Programming in C#

E06 Advanced topic and practices

Yu Guan | Microsoft MVP
2018

AGENDA

- Attributes
- Reflection
- Encryption
- Manage assemblies
- Practice project

Attributes

- Attributes add metadata to your program. Metadata is information about the types defined in a program. All .NET assemblies contain a specified set of metadata that describes the types and type members defined in the assembly. You can add custom attributes to specify any additional information that is required.
- You can apply one or more attributes to entire assemblies, modules, or smaller program elements such as classes and properties.
- Attributes can accept arguments in the same way as methods and properties.
- Your program can examine its own metadata or the metadata in other programs by using reflection.

Using Attributes

- Attributes can be placed on most any declaration, though a specific attribute might restrict the types of declarations on which it is valid. In C#, you specify an attribute by placing the name of the attribute, enclosed in square brackets ([]), above the declaration of the entity to which it applies.

```
[System.Serializable]
public class SampleClass
{
    // Objects of this type can be serialized.
}
```

Common Uses for Attributes

- Marking methods using the WebMethod attribute in Web services to indicate that the method should be callable over the SOAP protocol. For more information, see WebMethodAttribute.
- Describing how to marshal method parameters when interoperating with native code. For more information, see MarshalAsAttribute.
- Describing the COM properties for classes, methods, and interfaces.
- Calling unmanaged code using the DllImportAttribute class.
- Describing your assembly in terms of title, version, description, or trademark.
- Describing which members of a class to serialize for persistence.
- Describing how to map between class members and XML nodes for XML serialization.
- Describing the security requirements for methods.
- Specifying characteristics used to enforce security.
- Controlling optimizations by the just-in-time (JIT) compiler so the code remains easy to debug.
- Obtaining information about the caller to a method.

Reflection

- Reflection provides objects (of type Type) that describe assemblies, modules and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them.

```
// Using Reflection to get information from an Assembly:
System.Reflection.Assembly info = typeof(System.Int32).Assembly;
System.Console.WriteLine(info);
```

Reflection Overview

- When you have to access attributes in your program's metadata.
- For examining and instantiating types in an assembly.
- For building new types at runtime. Use classes in System.Reflection.Emit.
- For performing late binding, accessing methods on types created at run time.

Reflection Create Instance

```
public static object CreateInstance(this Type source, params object[] args)
{
    if (source == typeof(string))
    {
        return string.Empty;
    }

    object result = null;

    try
    {
        result = Activator.CreateInstance(source, args);
    }
    catch
    {
        result = FormatterServices.GetUninitializedObject(source);
    }

    return result;
}
```

Reflection Invoke Method

```
public static object InvokeMethod(this object source, string method, bool isPublicOnly = true, params object[] parameters)
{
    MethodInfo methodInfo = null;

    Type[] parameterTypes = parameters.Select(p => p.GetType()).ToArray();

    try
    {
        methodInfo = source.GetType().GetMethod(method, isPublicOnly ? PublicOnlyLookup : AllLookup, null, CallingConventions.Any, parameterTypes, null);
    }
    catch (AmbiguousMatchException)
    {
        var methods = source.GetType().GetMethods(isPublicOnly ? PublicOnlyLookup : AllLookup).Where(p => p.Name.Equals(method, StringComparison.OrdinalIgnoreCase) && !p.IsGenericMethod);

        foreach (MethodInfo item in methods)
        {
            var methodParameters = item.GetParameters();

            if (methodParameters.Length != parameters.Length)
            {
                continue;
            }

            for (int i = 0; i < methodParameters.Length; i++)
            {
                if (methodParameters[i].ParameterType != parameterTypes[i])
                {
                    break;
                }
            }

            methodInfo = item;

            break;
        }
    }

    return methodInfo.Invoke(source, parameters);
}
```

Reflection Get/Set Property

```
public static object GetPropertyValue(this object source, string propertyName,
bool isPublicOnly = true, params object[] index)
{
    return source.GetType().GetProperty(propertyName, isPublicOnly ?
PublicOnlyLookup : AllLookup).GetValue(source, index);
}


public static object SetPropertyValue(this object source, string propertyName,
object value, bool isPublicOnly = true, params object[] index)
{
    source.GetType().GetProperty(propertyName, isPublicOnly ? PublicOnlyLookup :
AllLookup).SetValue(source, value, index);

    return source;
}
```

Reflection and attribute

```
public static string ToDescriptionString(this Enum source)
{
    DescriptionAttribute descriptionAttribute =
source.GetType().GetField(source.ToString()).GetCustomAttributes(typeof(Description
Attribute), false).FirstOrDefault() as DescriptionAttribute;
    return (descriptionAttribute != null) ? descriptionAttribute.Description :
source.ToString();
}
```

Encryption

- Symmetric Encryption
- Asymmetric Encryption
- Hash

Symmetric Encryption

```
string Encrypt(string clearText)
{
    string EncryptionKey = "MAKV2SPBNI99212";
    byte[] clearBytes = Encoding.Unicode.GetBytes(clearText);
    using (Aes encryptor = Aes.Create())
    {
        Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, new byte[] { 0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65,
0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });
        encryptor.Key = pdb.GetBytes(32);
        encryptor.IV = pdb.GetBytes(16);
        using (MemoryStream ms = new MemoryStream())
        {
            using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateEncryptor(), CryptoStreamMode.Write))
            {
                cs.Write(clearBytes, 0, clearBytes.Length);
                cs.Close();
            }
            clearText = Convert.ToBase64String(ms.ToArray());
        }
    }
    return clearText;
}
```

Symmetric Decryption

```
string Decrypt(string cipherText)
{
    string EncryptionKey = "MAKV2SPBNI99212";
    byte[] cipherBytes = Convert.FromBase64String(cipherText);
    using (Aes encryptor = Aes.Create())
    {
        Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, new byte[] { 0x49, 0x76, 0x61, 0x6e, 
0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });
        encryptor.Key = pdb.GetBytes(32);
        encryptor.IV = pdb.GetBytes(16);
        using (MemoryStream ms = new MemoryStream())
        {
            using (CryptoStream cs = new CryptoStream(ms, 
encryptor.CreateDecryptor(), CryptoStreamMode.Write))
            {
                cs.Write(cipherBytes, 0, cipherBytes.Length);
                cs.Close();
            }
            cipherText = Encoding.Unicode.GetString(ms.ToArray());
        }
    }
    return cipherText;
}
```

Asymmetric Encryption

```
public static byte[] EncryptRSA(this byte[] source, string key = null,
bool fOAEP = false)
{
    if (source == null)
    {
        return null;
    }

    using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(new
CspParameters { KeyContainerName = key ?? string.Empty }))
    {
        return rsa.Encrypt(source, fOAEP);
    }
}
```

Asymmetric Decryption

```
public static byte[] DecryptRSA(this byte[] source, string key = null,
bool fOAEP = false)
{
    if (source == null)
    {
        return null;
    }

    using (RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(new CspParameters { KeyContainerName = key ??
string.Empty }))
    {
        return rsa.Decrypt(source, fOAEP);
    }
}
```

Hash

```
public static byte[] HashMD5(this byte[] source)
{
    using (MD5 hasher = MD5.Create())
    {
        return hasher.ComputeHash(source);
    }
}
```

What is assemblies

- An Assembly is a basic unit of application deployment and versioning.
- An Assembly is also called the building block of a .Net application.
- An Assembly is either a .exe or .dll file.

Assembly structure

- Assembly manifest (name,language,version).
- CIL code (logic part).
- Type information (Datatype).
- Resources.

Assembly Versioning

- Assembly Versioning is a new feature introduced in .Net that allows two versions of the same component that exists in a single machine and in a single folder side-by-side.
- A version number is assigned by the programmer and is not provided or controlled by .Net software.

```
// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Revision
//      Build Number
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("2.17.8")]
////[assembly: AssemblyFileVersion("1.0.0.0")]
```

Assembly and reflection

```
public static Type Load(string assemblyFile, string typeFullName, bool ignoreCase)
{
    if (string.IsNullOrEmpty(assemblyFile))
    {
        throw new ArgumentNullException("assemblyFile");
    }

    if (string.IsNullOrEmpty(typeFullName))
    {
        throw new ArgumentNullException("typeFullName");
    }

    if (!File.Exists(assemblyFile))
    {
        throw new FileNotFoundException("The specified assembly file does not exist.", assemblyFile);
    }

    try
    {
        Assembly assembly = Assembly.Load(File.ReadAllBytes(assemblyFile));
        return assembly.GetType(typeFullName, true, ignoreCase);
    }
    catch (Exception e)
    {
        InternalLogger.Log(e);
        throw;
    }
}
```

Practice project

- Architect and design
- Implementation
- Test

Recap

- Exam Ref 70-483
  - CHAPTER 3 Debug applications and implement security
    - Objective 3.2 Perform symmetric and asymmetric encryption
    - Objective 3.3 Manage assemblies

THANK YOU!

Q&A