**接口（C# 编程指南）**

接口包含[类](https://msdn.microsoft.com/zh-CN/library/0b0thckt.aspx)或[结构](https://msdn.microsoft.com/zh-CN/library/ah19swz4.aspx)可以实现的一组相关功能的定义。

例如，使用接口可以在类中包括来自多个源的行为。 由于C#语言不支持多重继承，所以该功能很重要。 此外，如果要模拟结构的继承，也必须使用接口，因为它们无法实际从另一个结构或类继承。

可使用 [interface](https://msdn.microsoft.com/zh-CN/library/87d83y5b.aspx) 关键字定义接口，如以下示例所示。

interface IEquatable<T>

{

bool Equals(T obj);

}

实现 [IEquatable<T>](https://msdn.microsoft.com/zh-CN/library/ms131187.aspx) 接口的任何类或结构都必须包含与该接口指定的签名匹配的 [Equals](https://msdn.microsoft.com/zh-CN/library/ms131190.aspx) 方法的定义。因此，可以依靠实现 IEquatable<T> 的类来包含 Equals 方法，类的实例可以通过该方法确定它是否等于相同类的另一个实例。

IEquatable<T> 的定义不为 Equals 提供实现。该接口仅定义签名。这样，C# 中的接口便类似于其中所有方法都是抽象方法的抽象类。但是，类或结构可以实现多个接口，但是类只能继承单个类（抽象或不抽象）。因此，使用接口可以在类中包括来自多个源的行为。

有关抽象类的详细信息，请参阅[抽象类、密封类及类成员](https://msdn.microsoft.com/zh-CN/library/ms173150.aspx)。

接口可以包含方法、属性、事件、索引器或这四种成员类型的任意组合。有关示例的链接，请参阅[相关章节](https://wizardforcel.gitbooks.io/msdn-csharp/content/guide/86.html#BKMK_RelatedSections)。接口不能包含常量、字段、运算符、实例构造函数、析构函数或类型。接口成员会自动成为公共成员，不能包含任何访问修饰符。成员也不能是[静态](https://msdn.microsoft.com/zh-CN/library/98f28cdx.aspx)成员。

若要实现接口成员，实现类的对应成员必须是公共、非静态，并且具有与接口成员相同的名称和签名。

当类或结构实现接口时，类或结构必须为该接口定义的所有成员提供实现。接口本身不提供类或结构可以通过继承基类功能的方式来继承的任何功能。但是，如果基类实现接口，则从基类派生的任何类都会继承该实现。

下面的示例演示 IEquatable<T> 接口的实现。实现类 Car 必须提供 [Equals](https://msdn.microsoft.com/zh-CN/library/ms131190.aspx) 方法的实现。

public class Car : IEquatable<Car>

{

public string Make {get; set;}

public string Model { get; set; }

public string Year { get; set; }

// Implementation of IEquatable<T> interface

public bool Equals(Car car)

{

if (this.Make == car.Make &&

this.Model == car.Model &&

this.Year == car.Year)

{

return true;

}

else

return false;

}

}

类的属性和索引器可以为接口中定义的属性或索引器定义额外的访问器。例如，一个接口可能会声明一个具有 [get](https://msdn.microsoft.com/zh-CN/library/ms228503.aspx) 访问器的属性。实现该接口的类可以声明同时具有 **get** 和 [set](https://msdn.microsoft.com/zh-CN/library/ms228368.aspx) 访问器的相同属性。但是，如果属性或索引器使用显式实现，则访问器必须匹配。有关显式实现的详细信息，请参阅[显式接口实现（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173157.aspx)和[接口属性（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/64syzecx.aspx)。

接口可以实现其他接口。类可能通过它继承的基类或通过其他接口实现的接口来多次包含某个接口。但是，类只能提供接口的实现一次，并且仅当类将接口作为类定义的一部分 (class ClassName : InterfaceName) 进行声明时才能提供。如果由于继承实现接口的基类而继承了接口，则基类会提供接口的成员的实现。但是，派生类可以重新实现接口成员，而不是使用继承的实现。

基类还可以使用虚拟成员实现接口成员。在这种情况下，派生类可以通过重写虚拟成员来更改接口行为。有关虚拟成员的详细信息，请参阅[多态性](https://msdn.microsoft.com/zh-CN/library/ms173152.aspx)。

**接口摘要**

接口具有以下属性：

* 接口类似于抽象基类。实现接口的任何类或结构都必须实现其所有成员。
* 接口无法直接进行实例化。其成员由实现接口的任何类或结构来实现。
* 接口可以包含事件、索引器、方法和属性。
* 接口不包含方法的实现。
* 一个类或结构可以实现多个接口。一个类可以继承一个基类，还可实现一个或多个接口。

**本节内容**

[显式接口实现（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173157.aspx)

说明如何创建特定于接口的类成员。

[如何：显式实现接口成员（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/44a9ty12.aspx)

提供有关如何显式实现接口的成员的示例。

[如何：显式实现两个接口的成员（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/4taxa8t2.aspx)

提供有关如何使用继承显式实现接口的成员的示例。

**相关章节**

* [接口属性（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/64syzecx.aspx)
* [接口中的索引器（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/tkyhsw31.aspx)
* [如何：实现接口事件（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ak9w5846.aspx)
* [类和结构（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173109.aspx)
* [继承（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173149.aspx)
* [方法（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173114.aspx)
* [多态性（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173152.aspx)
* [抽象类、密封类及类成员（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173150.aspx)
* [属性（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/x9fsa0sw.aspx)
* [事件（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/awbftdfh.aspx)
* [索引器（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/6x16t2tx.aspx)

**重要章节**

[Learning C# 3.0: Master the Fundamentals of C# 3.0（学习 C# 3.0：掌握 C# 3.0 的基本知识）](https://msdn.microsoft.com/library/orm-9780596521066-01.aspx)中的 [Interfaces（接口）](https://msdn.microsoft.com/library/orm-9780596521066-01-13.aspx)

**请参阅**

[C# 编程指南](https://msdn.microsoft.com/zh-CN/library/67ef8sbd.aspx)

[继承（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173149.aspx)

## 完整实例

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace csharpbasic\_interface  {  interface IEquatable<T>  {  bool Equals(T t);  }  public class Car : IEquatable<Car>  {  public string Make { get; set; }  public string Model { get; set; }  public string Year { get; set; }  public Car(string make, string model, string year)  {  Make = make;  Model = model;  Year = year;  }  public bool Equals(Car t)  {  return (this.Make == t.Make && this.Model == t.Model && this.Year == t.Year);  }  }  internal class Program  {  static void TestInterface()  {  Car c1 = new Car("TOYOTA", "COROLLA", "1987");  Car c2 = new Car("TOYOTA", "CAMERY", "1989");  Console.WriteLine(c1.Equals(c2));//False  Car c3 = new Car("TOYOTA", "COROLLA", "1987");  Console.WriteLine(c3.Equals(c1));//True  }  static void Main(string[] args)  {  TestInterface();  }  }  } |

# 显式接口实现（C# 编程指南）

如果[类](https://msdn.microsoft.com/zh-CN/library/0b0thckt.aspx)实现两个接口，并且这两个接口包含具有相同签名的成员，那么在类中实现该成员将导致两个接口都使用该成员作为它们的实现。在下面的示例中，所有对 Paint 调用方法相同。

class Test

{

static void Main()

{

SampleClass sc = new SampleClass();

IControl ctrl = (IControl)sc;

ISurface srfc = (ISurface)sc;

// The following lines all call the same method.

sc.Paint();

ctrl.Paint();

srfc.Paint();

}

}

interface IControl

{

void Paint();

}

interface ISurface

{

void Paint();

}

class SampleClass : IControl, ISurface

{

// Both ISurface.Paint and IControl.Paint call this method.

public void Paint()

{

Console.WriteLine("Paint method in SampleClass");

}

}

// Output:

// Paint method in SampleClass

// Paint method in SampleClass

// Paint method in SampleClass

然而，如果两个[接口](https://msdn.microsoft.com/zh-CN/library/87d83y5b.aspx)成员执行不同的函数，那么这可能会导致其中一个接口的实现不正确或两个接口的实现都不正确。可以显式地实现接口成员 -- 即创建一个仅通过该接口调用并且特定于该接口的类成员。这是使用接口名称和一个句点命名该类成员来实现的。例如：

public class SampleClass : IControl, ISurface

{

void IControl.Paint()

{

System.Console.WriteLine("IControl.Paint");

}

void ISurface.Paint()

{

System.Console.WriteLine("ISurface.Paint");

}

}

类成员 **IControl.Paint** 只能通过 **IControl** 接口使用，**ISurface.Paint** 只能通过 **ISurface** 使用。两个方法实现都是分离的，都不可以直接在类中使用。例如：

// Call the Paint methods from Main.

SampleClass obj = new SampleClass();

//obj.Paint(); // Compiler error.

IControl c = (IControl)obj;

c.Paint(); // Calls IControl.Paint on SampleClass.

ISurface s = (ISurface)obj;

s.Paint(); // Calls ISurface.Paint on SampleClass.

// Output:

// IControl.Paint

// ISurface.Paint

显式实现还用于解决两个接口分别声明具有相同名称的不同成员（如属性和方法）的情况：

interface ILeft

{

int P { get;}

}

interface IRight

{

int P();

}

为了同时实现两个接口，类必须对属性 P 和/或方法 P 使用显式实现以避免编译器错误。例如：

class Middle : ILeft, IRight

{

public int P() { return 0; }

int ILeft.P { get { return 0; } }

}

## 实例2

本示例声明一个 [接口](https://msdn.microsoft.com/zh-CN/library/87d83y5b.aspx)IDimensions 和一个类 Box，该类显式实现接口成员 getLength 和 getWidth。通过接口实例 dimensions 访问这些成员。

interface IDimensions

{

float getLength();

float getWidth();

}

class Box : IDimensions

{

float lengthInches;

float widthInches;

Box(float length, float width)

{

lengthInches = length;

widthInches = width;

}

// Explicit interface member implementation:

float IDimensions.getLength()

{

return lengthInches;

}

// Explicit interface member implementation:

float IDimensions.getWidth()

{

return widthInches;

}

static void Main()

{

// Declare a class instance box1:

Box box1 = new Box(30.0f, 20.0f);

// Declare an interface instance dimensions:

IDimensions dimensions = (IDimensions)box1;

// The following commented lines would produce compilation

// errors because they try to access an explicitly implemented

// interface member from a class instance:

//System.Console.WriteLine("Length: {0}", box1.getLength());

//System.Console.WriteLine("Width: {0}", box1.getWidth());

// Print out the dimensions of the box by calling the methods

// from an instance of the interface:

System.Console.WriteLine("Length: {0}", dimensions.getLength());

System.Console.WriteLine("Width: {0}", dimensions.getWidth());

}

}

/\* Output:

Length: 30

Width: 20

\*/

## 可靠编程

* 请注意 **Main** 方法中下列代码行被注释掉，因为它们将产生编译错误。显式实现的接口成员不能从[类](https://msdn.microsoft.com/zh-CN/library/0b0thckt.aspx)实例访问：
* //System.Console.WriteLine("Length: {0}", box1.getLength());
* //System.Console.WriteLine("Width: {0}", box1.getWidth());
* 还请注意，**Main** 方法中的下列代码行成功输出框的尺寸，因为这些方法是从接口实例调用的：
* System.Console.WriteLine("Length: {0}", dimensions.getLength());

System.Console.WriteLine("Width: {0}", dimensions.getWidth());

## 实例3

显式[接口](https://msdn.microsoft.com/zh-CN/library/87d83y5b.aspx)实现还允许程序员实现具有相同成员名称的两个接口，并为每个接口成员各提供一个实现。本示例同时以公制单位和英制单位显示框的尺寸。Box [类](https://msdn.microsoft.com/zh-CN/library/0b0thckt.aspx)实现 IEnglishDimensions 和 IMetricDimensions 两个接口，它们表示不同的度量系统。两个接口有相同的成员名称 Length 和 Width。

// Declare the English units interface:

interface IEnglishDimensions

{

float Length();

float Width();

}

// Declare the metric units interface:

interface IMetricDimensions

{

float Length();

float Width();

}

// Declare the Box class that implements the two interfaces:

// IEnglishDimensions and IMetricDimensions:

class Box : IEnglishDimensions, IMetricDimensions

{

float lengthInches;

float widthInches;

public Box(float length, float width)

{

lengthInches = length;

widthInches = width;

}

// Explicitly implement the members of IEnglishDimensions:

float IEnglishDimensions.Length()

{

return lengthInches;

}

float IEnglishDimensions.Width()

{

return widthInches;

}

// Explicitly implement the members of IMetricDimensions:

float IMetricDimensions.Length()

{

return lengthInches \* 2.54f;

}

float IMetricDimensions.Width()

{

return widthInches \* 2.54f;

}

static void Main()

{

// Declare a class instance box1:

Box box1 = new Box(30.0f, 20.0f);

// Declare an instance of the English units interface:

IEnglishDimensions eDimensions = (IEnglishDimensions)box1;

// Declare an instance of the metric units interface:

IMetricDimensions mDimensions = (IMetricDimensions)box1;

// Print dimensions in English units:

System.Console.WriteLine("Length(in): {0}", eDimensions.Length());

System.Console.WriteLine("Width (in): {0}", eDimensions.Width());

// Print dimensions in metric units:

System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());

System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());

}

}

/\* Output:

Length(in): 30

Width (in): 20

Length(cm): 76.2

Width (cm): 50.8

\*/

## 可靠编程

如果希望默认度量采用英制单位，请正常实现 Length 和 Width 这两个方法，并从 IMetricDimensions 接口显式实现 Length 和 Width 方法：

// Normal implementation:

public float Length()

{

return lengthInches;

}

public float Width()

{

return widthInches;

}

// Explicit implementation:

float IMetricDimensions.Length()

{

return lengthInches \* 2.54f;

}

float IMetricDimensions.Width()

{

return widthInches \* 2.54f;

}

这种情况下，可以从类实例访问英制单位，而从接口实例访问公制单位：

public static void Test()

{

Box box1 = new Box(30.0f, 20.0f);

IMetricDimensions mDimensions = (IMetricDimensions)box1;

System.Console.WriteLine("Length(in): {0}", box1.Length());

System.Console.WriteLine("Width (in): {0}", box1.Width());

System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());

System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());

}

## 请参阅

[C# 编程指南](https://msdn.microsoft.com/zh-CN/library/67ef8sbd.aspx)

[类和结构（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173109.aspx)

[接口（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173156.aspx)

[继承（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173149.aspx)