# 转换运算符（C# 编程指南）

C# 允许程序员在类或结构上声明转换，以便类或结构与其他类或结构或者基本类型进行相互转换。转换的定义方法类似于运算符，并根据它们所转换到的类型命名。要转换的参数类型或转换结果的类型必须是（不能两者同时都是）包含类型。

class SampleClass

{

public static explicit operator SampleClass(int i)

{

SampleClass temp = new SampleClass();

// code to convert from int to SampleClass...

return temp;

}

}

## 转换运算符概述

转换运算符具有以下特点：

* 声明为 **implicit** 的转换在需要时自动进行。
* 声明为 **explicit** 的转换需要调用强制转换。
* 所有转换都必须声明为 **static**。

# 用户定义的显式和隐式转换运算符

用户定义类型可以定义从或到另一个类型的自定义隐式或显式转换。 隐式转换无需调用特殊语法，并且可以在各种情况（例如，在赋值和方法调用中）下发生。 预定义的 C# 隐式转换始终成功，且永远不会引发异常。 用户定义隐式转换也应如此。 如果自定义转换可能会引发异常或丢失信息，请将其定义为显式转换。

[is](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/type-testing-and-cast#is-operator) 和 [as](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/type-testing-and-cast#as-operator) 运算符不考虑使用用户定义转换。 [强制转换表达式](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/type-testing-and-cast#cast-expression)用于调用用户定义显式转换。

operator 和 implicit 或 explicit 关键字分别用于定义隐式转换或显式转换。 定义转换的类型必须是该转换的源类型或目标类型。 可用两种类型中的任何一种类型来定义两种用户定义类型之间的转换。

下面的示例展示如何定义隐式转换和显式转换：

C#复制

using System;

public readonly struct Digit

{

private readonly byte digit;

public Digit(byte digit)

{

if (digit > 9)

{

throw new ArgumentOutOfRangeException(nameof(digit), "Digit cannot be greater than nine.");

}

this.digit = digit;

}

public static implicit operator byte(Digit d) => d.digit;

public static explicit operator Digit(byte b) => new Digit(b);

public override string ToString() => $"{digit}";

}

public static class UserDefinedConversions

{

public static void Main()

{

var d = new Digit(7);

byte number = d;

Console.WriteLine(number); // output: 7

Digit digit = (Digit)number;

Console.WriteLine(digit); // output: 7

}

}

从 C# 11 开始，可以定义 checked 显式转换运算符。 有关详细信息，请参阅[算术运算符](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/arithmetic-operators)一文的[用户定义的 checked 运算符](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/arithmetic-operators#user-defined-checked-operators)部分。

operator 关键字也可用于重载预定义的 C# 运算符。 有关详细信息，请参阅[运算符重载](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/operator-overloading)。

# 强制转换和类型转换（C# 编程指南）

由于 C# 是在编译时静态类型化的，因此变量在声明后就无法再次声明，或无法分配另一种类型的值，除非该类型可以隐式转换为变量的类型。 例如，string 无法隐式转换为 int。 因此，在将 i 声明为 int 后，无法将字符串“Hello”分配给它，如以下代码所示：

C#复制

int i;

// error CS0029: can't implicitly convert type 'string' to 'int'

i = "Hello";

但有时可能需要将值复制到其他类型的变量或方法参数中。 例如，可能需要将一个整数变量传递给参数类型化为 double 的方法。 或者可能需要将类变量分配给接口类型的变量。 这些类型的操作称为类型转换。 在 C# 中，可以执行以下几种类型的转换：

* **隐式转换**：不需要特殊语法，因为转换始终会成功，并且不会丢失任何数据。 示例包括从较小整数类型到较大整数类型的转换以及从派生类到基类的转换。
* **显式转换（强制转换）** ：必须使用[强制转换表达式](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/type-testing-and-cast#cast-expression)，才能执行显式转换。 在转换中可能丢失信息时或在出于其他原因转换可能不成功时，必须进行强制转换。 典型的示例包括从数值到精度较低或范围较小的类型的转换和从基类实例到派生类的转换。
* **用户定义的转换**：用户定义的转换使用你可以定义的特殊方法，以支持在不具有基类和派生类关系的自定义类型之间实现显式和隐式转换。 有关详细信息，请参阅[用户定义转换运算符](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/user-defined-conversion-operators)。
* **使用帮助程序类进行转换**：若要在非兼容类型（如整数和 [System.DateTime](https://learn.microsoft.com/zh-cn/dotnet/api/system.datetime) 对象，或十六进制字符串和字节数组）之间转换，可使用 [System.BitConverter](https://learn.microsoft.com/zh-cn/dotnet/api/system.bitconverter) 类、[System.Convert](https://learn.microsoft.com/zh-cn/dotnet/api/system.convert) 类和内置数值类型的 Parse 方法（如 [Int32.Parse](https://learn.microsoft.com/zh-cn/dotnet/api/system.int32.parse)）。 有关详细信息，请参见[如何将字节数组转换为 int](https://learn.microsoft.com/zh-cn/dotnet/csharp/programming-guide/types/how-to-convert-a-byte-array-to-an-int)、[如何将字符串转换为数字](https://learn.microsoft.com/zh-cn/dotnet/csharp/programming-guide/types/how-to-convert-a-string-to-a-number)和[如何在十六进制字符串与数值类型之间转换](https://learn.microsoft.com/zh-cn/dotnet/csharp/programming-guide/types/how-to-convert-between-hexadecimal-strings-and-numeric-types)。

## 隐式转换

对于内置数值类型，如果要存储的值无需截断或四舍五入即可适应变量，则可以进行隐式转换。 对于整型类型，这意味着源类型的范围是目标类型范围的正确子集。 例如，[long](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/integral-numeric-types) 类型的变量（64 位整数）能够存储 [int](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/integral-numeric-types)（32 位整数）可存储的任何值。 在下面的示例中，编译器先将右侧的 num 值隐式转换为 long 类型，再将它赋给 bigNum。

C#复制

// Implicit conversion. A long can

// hold any value an int can hold, and more!

int num = 2147483647;

long bigNum = num;

有关所有隐式数值转换的完整列表，请参阅[内置数值转换](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/numeric-conversions)一文的[隐式数值转换表](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/numeric-conversions#implicit-numeric-conversions)部分。

对于引用类型，隐式转换始终存在于从一个类转换为该类的任何一个直接或间接的基类或接口的情况。 由于派生类始终包含基类的所有成员，因此不必使用任何特殊语法。

C#复制

Derived d = new Derived();

// Always OK.

Base b = d;

## 显式转换

但是，如果进行转换可能会导致信息丢失，则编译器会要求执行显式转换，显式转换也称为强制转换。 强制转换是显式告知编译器以下信息的一种方式：你打算进行转换且你知道可能会发生数据丢失，或者你知道强制转换有可能在运行时失败。 若要执行强制转换，请在要转换的值或变量前面的括号中指定要强制转换到的类型。 下面的程序将 [double](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types) 强制转换为 [int](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/integral-numeric-types)。如不强制转换，程序将无法编译。

class Test

{

static void Main()

{

double x = 1234.7;

int a;

// Cast double to int.

a = (int)x;

System.Console.WriteLine(a);

}

}

// Output: 1234

有关支持的显式数值转换的完整列表，请参阅[内置数值转换](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/numeric-conversions)一文的[显式数值转换](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/numeric-conversions#explicit-numeric-conversions)部分。

对于引用类型，如果需要从基类型转换为派生类型，则必须进行显式强制转换：

C#复制

// Create a new derived type.

Giraffe g = new Giraffe();

// Implicit conversion to base type is safe.

Animal a = g;

// Explicit conversion is required to cast back

// to derived type. Note: This will compile but will

// throw an exception at run time if the right-side

// object is not in fact a Giraffe.

Giraffe g2 = (Giraffe)a;

引用类型之间的强制转换操作不会更改基础对象的运行时类型；它只更改用作对该对象引用的值的类型。 有关详细信息，请参阅[多态性](https://learn.microsoft.com/zh-cn/dotnet/csharp/fundamentals/object-oriented/polymorphism)。

## 运行时的类型转换异常

在某些引用类型转换中，编译器无法确定强制转换是否会有效。 正确进行编译的强制转换操作有可能在运行时失败。 如下面的示例所示，类型转换在运行时失败将导致引发 [InvalidCastException](https://learn.microsoft.com/zh-cn/dotnet/api/system.invalidcastexception)。

class Animal

{

public void Eat() => System.Console.WriteLine("Eating.");

public override string ToString() => "I am an animal.";

}

class Reptile : Animal { }

class Mammal : Animal { }

class UnSafeCast

{

static void Main()

{

Test(new Mammal());

// Keep the console window open in debug mode.

System.Console.WriteLine("Press any key to exit.");

System.Console.ReadKey();

}

static void Test(Animal a)

{

// System.InvalidCastException at run time

// Unable to cast object of type 'Mammal' to type 'Reptile'

Reptile r = (Reptile)a;

}

}

Test 方法有一个 Animal 形式参数，因此，将实际参数 a 显式强制转换为 Reptile 会造成危险的假设。 更安全的做法是不要做出假设，而是检查类型。 C# 提供 [is](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/type-testing-and-cast#is-operator) 运算符，使你可以在实际执行强制转换之前测试兼容性。 有关详细信息，请参阅[如何使用模式匹配以及 as 和 is 运算符安全地进行强制转换](https://learn.microsoft.com/zh-cn/dotnet/csharp/fundamentals/tutorials/safely-cast-using-pattern-matching-is-and-as-operators)。