**可以为 null 的类型（C# 编程指南）**

可以为 null 的类型是 [System.Nullable<T>](https://msdn.microsoft.com/zh-cn/library/b3h38hb0.aspx) 结构的实例。可以为 null 的类型可以表示其基础值类型正常范围内的值，再加上一个 **null** 值。例如，Nullable<Int32> 读作“可以为 null 的 Int32”，可以将其赋值为 -2147483648 到 2147483647 之间的任意值，也可以将其赋值为 **null** 值。可以赋给 Nullable<bool> 的值包括 [true](https://msdn.microsoft.com/zh-cn/library/eahhcxk2.aspx)、[false](https://msdn.microsoft.com/zh-cn/library/67bxt5ee.aspx) 或 [null](https://msdn.microsoft.com/zh-cn/library/edakx9da.aspx)。在处理数据库和其他包含不可赋值的元素的数据类型时，将 **null** 赋值给数值类型或布尔型的功能特别有用。例如，数据库中的布尔型字段可以存储值 **true** 或 **false**，或者，该字段也可以未定义。

class NullableExample

{

static void Main()

{

int? num = null;

// Is the HasValue property true?

if (num.HasValue)

{

System.Console.WriteLine("num = " + num.Value);

}

else

{

System.Console.WriteLine("num = Null");

}

// y is set to zero

int y = num.GetValueOrDefault();

// num.Value throws an InvalidOperationException if num.HasValue is false

try

{

y = num.Value;

}

catch (System.InvalidOperationException e)

{

System.Console.WriteLine(e.Message);

}

}

}

此示例将显示输出：

num = Null

Nullable object must have a value.

### 可空类型有一个HasValue属性来判断它是否有值

### 可空类型有一个GetValueOrDefault(),可以得到这个类型的默认初始值.如可空int变量的GetValueOrDefault()返回0.

### 在c#中变量有一个Value属性可以获取变量的值

### 可空类型是指值类型而不是引用类型,虽然引用类型也支持null值

有关更多示例，请参见[使用可以为 null 的类型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/2cf62fcy.aspx)。

# 可以为 null 的类型概述

可以为 null 的类型具有以下特性：

* 可以为 null 的类型表示可被赋值为 **null** 值的值类型变量。无法创建基于引用类型的可以为 null 的类型。（引用类型已支持 **null** 值。）
* 语法 **T?** 是 [Nullable<T>](https://msdn.microsoft.com/zh-cn/library/b3h38hb0.aspx) 的简写，此处的 **T** 为值类型。这两种形式可以互换。
* 为可以为 null 的类型赋值的方法与为一般值类型赋值的方法相同，如 int? x = 10; 或 double? d = 4.108。对于可以为 null 的类型，也可向其赋 **null**: int? x = null. 值
* 如果基础类型的值为 **null**，请使用 [Nullable<T>.GetValueOrDefault](https://msdn.microsoft.com/zh-cn/library/72cec0e0.aspx) 方法返回该基础类型所赋的值或默认值，例如 int j = x.GetValueOrDefault();
* 将 [HasValue](https://msdn.microsoft.com/zh-cn/library/sksw8094.aspx) 和 [Value](https://msdn.microsoft.com/zh-cn/library/ydkbatt6.aspx) 只读属性用于测试是否为空和检索值，如下面的示例所示：if(x.HasValue) j = x.Value;
  + 如果此变量包含值，则 **HasValue** 属性返回 **true**；或者如果是 **null** 则返回 **false**。
  + 如果已赋值，则 **Value** 属性返回该值。否则，将引发 [System.InvalidOperationException](https://msdn.microsoft.com/zh-cn/library/system.invalidoperationexception.aspx)。
  + **HasValue** 的默认值为 **false**。 **Value** 属性没有默认值。
  + 还可以将 **==** 和 **!=** 操作数用于可为 null 的类型，如下面的示例所示：if (x != null) y = x;
* 使用 **??** 算符分配默认值，在将当前值为 **null** 的可以为 null 的类型赋值给不可以为 null 的类型时，将应用该默认值，如 int? x = null; int y = x ?? -1;
* 不允许使用嵌套的可以为 null 的类型。将不编译下面一行：Nullable<Nullable<int>> n;

**使用可以为 null 的类型（C# 编程指南）**

可以为 null 的类型可以表示基础类型的所有值，另外还可以表示 [null](https://msdn.microsoft.com/zh-cn/library/edakx9da.aspx) 值。可以为 null 的类型可通过下面两种方式中的一种声明：

System.Nullable<T> variable

- 或 -

T?variable

T 是可以为 null 的类型的基础类型。 T 可以是包括 **struct** 在内的任何值类型；但不能是引用类型。

有关可能使用可以为 null 的类型的示例，请考虑普通的布尔变量如何能够具有两个值：true 和 false。不存在表示“未定义”的值。在很多编程应用中（最突出的是数据库交互），变量可以以未定义的状态出现。例如，数据库中的某个字段可能包含值 true 或 false，但是它也可能根本不包含值。同样，可以将引用类型设置为 **null**，以指示它们未初始化。

这种不一致会导致额外的编程工作，如使用附加变量来存储状态信息、使用特殊值，等等。可以为 null 的类型修饰符使 C# 能够创建表示未定义值的值类型变量。

**可以为 null 的类型示例**

任何值类型都可用作可以为 null 的类型的基础。例如：

int? i = 10;

double? d1 = 3.14;

bool? flag = null;

char? letter = 'a';

int?[] arr = new int?[10];

**可以为 null 的类型的成员**

可以为 null 的类型的每个实例都具有两个公共的只读属性：

* **HasValue**

HasValue 属于 **bool** 类型。当变量包含非 null 值时，它被设置为 **true**。

* **Value**

Value 的类型与基础类型相同。如果 HasValue 为 **true**，则说明 Value 包含有意义的值。如果 HasValue 为 **false**，则访问 Value 将引发 [InvalidOperationException](https://msdn.microsoft.com/zh-cn/library/system.invalidoperationexception.aspx)。

在此示例中，HasValue 成员用于在尝试显示变量之前测试它是否包含值。

int? x = 10;

if (x.HasValue)

{

System.Console.WriteLine(x.Value);

}

else

{

System.Console.WriteLine("Undefined");

}

也可以如下面的示例所示对值进行测试：

int? y = 10;

if (y != null)

{

System.Console.WriteLine(y.Value);

}

else

{

System.Console.WriteLine("Undefined");

}

**显式转换**

可以为 null 的类型可强制转换为常规类型，方法是使用强制转换来显式转换或者通过使用 Value 属性来转换。例如：

int? n = null;

//int m1 = n; // Will not compile.

int m2 = (int)n; // Compiles, but will create an exception if n is null.

int m3 = n.Value; // Compiles, but will create an exception if n is null.

如果两种数据类型之间定义了用户定义的转换，则同一转换也可用于这些数据类型的可以为 null 的版本。

**隐式转换**

可使用 **null** 关键字将可以为 null 的类型的变量设置为 null，如以下示例所示：

int? n1 = null;

从普通类型到可以为 null 的类型的转换是隐式的。

int? n2;

n2 = 10; // Implicit conversion.

**运算符**

可以为 null 的类型还可以使用预定义的一元和二元运算符，以及现有的任何用户定义的值类型运算符。如果操作数为 null，这些运算符将产生一个 null 值；否则运算符将使用包含的值来计算结果。例如：

int? a = 10;

int? b = null;

a++; // Increment by 1, now a is 11.

a = a \* 10; // Multiply by 10, now a is 110.

a = a + b; // Add b, now a is null.

在对可以为 null 的类型执行比较时，如果其中一个可以为 null 的类型的值为 null，但另外一个类型的值不为 null，则除 **!=**（不等于）外，所有比较的结果都将为 **false**。一定不要以为由于一个特定比较的结果为 **false**，相反的情况就会为 **true**。在以下示例中，10 不大于、小于或等于 null。只有 num1 != num2 的计算结果为 **true**。

int? num1 = 10;

int? num2 = null;

if (num1 >= num2)

{

Console.WriteLine("num1 is greater than or equal to num2");

}

else

{

// This clause is selected, but num1 is not less than num2.

Console.WriteLine("num1 >= num2 returned false (but num1 < num2 also is false)");

}

if (num1 < num2)

{

Console.WriteLine("num1 is less than num2");

}

else

{

// The else clause is selected again, but num1 is not greater than

// or equal to num2.

Console.WriteLine("num1 < num2 returned false (but num1 >= num2 also is false)");

}

if (num1 != num2)

{

// This comparison is true, num1 and num2 are not equal.

Console.WriteLine("Finally, num1 != num2 returns true!");

}

// Change the value of num1, so that both num1 and num2 are null.

num1 = null;

if (num1 == num2)

{

// The equality comparison returns true when both operands are null.

Console.WriteLine("num1 == num2 returns true when the value of each is null");

}

/\* Output:

\* num1 >= num2 returned false (but num1 < num2 also is false)

\* num1 < num2 returned false (but num1 >= num2 also is false)

\* Finally, num1 != num2 returns true!

\* num1 == num2 returns true when the value of each is null

\*/

如果两个可以为 null 的类型的值均为 null，则其相等比较的计算结果为 **true**。

**??运算符**

**??** 运算符定义在将可以为 null 的类型分配给非可以为 null 的类型时返回的默认值。

int? c = null;

// d = c, unless c is null, in which case d = -1.

int d = c ?? -1;

此运算符还可用于多个可以为 null 的类型。例如：

int? e = null;

int? f = null;

// g = e or f, unless e and f are both null, in which case g = -1.

int g = e ?? f ?? -1;

**bool?type**

**bool?** 可以为 null 的类型可以包含三个不同的值：[true](https://msdn.microsoft.com/zh-cn/library/eahhcxk2.aspx)、[false](https://msdn.microsoft.com/zh-cn/library/67bxt5ee.aspx) 和 [null](https://msdn.microsoft.com/zh-cn/library/edakx9da.aspx)。有关如何从 bool? 强制转换为 bool 的信息，请参见[如何：安全地将 bool? 强制转换为 bool（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb384091.aspx)。

可以为 null 的布尔值类似于 SQL 中使用的布尔变量类型。若要确保由 **&** 和 产生的结果**|** 运算符与 SQL 中的三值布尔类型一致，提供了以下预定义的运算符：

bool?operator &(bool?x, bool?y)

bool?operator |(bool?x, bool?y)

下表中列出了这些运算符的结果：

| **X** | **y** | **x&y** | **x|y** |
| --- | --- | --- | --- |
| true | true | true | true |
| true | false | false | true |
| true | null | null | true |
| false | true | false | true |
| false | false | false | false |
| false | null | false | null |
| null | true | null | true |
| null | false | false | null |
| null | null | null | null |

**请参阅**

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

[可以为 null 的类型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/1t3y8s4s.aspx)

[装箱可以为 null 的类型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms228597.aspx)

[可以为 Null 的值类型 (Visual Basic)](https://msdn.microsoft.com/zh-cn/library/ms235245.aspx)

# 装箱可以为 null 的类型（C# 编程指南）

基于可以为 null 的类型的对象只在该对象为非空时装箱。如果 [HasValue](https://msdn.microsoft.com/zh-cn/library/sksw8094.aspx) 为 **false**，则将对象引用赋值为 **null**，而不进行装箱。例如：

bool? b = null;

object o = b;

// Now o is null.

如果对象非空，也就是说，如果 [HasValue](https://msdn.microsoft.com/zh-cn/library/sksw8094.aspx) 为 **true**，则会发生装箱过程，但只将可以为 null 的对象所基于的基础类型装箱。如果将非空的可以为 null 的值类型装箱，将使值类型本身（而不是包装该值类型的 [System.Nullable<T>](https://msdn.microsoft.com/zh-cn/library/b3h38hb0.aspx)）装箱。例如：

bool? b = false;

int? i = 44;

object bBoxed = b; // bBoxed contains a boxed bool.

object iBoxed = i; // iBoxed contains a boxed int.

对于那些通过装箱非可以为 null 的类型而创建的类型来说，两种装箱对象是完全相同的。并且，像非可以为 null 的装箱类型一样，可以将它们取消装箱，使其成为可以为 null 的类型，如以下示例所示：

bool? b2 = (bool?)bBoxed;

int? i2 = (int?)iBoxed;

## 备注

可以为 null 的类型在装箱时的行为具有两个优点：

1. 可以测试可以为 null 的对象及其装箱的对应项是否为空：
2. bool? b = null;
3. object boxedB = b;
4. if (b == null)
5. {
6. // True.
7. }
8. if (boxedB == null)
9. {
10. // Also true.
11. }
12. 装箱的可以为 null 的类型完全支持基础类型的功能：
13. double? d = 44.4;
14. object iBoxed = d;
15. // Access IConvertible interface implemented by double.
16. IConvertible ic = (IConvertible)iBoxed;
17. int i = ic.ToInt32(null);

string str = ic.ToString();

# 标识可以为 null 的类型（C# 编程指南）

可以使用 C# [typeof](https://msdn.microsoft.com/zh-cn/library/58918ffs.aspx) 运算符来创建表示可以为 null 的类型的 [Type](https://msdn.microsoft.com/zh-cn/library/system.type.aspx) 对象：

System.Type type = typeof(int?);

还可以使用 [System.Reflection](https://msdn.microsoft.com/zh-cn/library/system.reflection.aspx) 命名空间的类和方法来生成表示可以为 null 的类型的 [Type](https://msdn.microsoft.com/zh-cn/library/system.type.aspx) 对象。但是，如果您尝试使用 [GetType](https://msdn.microsoft.com/zh-cn/library/system.object.gettype.aspx) 方法或 **is** 运算符在运行时获得可以为 null 的类型变量的类型信息，得到的结果是表示基础类型而不是可以为 null 的类型本身的 [Type](https://msdn.microsoft.com/zh-cn/library/system.type.aspx) 对象。

如果对可以为 null 的类型调用 **GetType**，则在该类型被隐式转换为 [Object](https://msdn.microsoft.com/zh-cn/library/system.object.aspx) 时将执行装箱操作。因此，[GetType](https://msdn.microsoft.com/zh-cn/library/system.object.gettype.aspx) 总是返回表示基础类型而不是可以为 null 的类型的 [Type](https://msdn.microsoft.com/zh-cn/library/system.type.aspx) 对象。

int? i = 5;

Type t = i.GetType();

Console.WriteLine(t.FullName); //"System.Int32"

C# 的 [is](https://msdn.microsoft.com/zh-cn/library/scekt9xw.aspx) 运算符还可以作用于可以为 null 的的基础类型。因此，不能使用 **is** 来确定变量是否为可以为 null 的类型。下面的示例演示 **is** 运算符将 Nullable<int> 变量视为 int 变量。

static void Main(string[] args)

{

int? i = 5;

if (i is int) // true

//…

}

使用下面的代码来确定 [Type](https://msdn.microsoft.com/zh-cn/library/system.type.aspx) 对象是否表示可以为 null 的类型。请记住，如果 **Type** 对象是通过调用 [GetType](https://msdn.microsoft.com/zh-cn/library/system.object.gettype.aspx) 返回的，则此代码始终返回 false，如本主题中先前所述。

if (type.IsGenericType && type.GetGenericTypeDefinition() == typeof(Nullable<>)) {…}

# 安全地将 bool? 强制转换为 bool（C# 编程指南）

**bool?** 可以为 null 的类型可以包含三个不同的值：**true**、**false** 和 **null**。因此，**bool?** 类型不能用于条件语句，如 **if**、**for** 或 **while**。例如，以下代码会导致编译器错误。

bool? b = null;

if (b) // Error CS0266.

{

}

这是不允许的，因为 **null** 在条件上下文中的含义并不清楚。若要在条件语句中使用 **bool?**，请首先检查其 [HasValue](https://msdn.microsoft.com/zh-cn/library/sksw8094.aspx) 属性以确保其值不是 **null**，然后将它强制转换为 **bool**。有关更多信息，请参见 [bool](https://msdn.microsoft.com/zh-cn/library/c8f5xwh7.aspx)。如果对使用 **null** 值的 **bool?** 执行强制转换，则在条件测试中将引发 [InvalidOperationException](https://msdn.microsoft.com/zh-cn/library/system.invalidoperationexception.aspx)。下面的示例演示了一种从 **bool?** 安全地强制转换为 **bool** 的方法：

bool? test = null;

// Other code that may or may not

// give a value to test.

if(!test.HasValue) //check for a value

{

// Assume that IsInitialized

// returns either true or false.

test = IsInitialized();

}

if((bool)test) //now this cast is safe

{

// Do something.

}