# 9.0

# 一、.NET的体系结构

## 1.JIT编译器，提高性能

JIT编辑器并不是编译整个程序，而是只编译运行部分的代码。

## 2.IL，.net编译语言

IL可以指Intermediate Language，同MSIL(Microsoft Intermediate Language),是将.NET代码转化为机器语言的一个中间语言的缩写。

使用IL不仅支持平台无关性，还支持语言的互操作性。

互操作性是指能将包括C#的任何语言编译成中间语言，如VB，VC++

### 主要特征

a)面向对象和使用接口

b)值类型和引用类型之间的显著差异：

引用类型的实例存储在“托管堆”中，而值类型存储在堆栈中（如果值类型在引用类型中声明为字段，它们内联存储在堆中）

c)强数据类型化：

通用类型系统(CTS)、公共语言规范(CLS)

垃圾回收：.net运行库采用了垃圾回收器，其工作原理是：检查对存储在托管堆上的对象的引用，无引用时将其回收。

进程不能直接访问物理内存，因此一个进程不可能访问分配给另一个进程的内存，也就是一个进程出错不会影响另一个进程。

每个应用程序域大致对应一个应用程序，不同的可执行文件都运行在同一个进程中，数据是可以共享的。

d)使用异常来处理错误

e)使用特性(attribute)

## 3.语言的互操作性

语言互操作性的真正含义是：用一种语言编写的类应能直接与用另一种语言编写的类通信。

1. 用一种语言编写的类应能继承用另一种语言编写的类
2. 一个类应能包含另一个类的实例，而不管两个类是使用什么语言编写的
3. 一个对象应能直接调用用其他语言编写的另一个对象的方法
4. 对象（或对象的引用）应能在方法之间传递
5. 在不同的语言之间调用方法时，应能在调试器中交替调试这些方法调用，即调试不同语言编写的源代码

.NET和中间语言已经实现了这个目标，VS IDE提供了这样的工具

公共语言规范和通用类型系统一起确保语言的互操作性

### 1）VB

VB.NET与C#之间的区别很小，主要是使用花括号还是END语句的个人喜好问题

### 2）VC++

VC++.NET要在.NET framework中运行，只需要在代码开头添加以下命令即可

#using <mscorlib.dll>

### 3）VF#

VF#是一门函数语言，而不是面向对象，有助于用来编写复杂的算法。

### 4）COM和COM+

### 5）Windows运行库

这个运行库可以在VB、C#、C++和JS中使用。它的多样性是通过使用语言投影实现的。

## 4.数据类型

### 1）值类型与引用类型

如果要把自己的类型定义为值类型，就应该把它声明为一个结构。

### 2）预定义类型

整型：sbyte、short、int、long、byte、ushort、uint、ulong

所有的整型都能被赋予十进制或者十六进制的值，后者需要0x前缀

若没有显式的声明，则默认为int，当然也可以直接指定值，如long l=1234L，ulong ul=1234UL。建议使用大写的U和L，否则小写l容易与数字1混淆。

浮点型：double和float，指定float值一般后面加F（f）

decimal：指定值加M

char：除了带单引号的字符字面量外，还可用4位十六进制的Unicode值，带有数据类型转换的整数值或者十六进制数，还可用转义序列

string：字符串中允许包含换行符，在代码中一个字符串可以分两行写，显示的时候也会出现换行

## 5.C#的其他基础

**goto**语句可以跳转到指定行，但不能跳入像for这样的代码块中，也不能跳出try-catch后面的finially。一般不允许使用goto

**Enum.Parse**可以将字符串转化为枚举，通常三个参数：第一个是typeof获得的枚举类型，第二个是字符串，第三个表示是否不区分大小写

**using**可以给命名空间取别名，using 别名=命名空间。当使用别名时，调用不再使用.，而是::出类

**多个Main方法**：csc DoubleMain.cs时会报错，因为分辨不出哪个Main方法为程序入口点。csc DoubleMain.cs /main:Wrox.MathExample即可指定哪个类的Main方法为入口点，从而避免报错。

Main方法的参数可以通过命令行来传送选项

**占位符{n}**：还可以调整文本宽度，{n,w}其中w就是文本宽度。另外在宽度后面使用某种格式字符串，即可实现可选精度

**global关键字**：如果直接写命名空间点出类，那么程序会就近选择匹配的命名空间，而若是global::命名空间点出类，那么程序将从全局开始搜索匹配的命名空间。

**内部注释**：即/\*\*/可以用在某句代码内部。

**预处理指令**：#define（定义类似于VS点开项目属性中的条件编译符号）、if#elif#endif、#warning编译到此句会提示但继续编译、#error编译到此句将抛出异常、#region#endregion、#line、#pragma

**标识符**：如果需要把一个保留字用作标识符，那么可以在标识符的前面加上前缀符号@。标识符也可以包含unicode字符，用语法\uXXXX来指定，如\_test与\u005ftest可以互换，因为\_的Unicode代码就是005f

## 6.csc命令

#### 暂时没有看懂

## 7.C#的面向对象

### 1）类与结构

类是存在堆中的引用类型，而结构是存在栈上的值类型。

类中有个和构造函数差不多的东西，成为终结器，其名称也与类名一致，只是前面多了个~，当有某个对象不需要时调用它。

#### A）数据成员

按引用传递效率更高，因为在按值传递时，必须复制大量的数据。

#### B）函数成员

命名参数：void test(string a，string b)参时可以test(b:””,a:””），主要目的是使用在可选参数中。

#### 属性的访问性，暂时看不懂。

C#可以给类编写无参数的静态构造函数。只执行一次。而且无参数的实例构造函数和静态构造函数可以在同一个类中同时定义！

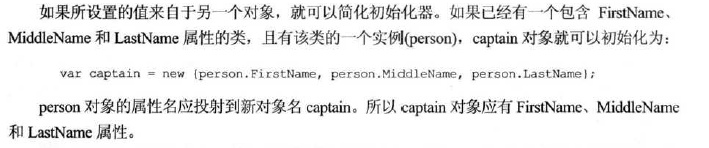
#### C）只读字段

readonly比const灵活，其规则是只可以在构造函数中给只读字段赋值，且不一定要是静态字段。

### 2）匿名类型

var不止能用于弱类型，还能用作隐式类型化的变量。

#### 没用过，有点难理解，需要用一下才知道。



### 3）结构

结构是值类型，不是引用类型。存储在栈中或者存储为内联（如果它们是存储在堆中的另一个对象的一部分。内联函数是指用inline关键字修饰的函数。在类内定义的函数被默认成内联函数。暂时理解为，在类中定义的结构）。

结构的性能与类相比，有正面也有负面，正面是分配内存与释放内存都比较快，负面是传递时是整个复制，而不是指传递引用。当然，结构作为参数传递时，使用ref，也是传递引用

定义结构对象，可以new，也可以不new。因为这里的new仅仅是调用构造函数，不分配堆内存。

结构不允许自定义无参数的构造方法，且结构中实例字段不能初始化

### 4）弱引用

弱引用需要使用WeakReference类来创建，并且需要先判断对象是否存在（属性IsAlive）。其属性Target会返回一个强引用，不为null则可用。

### 5）部分类partial

### 6）Object类

扩展方法：需要先创建一个静态类和一个静态方法。此静态方法要作扩展方法，第一个参数是要扩展的类型，放在this关键字的后面。

如果类中已经有与扩展方法同名的方法，则实例方法优先调用

## 8.继承

结构不能实现继承，但可以多重接口继承。

### virtual虚方法：

重写只对类中的实例函数成员有意义。

### 隐藏方法：

派生类和基类中有同签名的方法，但不是虚方法，那么基类的此方法会被隐藏。如果需要使派生类的此方法隐藏，则需要在派生类的此方法的定义前加上new关键字。

### 抽象类（abstract）：

不能实例化、不能有方法实现、必须在派生类中实现方法。

### 密封类（sealed）：

不能被继承。



this指的是当前类，base指的基类

## 9.泛型

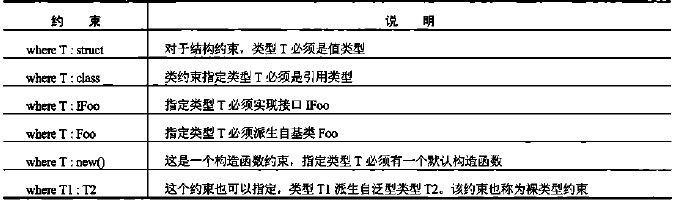
泛型的四大主题：默认值、约束、继承、静态成员

### Default关键字：

把null赋予引用类型，把0赋予值类型

### 约束

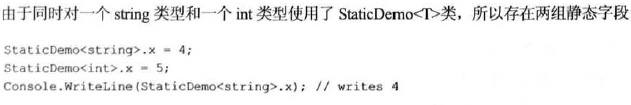
在定义泛型类型后面加上where T:Interface约束



不太明白：只能为默认构造函数定义构造函数约束，不能为其他构造函数定义构造函数约束

### 继承

泛型类可被继承，但是被继承时，必须指定基类的泛型类型。接口如是。



### 静态成员

泛型类的静态成员只能在类的一个实例中共享。

泛型可以在一种语言中定义，在任何其他.NET语言中使用

### 协变与抗变：见下文

## 10.NuGet包

NuGet包是一个.zip文件，其中包含程序集、配置信息和PowerShell脚本。

# 10

# 一、.NET

## 1.编译

### 1）.NET4.6

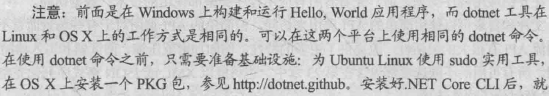
如果想使用csc命令在命令行上编译代码，需要设置某些环境变量：



### 2）.NET Core CLI



此工具可以跨平台使用



## 2.类

### 1）属性

自动实现的只读属性



表达式体属性



### 2）可空类型

在结构后面添加?即可将一个结构设置为可空类型，即这个对象可以被赋予null，如int?

??表示判断前面的变量是否为null，是则得到后面的值，否则取前面的变量值。

## 3.枚举

### 1）枚举与位（看不懂）

可以使用enum类型把多个选项分给一个变量，而不仅仅是一个枚举常量，而分配给常量的值必须是不同的位，Flags属性需要用枚举设置。设置不同的位，也可以结合单个位来包括多个值。



### 2）方法GetNames和GetValues

GetNames方法可以获取枚举所有名称的字符串数组，而GetValues可以获取枚举所有值的此枚举数组，参数均是typeof获得的类型。

## 3.接口



### 还是没有看懂啊，抽象类可以有实现代码的抽象成员？

暂时理解为，抽象类中可以存在非抽象方法。

## 4.资源

### 1）后台内存管理

32位处理器上的每个进程都可以使用4GB的内存。

栈是向下填充的，即从高内存地址向低内存地址填充。而堆上的内存是向上分配的。当引用变量超出作用域时，它会从栈中删除，但引用对象的数据仍保存在堆中，直到程序终止或者被回收。

垃圾回收器使用到了托管堆，只要它释放了能释放的所有对象，就会把其他对象移动回堆的端部，再次形成一个连续的内存块。而且在.net下，较大的对象有自己的托管堆，称为大对象堆。

System.GC.Collect方法可以强迫垃圾回收器运行。GCSetting.LatencyMode可以控制垃圾回收器进行回收的方式。

### 2）非托管资源的处理

析构函数：C#编译器在编译析构函数时，会隐式地把析构函数的代码编译为等价于重写Finalize（父类的方法）方法的代码。

没有析构函数的对象会在垃圾回收器中的一次处理中从内存中删除，但有析构函数的对象需要两次处理才能销毁。因此在C#中建议使用System.Idisposable接口替代析构函数。

调用GC.SuppressFinalize方法告诉垃圾回收器有一个类不再需要调用其析构函数，意味着垃圾回收器任务这个对象没有析构函数。

### 3）不安全的代码

用unsafe关键字编写不安全的代码，用于使用指针。

可以在变量声明前添加（仅限全局变量），也可以在类或结构定义前添加，还可以在某段代码前添加并把代码写入其后花括号。

#### 指针：

在变量声明时，\*表示声明一个指针，就是存储特定类型的变量的地址

变量声明后，&表示取地址（寻址运算符），\*表示获取地址的内容（间接寻址运算符，有时是取消引用运算符）

为避免垃圾回收器处理数据类型时出现问题，指针只能声明为非托管类型，即指针不能指向任何引用类型，可以理解为不能指向类。

要得到指针指向的地址，只需要做一次强制转换。

checked关键字不能用于涉及指针的转换。

如果要维护一个指针，但不希望指定它指向的数据类型，就可以为指针声明为void：void\* pTest

我的理解：指针算术的运算时，如加法运算，指针的值是等于原值加上某个数和指针指向的类型所占的字节数的乘积。

sizeof可以得到某个数据类型所占的字节数。

通过指针访问结构的成员，可以在指针前加\*取到值然后直接点出，也可以使用指针成员访问运算符->

然而，可能需要指针访问类对象中的某个结构属性，又不能直接声明指针指向该结构属性（因为它包含在类对象中），此时需要使用到fixed关键字，语法是：fixed(声明块){使用块}，若要声明多个指针可多个fixed()但只提供一个{}。另外fixed也可以嵌套。

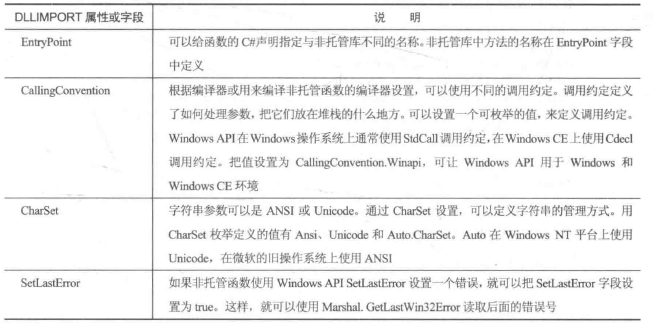
##### 使用指针优化性能

为了创建一个高性能的数组，需要使用另一个关键字：stackalloc。stackalloc命令指示.net运行库在栈上分配一定量的内存。如double数组的指针声明：double\* pDoubles=stackalloc double[10];调用时\*pDoubles指向的是第一个内存块的值，即数组的第一个元素，要获取或设置第X个值则要这么写\*(pDoubles+X)，也可以写成PDoubles[X]。

**注意**：在普通数组中超出下标范围就会抛异常，而在指针中就不会。如上式pDoubles[20]这个元素的20已超出界限10，它不会抛异常，因为pDoubles是一个指针地址，也就是说pDoubles[20]依然是指向一个地址，但是这个地址已经不在自定义的数组内存范围内，而是指向了一个不知名数组（是否存储其他数据未知），可能导致篡改了重要数据。

### 4）平台调用

不理解：C++有不同的Boolean数据类型（如本机bool和Windows定义的BOOL有不同的值），需要特性[MarshalAs]指定.NET类型bool应该映射为哪个本机类型，语法：在DllImport后加一行[return:MarshalAs(UnmanagedType.Bool)]



## 5.泛型（续）

### 1）协变与抗变（完全没搞懂）

没搞懂：参数类型是协变的（理解协变为派生类对象可以赋予基类）。方法的返回类型是抗变的（理解抗变为基类对象不一定可以赋予派生类）。（明明都是协变的）

#### 泛型接口的协变

使用out关键字，返回类型只能是T？

#### 泛型接口的抗变（逆变）

使用in关键字，只能把泛型类型T用作方法的输入？

第二次理解：泛型没有继承，而泛型的类型有继承。

out是协变，指子类的泛型集合可以赋值在父类的泛型集合，此时访问时只能访问到父类的元素，所以是很安全的，即协变。协变一般用在方法的返回值。

In是抗变，逻辑与协变相反。按照常理来说，协变是安全的，抗变是不安全的。但抗变其实也是为了安全，本质还是协变（因为，抗变先定义一个委托方法，方法中使用的还是父类的东西，所以即使后面赋值给一个子类，子类接收的参数最后还是调用父类的继承方法）。

因此，传入参数的时候才是逆变，返回值都是协变。