia de la matriz **MethodInfo**. Ahora, iterando a través de todos los elementos de **infoMetodo**, podemos obtener el tipo de retorno de ese método usando **metodo.ReturnType.FullName**. Después, para obtener todos los parámetros de ese método, llamamos a **metodo.GetParameters()**. Con el método **parametro**, podemos obtener el tipo de parámetro y su nombre.

Obtención de información de campos

```
// Imprimir lista de campos con su tipo.
        static void ListaCampos(Type tipo)
        {
            Console.WriteLine("***********Campos************");

            // Obtener todos los campos
            FieldInfo[] infoCampos = tipo.GetFields();
            foreach (var campo in infoCampos)
            {
                Console.WriteLine(String.Format
                ("{0}  {1}\n", campo.FieldType.Name, campo.Name));
            }
            Console.ReadKey();

        }
```

Para obtener todos los campos, vamos a usar **GetFields()** que nos devuelve un objeto **FieldInfo[]**, y con el objeto **campo**, podemos obtener su información.

Obtener información de la propiedad

```
/// Imprimir lista de propiedades con su tipo.
        static void ListOfProperty(Type type)
        {
            Console.WriteLine("***********Propiedades************");

            // Obtener todas las propiedades
            PropertyInfo[] infoPropiedades = type.GetProperties();
            foreach (var propiedad in infoPropiedades)
            {
                Console.WriteLine(String.Format
                ("{0}  {1}\n", propiedad.PropertyType.Name, propiedad.Name));
            }
            Console.ReadKey();
         }
```

FIN