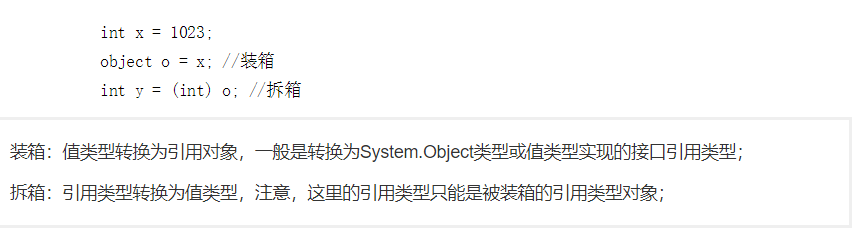
类型转换：  
类型转换包括显示转换和隐式转换，其类型转换的基本规则如下：

* 在任何类型都可以安全的转换为其基类类型，可以由隐式转换来完成。
* 任何类型转换为其派生类型时，必须进行显示转换，转换的规则是：（类型名）对象名。
* 使用GetType可以取得任何对象的精确类型
* 基本类型可以使用Covert类实现类型转换
* 除了string，其他类型都有parse方法，用于将字符串类型转换为对应的基本类型
* 值类型和引用类型的转换机制称为装箱（boxing）和拆箱（unboxing）（拆箱和装箱就是值类型和引用类型的转换，它是值类型和引用类型之间的桥梁）



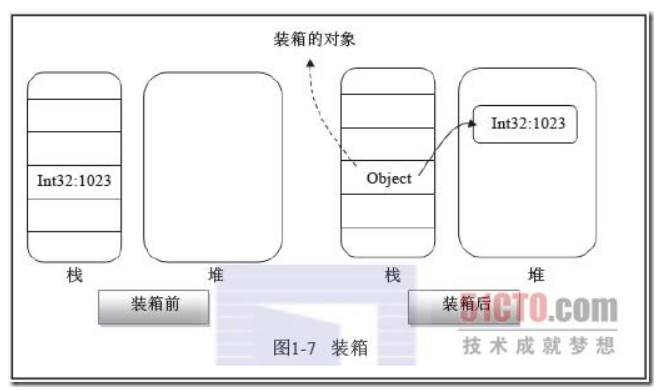
装箱的过程：

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，在加上额外固定空间（引用类型的标配：TypeHandler和同步索引快）
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。

拆箱：

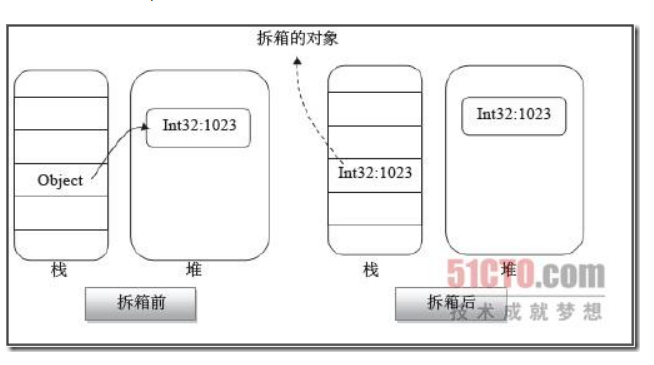
Int x = 1023;

Object o = x; // 装箱

Int y = (int)o; // 拆箱

拆箱就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，如果检测不合法，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。



using System;

namespace codeTest

{

class Program

{

static void Main(string[] args)

{

//任何类型都可以安全的转换为其基类类型，可以由隐式转换来完成；

int a = 1;

object aa = a;

//任何类型转换为其派生类型时，必须进行显示转换，转换的规则是：（类型名）对象名；

object b = 2;

int bb = (int)b;

//使用GetType可以取得任何对象的精确类型；

bb.GetType();

//基本类型可以使用Convert类实现类型转换；

int cc = Convert.ToInt32("44");

double dd = Convert.ToDouble("44");

//除了string以外的其他类型都有Parse方法，用于将字符串类型转换为对应的基本类型；

decimal ee = decimal.Parse("77");

//值类型和引用类型的转换机制称为装箱（boxing）和拆箱（unboxing）

object f = 60;

int ff = (int)f;

// is 和 as 关键字

//检查对象类型的兼容性，并返回结果，true或者false；

//不会抛出异常；

//如果对象为null，则返回值永远为false。

object \_A = new B();

if (\_A is A)

{

Console.WriteLine("is\_B类型兼容A类型");

B \_b = (B)\_A;

}

//检查对象类型的兼容性，并返回结果，如果不兼容就返回null；

//不会抛出异常；

//如果结果判断为空，则强制执行类型转换将抛出NullReferenceException异常。

if (\_A as A !=null)

{

Console.WriteLine("as\_B类型兼容A类型");

B \_bb = (B)\_A;

}

Console.ReadLine();

}

class A

{

}

class B : A

{

}

}

}

在程序中，进行类型转换是常见的事情，C#支持基本的强制类型转换方法，如：

Object obj1 = new NewType();

NewType newValue = (NewType)obj1;

这样强制转换的时候，这个过程是不安全的，因此需要try-catch语句进行保护，这样一来，比较安全的代码方法应如下所示：

Object obj1 = new NewType();

NewType newValue = null;

try

{

newValue = (NewType)obj1;

}

catch(Exception e)

{

MessageBox.show(e.Message);

}

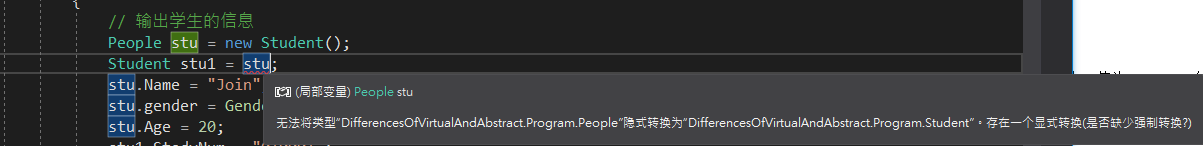
但是上面的写法在C#中已经过时了，也是比较低效的写法，比较高效且时尚的写法是用as操作符，如下：

Object obj1 = new NewType();

NewType newValue = obj1 as NewType;

// 这个语句中obj1是一个object类型的，值为NewType的实例。

但是下一句赋值时，object类型的对象必须显示转换为NewType类型才能进行赋值。



安全性：

As操作符不会做过的转换操作，当需要转化对象的类型属于转换目标类型或者转换目标类型的派生类型时，那么此转换操作才能成功，并且不产生新的对象（当不成功时，会返回null）。因此用as进行类型转换是安全的。

效率：

当用as操作符进行类型转换时，首先判断当前对象的类型，当类型满足要求后才进行转换，而传统的类型转换方式，是用当前对象直接去转换，而且为了保护转换成功，要加上try-catch，所以相对来说，as的效率更高。

使用需要注意的是

1. 不同类型之间进行类型转化是会发生错误的。

Object o = new NewType();

NewType1 type = o as NewType1;

1. 不能应用在值类型数据。

Object o = 11;

Int value = o as int;

对于2，在C#中可以使用is操作符，在加上老式的类型转换操作，就可以安全完成转换。

Object obj1 = 11;

If (objTest is int)

{

Int nValue = (int) obj1;

}

is操作符：检查对象是否与给定的类型兼容。

如下代码可以确定MyObject类型的一个实例，或者对象是否从MyObject派生的一个类型：

If (obj is MyObject){}

如果所提供的表达式非空，并且所提供的对象可以强制转换为所提供的类型而不会发生异常，其is表达式的结果就是true。

如果对象引用是null，is运算符总是返回false，因为没有可检查其类型的对象。

C#中的数据有两种类型：引用类型（reference types）和值类型（value types）。简单类型（包括int，long，double等）和结构（Structs）都是值类型，而其他的**类**都是引用类型。简单类型在传值的时候会做复制操作，而引用类型只是传递引用，就跟C++中的指针一样。在C++中，structs和类基本相同（expect that the default inheritance and default access are public rather than private）。而在C#中，struct和类有很大区别（由上所说一个是值类型，一个是引用类型）。