[**C# 装箱和拆箱[整理]**](http://www.cnblogs.com/huashanlin/archive/2007/05/16/749359.html)

1、  
      装箱和拆箱是一个抽象的概念   
2、  
      装箱是将值类型转换为引用类型 ；拆箱是将引用类型转换为值类型   
      利用装箱和拆箱功能，可通过允许值类型的任何值与Object 类型的值相互转换，将值类型与引用类型链接起来   
例如：   
int val = 100;   
object obj = val;   
Console.WriteLine (“对象的值 = {0}", obj);   
这是一个装箱的过程，是将值类型转换为引用类型的过程   
  
int val = 100;   
object obj = val;   
int num = (int) obj;   
Console.WriteLine ("num: {0}", num);   
这是一个拆箱的过程，是将值类型转换为引用类型，再由引用类型转换为值类型的过程   
  
注：被装过箱的对象才能被拆箱  
3、  
      .NET中，数据类型划分为值类型和引用(不等同于C++的指针)类型，与此对应，内存分配被分成了两种方式，一为栈，二为堆，注意：是托管堆。  
      值类型只会在栈中分配。  
      引用类型分配内存与托管堆。  
      托管堆对应于垃圾回收。

**4：装箱/拆箱是什么？**装箱：用于在垃圾回收堆中存储值类型。装箱是值类型到 object 类型或到此值类型所实现的任何接口类型的隐式转换。   
拆箱：从 object 类型到值类型或从接口类型到实现该接口的值类型的显式转换。

**5：为何需要装箱？(为何要将值类型转为引用类型？)**一种最普通的场景是，调用一个含类型为Object的参数的方法，该Object可支持任意为型，以便通用。当你需要将一个值类型(如Int32)传入时，需要装箱。   
另一种用法是，一个非泛型的容器，同样是为了保证通用，而将元素类型定义为Object。于是，要将值类型数据加入容器时，需要装箱。

**6：装箱/拆箱的内部操作。**装箱：   
对值类型在堆中分配一个对象实例，并将该值复制到新的对象中。按三步进行。   
第一步：新分配托管堆内存(大小为值类型实例大小加上一个方法表指针和一个SyncBlockIndex)。   
第二步：将值类型的实例字段拷贝到新分配的内存中。   
第三步：返回托管堆中新分配对象的地址。这个地址就是一个指向对象的引用了。   
有人这样理解：如果将Int32装箱，返回的地址，指向的就是一个Int32。我认为也不是不能这样理解，但这确实又有问题，一来它不全面，二来指向Int32并没说出它的实质(在托管堆中)。   
拆箱：  
检查对象实例，确保它是给定值类型的一个装箱值。将该值从实例复制到值类型变量中。   
有书上讲，拆箱只是获取引用对象中指向值类型部分的指针，而内容拷贝则是赋值语句之触发。我觉得这并不要紧。最关键的是检查对象实例的本质，拆箱和装箱的类型必需匹配，这一点上，在IL层上，看不出原理何在，我的猜测，或许是调用了类似GetType之类的方法来取出类型进行匹配(因为需要严格匹配)。

**7：装箱/拆箱对执行效率的影响**显然，从原理上可以看出，装箱时，生成的是全新的引用对象，这会有时间损耗，也就是造成效率降低。   
那该如何做呢？   
首先，应该尽量避免装箱。   
比如上例2的两种情况，都可以避免，在第一种情况下，可以通过重载函数来避免。第二种情况，则可以通过泛型来避免。   
当然，凡事并不能绝对，假设你想改造的代码为第三方程序集，你无法更改，那你只能是装箱了。   
对于装箱/拆箱代码的优化，由于C#中对装箱和拆箱都是隐式的，所以，根本的方法是对代码进行分析，而分析最直接的方式是了解原理结何查看反编译的IL代码。比如：在循环体中可能存在多余的装箱，你可以简单采用提前装箱方式进行优化。

**8：对装箱/拆箱更进一步的了解**装箱/拆箱并不如上面所讲那么简单明了，比如：装箱时，变为引用对象，会多出一个方法表指针，这会有何用处呢？   
我们可以通过示例来进一步探讨。   
举个例子。   
Struct A : ICloneable   
{   
public Int32 x;   
public override String ToString() {   
return String.Format(”{0}”,x);   
}   
public object Clone() {   
return MemberwiseClone();   
}   
}   
static void main()   
{   
A a;   
a.x = 100;   
Console.WriteLine(a.ToString());   
Console.WriteLine(a.GetType());   
A a2 = (A)a.Clone();   
ICloneable c = a2;   
Ojbect o = c.Clone();   
}   
5.0：a.ToString()。编译器发现A重写了ToString方法，会直接调用ToString的指令。因为A是值类型，编译器不会出现多态行为。因此，直接调用，不装箱。(注：ToString是A的基类System.ValueType的方法)   
5.1：a.GetType()，GetType是继承于System.ValueType的方法，要调用它，需要一个方法表指针，于是a将被装箱，从而生成方法表指针，调用基类的System.ValueType。(补一句，所有的值类型都是继承于System.ValueType的)。   
5.2：a.Clone()，因为A实现了Clone方法，所以无需装箱。   
5.3：ICloneable转型：当a2为转为接口类型时，必须装箱，因为接口是一种引用类型。   
5.4：c.Clone()。无需装箱，在托管堆中对上一步已装箱的对象进行调用。   
附：其实上面的基于一个根本的原理，因为未装箱的值类型没有方法表指针，所以，不能通过值类型来调用其上继承的虚方法。另外，接口类型是一个引用类型。对此，我的理解，该方法表指针类似C++的虚函数表指针，它是用来实现引用对象的多态机制的重要依据。

**9：如何更改已装箱的对象**对于已装箱的对象，因为无法直接调用其指定方法，所以必须先拆箱，再调用方法，但再次拆箱，会生成新的栈实例，而无法修改装箱对象。有点晕吧，感觉在说绕口令。还是举个例子来说：(在上例中追加change方法)   
public void Change(Int32 x) {   
this.x = x;   
}   
调用：   
A a = new A();   
a.x = 100;   
Object o = a; //装箱成o，下面，想改变o的值。   
((A)o).Change(200); //改掉了吗？没改掉。   
没改掉的原因是o在拆箱时，生成的是临时的栈实例A，所以，改动是基于临时A的，并未改到装箱对象。   
(附：在托管C++中，允许直接取加拆箱时第一步得到的实例引用，而直接更改，但C#不行。)   
那该如何是好？   
嗯，通过接口方式，可以达到相同的效果。   
实现如下：   
interface IChange {   
void Change(Int32 x);   
}   
struct A : IChange {   
…   
}   
调用：   
((IChange)o).Change(200);//改掉了吗？改掉了。   
为啥现在可以改？   
在将o转型为IChange时，这里不会进行再次装箱，当然更不会拆箱，因为o已经是引用类型，再因为它是IChange类型，所以可以直接调用Change，于是，更改的也就是已装箱对象中的字段了，达到期望的效果。  
  
**10、**--------------------------  
      将值类型转换为引用类型，需要进行装箱操作(boxing)：

1、首先从托管堆中为新生成的引用对象分配内存。

2、然后将值类型的数据拷贝到刚刚分配的内存中。

3、返回托管堆中新分配对象的地址。

可以看出，进行一次装箱要进行分配内存和拷贝数据这两项比较影响性能的操作。

将引用内型转换为值内型，需要进行拆箱操作(unboxing)：

1、首先获取托管堆中属于值类型那部分字段的地址，这一步是严格意义上的拆箱。

2、将引用对象中的值拷贝到位于线程堆栈上的值类型实例中。

经过这2步，可以认为是同boxing是互反操作。严格意义上的拆箱，并不影响性能，但伴随这之后的拷贝数据的操作就会同boxing操作中一样影响性能。  
  
**11、**-------------------------  
NET的所有类型都是由基类System.Object继承过来的，包括最常用的基础类型：int, byte, short，bool等等，就是说所有的事物都是对象。如果申明这些类型得时候都在堆(HEAP)中分配内存，会造成极低的效率！(个中原因以及关于堆和栈得区别会在另一篇里单独得说说！)  
.NET如何解决这个问题得了？正是通过将类型分成值型(value)和引用型(regerencetype)，C#中定义的值类型包括原类型（Sbyte、Byte、Short、Ushort、Int、Uint、Long、Ulong、Char、Float、Double、Bool、Decimal）、枚举(enum)、结构(struct)，引用类型包括：类、数组、接口、委托、字符串等。  
值型就是在栈中分配内存，在申明的同时就初始化，以确保数据不为NULL；  
引用型是在堆中分配内存，初始化为null，引用型是需要GARBAGE COLLECTION来回收内存的，值型不用，超出了作用范围，系统就会自动释放！  
下面就来说装箱和拆箱的定义！  
装箱就是隐式的将一个值型转换为引用型对象。比如：  
int i=0;  
Syste.Object obj=i;  
这个过程就是装箱！就是将i装箱！  
拆箱就是将一个引用型对象转换成任意值型！比如：  
int i=0;  
System.Object obj=i;  
int j=(int)obj;  
这个过程前2句是将i装箱，后一句是将obj拆箱！

分类: [**编程理论概念**](http://www.cnblogs.com/huashanlin/category/72507.html)