**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，或者无法用于存储其他类型的值，除非该类型可以转换为变量的类型。例如，不存在从整数到任意字符串的转换。因此，将 i 声明为整数后，就无法将字符串“Hello”赋予它，如下面的代码中所示。

int i;

i = "Hello"; // Error: "Cannot implicitly convert type 'string' to 'int'"

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，您可能有一个整数变量，您需要将该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赋给接口类型的变量。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：由于该转换是一种安全类型的转换，不会导致数据丢失，因此不需要任何特殊的语法。例如，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都是这样的转换。
* 显式转换（强制转换）：显式转换需要强制转换运算符。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。典型的例子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：可以定义一些特殊的方法来执行用户定义的转换，从而使不具有基类–派生类关系的自定义类型之间可以显式和隐式转换。有关更多信息，请参见 [转换运算符（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/09479473.aspx)。
* 使用帮助程序类的转换：若要在不兼容的类型之间进行转换，例如在整数与 [System.DateTime](https://msdn.microsoft.com/zh-cn/library/system.datetime.aspx) 对象之间转换，或者在十六进制字符串与字节数组之间转换，则可以使用 [System.BitConverter](https://msdn.microsoft.com/zh-cn/library/system.bitconverter.aspx) 类、[System.Convert](https://msdn.microsoft.com/zh-cn/library/system.convert.aspx) 类和内置数值类型的 **Parse** 方法，例如 [Int32.Parse](https://msdn.microsoft.com/zh-cn/library/system.int32.parse.aspx)。有关更多信息，请参见[如何：将字节数组转换为 int（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb384066.aspx)、[如何：将字符串转换为数字（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb397679.aspx)和[如何：在十六进制字符串与数值类型之间转换（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb311038.aspx)。

**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。例如，[long](https://msdn.microsoft.com/zh-cn/library/ctetwysk.aspx) 类型的变量（8 字节整数）能够存储 [int](https://msdn.microsoft.com/zh-cn/library/5kzh1b5w.aspx)（在 32 位计算机上为 4 字节）可存储的任何值。在下面的示例中，编译器先将右侧的值隐式转换为 **long** 类型，再将它赋给 bigNum。

// Implicit conversion. num 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，请参见[隐式数值转换表（C# 参考）](https://msdn.microsoft.com/zh-cn/library/y5b434w4.aspx)。

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。

Derived d = new Derived();

Base b = d; // Always OK.

**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”。强制转换是显式通知编译器您打算进行转换且您知道可能会发生数据丢失的一种方式。若要执行强制转换，请在要转换的值或变量前面的圆括号中指定要强制转换到的类型。下面的程序将 [double](https://msdn.microsoft.com/zh-cn/library/678hzkk9.aspx) 强制转换为 [int](https://msdn.microsoft.com/zh-cn/library/5kzh1b5w.aspx)。如不强制转换则该程序不会进行编译。

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列表，请参见[显式数值转换表（C# 参考）](https://msdn.microsoft.com/zh-cn/library/yht2cx7b.aspx)。

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：

// 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的引用的值的类型。有关更多信息，请参见 [多态性（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173152.aspx)。

**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，类型强制转换在运行时失败将导致引发 [InvalidCastException](https://msdn.microsoft.com/zh-cn/library/system.invalidcastexception.aspx)。

using System;

class Animal

{

public void Eat() { Console.WriteLine("Eating."); }

public override string ToString()

{

return "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)

{

// Cause InvalidCastException at run time

// because Mammal is not convertible to Reptile.

Reptile r = (Reptile)a;

}

}

C# 提供 [is](https://msdn.microsoft.com/zh-cn/library/scekt9xw.aspx) 和 [as](https://msdn.microsoft.com/zh-cn/library/cscsdfbt.aspx) 运算符，使您可以在实际执行强制转换之前测试兼容性。有关更多信息，请参见 [如何：使用 as 和 is 运算符安全地进行强制转换（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/cc488006.aspx)。