[1. 关键字 1](#_Toc465870502)

[1.1. ReadOnly 1](#_Toc465870503)

[1.1-1. 定义 1](#_Toc465870504)

[1.1-2. 是字段才有的关键字，属性不能使用 1](#_Toc465870505)

[1.1-3. 只读字段只能在构造函数或者变量声明时赋值 1](#_Toc465870506)

[1.1-4. ReadOnly Field 也可以赋值很多次 1](#_Toc465870507)

[1.1-5. 用readonly 标识的集合仍可以添加和修改元素 2](#_Toc465870508)

[1.1-6. 用readonly标识的引用类型字段仍可以修改引用类型的值 3](#_Toc465870509)

[1.1-7. 针对struct， 内嵌的属性自然是不能更改的 3](#_Toc465870510)

[1.1-8. No readonly reference instance 4](#_Toc465870511)

[1.1-9. ReadOnlyCollection 5](#_Toc465870512)

[1.1-10. ReadOnly和Const 区别 5](#_Toc465870513)

[2. 语法 6](#_Toc465870514)

[2.1. Int 和 Char的互相转换 6](#_Toc465870515)

[2.1-1. Int to char 6](#_Toc465870516)

[2.1-2. Char to int 6](#_Toc465870517)

[3. Class 9](#_Toc465870518)

[3.1. 互相引用或者递归引用自身时的堆栈溢出 9](#_Toc465870519)

[3.1-1. Class在构造的时候互联引用，堆栈溢出 9](#_Toc465870520)

[3.1-2. 递归引用自身，堆栈溢出 10](#_Toc465870521)

[3.1-3. 引用自身的第二种写法，仍然堆栈溢出 10](#_Toc465870522)

[3.1-4. 引用自身的正确写法 11](#_Toc465870523)

[4. Struct 11](#_Toc465870524)

[4.1. 定义 11](#_Toc465870525)

[4.2. Struct 可以实现interface 11](#_Toc465870526)

[4.3. Struct 不允许有Protected 的修饰符 12](#_Toc465870527)

[4.4. Struct 不能显式声明无参的构造函数 12](#_Toc465870528)

[4.5. Struct 不允许实例字段的初始化，static 字段可以 13](#_Toc465870529)

[4.6. Struct 可以不使用new ，但作为局部变量时必须先初始化 14](#_Toc465870530)

[4.7. Struct作为全局变量时，编译器允许不初始化就是用 15](#_Toc465870531)

[4.8. 使用new 关键字生成的struct， 编译器认为其已经初始化过了 15](#_Toc465870532)

[4.9. 不管用不用new， struct 作为值类型都是生成在栈上 15](#_Toc465870533)

[4.10. Struct 作为引用对象的filed 时，struct 各字段不能单独赋值 15](#_Toc465870534)

[4.11. Class 和 struct的区别 19](#_Toc465870535)

[4.11-1. 从使用场景的角度 19](#_Toc465870536)

[4.11-2. 从语法的角度 19](#_Toc465870537)

[5. Interface 19](#_Toc465870538)

[5.1. Interface 中的方法都不需要加public 关键字，也不能加 19](#_Toc465870539)

[6. Enum 20](#_Toc465870540)

[6.1. 枚举在IL内部被转换为int32 的值类型 20](#_Toc465870541)

[6.1-1. 不过在Class Enum 内部应该用的uLong 20](#_Toc465870542)

[6.2. 枚举的Tostring() 显示自己的名称 21](#_Toc465870543)

[6.3. FlagsAtribute 21](#_Toc465870544)

[6.4. 枚举内部实现 22](#_Toc465870545)

[6.4-1. 枚举继承自System.Enum 类 22](#_Toc465870546)

[6.4-2. Values 和 names 数组 23](#_Toc465870547)

[6.4-3. Tostring() 23](#_Toc465870548)

[6.5. C#编译器不允许直接继承System.Enum 24](#_Toc465870549)

[6.6. 可以把一个不存在的value 值强制转化为Enum 25](#_Toc465870550)

[6.7. Enum 项目可以赋负值 26](#_Toc465870551)

[7. Collections 27](#_Toc465870552)

[7.1. C#没有List类 27](#_Toc465870553)

[7.2. Array, ArrayList, List<T> 27](#_Toc465870554)

[7.2-1. 这三个其实都不能叫链表，而都应该叫数组 28](#_Toc465870555)

[8. IO 28](#_Toc465870556)

[8.1. 默认的Encoding为UTF8 28](#_Toc465870557)

[8.1-1. UTF8 BOM 是什么 29](#_Toc465870558)

[8.2. 默认的BufferSize 为1024 30](#_Toc465870559)

[8.3. End of File 30](#_Toc465870560)

[8.3-1. ReadFile 遇到end 时返回0 而不是-1 30](#_Toc465870561)

[8.3-2. End of File 总是单独读的，只有所有字节读完了才会遇到end of file 31](#_Toc465870562)

[8.4. Read 的时候，Byte[] 即使入参，也是出参 32](#_Toc465870563)

[8.5. Read 自己有一个Buffer 32](#_Toc465870564)

[8.6. Encoding 是类而不是枚举 33](#_Toc465870565)

[8.6-1. ASCII， UTF8等是Encoding类的static 成员 33](#_Toc465870566)

[8.6-2. Encoding 类都有一个相关的Encoder 和Decoder类 33](#_Toc465870567)

[9. 关键类和结构 34](#_Toc465870568)

[9.1. Convert 34](#_Toc465870569)

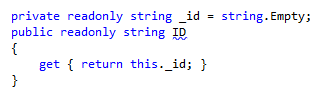
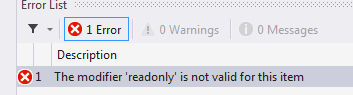
# 关键字

## ReadOnly

### 定义

The readonly keyword is a modifier that you can use on **fields**. When a field declaration includes a readonly modifier, assignments to the fields introduced by the declaration can only occur as part of the **declaration or in a constructor** in the same class.

### 是字段才有的关键字，属性不能使用

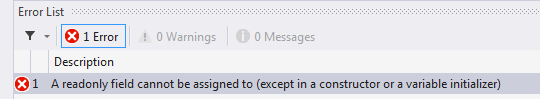
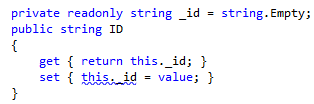
 

因为属性本质上是方法，方法谈不上只读还是可读写。属性是对字段（Field）的封装，只有字段，也就是数据才有只读，可读写的概念。

方法通过是否设置set 方法来实现对字段是否可读写的封装

### 只读字段只能在构造函数或者变量声明时赋值

否则编译器就会报错： A readonly field cannot be assigned to (except in a constructor or a variable initializer)



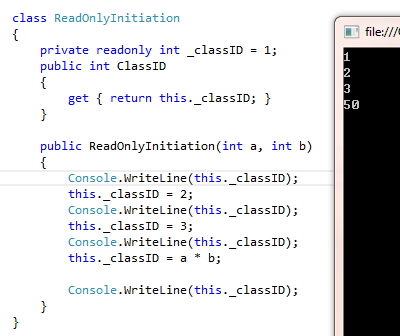
所以只读字段对应 的属性也不能有set方法

* For a instance field, in the instance constructors of the class
* for a static field, in the static constructor of the class

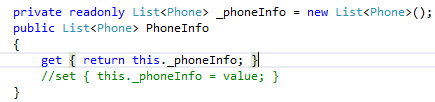
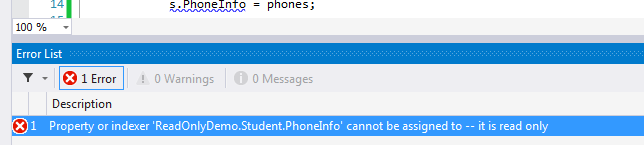
### ReadOnly Field 也可以赋值很多次

只要是这种赋值发生在field 声明或者在构造函数内部就可以。Readonly 强调的是一旦对象初始化完成，就不能再修改。而不是说只能进行一次赋值。

下面这段代码完全没问题

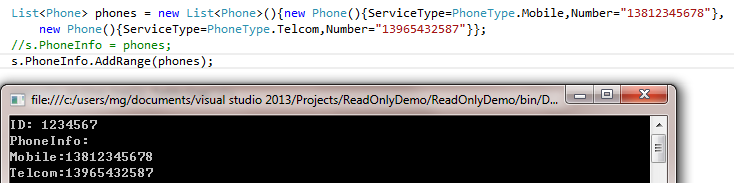


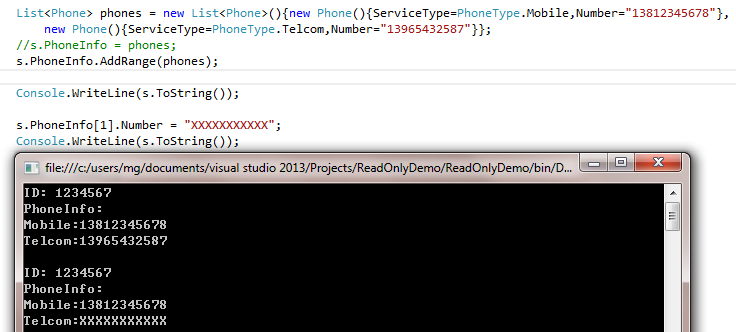
### 用readonly 标识的集合仍可以添加和修改元素

给只读集合赋值会报错。

但是给集合添加对象不会出错

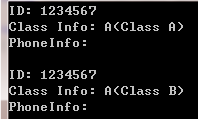
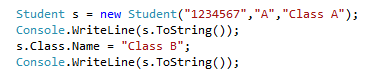
还可以改变集合里面每个元素的属性



还可以对collection的每一个item重新赋值



### 用readonly标识的引用类型字段仍可以修改引用类型的值

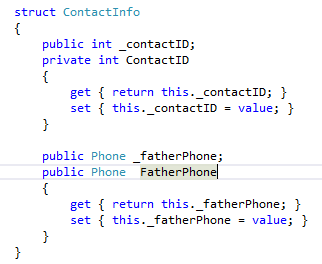
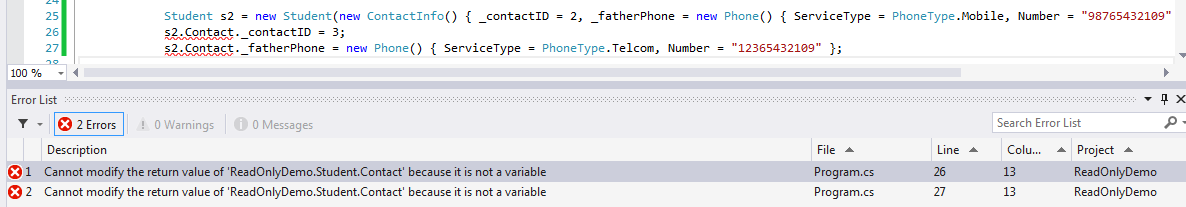
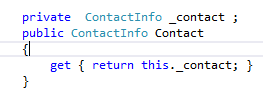


属性里面的内容仍然可以修改

### 针对struct， 内嵌的属性自然是不能更改的

这是和boxing， un-boxing机制相关的。

而struct本身如果用readonly 修饰，自然是不能改的

### No readonly reference instance

there's a big difference between making the field readonly and making the object itself immutable.

<http://stackoverflow.com/questions/681287/how-to-make-a-reference-type-property-readonly>

C#本身并不存在一种机制可以让object itself 本身不能被改变。有一种替代的方案是immutable 类型。但是.net 框架并没有提供一种现成的类或者attribute 可以很方便的声明一个immutable 的类型。所有的机制都需要开发者自己去完成

### ReadOnlyCollection

命名空间在System.Collections.ObjectModel， 而不是System.Collections.Generic 或者System.Collectios;



该类型没有add 或者insert 方法，所以无法直接往里面添加元素。

### ReadOnly和Const 区别

A const field can only be initialized at the declaration of the field

A readonly field can be initialized either at the declaration or in a constructor

readonly fields can have different values depending on the constructor used.

while a const field is a compile-time constant, the readonly field can be used for runtime constants （readonly 的使用场景是运行时常量）

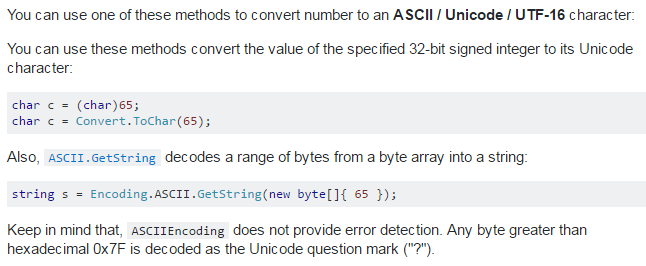
比如系统开始运行的时刻：在系统一经运行开始就不能再更改，但是又不是一个编译时可以知道的常量。



# 语法

## Int 和 Char的互相转换

### Int to char



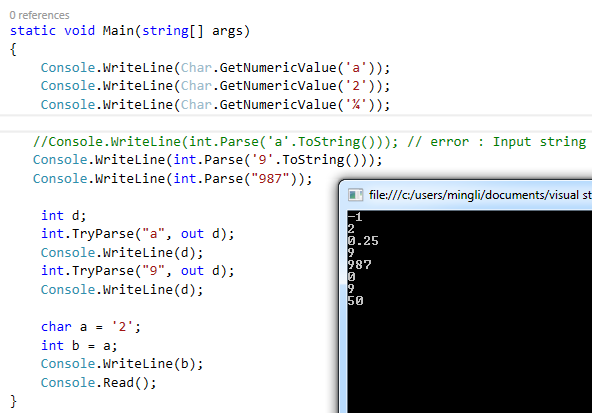
### Char to int

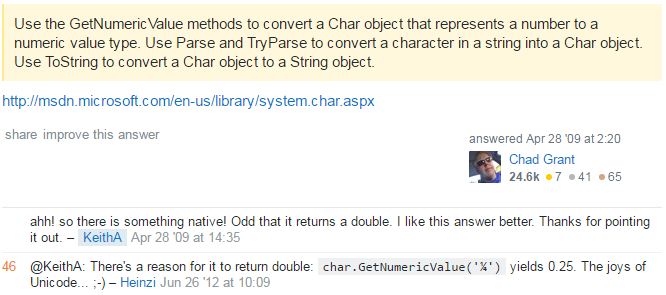
其实这里面分两种情况 这个char 本身是数字形式的，

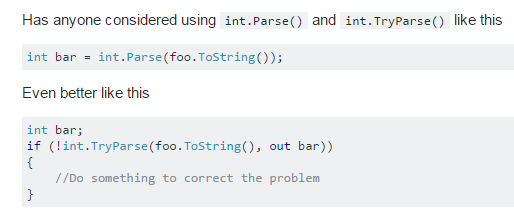
比如’1’, ‘2’, ‘9’ 等，那么它转化为int 就是对应的转化为 1， 2， 9

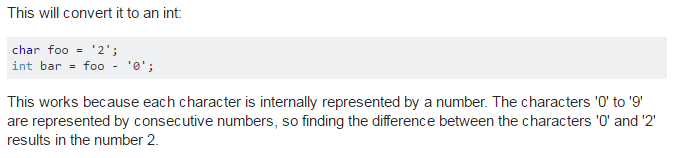
如果用’a’ , 因为’a’ 并不是一个有效的number (0~9)， 所以’a’ 不能转换为int，强转则报错

另一种情况是取的字符的ASCII 编码， 这时不仅字符是不是’1’ 到’9’ 都可以。‘a’也可以。而且’2’ 也不在得到十进制的2，而是得到50(0x32)





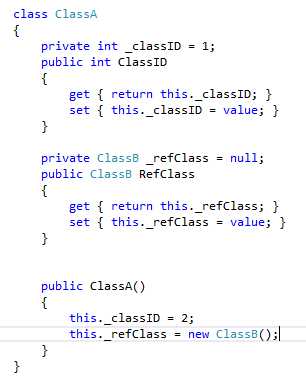
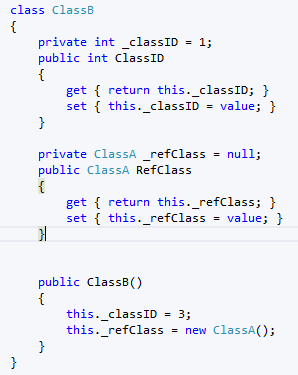
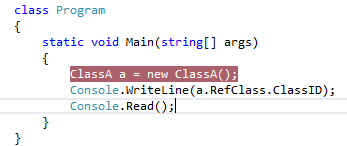


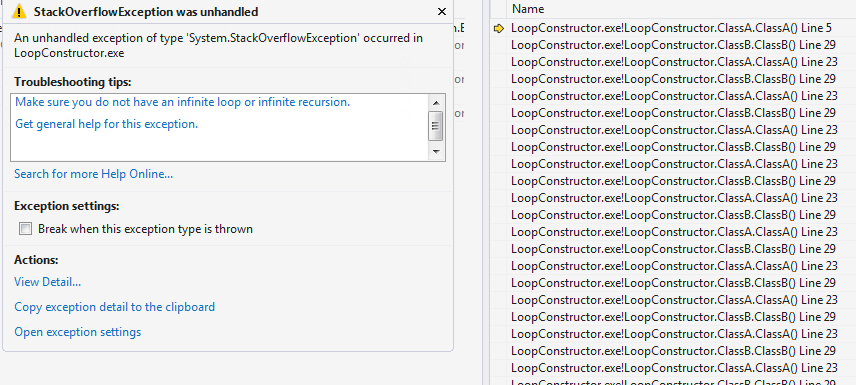


# Class

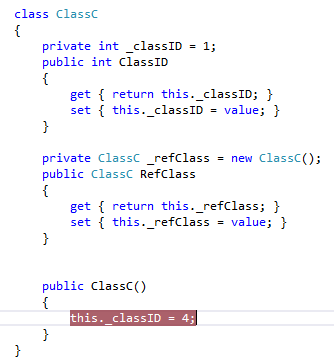
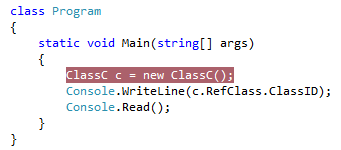
## 互相引用或者递归引用自身时的堆栈溢出

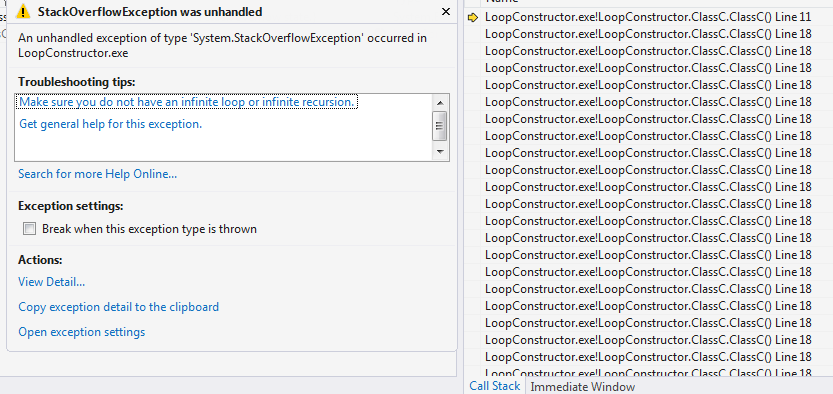
### Class在构造的时候互联引用，堆栈溢出

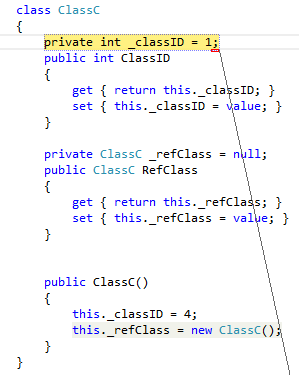
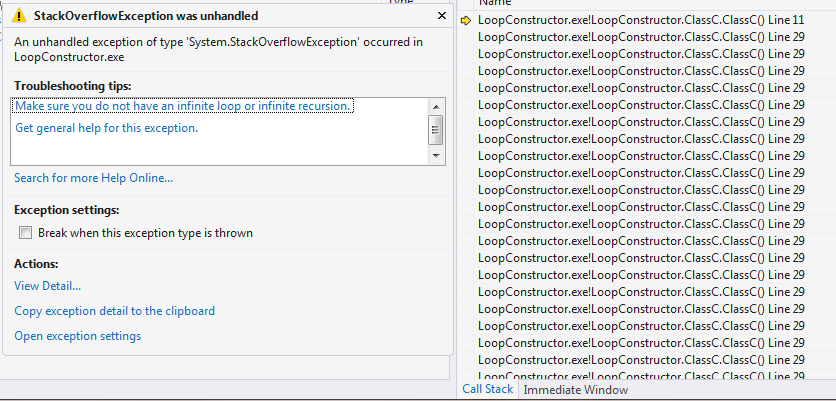


### 递归引用自身，堆栈溢出

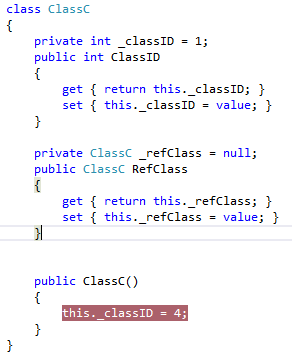
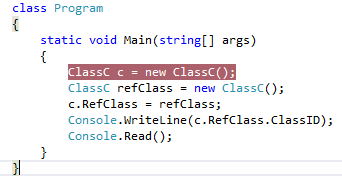
 



### 引用自身的第二种写法，仍然堆栈溢出

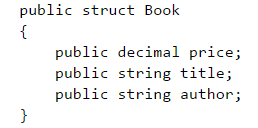
### 引用自身的正确写法

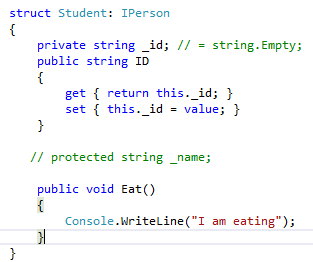
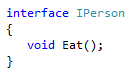
# Struct

## 定义

A struct type is a value type that is typically used to encapsulate small groups of related variables, such as the coordinates of a rectangle or the characteristics of an item in an inventory。

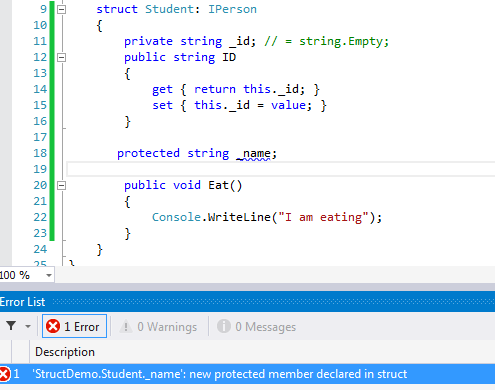


## Struct 可以实现interface

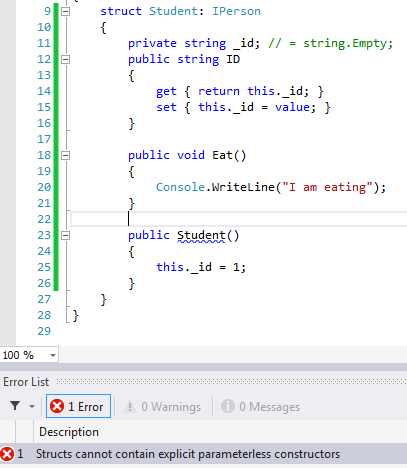
 

## Struct 不允许有Protected 的修饰符

因为Struct 不允许相互继承，所以Protected 也就没有了意义。不过private，public 没问题。

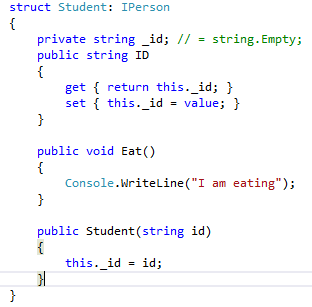


## Struct 不能显式声明无参的构造函数

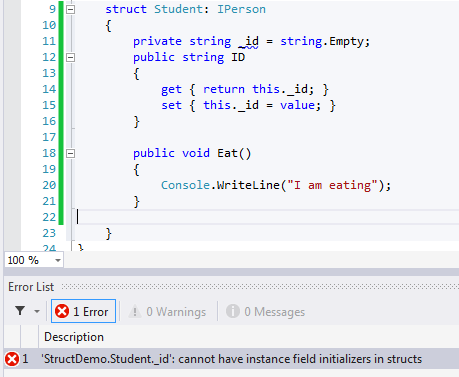


编译器保证struct 必定有一个无参的构造函数（应该是当不使用new 关键字生成struct时调用的）。在这个构造函数里，所有的值类型被初始化为默认值，所有的引用类型被初始化为null。

带参数的构造函数是允许的

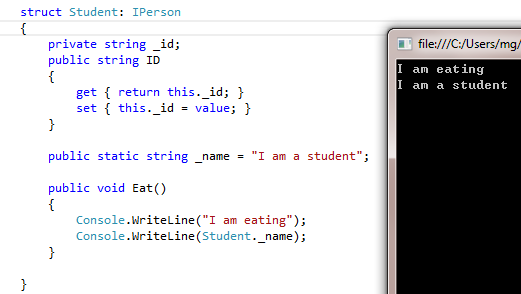


## Struct 不允许实例字段的初始化，static 字段可以



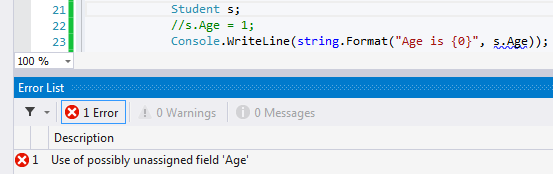
所有的初始化都在默认构造函数里自动做

the default value of a struct consists of the value that results from setting all value type fields to their default value and all reference type fields to null. For this reason, a struct does not permit instance field declarations to include variable initializers. This restriction applies only to instance fields. Static fields of a struct are permitted to include variable initializers.

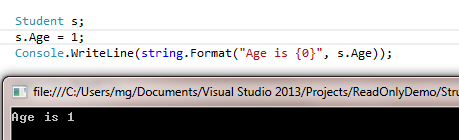


## Struct 可以不使用new ，但作为局部变量时必须先初始化

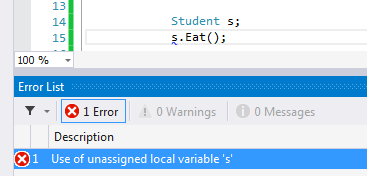
否则会报未初始化错误，虽然其实编译器会调用struct的默认构造函数进行初始化。



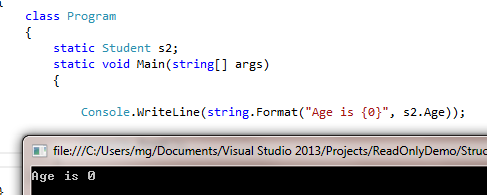
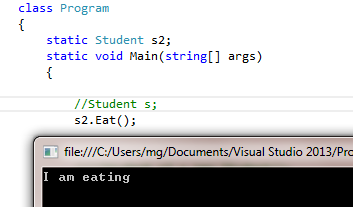
在使用时先赋一下值就可以了



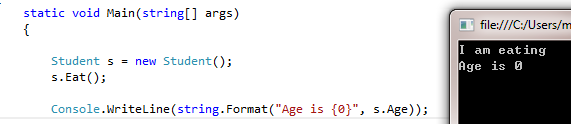
也不允许调用方法， 此时只能用new 关键字了



## Struct作为全局变量时，编译器允许不初始化就是用

## 使用new 关键字生成的struct， 编译器认为其已经初始化过了



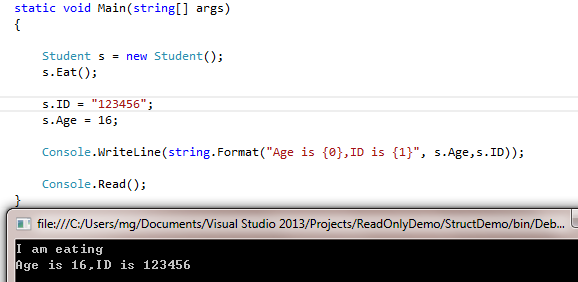
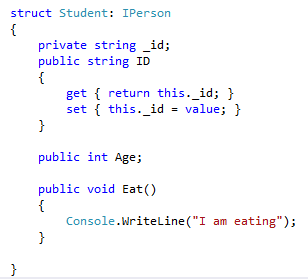
## 不管用不用new， struct 作为值类型都是生成在栈上

当然这是单独使用struct的情况。

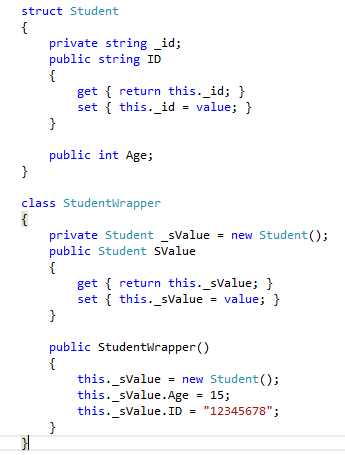
如果struct 作为别的一个引用类型的字段，那么它的值是生成在堆上的。

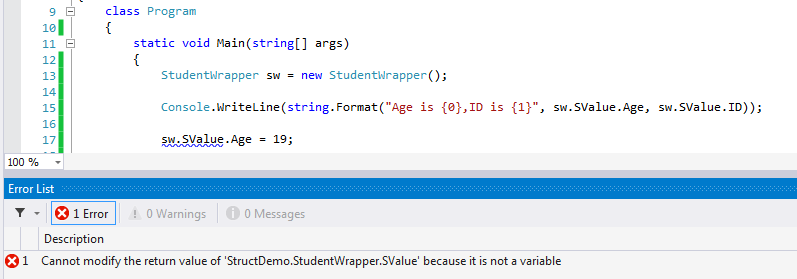
## Struct 作为引用对象的filed 时，struct 各字段不能单独赋值

当struct 单独使用时，其每个字段是可以单独赋值的。

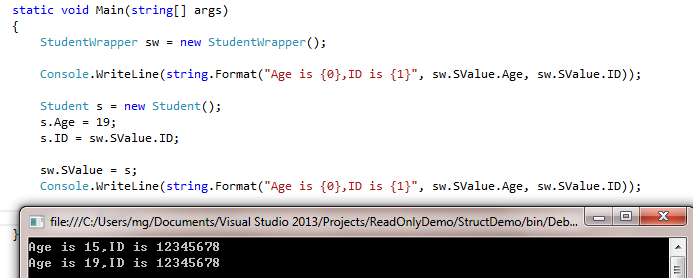
 

但是当struct 作为引用对象的field 时，就不能给struct的各个字段单位赋值了。 这是由于.net 的boxing-unboxing机制造成的。由于student是一个值类型，sw.SValue是在栈上复制了一份sw 的sValue的完整拷贝。此时sw.SValue 和 sw在堆中的sValue 的值已经完全没有关系了。直接读是没问题的，但当试图修改时，显然修改不能体现在堆上，所以编译器在编译阶段就直接出错了。



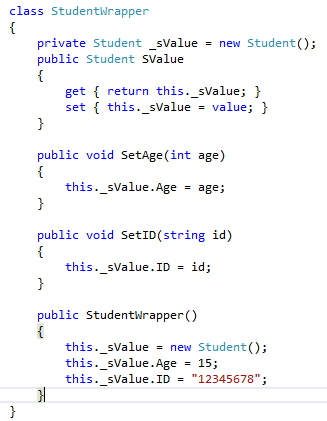


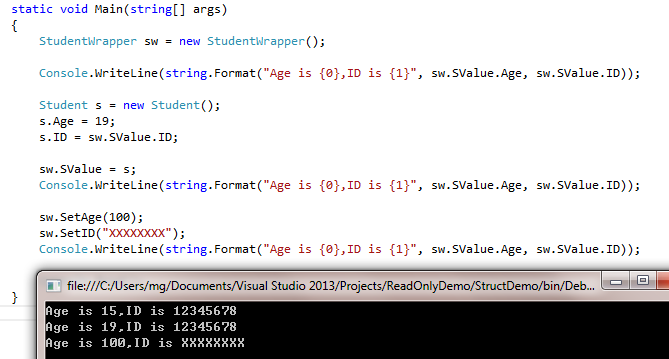
解决方案一： 整体赋值



解决方案二： 使用set 函数

根据编译起的错误可知，报的错误是StudentWrapper.SValue 的返回值不是一个变量所以不能更改，那我不使用这个返回值不就行了？ 而在StudentWrapper的构造函数里，我们看到是可以对SValue 的各个field 单独赋值的。所以我们可以考虑使用类似set 函数的方法





## Class 和 struct的区别

### 从使用场景的角度

Structs can also contain constructors, constants, fields, methods, properties, indexers, operators, events, and nested types, although if several such members are required, you should consider making your type a class instead.（这是微软的建议：https://msdn.microsoft.com/en-us/library/ah19swz4.aspx）

struct 有它自己的使用场景，如果你把它当成了class 来用，那为什么不一开始就定义成class得了

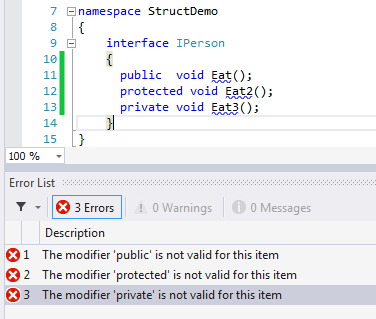
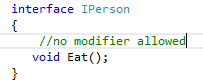
### 从语法的角度

1. Class 必须要使用new 关键字生成实例。 Struct 则可以使用new 也可以不用，就如同定义一个int 类型一样
2. Class 可以显式声明不带参数的构造函数，Struct 不可以。
3. Class 可以继承别的Class，Struct 不支持彼此间的继承

# Interface

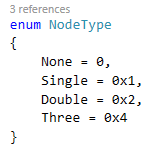
## Interface 中的方法都不需要加public 关键字，也不能加

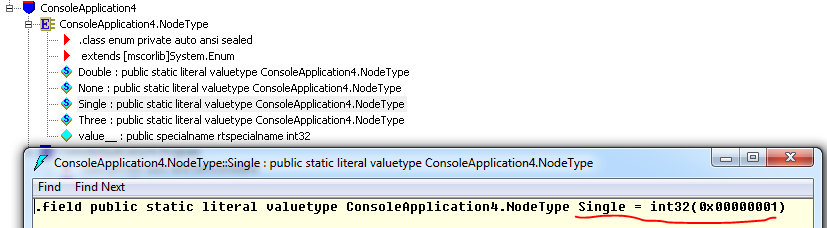
事实上interface 中的方法不允许加任何的修饰符。只能用默认的就是public

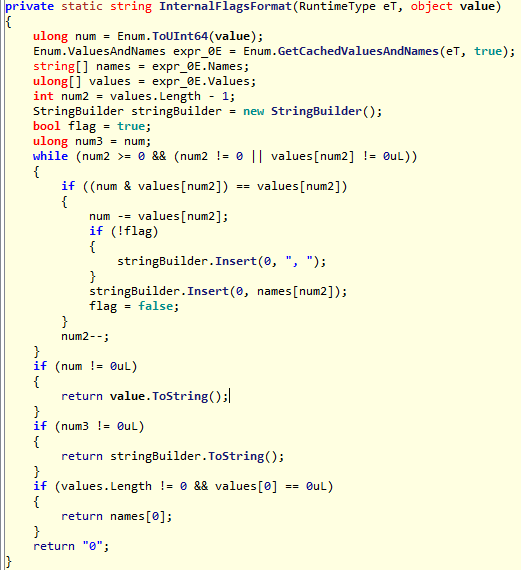
# Enum

## 枚举在IL内部被转换为int32 的值类型

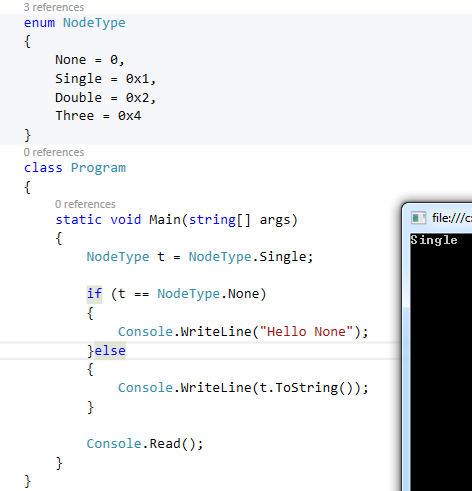




### 不过在Class Enum 内部应该用的uLong



## 枚举的Tostring() 显示自己的名称

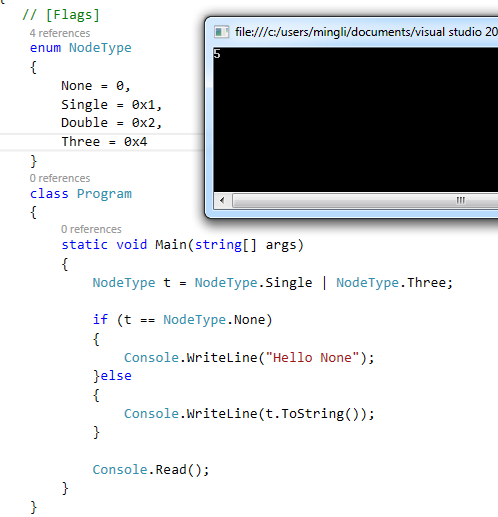
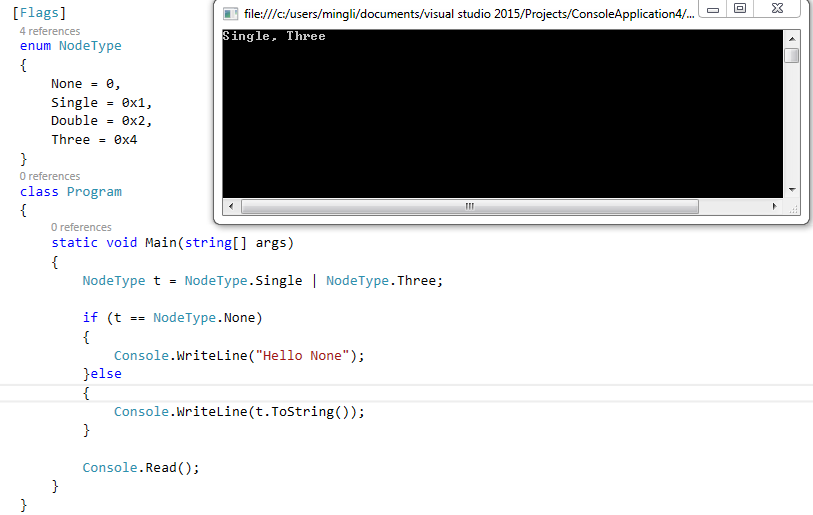


## FlagsAtribute

用来指示Enum是否可以被视为一组bit 位的flag，相互间可以与或者或。其实由于Enum 内部被实现为int32，无论加不加这个标志，都是可以与或的。加的目的是更加清晰，尤其是Tostring的时候。

对于组合的enum, 加flag的时候，显示的是名字的组合

不加flag的时候，显示的是int值， 因为找不到该值对应的name 标签

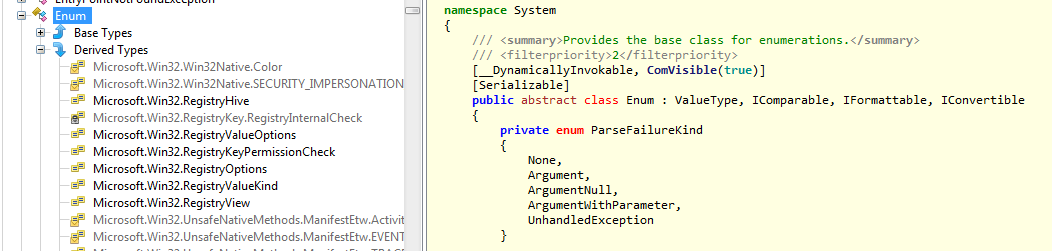
 

原因是在Enum 的tostring 里，会根据Enum是否加了Flags属性标签来采用不同的方法

## 枚举内部实现

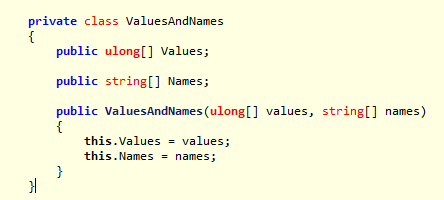
### 枚举继承自System.Enum 类

注意是一个abstract 类， 不能直接实例化，必须通过子类继承



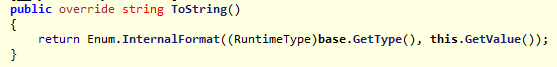
### Values 和 names 数组

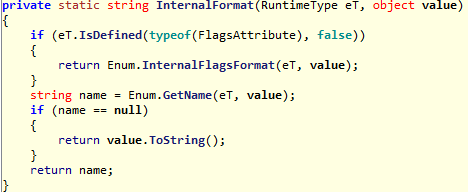
Enum 有一个values 数组和一个 string 数组来存储 值和name



### Tostring()

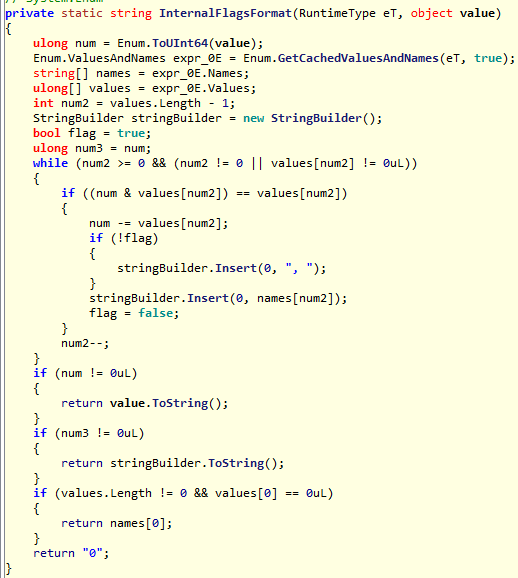
会根据是否实现flag 来用不同的方法来tostring。





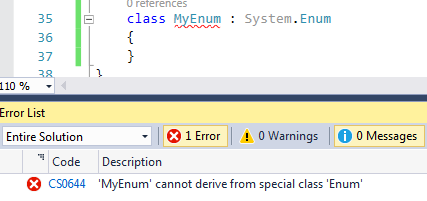
没有flag的时候，通过value 找到name，有则显示，没有则直接显示value 值

有flag的时候，会把那么一个个拼起来

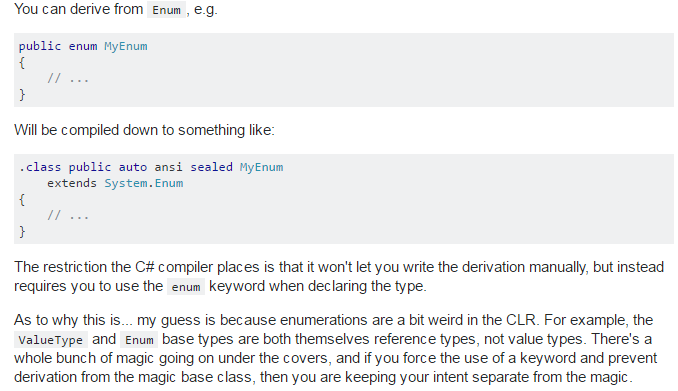


## C#编译器不允许直接继承System.Enum

System.Enum 是个抽象类，必须通过子类继承才能实例化。但是C#编译器不允许直接继承。

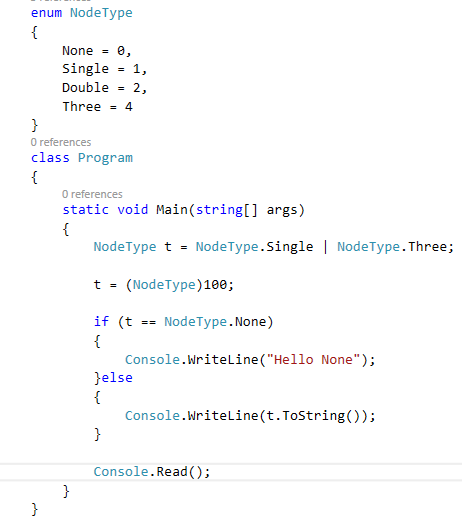


继承是编译器内部隐式做的：



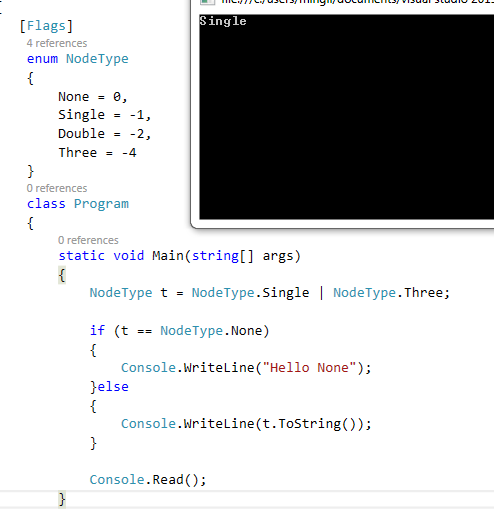
## 可以把一个不存在的value 值强制转化为Enum

编译和运行并不会报错， 这其实和几个enum 项目做与是一个效果



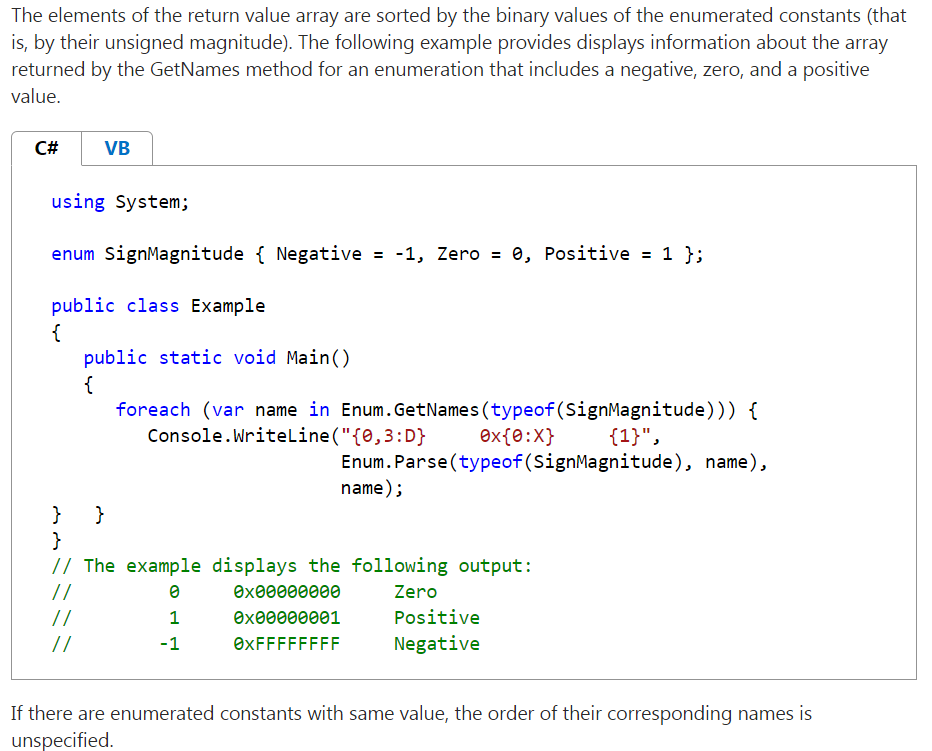
## Enum 项目可以赋负值

如果你愿意的话，虽然结果怪怪的，可能是有符号数转成无符号数的转换问题。



微软给的官方例子中就有一个把枚举项设置为-1的

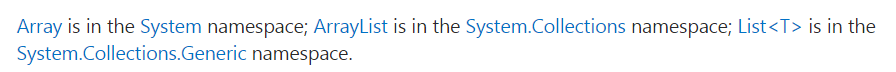
<https://msdn.microsoft.com/en-us/library/system.enum.getnames.aspx>



# Collections

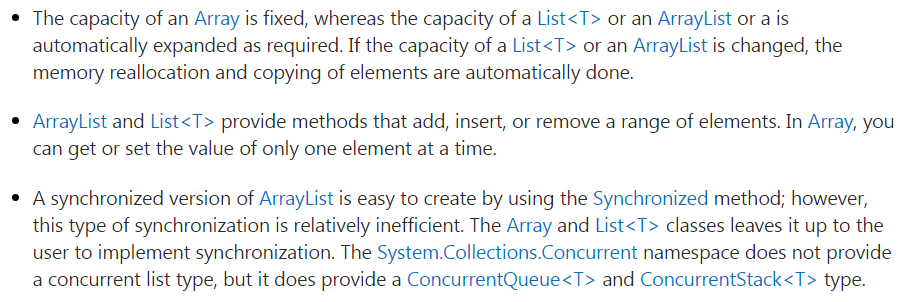
## C#没有List类

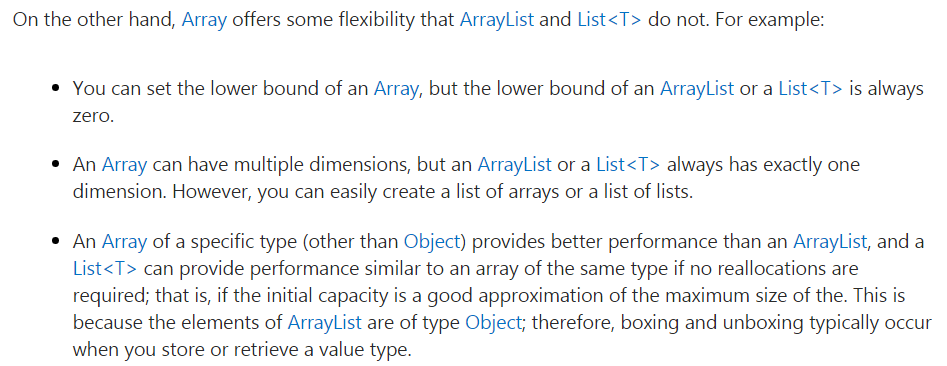
C#根本就没有List 类， 只有List<T> 泛型类，或者IList 接口



## Array, ArrayList, List<T>

ArrayList 只处理Object 类型的对象，所以可能会对值类型进行不必要的拆装箱





### 这三个其实都不能叫链表，而都应该叫数组

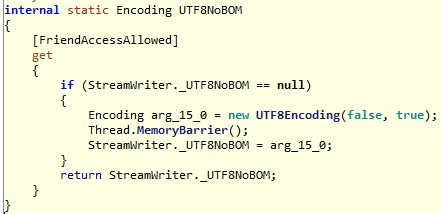
虽然名字里有List，但内部都是用数组来实现的，和链表LinkedList 还是有本质区别的

# IO

## 默认的Encoding为UTF8

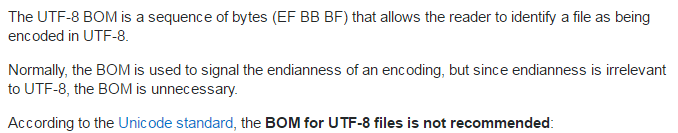
以StreamWriter 为例，默认的Encoding 是StreamWriter.UTF8NoBOM

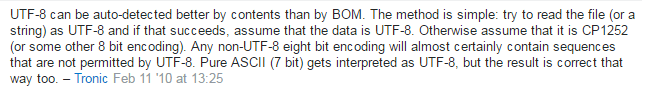
而UTF8NoBOM



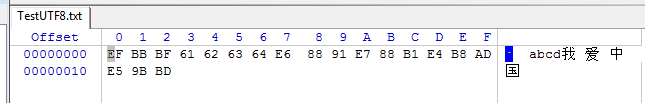
### UTF8 BOM 是什么

Byte Order Mark

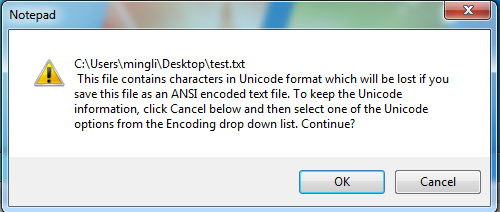




下面是以UTF-8 格式存的abcd我爱中国文件。可以看到刚开始的三个字节是EFBB BF



而同一段文本存为ANSI格式的时候





存的时候说会Lost，其实是真的Lost了，因为所有的字符都被存成了？，是真正的？，字节都丢了。所以，字符和编码是两个方面。一方面，字符的编码是什么，另一方面，每个编码格式都有自己的字符集。不在这个字符集内的字符是无法用这种编码格式编码的，那就会用一个自己字符集内的有效的字符的编码来代替那个无法编码的字符，所以信息丢失的时候是不可逆的

## 默认的BufferSize 为1024

仍然以StreamWriter 为例





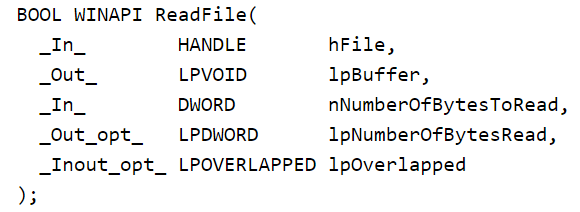
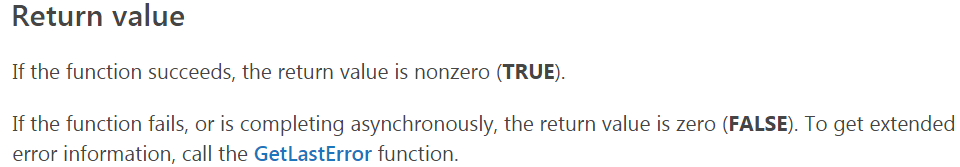
## End of File

### ReadFile 遇到end 时返回0 而不是-1

C# 中所有FileStream的Read 最后都其实是转换为对Win32的ReadFile API的调用



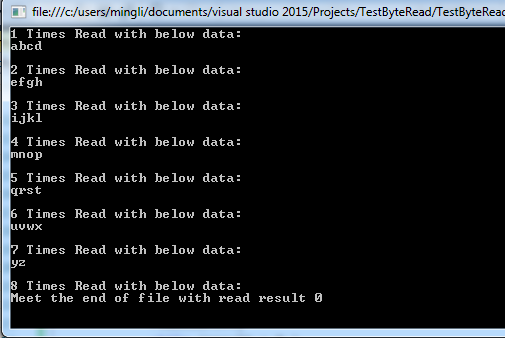
而在Win32定义中，ReadFile 只会返回False（0）和True（NonZero,其实是读到的字节数）

### End of File 总是单独读的，只有所有字节读完了才会遇到end of file

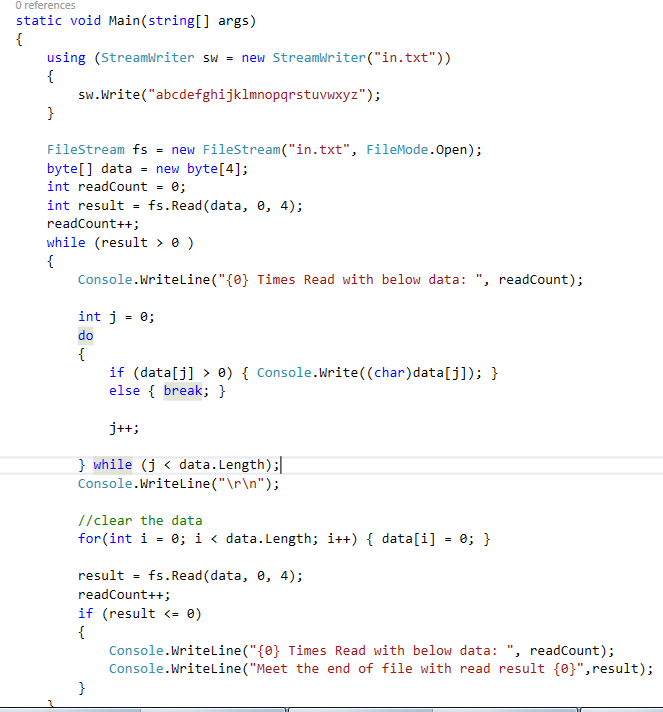
[end of file.cs](IO/end%20of%20file.cs)

以读26个英文字母为例，如果一次读4个字符，一共需要读8次，第7次其实只返回两个字符。第八次读时会遇到end of file



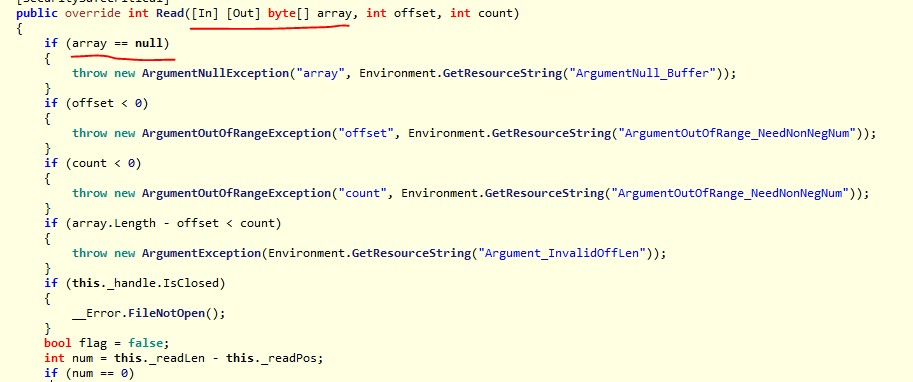
而不是第7次读的时候，读两个字符，然后遇到end of file

End of file 绝对不会和有效的字符混在一块返回。所以，要么读到字符，哪怕是一个byte，要么遇到end of file



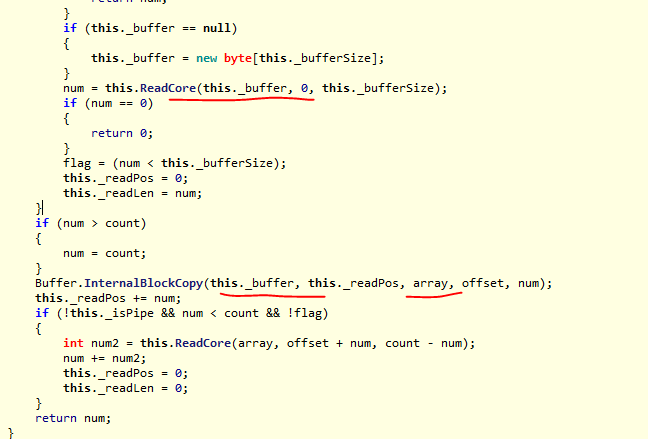
## Read 的时候，Byte[] 即使入参，也是出参

所以一定要初始化

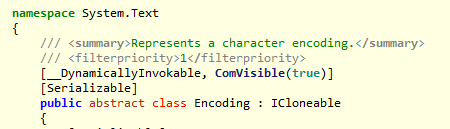


## Read 自己有一个Buffer

以FileStream的Read 为例，其实有一种场景是先把内容read 到自己的Buffer内部，然后再Copy到传递进来的Byte[]中，而不是一开始就直接使用传递进来的Byte[]

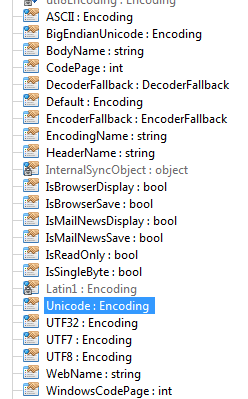
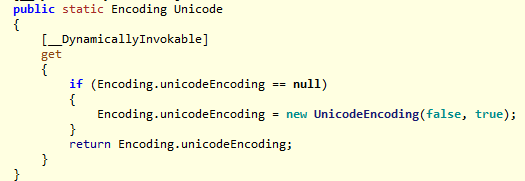
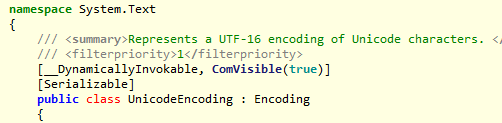


## Encoding 是类而不是枚举



### ASCII， UTF8等是Encoding类的static 成员

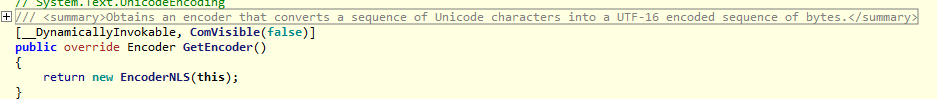
其本身也是一个Encoding 类

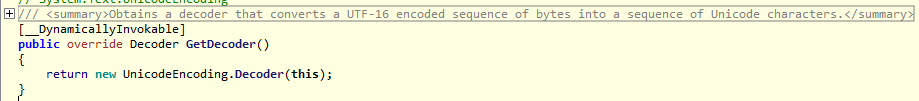
  

### Encoding 类都有一个相关的Encoder 和Decoder类

Encoder 是把字符序列转换为byte 序列，Decoder 是把Byte序列转换为字符序列







# 关键类和结构

## Convert