A Summary for C# refs

Notebook:	C#		
Created:	6/11/2020 8:27 PM	**Updated:**	6/19/2020 9:42 PM
Author:	gggggzd@hotmail.com		

Summary
1.1

1. Parallel
    1. Parallel.Invoke()
    2. Parallel.ForEach() / Paraller.For()
        1. 传入ParallelLoopState参数来manage
2. Prallel LINQ
    1. from xxx in xxx.AsPrallel()
    2. from xxx in xxx.AsPrallel().WithDegreeOfParalleism(4).WithExecutionMode(ParallelExecutionMode.ForceParalle
    3. in xxx.AsParaller().AsOrder()
    4. selete xxx.AsSequential()
    5. result.ForAll(item => DoSomeWork(item))
    6. Exception in queries
3. Task
    1. Create a Task
        1. Task newTask = new Task(() => DoWork());
        2. Task.Run(()=>DoWork());
    2. return a value from a task:Task<int> task = Task.Run(() => { return CalculateResult(); });
    3. Task.Waitall()/ WaitAny()
    4. task.ContinueWith()
    5. Child tasks
    6. Cancelling a long-running task：
        1. There is an important difference between threads and tasks, in that a Thread can be aborted at any time, whereas a Task must monitor a cancellation token so that it will end when told to.

2.

```
using System;
using System.Threading;
using System.Threading.Tasks;
namespace LISTING_1_49_cancel_a_task
{
    class Program
    {
        static CancellationTokenSource cancellationTokenSource =
        new CancellationTokenSource();

        static void Clock()
        {
            while (!cancellationTokenSource.IsCancellationRequested)
            {
                Console.WriteLine("Tick");
                Thread.Sleep(500);
            }
        }

        static void Main(string[] args)
        {
            Task.Run(() => Clock());
            Console.WriteLine("Press any key to stop the clock");
            Console.ReadKey();
            cancellationTokenSource.Cancel();
            Console.WriteLine("Clock stopped");
            Console.ReadKey();
        }
    }
}
```

3. A Task can indicate that it has been cancelled by raising an exception. This can be useful if a task is started in one place and monitored in another.

```
static void Clock(CancellationToken cancellationToken)
{
    int tickCount = 0;

    while (!cancellationToken.IsCancellationRequested && tickCount<20)
    {
        tickCount++;
        Console.WriteLine("Tick");
        Thread.Sleep(500);
    }

    cancellationToken.ThrowIfCancellationRequested();
}
```

4. Threads
    1. Create a thread: 先new Thread(some method),再call Strat()
    2. Passing data into a thread
        1. ParameterizedThreadStart ps = new ParameterizedThreadStart(WorkOnData);Thread thread = new Thread(ps);
        2. 新建thread的时候不直接传给他目标方法名字，而是传给他一个能接收参数的Lambda表达式，表达式里调用方法。
    3. Abort a thread
        1. thread.Abort()
        2. 使用flag
    4. Thread Synchronization using join
    5. Thread data storage and ThreadLocal
    6. Thread execution context
    7. Thread pools

5. Tasks and the User Interface:例子

1.2

1. Locks

```
static object sharedTotalLock = new object();

static void addRangeOfValues(int start, int end)
{
    while (start < end)
    {
        lock (sharedTotalLock)
        {
            sharedTotal = sharedTotal + items[start];
        }
        start++;
    }
}
```

1.

2. Monitors: the atomic code is enclosed in calls of *Monitor.Enter* and *Monitor.Exit*.

1.

```
static object sharedTotalLock = new object();

static void addRangeOfValues(int start, int end)
{
    long subTotal = 0;

    while (start < end)
    {
        subTotal = subTotal + items[start];
        start++;
    }

    Monitor.Enter(sharedTotalLock);
    sharedTotal = sharedTotal + subTotal;
    Monitor.Exit(sharedTotalLock);
}
```

2. Monitor可以用 if(TryEnter(lockObject))

3. Interlocked operations：比lock更好

1. This provides a set of thread-safe operations that can be performed on a variable. These include increment, decrement, exchange (swap a variable with another), and add.
2. 例子：

```
static void addRangeOfValues(int start, int end)
{
    long subTotal = 0;

    while (start < end)
    {
        subTotal = subTotal + items[start];
        start++;
    }

    Interlocked.Add(ref sharedTotal, subTotal);
}
```

4. Volatile variables

1. 例子：先执行第一个用到x之后用到y，再用到x,编译器会交换第一个和第二个语句的执行顺序，可能会导致多线程问题，C#可能会交换第一个和第二个语句的执行顺序,用violate关键字解决:

```
int x;
int y=0;
x = 99;
y = y + 1;
Console.WriteLine("The answer is: {0}", x);
```

5. Implementing thread-safe methods

1. Thread safety and method parameters：值类型参数传入的是copy，类型安全，引用类型传入的是引用
2. 多个线程访问一个类里的方法的时候要考虑线程安全

Skill 2.4 Create and implement a class hierarchy

1. Create and implement classes based on the IComparable, IEnumerable, IDisposable, and IUnknown interfaces
    1. **IComparable**:用来确定Sort的时候Object的顺序
        1. 类里实现public int CompareTo(object obj):如果<0则放在被比较对象之前，等于0则放在相同位置，<0则放在之后。

```
public class BankAccount : IAccount, IComparable
{
    public int CompareTo(object obj)
    {
        // if we are being compared with a null object we are definitely after it
        if (obj == null) return 1;

        // Convert the object reference into an account reference
        IAccount account = obj as IAccount;

        // as generates null if the conversion fails
        if (account == null)
            throw new ArgumentException("Object is not an account");

        // use the balance value as the basis of the comparison
        return this.balance.CompareTo(account.GetBalance());
    }

}
```

    2. IEnumerable: GetEnmerator返回enumerator,enumerator有MoveNext():boolean方法，返回真则说明可以Move到下一个

```
namespace LISTING_2_41_Get_an_enumerator
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get an enumerator that can iterate through a string
            var stringEnumerator = "Hello world".GetEnumerator();

            while(stringEnumerator.MoveNext())
            {
                Console.Write(stringEnumerator.Current);
            }

            Console.ReadKey();
        }
    }
}
```

        1. Making an object enumerable：the **IEnumerable** interface (meaning it can be enumerated) and the **IEenumerator<int>** interface (meaning that it contains a call of **GetEnumerator** to get an enumerator from it).

```
class EnumeratorThing : IEnumerator<int>, IEnumerable
{
    int count;
    int limit;

    public int Current
    {
        get
        {
            return count;
        }
    }

    object IEnumerator.Current
    {
        get
        {
            return count;
        }
    }

    public void Dispose()
    {
    }

    public bool MoveNext()
    {
        if (++count == limit)
            return false;
        else

            return true;
    }

    public void Reset()
    {
        count = 0;
    }

    public IEnumerator GetEnumerator()
    {
        return this;
    }

    public EnumeratorThing(int limit)
    {
        count = 0;
        this.limit = limit;
    }
}
```

You can use an `EnumeratorThing` instance in a `foreach` loop:

```
   EnumeratorThing e = new EnumeratorThing(10);

foreach(int i in e)
    Console.WriteLine(i);
```

3. IDisposable

1. definition:

```
//
// Summary:
//     Provides a mechanism for releasing unmanaged resources.
public interface IDisposable
{
    //
    // Summary:
    //     Performs application-defined tasks associated with freeing, releasing, or
          resetting
    //     unmanaged resources.
    void Dispose();
}
```

2. DIspose方法不会在对象被内存中删去的时候自动调用，只能要么直接call Dispose，要么用using

Skill 2.5: Find, execute, and create types at runtime by using reflection

1. **Serializable**: means that the class is may be opened and read by a serializer.Note that some serializers, notably the XMLSerializer and the JSONSerializer, don't need classes to be marked as serializable before they can work with them.
2. **Conditional** attributes : used to activate and de-activate the contents of methods.
    1. code below shows how it is used. The symbols TERSE and VERBOSE can be used to select the level of logging that is performed by a program. If the TERSE symbol is defined the body of the terseReport method will be obeyed when the method is called. If the VERBOSE symbol is defined the body of the verboseReport method will be obeyed. The body of the reportHeader method will be obeyed if either the TERSE or the VERBOSE

symbols are defined because two attributes are combined before that method definition.

```
//#define TERSE
#define VERBOSE

using System;
using System.Diagnostics;

namespace LISTING_2_47_The_conditional_attribute
{
    class Program
    {
        [Conditional("VERBOSE"), Conditional("TERSE")]
        static void reportHeader()
        {
            Console.WriteLine("This is the header for the report");
        }

        [Conditional("VERBOSE")]
        static void verboseReport()
        {
            Console.WriteLine("This is output from the verbose report.");
        }

        [Conditional("TERSE")]
        static void terseReport()
        {
            Console.WriteLine("This is output from the terse report.");
        }

        static void Main(string[] args)
        {
            reportHeader();
            terseReport();
            verboseReport();
            Console.ReadKey();
        }
    }
}
```

2. conditional attritube 不阻止方法进入编译器，而是阻止执行
3. **Attribute.IsDefined():**来判断一个类是否attached了某个attribute, IsDefined接受两个参数：1. 要判断的类 2.某个attritube

```
if (Attribute.IsDefined(typeof(Person), typeof(SerializableAttribute)))
    Console.WriteLine("Person can be serialized");
```

1.
2. Note that although the attribute is called Serializable when it is used in the source code, the name of the class that implements the attribute has the text Attribute appended to it, so that the attribute class we are looking for is called SerializableAttribute. The convention of adding the “Attribute” to the end of attribute classes is one that should be followed when creating our own attributes.
4. Creating attribute classes

1. e.g.

```
class ProgrammerAttribute: Attribute
{
    private string programmerValue;

    public ProgrammerAttribute (string programmer)
    {
        programmerValue = programmer;
    }

    public string Programmer
    {
        get
        {
            return programmerValue;
        }
    }
}
```

2. 使用的时候：

```
 [ProgrammerAttribute(programmer:"Fred")]
class Person
{
    public string Name { get; set; }
}
```

5. Controlling the use of Attributes
   1. 可以用AttributeUseage来控制Attribute的使用，e.g.只用在class上：

```
[AttributeUsage(AttributeTargets.Class)]
class ProgrammerAttribute: Attribute
{
…
}
```

   2. e.g.用在field和Property上

```
 [AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
class FieldOrProp: Attribute
{
…
}
```

6. **GetCustomAttribute():** Read attributes

```
Attribute a = Attribute.GetCustomAttribute( typeof(Person),
typeof(ProgrammerAttribute));

ProgrammerAttribute p = (ProgrammerAttribute)a;

Console.WriteLine("Programmer: {0}", p.Programmer);
```

   1.

7. **Using reflection**
   1. **GetType():**get the type of an instance.

```
System.Type type;

Person p = new Person();
type = p.GetType();
Console.WriteLine("Person type: {0}", type.ToString());
```

      1. e.g.
   2. prints all the members of the Person.:

1. e.g.

```
using System;
using System.Reflection;

namespace LISTING_2_53_Investigating_a_type
{
    class Person
    {
        public string Name { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            System.Type type;

            Person p = new Person();
            type = p.GetType();

            foreach(MemberInfo member in type.GetMembers() )
            {
                Console.WriteLine(member.ToString());
            }

            Console.ReadKey();
        }
    }
}
```

>>>output:

```
System.String get_Name()
Void set_Name(System.String)
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
System.Type GetType()
Void .ctor()
System.String Name
```

3. Calling a method on an object by using reflection
    1. e.g.手动调用属性的Set方法

```
System.Type type;

Person p = new Person();
type = p.GetType();

MethodInfo setMethod = type.GetMethod("set_Name");
setMethod.Invoke(p, new object [] { "Fred" });

Console.WriteLine(p.Name);
```

8. Managed Extensibility Framework (MEF)
    1. code below shows how we can search an assembly for a particular type of component class. It can be used in a banking application to find all the classes that implement the IAccount interface.The code uses the IsAssignableFrom method to decide whether or not a given type implements the IAccount interface.

```
Assembly thisAssembly = Assembly.GetExecutingAssembly();

List<Type> AccountTypes = new List<Type>();

foreach ( Type t in thisAssembly.GetTypes() )
{
    if (t.IsInterface)
        continue;

    if(typeof(IAccount).IsAssignableFrom(t))
    {
        AccountTypes.Add(t);
    }
}
```

LINQ版本：

```
var AccountTypes = from type in thisAssembly.GetTypes()
                   where typeof(IAccount).IsAssignableFrom(type) && !type.IsInterface
                   select type;
```

2. It is possible to load an assembly from a file by using the Assembly.Load method. The statement below would load all the types in a file called BankTypes.dll

```
Assembly bankTypes = Assembly.Load("BankTypes.dll");
```

9. **Generate code at runtime by using CodeDOM and Lambda expressions**
    1. **CodeDOM**
        1. 创建CodeDOM

```
CodeCompileUnit compileUnit = new CodeCompileUnit();

// Create a namespace to hold the types we are going to create
CodeNamespace personnelNameSpace = new CodeNamespace("Personnel");

// Import the system namespace
personnelNameSpace.Imports.Add(new CodeNamespaceImport("System"));
// Create a Person class
CodeTypeDeclaration personClass = new CodeTypeDeclaration("Person");
personClass.IsClass = true;
personClass.TypeAttributes = System.Reflection.TypeAttributes.Public;

// Add the Person class to personnelNamespace

personnelNameSpace.Types.Add(personClass);

// Create a field to hold the name of a person
CodeMemberField nameField = new CodeMemberField("String", "name");
nameField.Attributes = MemberAttributes.Private;

// Add the name field to the Person class
personClass.Members.Add(nameField);

// Add the namespace to the document
compileUnit.Namespaces.Add(personnelNameSpace);
```

2. 用**CodeDomProvider**读取刚刚创建的**CodeDOM**

```
// Now we need to send our document somewhere
// Create a provider to parse the document
CodeDomProvider provider = CodeDomProvider.CreateProvider("CSharp");
// Give the provider somewhere to send the parsed output
StringWriter s = new StringWriter();
// Set some options for the parse - we can use the defaults
CodeGeneratorOptions options = new CodeGeneratorOptions();

// Generate the C# source from the CodeDOM
provider.GenerateCodeFromCompileUnit(compileUnit, s, options);
// Close the output stream
s.Close();

// Print the C# output
Console.WriteLine(s.ToString());
```

输出：

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Personnel {
    using System;


    public class Person {

        private String name;
    }
}
```

3. **Assembly**

1. 读取**Assmebly**的信息

```
using System;
using System.Reflection;

namespace LISTING_2_60_Assembly_object
{
    class Program
    {
        static void Main(string[] args)
        {
            Assembly assembly = Assembly.GetExecutingAssembly();

            Console.WriteLine("Full name: {0}", assembly.FullName);
            AssemblyName name = assembly.GetName();
            Console.WriteLine("Major Version: {0}", name.Version.Major);
            Console.WriteLine("Minor Version: {0}", name.Version.Minor);
            Console.WriteLine("In global assembly cache: {0}",
            assembly.GlobalAssemblyCache);
            foreach (Module assemblyModule in assembly.Modules)
            {
                Console.WriteLine("  Module: {0}", assemblyModule.Name);
                foreach (Type moduleType in assemblyModule.GetTypes())
                {
                    Console.WriteLine("     Type: {0}", moduleType.Name);
                    foreach (MemberInfo member in moduleType.GetMembers())
                {
                     Console.WriteLine("        Member: {0}", member.Name);
                }
            }
        }

        Console.ReadKey();
    }
}
}
```

2. **PropertyInfo**

```
using System;
using System.Reflection;

namespace LISTING_2_61_Property_info
{
    public class Person
    {
        public String Name { get; set; }

        public String Age { get; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Type type = typeof(Person);

            foreach (PropertyInfo p in type.GetProperties())
            {
                Console.WriteLine("Property name: {0}", p.Name);
                if(p.CanRead)
                {
                    Console.WriteLine("Can read");
                    Console.WriteLine("Set method: {0}", p.GetMethod);
                }
                if (p.CanWrite)
                {
                    Console.WriteLine("Can write");
                    Console.WriteLine("Set method: {0}", p.SetMethod);
                }
            }


                    Console.ReadKey();
                }
            }
        }
```

3. **MethodInfo**

```
using System;
using System.Reflection;

namespace LISTING_2_62_Method_reflection
{
    public class Calculator
    {
        public int AddInt(int v1, int v2)
        {
            return v1 + v2;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Get the type information for the Calculator class");
            Type type = typeof(Calculator);

            Console.WriteLine("Get the method info for the AddInt method");
            MethodInfo AddIntMethodInfo = type.GetMethod("AddInt");

            Console.WriteLine("Get the IL instructions for the AddInt method");
            MethodBody AddIntMethodBody = AddIntMethodInfo.GetMethodBody();

            // Print the IL instructions.
            foreach (byte b in AddIntMethodBody.GetILAsByteArray())
            {
                Console.Write(" {0:X}", b);
            }
            Console.WriteLine();

            Console.WriteLine("Create Calculator instance");
            Calculator calc = new Calculator();

            Console.WriteLine("Create parameter array for the method");


            object[] inputs = new object[] { 1, 2 };

            Console.WriteLine("Call Invoke on the method info");
            Console.WriteLine("Cast the result to an integer");
            int result = (int) AddIntMethodInfo.Invoke(calc, inputs);
            Console.WriteLine("Result of: {0}", result);

            Console.WriteLine("Call InvokeMember on the type");
            result = (int) type.InvokeMember("AddInt",
                        BindingFlags.InvokeMethod |
                        BindingFlags.Instance | BindingFlags.Public,
                        null,
                        calc,
                        inputs);
            Console.WriteLine("Result of: {0}", result);

            Console.ReadKey();
        }
    }
}
```

4. The **GetType** method can be called on any instance to obtain a reference to the type for that object, and the **typeof** method can be used on any type to obtain the type object that describes that type.

Skill 2.6: Manage the object life cycle

1. Garbage collection in .NET
    1. **objects**: heap v**alue types**: stack
    2. { int i = 0} 不会给GC带来工作，因为int是值类型存放在stack frame里
    3. 手动调用GC：GC.Collect();等待Collect()方法的时候Finalizer都完成：GC.WaitForPendingFinalizers():
    4. 忽略一个对象的Finalzier: GC.SuppressFinalize(p);取消忽略：GC.ReRegisterForFinalize(p);
    5. **finalizer**

1. . An object can contain a finalizer method that is invoked by the garbage collector in advance of that object being removed from memory.可以用来释放这个对象使用的资源。
2. e.g.

```
public class Person
{
    long[] personArray = new long[1000000];

    ~Person()
    {
        // This is where the person would be finalized
        Console.WriteLine("Finalizer called");
    }
}
```

3. 如果程序运行不缺少内存，可能finalizer永远不会被调用
4. finalizer方法在从内存中删除之前调用，Dispose方法被调用不会导致对象从内存中删除，Only objects that have no references to them are deleted.
5. Using IDisposable and finalization on the same object
    1. 手动调用Dispose或者using的时候就不让finalizer来finalize了，此时free both managed 和unmanaged objects，并且SuppressFinalize。GC调用finalizer的时候在finalizer里面调用Dispose，并且只free unmanaged objects

```
using System;

namespace LISTING_2_65_The_dispose_pattern
{
    class ResourceHolder : IDisposable
    {

        // Flag to indicate when the object has been
        // disposed
        bool disposed = false;

        public void Dispose()
        {
            // Call dispose and tell it that
            // it is being called from a Dispose call
            Dispose(true);
            GC.SuppressFinalize(this);
        }


        public virtual void Dispose(bool disposing)
        {
            // Give up if already disposed
            if (disposed)
                return;

            if(disposing)
            {
                // free any managed objects here
            }

            // Free any unmanaged objects here
        }

        ~ResourceHolder()
        {
            // Dispose only of unmanaged objects
            Dispose(false);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ResourceHolder r = new ResourceHolder();

            r.Dispose();
        }
    }
}
```

Skill 2.7: Manipulate strings

1. String是引用类型，但是一个String被创建了之后就不可以改变值，所有String的editing方法都是返回一个新的String，这样保证了线程安全，但是可以用StringBuilder来创建mutable string object
2. String Interning: The compiler will save program memory by making both s1 and s2 refer to the same string object with the content “hello.”

```
string s1 = "hello";
string s2 = "hello";
```

3. **Manipulate strings by using the StringBuilder, StringWriter, and StringReader classes**
    1. **StringBuilder**
        1. It can improve the speed of a program while at the same time making less work for the garbage collector
            1. 不用StringBuilder: 会产生一个中间String *firstName* + " "

```
string firstName = "Rob";
string secondName = "Miles";

string fullName = firstName + " " + secondName;
```

使用StringBuilder: The StringBuilder type provides methods that let a program treat a string as a mutable object.

```
StringBuilder fullNameBuilder = new StringBuilder();
fullNameBuilder.Append(firstName);
fullNameBuilder.Append(" ");
fullNameBuilder.Append(secondName);
Console.WriteLine(fullNameBuilder.ToString());
```

2. **StringWriter:**It implements the TextWriter interface, so programs can send their output into a string.

3. **StringReader:**It is a convenient way of getting string input into a program that would like to read its input from a stream.

2. Search strings
    1. Contains(string): boolean

```
string input = "      Rob Miles";

if(input.Contains("Rob"))
{
    Console.WriteLine("Input contains Rob");
}
```

    2. **StartsWith** and **EndsWith**
    3. **TrimStart** creates a new string with whitespace removed from the start, **TrimEnd** removes whitespace from the end of the string and Trim removes whitespace from both ends.

```
string input = "      Rob Miles";

string trimmedString = input.TrimStart();

if(trimmedString.StartsWith("Rob"))
{
    Console.WriteLine("Starts with Rob");
}
```

    4. **IndexOf** and **SubString**:The method **IndexOf** returns an integer which gives the position of the first occurrence of a character or string in a string. There is also a **LastIndexOf** method that will give the position of the last occurrence of a string. There are overloads for these methods that let you specify the start position for the search. You can use these position values with the SubString method to extract a particular substring from a string.
    5. Replace()

6. Split():The Split method can be used to split a string into a number of substrings.

```
string sentence = "The cat sat on the mat.";
string[] words = sentence.Split(' ');
foreach (string word in words)
{
    Console.WriteLine(word);
}
```

3. String comparison and cultures

```
// Default comparison fails because the strings are different
if (!"encyclopædia".Equals("encyclopaedia"))
    Console.WriteLine("Unicode encyclopaedias are not equal");
// Set the curent culture for this thread to EN-US
Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture("en-US");
// Using the current culture the strings are equal
if ("encyclopædia".Equals("encyclopaedia", StringComparison.CurrentCulture))
    Console.WriteLine("Culture comparison encyclopaedias are equal");
// We can use the IgnoreCase option to perform comparisions that ignore case
if ("encyclopædia".Equals("ENCYCLOPAEDIA", StringComparison.CurrentCultureIgnoreCase))
    Console.WriteLine("Case culture comparison encyclopaedias are equal");
if (!"encyclopædia".Equals("ENCYCLOPAEDIA", StringComparison.OrdinalIgnoreCase))
    Console.WriteLine("Ordinal comparison encyclopaedias are not equal");
```

1.

4. Enumerate string methods: You can regard a string as an array of characters.
5. Format strings
    1. e.g.

```
int i = 99;
double pi = 3.141592654;

Console.WriteLine(" {0,-10:D}{0, -10:X}{1,5:N2}", i, pi);
```

花括号内第一个是参数列表中的第几个item，第二个是占据多少个空间(-号代表左对齐)，：号后的是formatting information for the item. The formatting information for an integer string conversion can comprise the character ‘D’ meaning decimal string and ‘X’ meaning hexadecimal string. The formatting information for a floating-point number can comprise the character ‘N,’N2代表精度是2位

输出：

```
99          63              3.14
```

    2. IFormattable
        1. Any type that implements the **IFormattable** interface will contain a **ToString** method that can be used to request formatted conversion of values into a string.

```
class MusicTrack : IFormattable
{
    string Artist { get; set; }
    string Title { get; set; }

    // ToString that implements the formatting behavior
    public string ToString(string format, IFormatProvider formatProvider)
    {
        // Select the default behavior if no format specified
        if (string.IsNullOrWhiteSpace(format))
            format = "G";

        switch (format)
        {
            case "A": return Artist;
            case "T": return Title;
            case "G": // default format
            case "F": return Artist + " " + Title;
            default:
                throw new FormatException("Format specifier was invalid.");
        }
    }

    // ToString that overrides the behavior in the base class
    public override string ToString()
    {
        return Artist + " " + Title;
    }

    public MusicTrack(string artist, string title)
    {
        Artist = artist;
        Title = title;
    }
}
```

```
MusicTrack song = new MusicTrack(artist: "Rob Miles", title: "My Way");

Console.WriteLine("Track: {0:F}", song);
Console.WriteLine("Artist: {0:A}", song);
Console.WriteLine("Title: {0:T}", song);
```

3. Use string interpolation
    1. 原来：

    ```
    string name = "Rob";
    int age = 21;
    Console.WriteLine("Your name is {0} and your age is {1,-5:D}", name, age);
    ```

    2. 使用interpolation：

    ```
    Console.WriteLine($"Your name is {name} and your age is {age,-5:D}");
    ```