[1. 关键字 1](#_Toc463351390)

[1.1. ReadOnly 1](#_Toc463351391)

[1.1-1. 定义 1](#_Toc463351392)

[1.1-2. 是字段才有的关键字，属性不能使用 1](#_Toc463351393)

[1.1-3. 只读字段只能在构造函数或者变量声明时赋值 1](#_Toc463351394)

[1.1-4. ReadOnly Field 也可以赋值很多次 1](#_Toc463351395)

[1.1-5. 用readonly 标识的集合仍可以添加和修改元素 2](#_Toc463351396)

[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](#_Toc463351397)

[1.1-7. 针对struct， 内嵌的属性自然是不能更改的 3](#_Toc463351398)

[1.1-8. No readonly reference instance 4](#_Toc463351399)

[1.1-9. ReadOnly和Const 区别 5](#_Toc463351400)

[2. Class 5](#_Toc463351401)

[2.1. 互相引用或者递归引用自身时的堆栈溢出 5](#_Toc463351402)

[2.1-1. Class在构造的时候互联引用，堆栈溢出 5](#_Toc463351403)

[2.1-2. 递归引用自身，堆栈溢出 6](#_Toc463351404)

[2.1-3. 引用自身的第二种写法，仍然堆栈溢出 7](#_Toc463351405)

[2.1-4. 引用自身的正确写法 7](#_Toc463351406)

[3. Struct 7](#_Toc463351407)

[3.1. 定义 7](#_Toc463351408)

[3.2. Struct 可以实现interface 8](#_Toc463351409)

[3.3. Struct 不允许有Protected 的修饰符 8](#_Toc463351410)

[3.4. Struct 不能显式声明无参的构造函数 9](#_Toc463351411)

[3.5. Struct 不允许实例字段的初始化，static 字段可以 10](#_Toc463351412)

[3.6. Struct 可以不使用new ，但作为局部变量时必须先初始化 10](#_Toc463351413)

[3.7. Struct作为全局变量时，编译器允许不初始化就是用 12](#_Toc463351414)

[3.8. 使用new 关键字生成的struct， 编译器认为其已经初始化过了 12](#_Toc463351415)

[3.9. 不管用不用new， struct 作为值类型都是生成在栈上 12](#_Toc463351416)

[3.10. Struct 作为引用对象的filed 时，struct 各字段不能单独赋值 12](#_Toc463351417)

[3.11. Class 和 struct的区别 15](#_Toc463351418)

[3.11-1. 从使用场景的角度 15](#_Toc463351419)

[3.11-2. 从语法的角度 15](#_Toc463351420)

[4. Interface 16](#_Toc463351421)

[4.1. Interface 中的方法都不需要加public 关键字，也不能加 16](#_Toc463351422)

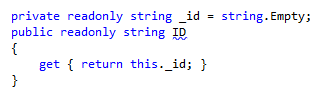
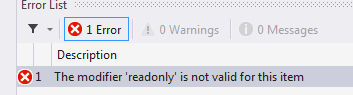
# 关键字

## 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

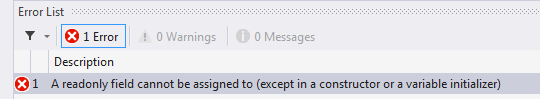
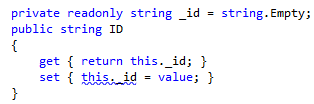
 

因为属性本质上是方法，方法谈不上只读还是可读写。属性是对字段（Field）的封装，只有字段，也就是数据才有只读，可读写的概念。

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)



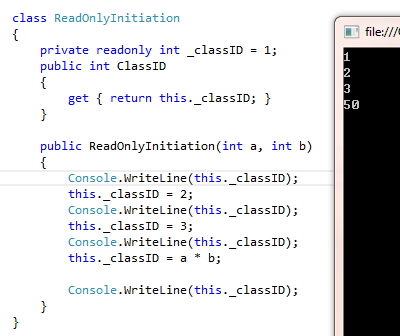
所以只读字段对应 的属性也不能有set方法

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

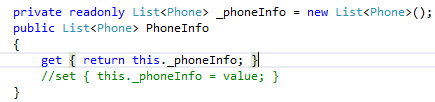
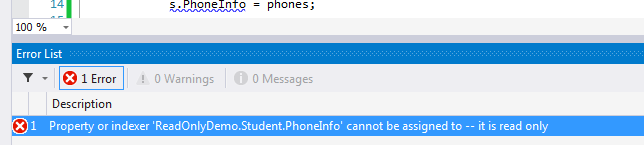
### ReadOnly Field 也可以赋值很多次

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

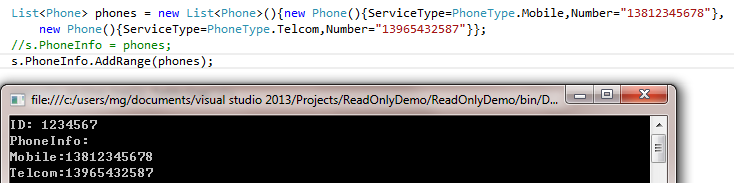


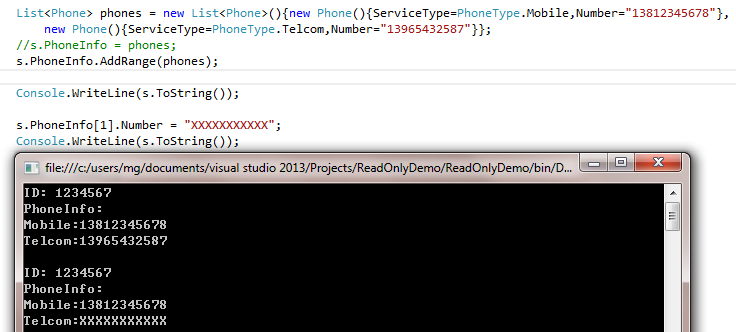
### 用readonly 标识的集合仍可以添加和修改元素

给只读集合赋值会报错。

但是给集合添加对象不会出错

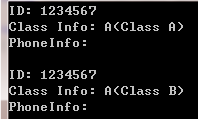
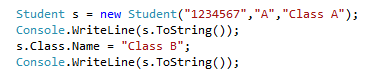
还可以改变集合里面每个元素的属性



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

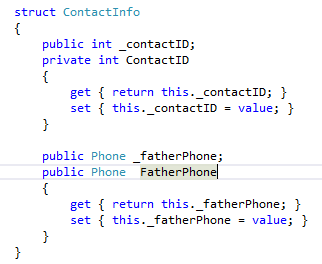
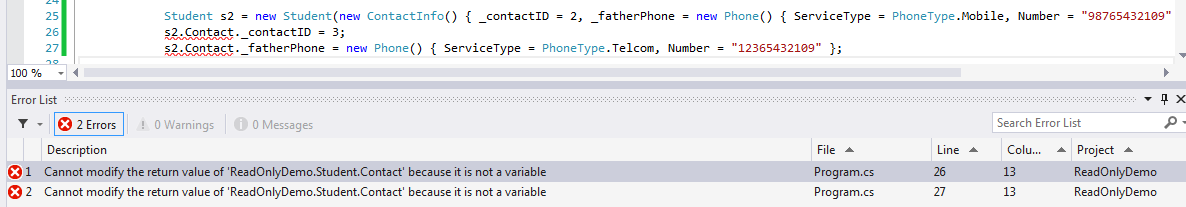
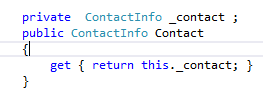


属性里面的内容仍然可以修改

### 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）

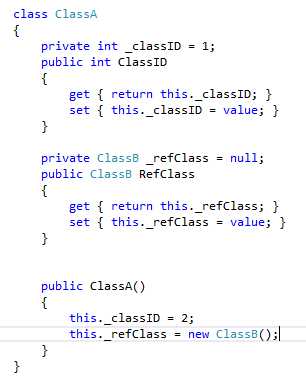
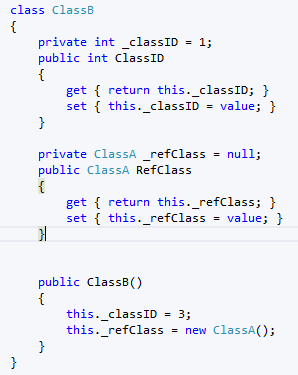
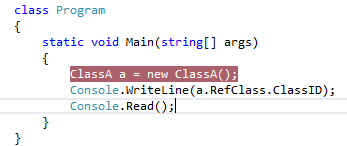
比如系统开始运行的时刻：在系统一经运行开始就不能再更改，但是又不是一个编译时可以知道的常量。

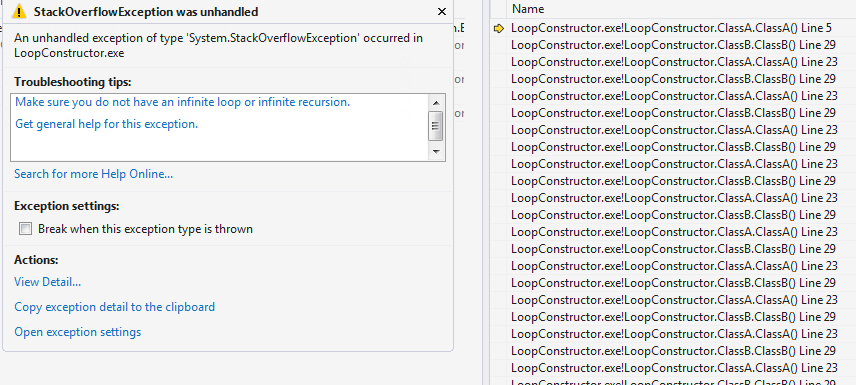


# Class

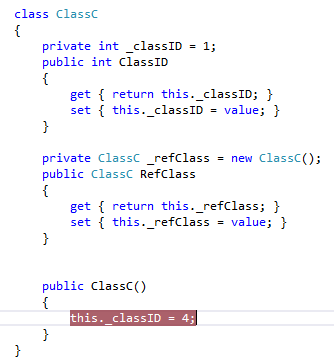
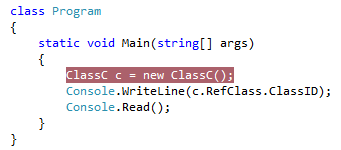
## 互相引用或者递归引用自身时的堆栈溢出

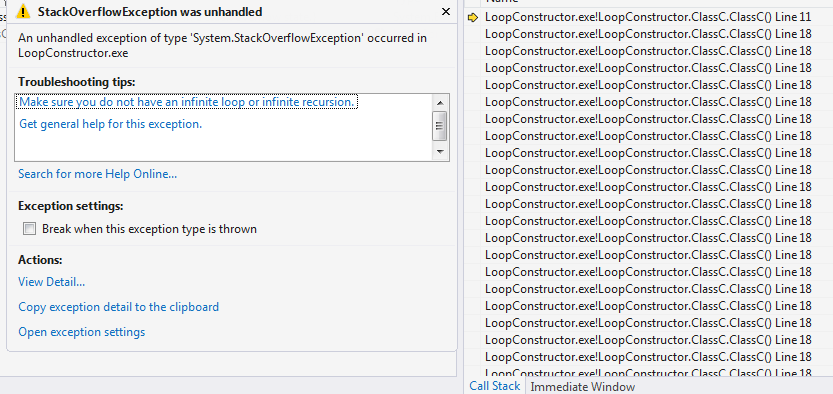
### Class在构造的时候互联引用，堆栈溢出

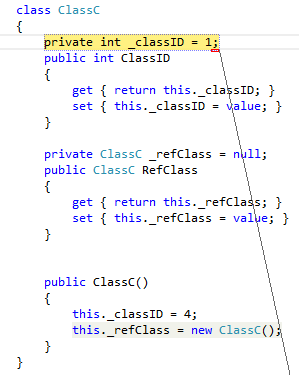
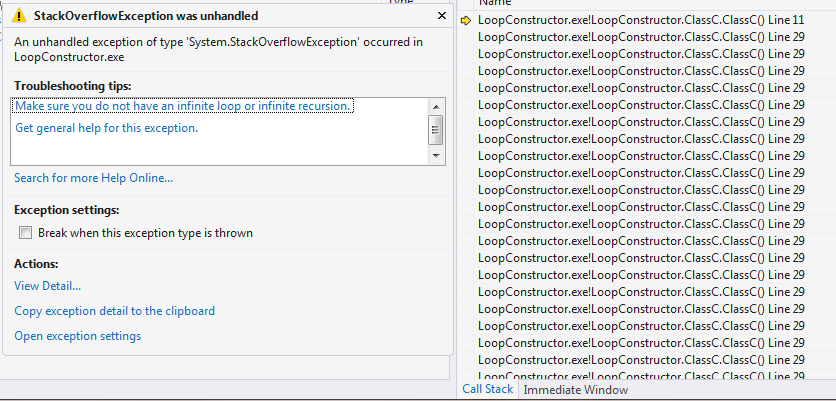


### 递归引用自身，堆栈溢出

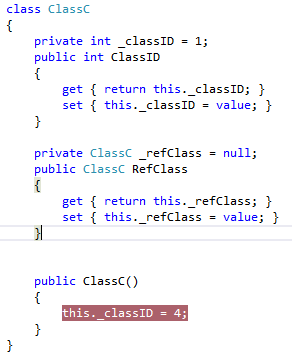
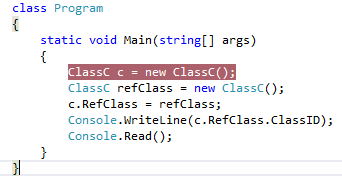
 



### 引用自身的第二种写法，仍然堆栈溢出

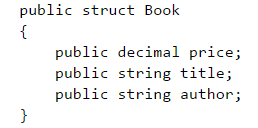
### 引用自身的正确写法

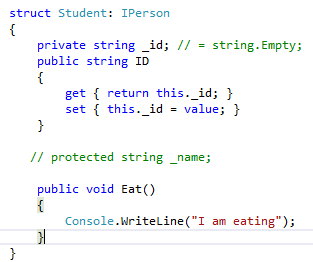
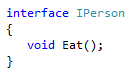
# Struct

## 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。

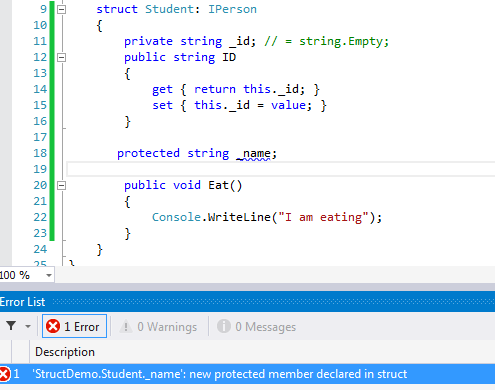


## Struct 可以实现interface

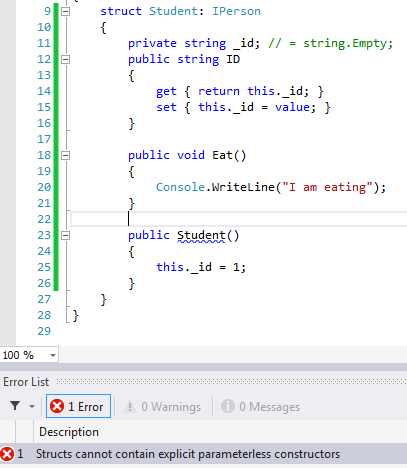
 

## Struct 不允许有Protected 的修饰符

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。

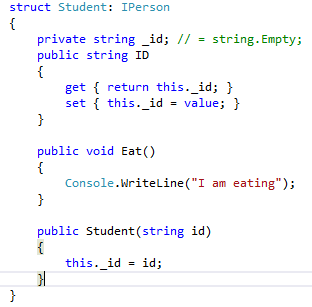


## Struct 不能显式声明无参的构造函数

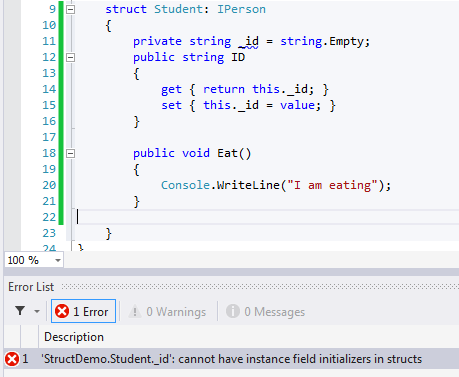


编译器保证struct 必定有一个无参的构造函数（应该是当不使用new 关键字生成struct时调用的）。在这个构造函数里，所有的值类型被初始化为默认值，所有的引用类型被初始化为null。

带参数的构造函数是允许的

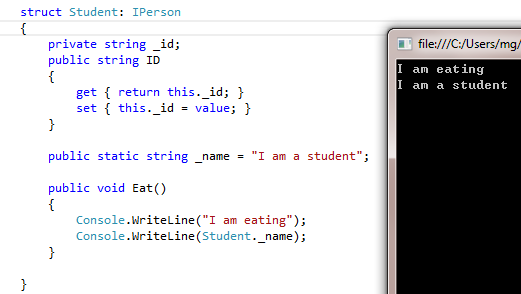


## Struct 不允许实例字段的初始化，static 字段可以



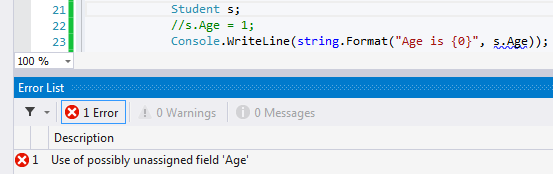
所有的初始化都在默认构造函数里自动做

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.

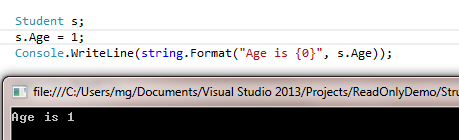


## Struct 可以不使用new ，但作为局部变量时必须先初始化

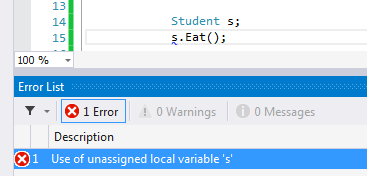
否则会报未初始化错误，虽然其实编译器会调用struct的默认构造函数进行初始化。



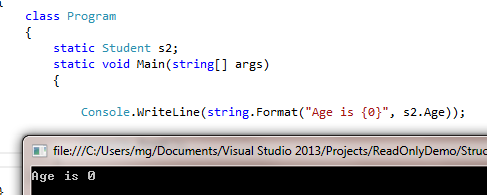
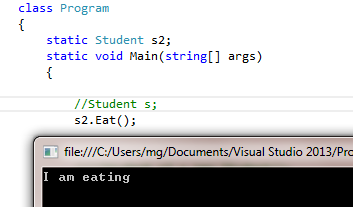
在使用时先赋一下值就可以了



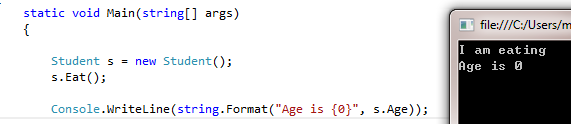
也不允许调用方法， 此时只能用new 关键字了



## 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



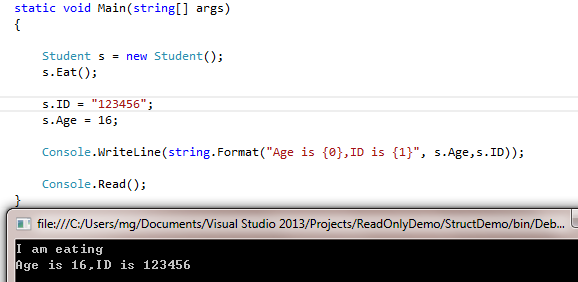
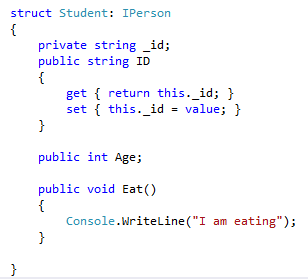
## 不管用不用new， struct 作为值类型都是生成在栈上

当然这是单独使用struct的情况。

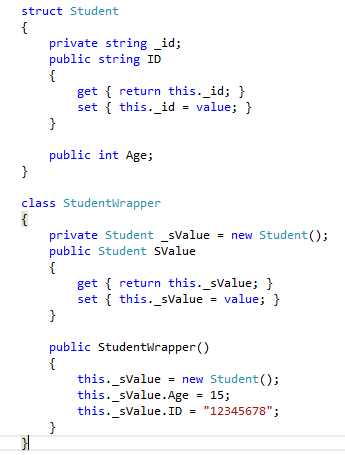
如果struct 作为别的一个引用类型的字段，那么它的值是生成在堆上的。

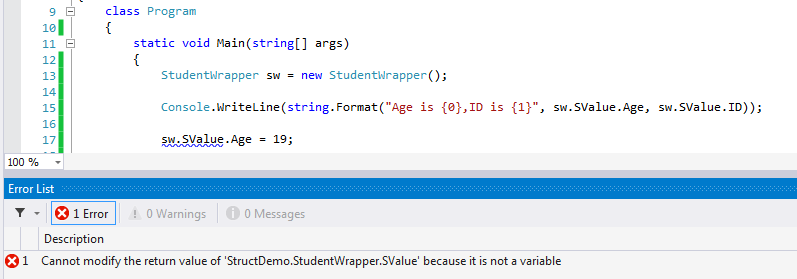
## Struct 作为引用对象的filed 时，struct 各字段不能单独赋值

当struct 单独使用时，其每个字段是可以单独赋值的。

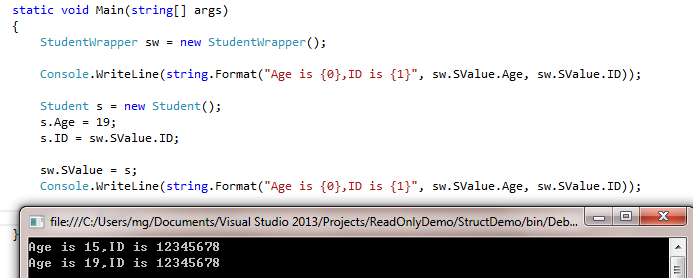
 

但是当struct 作为引用对象的field 时，就不能给struct的各个字段单位赋值了。 这是由于.net 的boxing-unboxing机制造成的。由于student是一个值类型，sw.SValue是在栈上复制了一份sw 的sValue的完整拷贝。此时sw.SValue 和 sw在堆中的sValue 的值已经完全没有关系了。直接读是没问题的，但当试图修改时，显然修改不能体现在堆上，所以编译器在编译阶段就直接出错了。



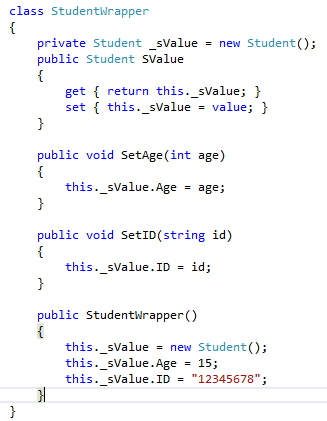


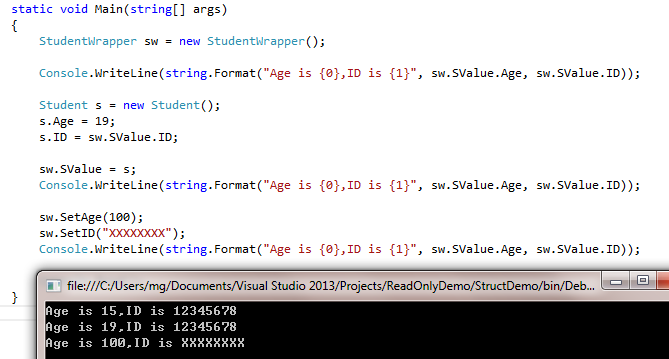
解决方案一： 整体赋值



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

