**装箱和拆箱是值类型和引用类型之间相互转换是要执行的操作。**

1. 装箱在值类型向引用类型转换时发生

2. 拆箱在引用类型向值类型转换时发生

光上述两句话不难理解，但是往深处了解，就需要一些篇幅来解释了。

我们先看装箱时都会发生什么事情，下面是一行最简单的装箱代码

object obj = 1;

这行语句将整型常量1赋给object类型的变量obj； 众所周知常量1是值类型，值类型是要放在栈上的，而object是引用类型，它需要放在堆上；要把值类型放在堆上就需要执行一次装箱操作。

这行语句的IL代码如下，请注意注释部分说明：

1. .locals init (
2. [0] object objValue
3. ) //以上三行IL表示声明object类型的名称为objValue的局部变量
4. IL\_0000: nop
5. IL\_0001: ldc.i4.s 9 //表示将整型数9放到栈顶
6. IL\_0003: box [mscorlib]System.Int32 //执行IL box指令，在内存堆中申请System.Int32类型需要的堆空间
7. IL\_0008: stloc.0 //弹出堆栈上的变量，将它存储到索引为0的局部变量中

以上就是装箱所要执行的操作了，执行装箱操作时不可避免的要在堆上申请内存空间，并将堆栈上的值类型数据复制到申请的堆内存空间上，这肯定是要消耗内存和cpu资源的。我们再看下拆箱操作是怎么回事：

请看下面的C#代码：

上面的两行代码会执行一次装箱操作将整形数字常量4装箱成引用类型object变量objValue；然后又执行一次拆箱操作，将存储到堆上的引用变量objValue存储到局部整形值类型变量value中。

1. object objValue = 4;
2. int value = (int)objValue;

同样我们需要看下IL代码：

拆箱操作的执行过程和装箱操作过程正好相反，是将存储在堆上的引用类型值转换为值类型并给值类型变量。

1. .locals init (
2. [0] object objValue,
3. [1] int32 'value'
4. ) //上面IL声明两个局部变量object类型的objValue和int32类型的value变量
5. IL\_0000: nop
6. IL\_0001: ldc.i4.4 //将整型数字4压入栈
7. IL\_0002: box [mscorlib]System.Int32 //执行IL box指令，在内存堆中申请System.Int32类型需要的堆空间
8. IL\_0007: stloc.0 //弹出堆栈上的变量，将它存储到索引为0的局部变量中
9. IL\_0008: ldloc.0//将索引为0的局部变量（即objValue变量）压入栈
10. IL\_0009: unbox.any [mscorlib]System.Int32 //执行IL 拆箱指令unbox.any 将引用类型object转换成System.Int32类型
11. IL\_000e: stloc.1 //将栈上的数据存储到索引为1的局部变量即value

**装箱操作和拆箱操作是要额外耗费cpu和内存资源的，所以在c# 2.0之后引入了泛型来减少装箱操作和拆箱操作消耗。**