.NET 面试题解析(02)-拆箱与装箱

系列文章目录地址：

.NET 面试题解析(00)-开篇来谈谈面试 & 系列文章索引

装箱和拆箱几乎是所有面试题中必考之一,看上去简单,就往往容易被忽视。其实它一点都不简单的，一个简单的问题也可以从多个层次来解读。

常见面试题目：

1.什么是拆箱和装箱？

2.什么是箱子？

3.箱子放在哪里？

4.装箱和拆箱有什么性能影响？

5.如何避免隐身装箱？

6.箱子的基本结构？

7.装箱的过程？

8.拆箱的过程？

9.下面这段代码输出什么？共发生多少次装箱？多少次拆箱？

```
int i = 5;
object obj = i;
IFormattable ftt = i;
Console.WriteLine(System.Object.ReferenceEquals(i, obj));
Console.WriteLine(System.Object.ReferenceEquals(i, ftt));
Console.WriteLine(System.Object.ReferenceEquals(ftt, obj));
Console.WriteLine(System.Object.ReferenceEquals(i, (int)obj));
Console.WriteLine(System.Object.ReferenceEquals(i, (int)ftt));
```

深入浅出装箱与拆箱

有拆必有装，有装必有拆。

在上一文中我们提到，所有值类型都是继承自 System.ValueType，而 System.ValueType 又是来自何方呢,不难发现 System.ValueType 继承自 System.Object。因此 **Object 是.NET 中的万物之源**，几乎所有类型都来自她，这是装箱与拆箱的基础。

特别注意的是，本文与上一文有直接关联，需要先了解上一文中值类型与引用类型的原理，才可以更好理解本文的内容。

基本概念

拆箱与装箱就是值类型与引用类型的转换，她是值类型和引用类型之间的桥梁，他们可以相互转换的一个基本前提就是上面所说的：Object 是.NET 中的万物之源

先看看一个小小的实例代码：

```
            int x = 1023;
            object o = x; //装箱
            int y = (int) o; //拆箱
```

装箱：值类型转换为引用对象，一般是转换为 System.Object 类型或值类型实现的接口引用类型；

拆箱：引用类型转换为值类型，注意，这里的引用类型只能是被装箱的引用类型对象；

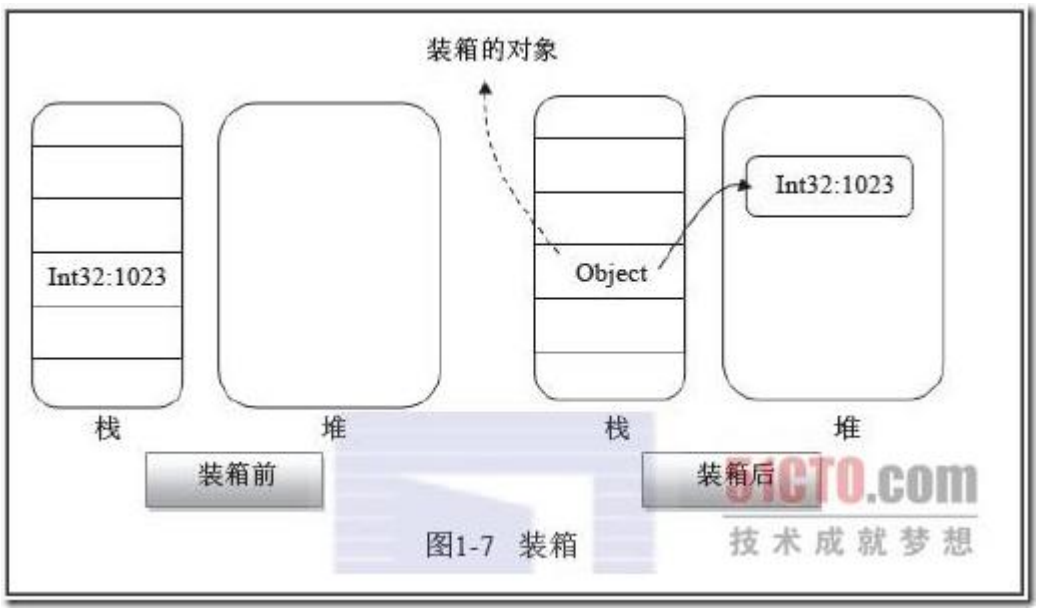
由于值类型和引用类型在内存分配的不同，从内存执行角度看，拆箱与装箱就势必存在内存的分配与数据的拷贝等操作，这也是装箱与拆箱性能影响的根源。

装箱的过程

```
int x = 1023;
object o = x; //装箱
```

装箱就是把值类型转换为引用类型，具体过程：

- 1.在堆中申请内存，内存大小为值类型的大小，再加上额外固定空间（引用类型的标配：TypeHandle 和同步索引块）；
- 2.将值类型的字段值（x=1023）拷贝新分配的内存中；
- 3.返回新引用对象的地址（给引用变量 object o）



如上图所示，装箱后内存有两个对象：一个是值类型变量 x，另一个就是新引用对象 o。装箱对应的 IL 指令为 box，上面装箱的 IL 代码如下图：

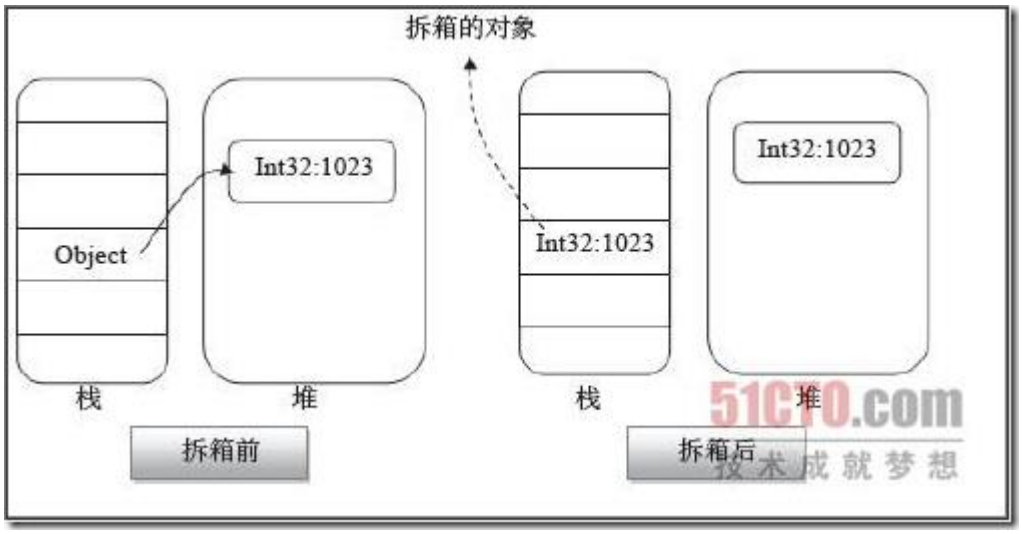
```
IL_0001:  ldc.i4      0x3ff  int a=1023
IL_0006:  stloc.0
IL_0007:  ldloc.0
IL_0008:  box         [mscorlib]System.Int32 object o=a
IL_000d:  stloc.1
```

拆箱的过程

```
int x = 1023;
object o = x; //装箱
int y = (int) o; //拆箱
```

明白了装箱，拆箱就是装箱相反的过程，简单的说是把装箱后的引用类型转换为值类型。具体过程：

- 1.检查实例对象（object o）是否有效，如是否为 null，其装箱的类型与拆箱的类型（int）是否一致，如检测不合法，抛出异常；
- 2.指针返回，就是获取装箱对象（object o）中值类型字段值的地址；
- 3.字段拷贝，把装箱对象（object o）中值类型字段值拷贝到栈上，意思就是创建一个新的值类型变量来存储拆箱后的值；



如上图所示，拆箱后，得到一个新的值类型变量 y，拆箱对应的 IL 指令为 unbox，拆箱的 IL 代码如下：

```
IL_000e:  ldloc.1
IL_000f:  unbox.any  [mscorlib]System.Int32   int y = (int) o; //拆箱
IL_0014:  stloc.2
```

装箱与拆箱总结及性能

装的的什么？拆的又是什么？什么是箱子？

通过上面深入了解了装箱与拆箱的原理，不难理解，只有值类型可以装箱，拆的就是装箱后的引用对象，箱子就是一个存放了值类型字段的引用对象实例，箱子存储在托管堆上。**只有值类型才有装箱、拆箱两个状态，而引用类型一直都在箱子里**。

关于性能

之所以关注装箱与拆箱，主要原因就是他们的性能问题，而且在日常编码中，经常有装箱与拆箱的操作，而且这些装箱与拆箱的操作往往是在不经意时发生。**一般来说，装箱的性能开销更大**，这不难理解，因为引用对象的分配更加复杂，成本也更高，值类型分配在栈上，分配和释放的效率都很高。装箱过程是需要创建一个新的引用类型对象实例，拆箱过程需要创建一个值类型字段，开销更低。

为了尽量避免这种性能损失，尽量使用泛型，在代码编写中也尽量避免隐式装箱。

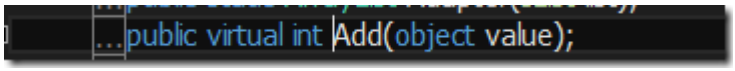
什么是隐式装箱？如何避免？

就是不经意的代码导致多次重复的装箱操作，看看代码就好理解了

```
int x = 100;
ArrayList arr = new ArrayList(3);
arr.Add(x);
```

```
arr.Add(x);
arr.Add(x);
```

这段代码共有多少次装箱呢？看看 Add 方法的定义：



再看看 IL 代码，可以准确的得到装箱的次数：

```
IL_0005:  newobj     instance void [mscorlib]System.Collections.ArrayList::.ctor(int
IL_000a:  stloc.1                                  创建ArrayList
IL_000b:  ldloc.1
IL_000c:  ldloc.0
IL_000d:  box        [mscorlib]System.Int32   第一次装箱
IL_0012:  callvirt   instance int32 [mscorlib]System.Collections.ArrayList::Add(obje
IL_0017:  pop
IL_0018:  ldloc.1
IL_0019:  ldloc.0
IL_001a:  box        [mscorlib]System.Int32    第二次装箱
IL_001f:  callvirt   instance int32 [mscorlib]System.Collections.ArrayList::Add(obje
IL_0024:  pop
IL_0025:  ldloc.1
IL_0026:  ldloc.0
IL_0027:  box        [mscorlib]System.Int32   第三次装箱
IL_002c:  callvirt   instance int32 [mscorlib]System.Collections.ArrayList::Add(obje
```

显示装箱可以避免隐式装箱，下面修改后的代码就只有一次装箱了。

```
int x = 100;
ArrayList arr = new ArrayList(3);
object o = x;
arr.Add(o);
arr.Add(o);
arr.Add(o);
```

题目答案解析:

1.什么是拆箱和装箱？

装箱就是值类型转换为引用类型，拆箱就是引用类型（被装箱的对象）转换为值类型。

2.什么是箱子？

就是引用类型对象。

3.箱子放在哪里？

托管堆上。

4.装箱和拆箱有什么性能影响？

装箱和拆箱都涉及到内存的分配和对象的创建，有较大的性能影响。

5.如何避免隐身装箱？

编码中，多使用泛型、显示装箱。

6.箱子的基本结构？

上面说了，箱子就是一个引用类型对象，因此她的结构，主要包含两部分：

- 值类型字段值；
- 引用类型的标准配置，引用对象的额外空间：TypeHandle 和同步索引块，关于这两个概念在本系列后面的文章会深入探讨。

7.装箱的过程？

- 1.在堆中申请内存，内存大小为值类型的大小，再加上额外固定空间（引用类型的标配：TypeHandle 和同步索引块）；
- 2.将值类型的字段值（x=1023）拷贝新分配的内存中；
- 3.返回新引用对象的地址（给引用变量 object o）

8.拆箱的过程？

- 1.检查实例对象（object o）是否有效，如是否为 null，其装箱的类型与拆箱的类型（int）是否一致，如检测不合法，抛出异常；
- 2.指针返回，就是获取装箱对象（object o）中值类型字段值的地址；
- 3.字段拷贝，把装箱对象（object o）中值类型字段值拷贝到栈上，意思就是创建一个新的值类型变量来存储拆箱后的值；

9.下面这段代码输出什么？共发生多少次装箱？多少次拆箱？

```
int i = 5;
object obj = i;
IFormattable ftt = i;
Console.WriteLine(System.Object.ReferenceEquals(i, obj));
Console.WriteLine(System.Object.ReferenceEquals(i, ftt));
Console.WriteLine(System.Object.ReferenceEquals(ftt, obj));
Console.WriteLine(System.Object.ReferenceEquals(i, (int)obj));
Console.WriteLine(System.Object.ReferenceEquals(i, (int)ftt));
```

上面代码输出如下，至于发生多少次装箱多少次拆箱，你猜？

```
False
False
False
False
False
```


.NET 面试题解析(00)-开篇来谈谈面试 & 系列文章索引

参考资料:

书籍：CLR via C#

书籍：你必须知道的.NET

1.4.2 装箱和拆箱：http://book.51cto.com/art/201012/237726.htm

分类: C#.NET

/*梦里花落知多少*/

关注 - 5

粉丝 - 136

+加关注

« 上一篇：.NET 面试题解析(01)-值类型与引用类型

» 下一篇：.NET 面试题解析(03)-string 与字符串操作

posted @ 2016-03-03 09:17 /*梦里花落知多少*/ 阅读(843) 评论(11) 编辑 收藏

评论列表

#1 楼 2016-03-03 10:02 cwwhy

不知道类的值类型属性，会不会有装箱拆箱的操作？值类型属性本身就是存储在堆上的，在转成 object 的时候会装箱吗？

支持(0)反对(0)

#2 楼 2016-03-03 10:54 lswtianliang

小伙子加油

支持(0)反对(0)

#3 楼[楼主] 2016-03-03 10:58 /*梦里花落知多少*/

@cwwhy

class 本来就在箱子里（堆上），肯定不会装箱了

支持(0)反对(0)

#4 楼 2016-03-03 11:21 cwwhy

就是说不是所有的值类型转成引用类型都会装箱？

支持(0)反对(0)

#5 楼 2016-03-03 11:28 大橙子小橘子

默默的右边点赞

支持(0)反对(0)

#6 楼 2016-03-03 14:18 jojoka

博主，您关于避免装箱的例子上仅仅是使用‘一个值’的的转换避免 3 次装箱。若需要 3 个值的情况下您的例子仍然需要 3 次装箱也就是没有避免隐形装箱的问题？相反在定义 ArrayList 的时候您有一个操作，即定义数据边界长度。这一步不也是在浪费动态数组的灵活性来牺牲效率吗？

支持(0)反对(0)

#7 楼 2016-03-03 17:43 花生与半仙

博主，你说的是拆箱只能是被装箱的引用类型对象，也就是不是经过装箱的引用类型对象不存在拆箱这个行为？

支持(0)反对(0)

#8 楼[楼主] 2016-03-03 17:55 /*梦里花落知多少*/

@jojoka

你观察的好仔细，

只是这个地方举得例子主要目的是说明“隐式装修”的问题，平时编码不注意的话会经常存在这种隐式装箱的问题。

支持(0)反对(0)

#9 楼[楼主] 2016-03-03 17:56 /*梦里花落知多少*/

@花生与半仙

对

支持(0)反对(0)

#10 楼 2016-03-03 21:03 花生与半仙

还有一个小问题，博主。类（class）是引用类型的，那么实例化以后这个实例化对象是在栈还是在托管堆上呢？是不是引用在栈上，但是实例话对象是放在托管堆上的呢？

支持(0)反对(0)

#11 楼[楼主] 2016-03-03 22:00 /*梦里花落知多少*/

@花生与半仙

对