无参属性

使用属性的好处：更好地遵从了数据封装的原则，不允许从外部修改字段（设为private），只可以通过属性读取和修改。

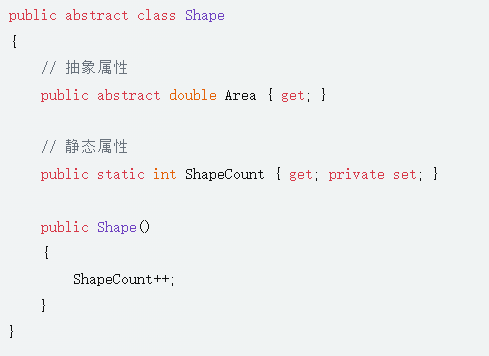
使用无参属性的规则：

1.每个属性都有名称和类型（类型不能的void）。

2.属性不能重载。

3.属性可以同时指定get和set方法，但是可以省略get或set来定义只读属性或者只写属性

关于实例属性、静态属性、抽象属性、虚属性的例子：



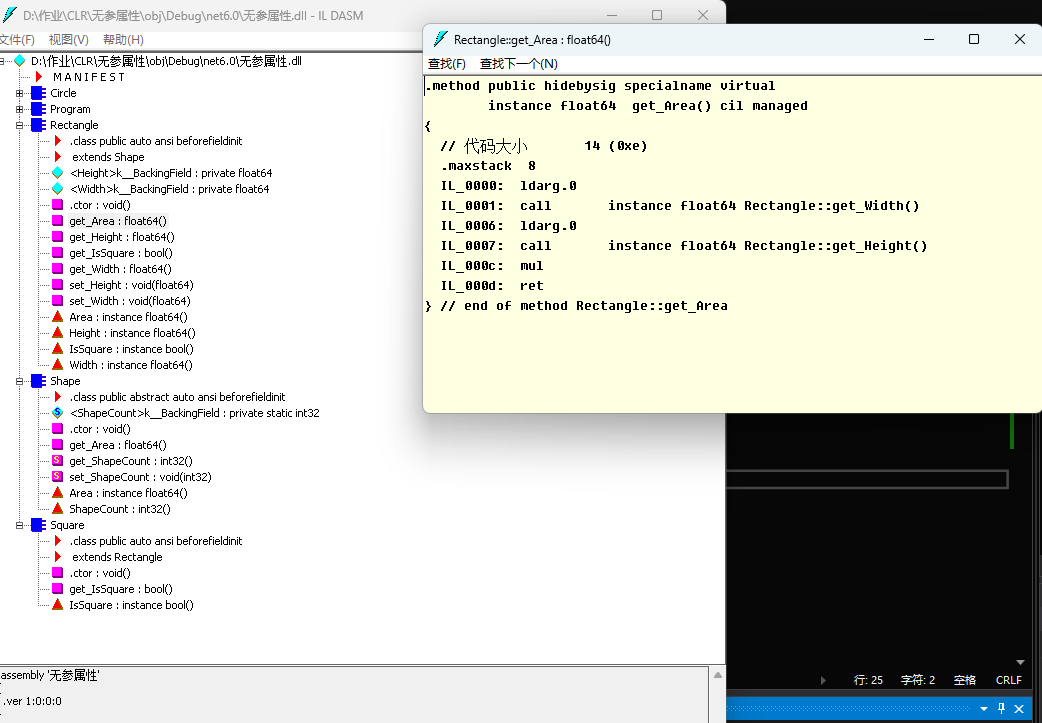
图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

Ps：属性实际上是在调用方法，以下是上面例子的IL代码



可以见到get变为了以get\_为前缀的方法，set也是一样

自动实现的属性，声明属性而不提供get/set的实现，C#会自动声明一个私有字段

下面是例子：



Ps：1.使用AIP，属性必然是可读可写的 2.一个属性中get和set必须同时使用AIP实现或者同时手动实现

AIP（自动实现属性）的一些缺陷：

1.必须在每个构造器方法中显式初始化每个AIP

2.AIP的支持字段名称由编译器决定，每次重新编译代码都可能更改这个名称

3.调试时无法在AIP的get和set方法上设置断点。而手动实现的属性可以。

如何合理地定义属性（与字段相比）？

1.如果定义属性最好同时提供get和set方法

2.属性方法可能抛出异常而字段访问不会

3.属性不能作为out或ref参数传给方法，而字段可以

4.属性方法执行可能花费较长事件（因为属性的get和set实际是方法），而字段访问总是立即完成

5.连续多次调用属性，属性方法每次都可能返回不同的值（这是为什么？）

6.属性方法可能造成一些副作用，而字段不会。比如当定义一个类型时，应确保无论以什么顺序设置属性，都不会影响类型的行为；比如带来额外的性能消耗

7.属性方法可能消耗额外的内存。

属性额外逻辑造成副作用性能消耗的例子：



执行了额外的判断逻辑和可能存在的抛出异常逻辑

属性定义的顺序影响结果的例子：

图形用户界面, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

连续调用一个属性，属性方法都可能返回不同值的例子：



同理用属性方法返回DeltaTime.Now每次也会返回不同的值，这是在字段上不会发生的。

对象和集合初始化器

存在的意义：可以在构造一个对象的同时设置对象的一些public属性或字段。

例子：





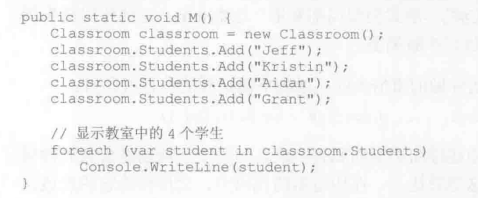
如果属性的类实现了IEnumerable或者IEnumerable<T>接口，属性就被认为是集合，而集合的初始化是相加操作（前提是提供了Add即相加的方法）

例子

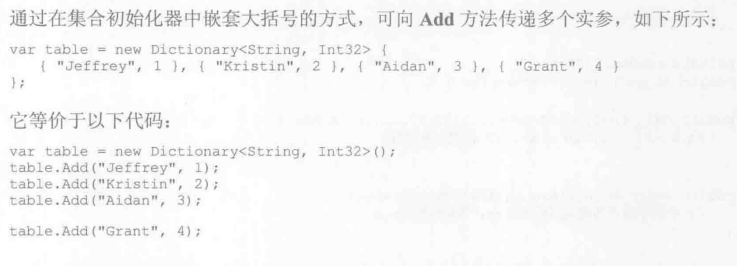


这样以来，初始化student属性是大括号里的元素就可以不断往后添加进列表

实际上这个初始化student代码会被编译器变成这样



而如果有的集合的Add需要获取多个实参，就需要嵌套大括号，例子如下：



匿名类型

匿名类型以及初始化的例子，书P209



C#编译器会通过等号右边的表达式推测类型（可以类比隐式类型局部变量var）

编译器支持用另外两种语法声明匿名函数中的属性，能根据变量推断属性名和类型：

文本

描述已自动生成

报纸上的文字

描述已自动生成

Ps：如果源代码中定义了多个匿名类型，而且这些类型具有相同的结构，那么只会创建一个匿名类型定义，但创建该类型的多个实例。

方法不能返回对匿名类型的引用（匿名类型在编译时会被赋予一个临时名称，但我们并不能获得这个每次编译都发生改变的名称）。

System.Tuple类型

元组类型被创建好之后就不可变了（所有属性只读）



文本

描述已自动生成

文本

描述已自动生成

有参属性

有参属性即索引器，它的get访问器方法接收一个或多个参数，set访问器方法接收两个或多个参数

定义有参属性需要在类型后加上this，后面的方括号“[ ]”内为参数。C#中索引器没有自己的名字，编译器生成代码时也不会按照名字区分索引器

索引器的例子：



本例中value为一个隐藏参数，代表想赋给“被索引元素”的新值或键值对中的值，方括号中是索引或者键值对中的键

有参属性的get访问器方法在生成IL代码时会生成以get\_Item为前缀的方法，set方法同理。

如果一个类型提供了一个Item属性，那么该类型就提供了索引器。

可以通过System.Runtime.CompilerServices.IndexerNameAttribute这个定制特性来重命名生成的访问器方法的名字，把Item替换成提供给这个定制特性的字符串

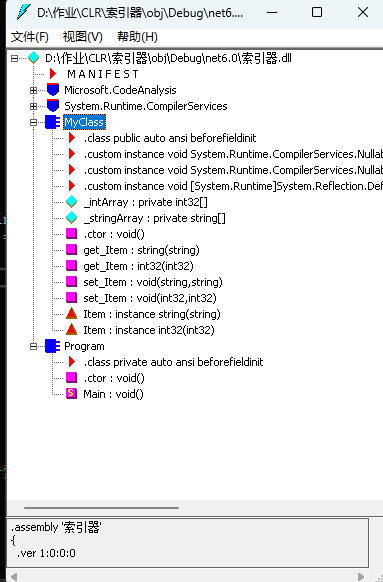
IndexerName特性允许定义多个相同签名的索引器，但C#不允许这么做，因为C#中不是通过名称来引用索引器（它是使用参数类型来确定访问的是哪个索引器），编译器不知道引用的是哪个索引器

例子



文本

描述已自动生成



通过Item:instance string(string) 和Item:instance Int32(Int32)可知，以及IL\_000a和IL\_001b，是通过参数类型来确定索引器的

调用属性访问器方法时的性能

对于简单的get和set访问器方法（例如AIP的get和set），JIT编译器会将代码内联，因此这种情况下使用属性不会造成性能上的损失（而属性如果执行了其他的逻辑就会造成性能的损失）。

属性访问器的可访问性

定义属性时如果两个访问器方法需要不同的访问性，C#要求必须为属性本身限制最小的访问性，两个访问器 只能选择一个来使用较大的（另一个只能使用和属性本身一样的修饰符）。

例子：



Set方法使用了限制更大的private，则get方法只能使用和属性本身一样的public

泛型属性访问器方法

C#不允许这么做。原因是

