[C#可空类型](http://www.cnblogs.com/wilber2013/p/4294492.html)

在程序开发中，有时候需要值类型也为可空类型，比如，在数据库中，我们可以把一个日期Datetime设置为null。

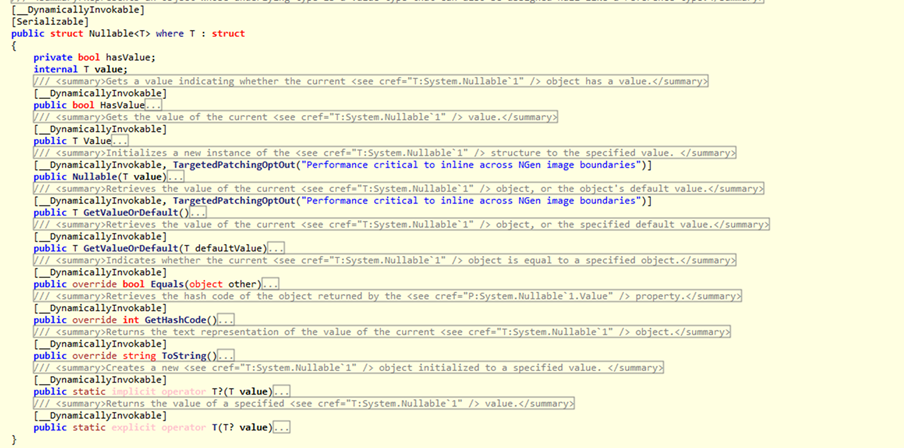
在C# 2.0中就出现了可空类型，允许值类型也可以为空（null），可空类型的实现基于C#泛型。

**可空类型基本知识**

可空类型的核心是System.Nullable<T>，同时静态类System.Nullable为可空类型提供了很多实用的方法。下面分别看看可空类型的这两个重要组成部分。

**System.Nullable<T>**

通过ILSpy我们可以查看这个类型的C#代码：



从上面的图中可以看到关于System.Nullable<T>的一些关键点：

1. Nullable<T>是一个泛型类型
2. 类型参数T有一个值类型的约束（根据值类型约束T : struct，T不能为可空类型，也就是说Nullable<Nullable<int>>是不允许的）
3. Nullable<T>是一个值类型（是一个struct）

对于任何具体的可空类型来说，T的类型为可空类型的**基础类型（underlying type）**，例如Nullable<int>的基础类型就是int。

通过上面代码还可以看到，Nullable<T>有两个重要的属性，HasValue和Value。通过它们可以了解可空类型是怎么工作的：

1. 如果一个可空值类型存在一个真正的值，那么Value就代表这个值本身，同时HasValue值为true
2. 如果一个可空值类型为空，那么HasValue为false，Value这是没有意义。

下面看一个可空类型的简单例子，进一步了解一下可空类型：

[复制代码](javascript:void(0);)

static void Display(Nullable<int> x)

{

Console.WriteLine("HasValue: {0}", x.HasValue);

if (x.HasValue)

{

Console.WriteLine("Value: {0}", x.Value);

Console.WriteLine("Explicit conversion: {0}", (int)x);

}

Console.WriteLine("GetValueOrDefault(): {0}", x.GetValueOrDefault());

Console.WriteLine("GetValueOrDefault(10): {0}", x.GetValueOrDefault(10));

Console.WriteLine("ToString(): {0}", x.ToString());

Console.WriteLine("GetHashCode(): {0}", x.GetHashCode());

Console.WriteLine();

}

static void Main(string[] args)

{

Nullable<int> x = 5;

Display(x);

x = new Nullable<int>(9);

Display(x);

x = new Nullable<int>();

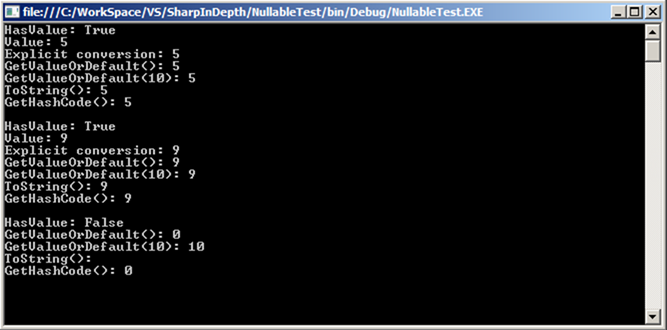
Display(x);

Console.Read();

}

[复制代码](javascript:void(0);)

程序的输出为：



通过这段代码可以看到HasValue和Value的使用，以及Nullable<T>中一些常用的方法。

注意，在这段代码中，下面两句的IL代码是一样的：

C#代码

Nullable<int> x = 5;

x = new Nullable<int>(9);

IL代码

IL\_0004: call instance void valuetype [mscorlib]System.Nullable`1<int32>::.ctor(!0)

IL\_0015: call instance void valuetype [mscorlib]System.Nullable`1<int32>::.ctor(!0)

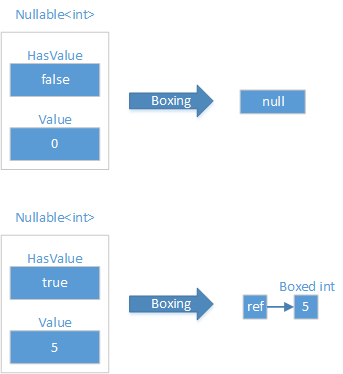
这里涉及了**包装（wrapping）**和**拆包（unwrapping）**的概念：将T的一个实例转换成Nullable<T>的一个实例的过程在C#中成为包装，相反的过程成为拆包。这个概念跟装箱和拆箱不一样，后面会看到Nullable<T>的装箱和拆箱。

**Nullable<T>的装箱和拆箱**

从前面的分析可以看到Nullable<T>是一个结构，也就是一个值类型。也就是说，当我们把可空类型转换成一个引用类型的时候需要进行装箱操作。

对于Nullable<T>的装箱和拆箱可以概括为：

* Nullable<T>的实例要么装箱为空引用，要么装箱成T的一个以装箱的值



* 已装箱的值可以拆箱成普通类型，或者拆箱为对于的可空类型
  + 拆箱一个空引用时，如果拆箱为普通类型，会抛出一个NullReferenceException的异常
  + 如果拆箱成恰当的可控类型，就会拆箱成一个没有值的Nullable<T>实例

看一个关于可空类型装箱和拆箱的例子：

[复制代码](javascript:void(0);)

static void Main(string[] args)

{

Nullable<int> x = 5;

//有值的可空类型装箱

object boxed = x;

Console.WriteLine(x.GetType());

//拆箱为普通类型

int normal = (int)boxed;

Console.WriteLine(normal);

//拆箱为可空类型

x = (Nullable<int>)boxed;

Console.WriteLine(x);

x = new Nullable<int>();

//空的可空类型装箱

boxed = x;

Console.WriteLine(boxed == null);

//拆箱为可空类型

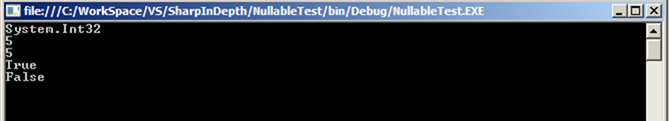
x = (Nullable<int>)boxed;

Console.WriteLine(x.HasValue);

}

[复制代码](javascript:void(0);)

输出：



**System.Nullable**

System.Nullable是一个静态类，只包含三个静态方法，大家可以通过ILSpy进行查看，这里就不上图了。

下面两个方法是比较方法：

public static int Compare<T>(T? n1, T? n2) where T : struct

public static bool Equals<T>(T? n1, T? n2) where T : struct

下面这个方法用来获得可空类型的基础类型：

public static Type GetUnderlyingType(Type nullableType)

**可空类型语法糖**

在C# 2.0中，我们可以使用?修饰符来表示可空类型。

下面的C#语句具有相同的IL代码。

Nullable<int> x = 5;

int? y = 5;

IL\_0004: call instance void valuetype [mscorlib]System.Nullable`1<int32>::.ctor(!0)

IL\_000d: call instance void valuetype [mscorlib]System.Nullable`1<int32>::.ctor(!0)

**使用null进行赋值和比较**

C#编译器允许使用null在比较和赋值中表示一个可空类型的空值。

对于下面的代码，通过IL可以发现"x == null"实际调用的是HasValue属性进行比较。

int? x = null;

Console.WriteLine(x == null);

IL\_000b: call instance bool valuetype [mscorlib]System.Nullable`1<int32>::get\_HasValue()

**总结**

C# 2.0中出现的可空类型解决了我们很多的问题，可空类型的相关知识还是比较容易理解的。

在使用中，我们可以直接使用?修饰符来创建可空值类型。

* [**可空类型基本知识**](http://www.cnblogs.com/wilber2013/p/4294492.html#_nav_0)
* [**System.Nullable<T>**](http://www.cnblogs.com/wilber2013/p/4294492.html#_nav_1)
* [**Nullable<T>的装箱和拆箱**](http://www.cnblogs.com/wilber2013/p/4294492.html#_nav_2)
* [**System.Nullable**](http://www.cnblogs.com/wilber2013/p/4294492.html#_nav_3)
* [**可空类型语法糖**](http://www.cnblogs.com/wilber2013/p/4294492.html#_nav_4)
* [**使用null进行赋值和比较**](http://www.cnblogs.com/wilber2013/p/4294492.html#_nav_5)
* [**总结**](http://www.cnblogs.com/wilber2013/p/4294492.html#_nav_6)

作者：[田小计划](http://home.cnblogs.com/u/wilber2013/)

出处：<http://www.cnblogs.com/wilber2013/>

本文版权归作者和博客园共有，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出原文连接，否则保留追究法律责任的权利。

如果觉得不错，请点击**推荐**和**关注**！