**C# Reflection**

In C#, reflection allows us to inspect and manipulate classes, constructors, methods, and fields at run time.

For example, we have defined a class named Student as:

class Student

{

// a field

public int age;

// method

public void score()

{

// .. some code

}

}

The above Student class consists of a field and a method. With the help of reflection, we can inspect this information at runtime.

We will learn about reflection below in detail.

**What is C# Reflection?**

In C#, there is a block called **Assembly** which is automatically generated by the compiler after the successful compilation of code. It consists of two parts: intermediate language and metadata.

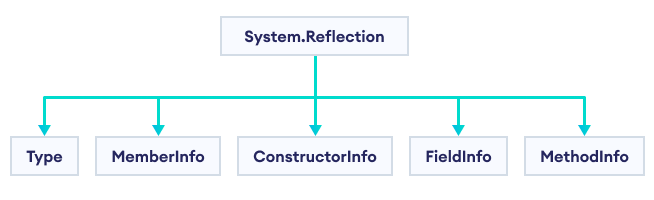
When we write C# code, it is not directly compiled into machine-level language. It is first compiled into an intermediate language which is packaged into **assembly**.

Similarly, the metadata consists of information about the types like classes, methods, constructors, etc. For example, a Student class consists of some methods and fields. All this information about the Student class is in the form of metadata.

Reflection is simply taking that metadata and inspecting the information inside it.

**C# Reflection Hierarchy**

C# provides System.Reflection namespace using which we can perform reflection. The System.Reflection namespace contains the classes like:

C# Reflection Hierarchy

The above image shows some of the classes under the System.Reflection namespace. Among these, we will demonstrate reflection using the Type class.

**C# Type Class**

The primary way to access the metadata is by using the Type class. This class provides methods and properties using which we can get information about a type declaration. For example,

using System;

using System.Reflection;

public class Program

{

public static void Main()

{

String studentName = "Jack";

// get the current type of studentName

Type studentNameType = studentName.GetType();

Console.WriteLine("Type is: " + studentNameType);

}

}

**Output**

Type is: System.String

In the above example, we have defined a String class variable studentName. Notice the code,

// get the current type of studentName

Type studentNameType = studentName.GetType();

The GetType() method returns the Type of the current data. Here, we have used GetType() with studentName and assigned it to the Type variable studentNameType.

The GetType() method returns System.String which is the current type of studentName.

**Example 1: C# Reflection to get Assembly**

The Type class provides a property called Assembly which generates the Assembly of the code. For example,

using System;

using System.Reflection;

class Program

{

static void Main()

{

// get typeof the Program class and load it to Type variable t

Type t = typeof(Program);

// get Assembly of variable t using the Assembly property

Console.WriteLine(t.Assembly);

}

}

**Output**

app, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

In the above example, notice the code,

// get type of Program and load it to Type variable t

Type t = typeof(Program);

Here, we have used the typeof operator that returns the type of the Program class and assigned it to Type variable t.

Then, we have used the Assembly property with t as t.Assembly that returns assembly of the Program class.

**Example 2: C# Reflection with Enumerable**

We can get information about the Enumerable type using the properties of the Type class. For example,

using System;

using System.Reflection;

class Program

{

static void Main()

{

// get typeof Enumerable and load it to Type variable t

Type t = typeof(Enumerable);

// the Type class properties return information about the Enumerable Type

Console.WriteLine("Name : {0}", t.Name);

Console.WriteLine("Namespace : {0}", t.Namespace);

Console.WriteLine("Base Type : {0}", t.BaseType);

}

}

**Output**

Name : Enumerable

Namespace : System.Linq

Base Type : System.Object

In the above example, we have used the Type class properties with the Enumerable type,

* t.Name - gets the name of the type
* t.Namespace - gets the namespace of the type
* t.BaseType - gets the base or parent type

**Example 3: C# Reflection with String**

We can also get information about the String type using the properties of the Type class. For example,

using System;

using System.Reflection;

class Program

{

static void Main()

{

// get typeof String and load it to Type variable t

Type t = typeof(String);

// the Type class properties return information about the String Type

Console.WriteLine("Name : {0}", t.Name);

Console.WriteLine("Namespace : {0}", t.Namespace);

Console.WriteLine("Base Type : {0}", t.BaseType);

}

}

**Output**

Name : String

Namespace : System

Base Type : System.Object

**Note:** We can get information about other types like *class*, *delegate*, *array* and so on using Reflection in C#..