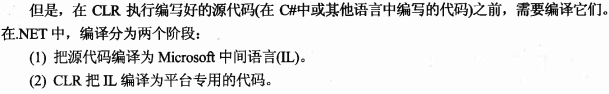
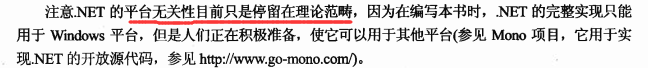
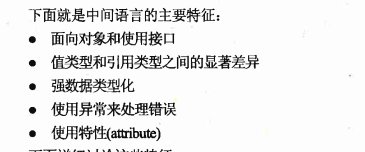
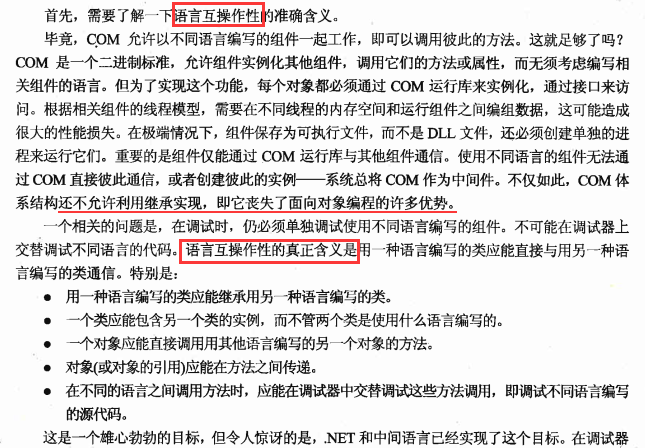
.NET FrameWork的 核心是其运行库执行环境,称为公共语言运行库CLR或.NET运行库。

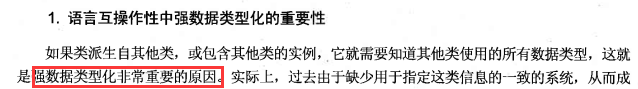




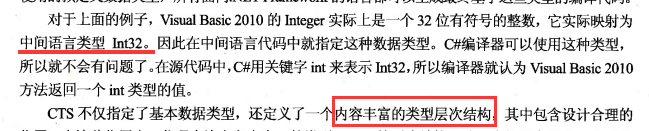


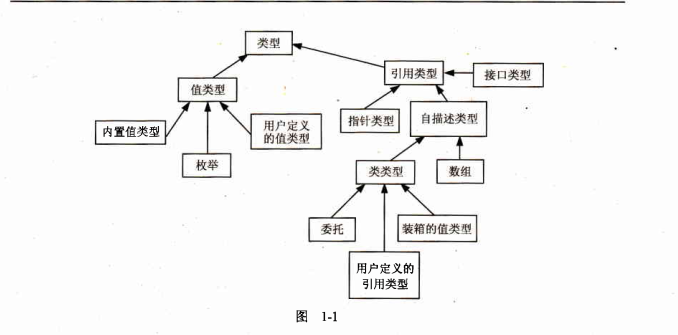




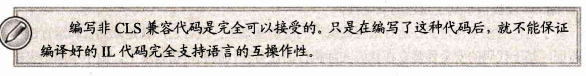


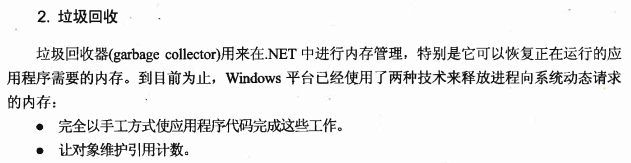


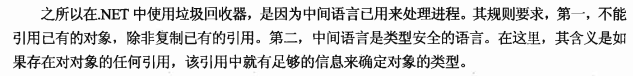












**元数据**：描述数据的数据。

GAC 全局程序缓存

JIT（just in time）编译器

第二章

语法

using System; 类似C++的using namespace xxx 或java中的import xxxx

标准的 system名 称空间包含了最常用的.NET类 型

C#没有用于输入和输出的内置关键字,而是完全依赖于.NET类

所有的 C#代码都必须包含在一个类中

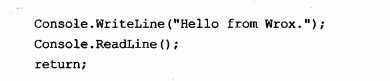
Main方法M大写。public static void Mian(){}

要么返回void要么int

C#中方法定义如下：

[modifiers] return\_type MethodName( [parameters] ){}

modifiers 例如public static



控制台读写：

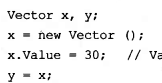
类型推断 (关键字var)：var someNumber = 0;

等价于 int someNumber = 0;

常量：const int a=0;

不能从一个变量中提取值来初始化常量

常量总是静态的，不必也不予许添加static修饰符



要创建对象必须使用new关键字；

这里xy都是引用类型：

if else

switch

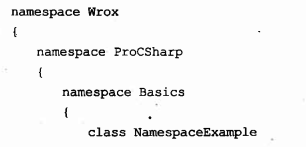
while do while

for

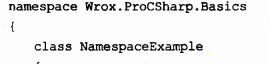
类似C++与java

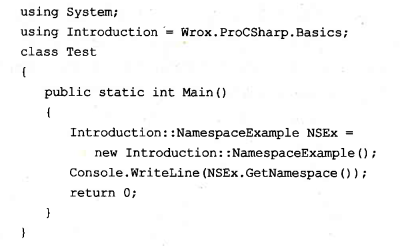
foreach(int temp in arrayOfInts){....}

不能再循环中改变temp的值



名称空间

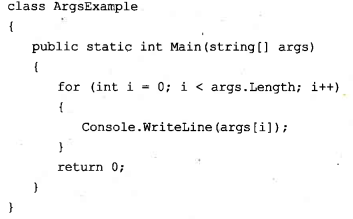


名称空间别名

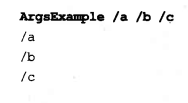


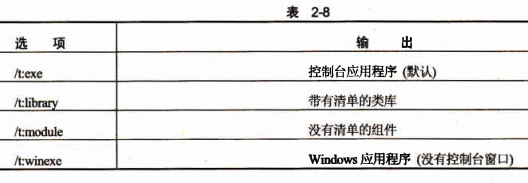
多个Main()方法

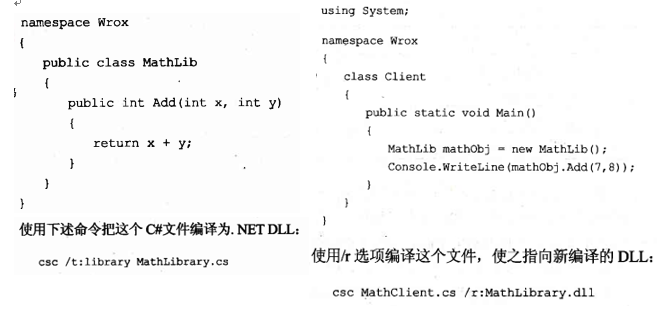
带参Main（）方法

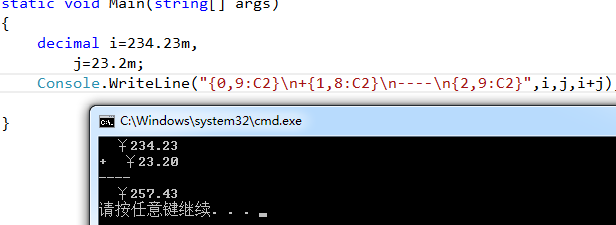


命令行运行与输出



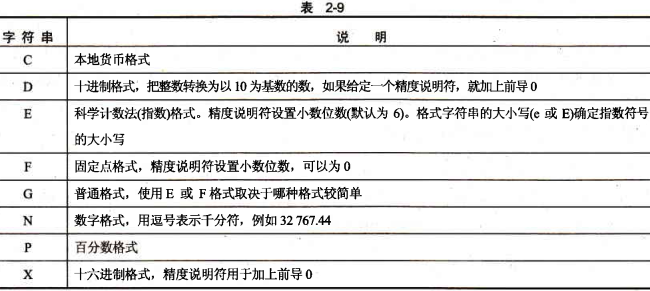




控制台

{0,9:C2}其中0指参数索引这里指代变量i，9指代占位数量，2指代小数位数。

C则是：



注释

以“///”的注释可以自动创建xml文档



预处理指令

#define 和#undef

告诉编译器，存在、删除给定的符号

#if、#elif、#else和#endif

使用define中的符号做条件

#error提醒消息并停止编译

#if DEBUG && RELEASE

#error “you’ve defined DEBUG and RELEASE simuteneously!”

#endif

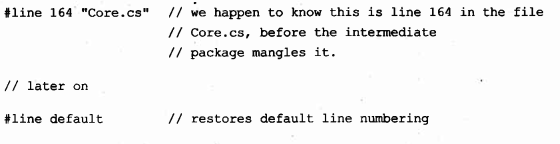
#warning不停止编译

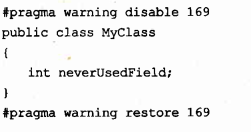
#warning “don’t forget to remove this line before boss check the code”

Console.WriteLine(“I hate the boss”);

#region和#endregion

#line可以用于改变编译器在警告和错误信息中显示的文件名和行号信息。



#pragma抑制或还原编译警告

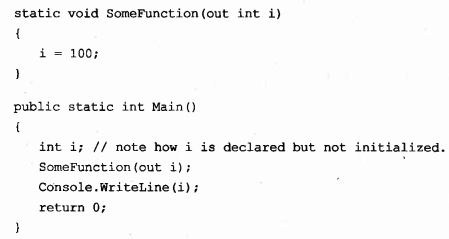
方法

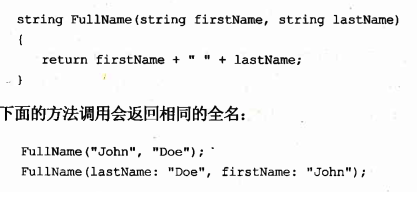
ref参数

static void SomeFunction(int [] ints, ref int i)参数i将会按引用传递。

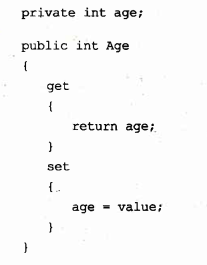
默认非引用类型都按值传递

out参数

 可以不初始化

命名参数

可以为参数提供默认值，但必须是方法定义的最后面的参数

类

属性

属性可以有公有的get访问器和

私有或受保护的 set访问器。

如果属性的 set和 get访 问器中没有任何逻辑,

就可以使用自动实现的属性。这种属性会自动实

现后备成员变量。如：



构造函数

没有必要给类提供构造函数。一般编译器会在后台创建默认构造函数

初始化字段为默认值，通常这就足矣。

如果声明了带参构造函数编译器就不会再生成默认构造函数

静态构造函数

通常在第一次调用类的任何成员之前执行静态构造函数。

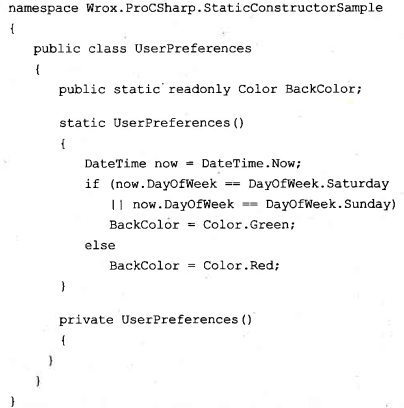
编写静态构造函数：

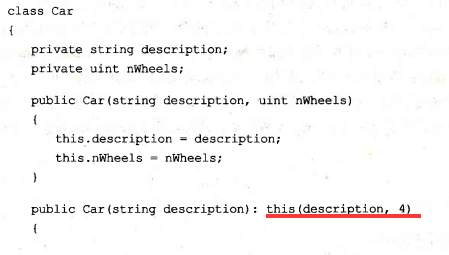
类中有一些静态字段或属性需要在第一次使用类之前，在外部源中初始化这些字段

没有访问修饰符，在加载类时，总由.NET调用

如果多个类都有静态构造函数,先执行哪个静态构造函数就不确定

如果任何静态字段有默认值,就会在调用静态构造函数之前指定它们





初始化构造器

初始化器中不能有多个调用

可以包含对直接基类的构造函数

的调用使用相同的语法,

但应使用 base关键字代替

this

只读字段

readonly关键字比const灵活，允许吧一个字段设置为常量，但还需要一些计算以确定初值。

可以在构造函数中给只读参数赋值。

如果把只读字段设置为静态，就必须显示声明他。

在构造函数中不必给只读字段赋值。 如果没有赋值,它的值就是其特定数据类型的默认值,或者在声明时给它初始化的值。

继承语法类似c++

class Myclass：MyBaseClass{...}

匿名类型

var aObject = new {FirstName=”ming”,LastName =”Morris”};

字段相同，类型就相同

即若另有var bObject=new {FirstName=”hg”,LastName =”Mo”};

则可以设置aObject=bObject;

结构

有时仅需要一个小的数据结构。此时,类提供的功能多于我们需要的功能,由于性能

原因,最好使用结构。

结构是值类型,不是引用类型

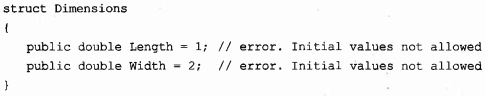
 合法的做法：



或者：

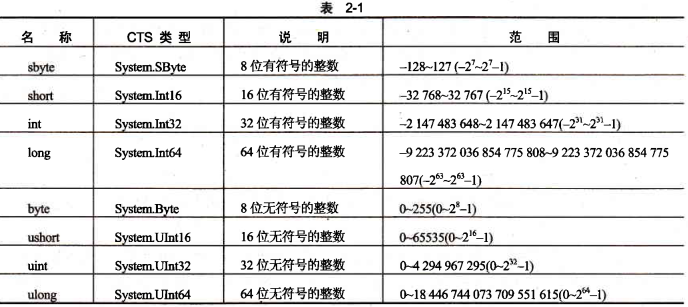
结构作为参数来传递或者把一个结构赋予另一结构结构的所有内容就被复制。

故当结构被作为参数传递时，应做ref参数传递，一避免性能损失。

结构中不能有初始值设定

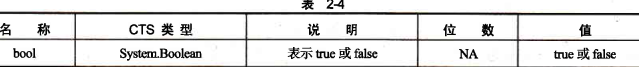
CTS预定义的类型：

值类型



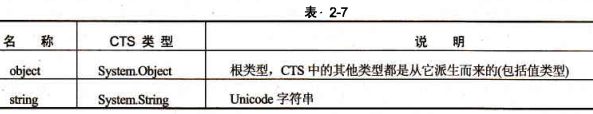


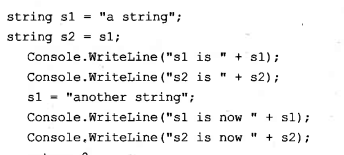






引用类型

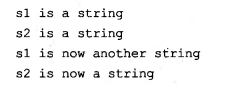


引用类似于指针但是

改变string的值会重新分配一个

新的空间。

输出为：



@的使用：string rowString=@”blabla”;

类似于c++的原始字符串字面量

部分类

用partial关键字在class前标记，可以在不同文件中编写同一个类。

如果声明类时使用了下面的关键字,这些关键字就必须应用于同一个类的所有部分：

public private protected internal abstract

sealed new 以及一般约束

在把部芬

类编译到类型中时,属 性(舢 注释、 接 口、 泛型类型的参数属性和成员会合并

静态类

只包含静态字段与方法

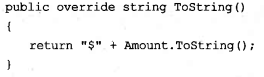
Object类

所有类都派生自System.Object

MemberwiseClone()复制对象并返回一个对对象副本的引用。（浅复制）

该方法不是虚方法,所以不能重写它的实现代码

ToString（）显示类的名称

 重写方法

扩展方法

没有类的源代码而需要为类添加功能

创建一个静态类，添加相应的方法

public static class MyExtensionClass{

public static void aExtensionMethod(this MyClass myclass,int arg){

myclass.somefield+=arg;

}

}

其调用方法与原来的类的调用一致，若拓展方法与类中某方法同名，则拓展的方法永不会被调用，即类中原有方法优先

第四章

C#类可以派生自另一个类和任意多个接口  
 C#不支持多重实现继承

结构可以支持多重接口继承

把一个基类函数声明为 virtual就可以在任何派生类中重写该函数

成员字段和静态函数都不能声明为virtaul

也可以把属性声明为virtual对于虚属性或重写属性,语法与非虚属性相同,但要在定义中添加关键字virtual

C#要求在派生类的函数重写另一个函数时,要使用override关键字显式声明

若编写了与父类同名的方法而未书写override关键字，编译器会给出警告，这时需要

隐藏父类方法，用new关键字注明该方法即可

可以使用base.BaseClasssMethdName()来访问父类的方法

抽象类

可以用abstract声明函数与类，如果包含抽象函数，则类也必须是抽象的。

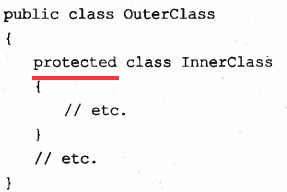
密封类

可以将方法或类声明为sealed让其不能被重写或继承

要在方法或属性上使用 sealed关键字,必须先从基类上把它声明为要重写的方法或属性。 如果基类上不希望有重写的方法或属性,就不要把它声明为virtual。

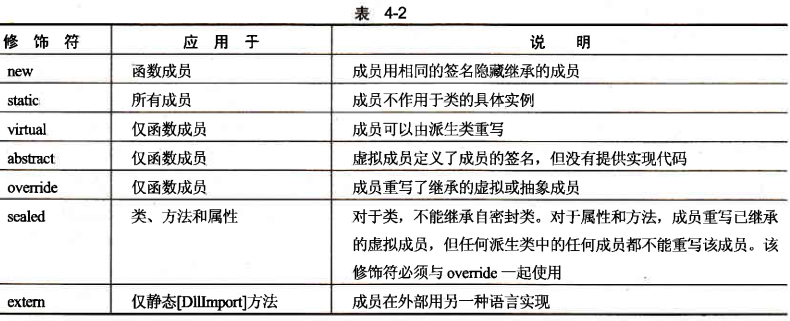
可见性修饰符





不能把类型定义为protected、private 和protected internal,因为这些修饰符对于包含在名称空间中的类型没有意义。 因此这些修饰符只能应用于成员。 但是,可以用这些修饰符定义嵌套的类型 即,包含在其他类型中的类型。

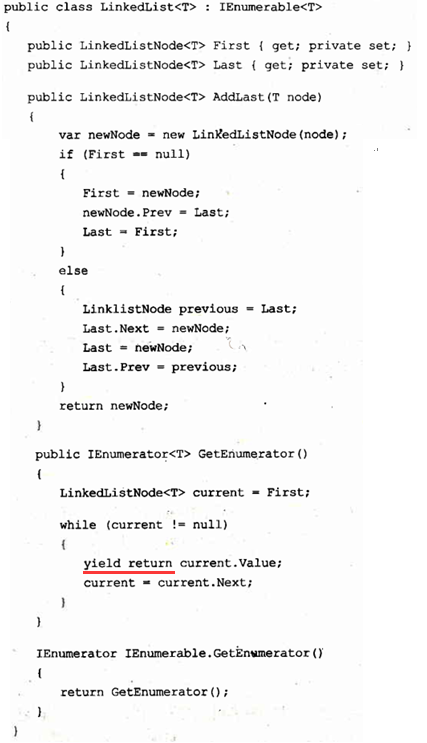
右例中InnerClass可以访问OuterClass的所有成员。

其他修饰符

接口

声明接口在语法上与声明抽象类完全相同,但不允许提供接口中任何成员的实现方式。一般情况下,接口只能包含方法、属性、索引器和事件的声明。

接口定义中还不允许声明关于成员的修饰符。

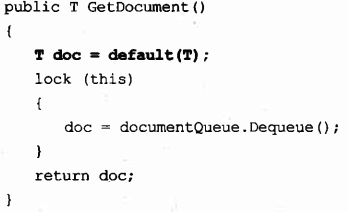


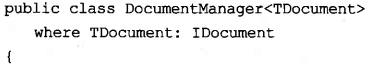
第五章 泛型

泛型类的定义会放在程序集中,所以用特定类型实例化泛型类不会在IL代码中复制这些类。但是,在JIT编译器把泛型类编译为本地代码时,会给每个值类型创建一个新类。

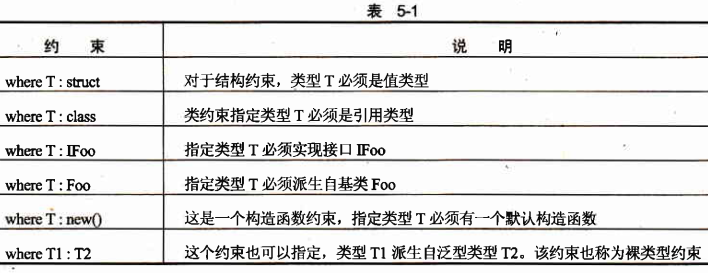
泛型类功能

默认值

default在这里给泛型类型初始化为null或0；

约束

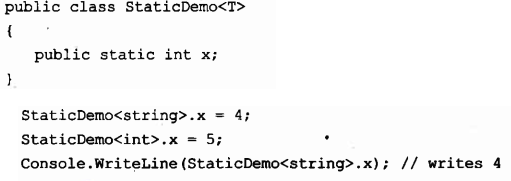
where子句指定泛型实现某接口



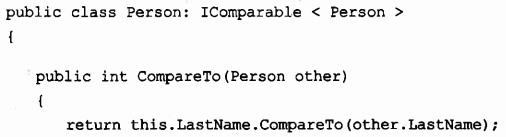
在where子句中只能定义基类、接口和默认构造函数。

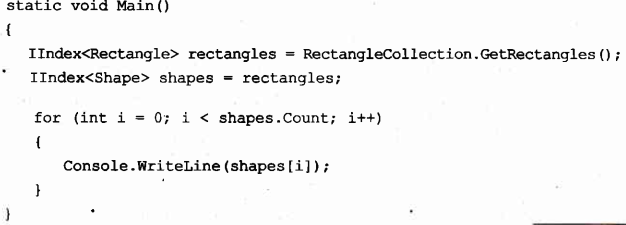
继承

从泛型类继承必须指定基类的泛型类型

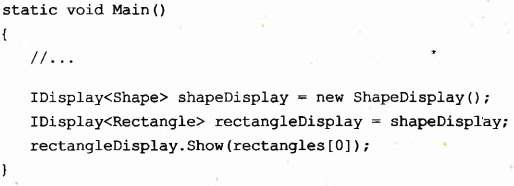
静态成员

对应不同泛型类型有各自的同名静态成员。（不同类型的泛型是不同的类）

泛型接口



泛型接口协变



泛型接口抗变

泛型结构Nullable<T> 可空类型.NET定义的一种类型

可空类型

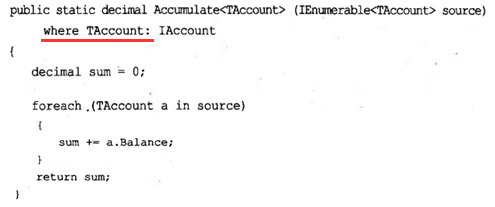
Nullable<int> x1; int ?x2;

 两种语法均可

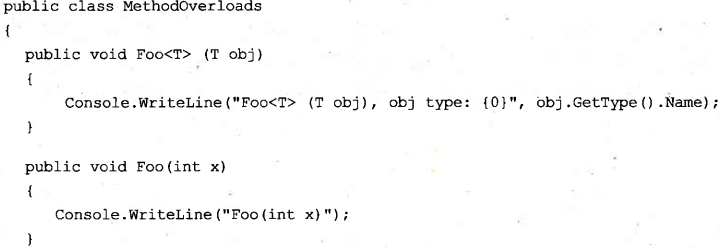
非空类型可隐式转换为可空

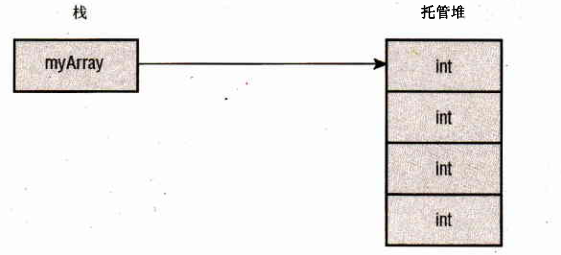
反过来可能会失败

因为x1可能空

泛型方法

带约束的泛型方法

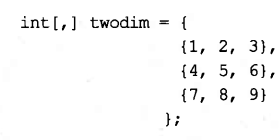
泛型特例化

第六章 数组

声明初始化

int[] myArray = new int[4];

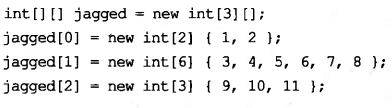
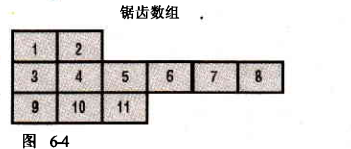
初始化器

数组初始化器只能在声明数组变量时使用不能在声明数组之后使用。

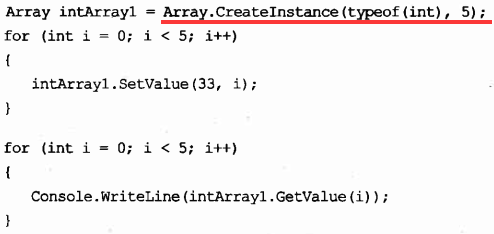
int[] myArray = new int[4]{4,7,5,6};

或者：int[] myArray={4,7,5,6};

锯齿数组

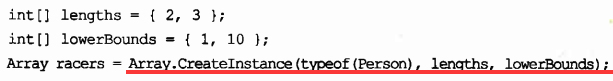


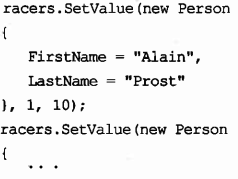
Array数组

创建

可以将已创建的数组强制转换成声明为int[]的数组

int[] intArray2 = (int[])intArrya1;



创建一个2\*3第一维基于1,第二维基于10的数组。

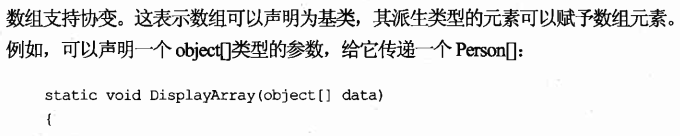
数组复制

还可以使用Array.Copy()方法创建浅表副本。

重要区别:Clone()方法会创建一个新数组,而Copy()方法必须传递阶数相同且有足够元素的己有数组

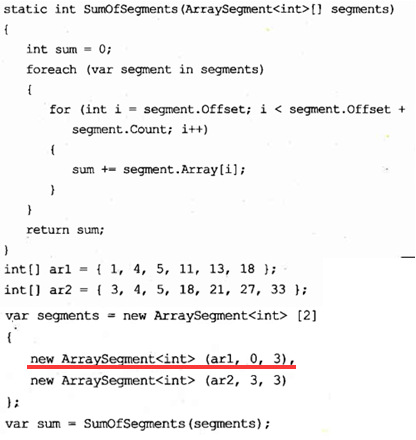
数组排序

Sort()方法使用快速排序算法，需要数组中的元素实现IComparable接口

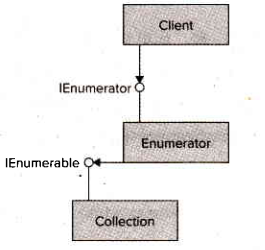
数组协变

数组协变只能用于引用类型不能用于值类型。

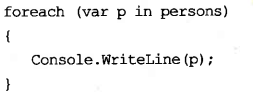
数组段ArraySegment<T>



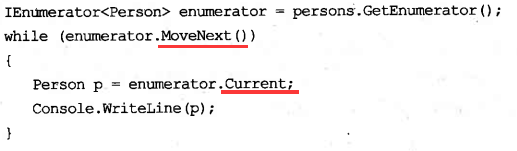
数组段不会复制数组元素，原数组的元素可以通过数组段访问，如果数组段元素改变，这些变化会反映到原数组中。

枚举

数组或集合实现带GetEumerator()方法的IEumerable接口。GetEumerator()方法返回一个实现IEmmble接 口的枚举。接着,fomch语句就可以使用IEmerbale接口迭代集合了。

foreach语句

C#编译器会将上代码解析为



yield语句

yield return语句返回集合的一个元素，并移动到下一个元素上。yield break可停止迭代。

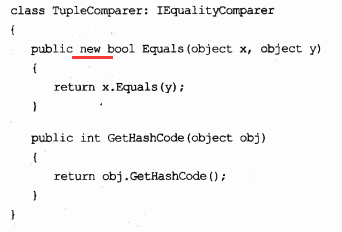
包含yield的语句的方法或属性也成为迭代块。

迭代块必须声明为返回IEnumerator或IEnumerable接口，或这些接口的泛型版本。

迭代块可以包含多个yield return或yield break但不能包含return

结构比较

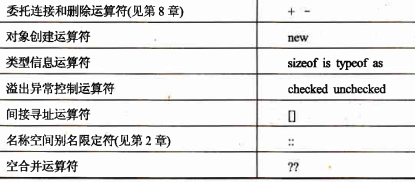
IstructuralEquatable IstructualComparable接口



**P159**

实现IequalityComparer接口的Equals()方法需要new修饰符或隐式实现接口，因为Object也定义了带两个参数的Equals()方法

第七章

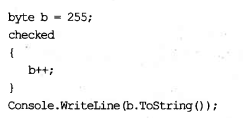
运算符

有4个运算符（sizeof、\*、->、和&）只能用于不安全的代码(这些代码忽略了C#的类型安全性检查)

++ --前置后置

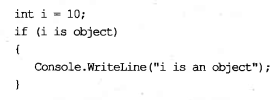
checked unchecked运算符

考虑如右面的代码，++操作会导致溢出CLR如何处理这个溢出取决于多种因素!

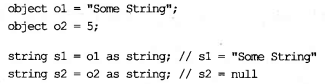
 可以标记代码块为checked，CLR就会执行溢出检查。

is运算符

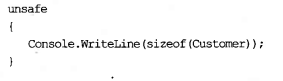
检查是否与特定类型兼容（表示对象或者该类型,或者派生自该类型）

 右例：int也是Object继承而来

as运算符

 显示类型转化。兼容则成功，不兼容则返回null

as运算符允许在一步中进行安全的类型转换，不需要先is

sizeof运算符

对于复杂类型(和非基元类型)使用sizeof运算符,就需要把代码放在unsafe块中。

typeof运算符

返回一个System.Type对象。

可空类型和运算符

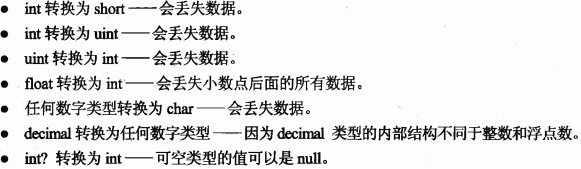
在比较可空类型时,只要有一个操作数是null,比较的结果就是false。

空合并运算符

第一个操作数必须是一个可空类型或引用类型;第二个操作数必须与第一个操作数的类型相同,或者可以隐含地转换为第一个操作数的类型。

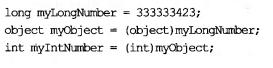
如果第一个操作数不是null,整个表达式就等于第一个操作数的值。如果第一个操作数是null,整个表达式就等于第二个操作数的值。

类型安全性

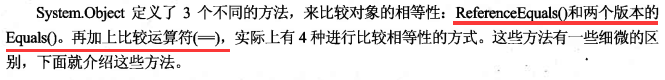


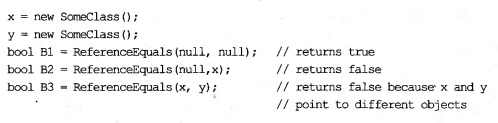




在拆箱时,须确保得到的值变量有足够的空间存储拆箱的值中的所有字节。

如：右例会抛出异常

对象相等性

ReferenceEquals()

虚拟的Equals()

静态的Equals()

==运算符 默认直接比较值或者引用