第一章

CLR是公共语言运行时，这个东西，在程序运行的时候负责程序的内存管理、垃圾回收、等等。

CIL是Assembly里面的程序的中间代码，和本机的代码无关。

包括了三大部分

程序集Assembly 里面是公共中间语言 CIL

公共语言基础结构 CLI——包括了CTS 和 CLS 是公共语言的规范

CLR公共语言运行库——包括了GC垃圾收集、JIT实时编译器、BCL基础类库或者叫做框架类库（为了程序编写提供了很多的方法和工具）。

第一章主要是介绍了DONET框架的情况。

第三章首先说C#是一组类型声明

对于EXE，类型声明要包含一个Main类

命名空间把相关的类型声明分组并且命名。

类型是一个用来创建数据结构的模板，模板并不是数据结构，但是详细说明了这个模板构造的对象的特征。

类型——一个标识符（名称）、结构（也是分配的空间）、行为（比如可以表示多大的数据）。

Int 这种都是类型模板，而最终要实际应用，还要创建实例化的对象。

类型包括了内置类型，object，也就是内置类型+类对象。

通过实例化一个类型，创建这个类型的实例化对象。——只要是使用new运算符的，应该说就是这种方式得到的结果。

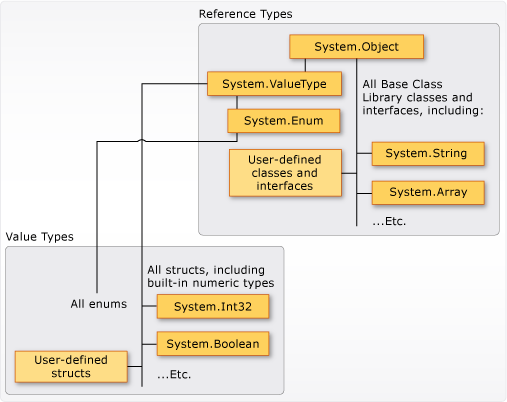
类型可以存储不同的数据项，简单的内置类型仅仅能存储一个数据项，而复杂数据类型或者是自定义的数据类型可以存储多个同类型的或者不同类型的数据项。这些东西，可以通过索引的方式实现检索。

上面的都是简单的数据类型，复杂的类型模板，比如说是类，是可以存储不同的数据项的，这些数据项都被称为是成员。这些成员的使用方式不是数组那样的下标索引方式，而是更加复杂的名称索引方式。类里面包含了两种成员，数据成员（数据）和函数成员（类型的行为）。

预定义类型

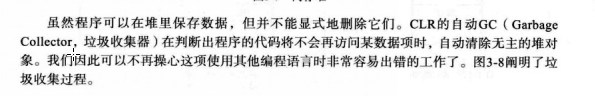
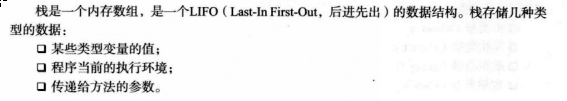
11种数值类型——有没有符号、浮点数3种，unicode字符类型char，bool

3种非简单类型——string object dynamtic



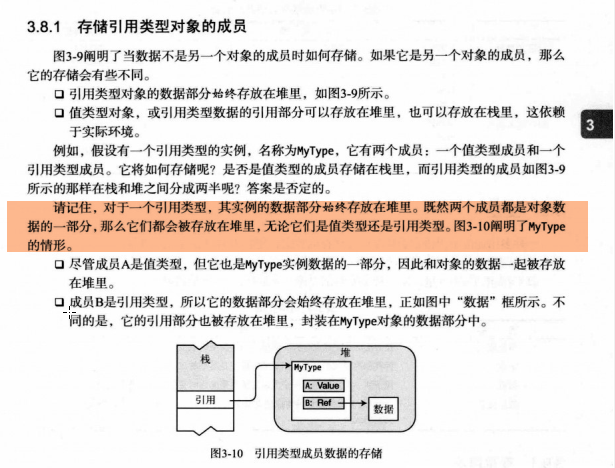
第三章主要是说明了C#语言中存在的类型，类型自己是存在着特征的。然后C#类型和.NET类型也是存在着紧密的关系的。

类型除了上面的DONET内置类型，还有用户自定义Class Struct Array Enum Delegate Interface类型。

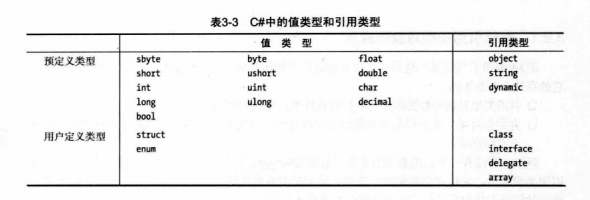


上面是从DoNet框架以及C#语言的角度来对于类型进行分类，我们还有一种角度就是从值类型和引用类型的角度对于数据项的类型进行定义。类型决定了对象在内存中的位置。

值类型和引用类型主要是自己的存储位置存在着差别。



也就是说引用类型永远是两个内存，也许一部分在栈里面，也许全部都是在堆里面，这种全部都在heap里面的情况，也是分两个部分在heap里面进行存储的。



之所以要引入类型，主要是不同的类型存在不同的动作和不同的数据项。一个值类型，也许自身仅仅能存储一个或者几个同类型的数据项，而一个引用类型，也许可以存储多个不同类型·的数据项，而且还顺带了这些数据项的方法。

类型、变量之间的差别

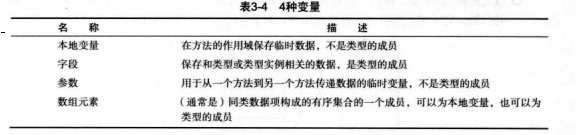
除了类型之外，还存在着一种东西就是变量，

变量允许程序存取数据？？

* 本地变量——在方法的作用域保存临时数据，不是类型的成员。

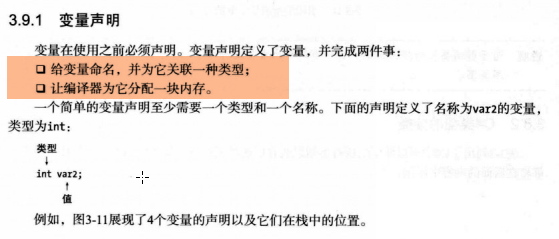
也就是临时变量，形式参数。方法声明时候的形式参数列表。

* 字段——保存和类型或者类型实例相关的数据，是类型的成员。
* 参数——从一个方法到另外一个方法传递数据的临时变量，不是类型的成员。
* 数组元素？？？



莫非是说——数据类型，然后在另外一个领域存在着变量这种东西？？？

方法的调用有两种方式，一个是直接调用，一个是声明为函数指针的委托的方式，然后进行调用，也就是用invoke，使用委托进行间接调用。



类和类对象的声明，一个是类成员的声明和初始化所对应的内存硬件的本质。

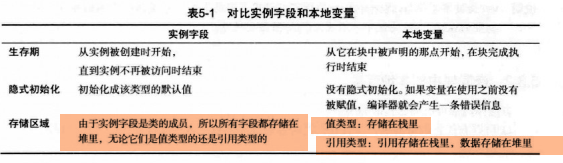
类的声明是一个引用操作，而一个类的初始化是在heap里面的，对于数据项和方法的内存的分配和初始化。也就是说，初始化或者是创建实例是分配内存的操作。

还是强调一下：注意类型和变量之间的无关性。

在C#中，同样 存在变量的生存期和作用域这两个东西。

本地变量——创建的目的是为了保存本地的临时数据。

生存期——函数块部分。



注意代码编译的过程，首先是使用编译器，然后是使用链接器，链接器对于代码的实际内容的要求更加的细致。

注意break是跳出当前的循环，然后continue是跳到当前循环的底部。也就是进行判断，是不是要进行下一步的循环。

方法体调用别的方法——主要是使用的·invoke。

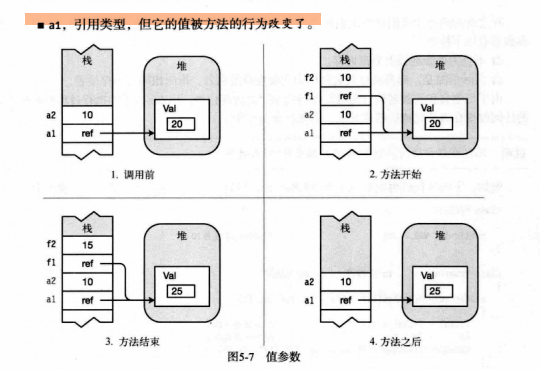
类型 产生并且定义一个变量 而变量和参数之间又是什么关系？？？？

* 形式参数 就是声明在方法的参数列表中的本地变量之一。
* 变量自己有类型和名称，可以被写入和读取。

形式参数和实际参数在运行的时候，可以体现出值类型参数和引用类型参数的不同之处。

值类型的参数，传递给形式参数的是一个副本，而引用参数传递给形式参数的是一个引用的副本，引用的副本指向的位置和原来一样，所以，最后的结果会修改原来的数值。

所以说，希望使用方法修改一个引用类型变量中的东西的时候比如你想修改一个类中的变量，无论这个变量是值类型还是引用类型，都是将整个类传递进去作为引用参数。



参数的主要作用是用来传进或者传出数据。Parameter。

参数的分类——值参数——将实参的值复制到形参的方法把数据传递给方法，不管是int还是MyClass，都是这种值参数。应该说是

* 值类型的实参传递给值参数，复制值类型的副本，不改变原来的值。
* 引用类型的实参传递给值参数，传递的是引用的副本，形式参数的实例化对象会通过这个副本改变实际参数的值。

形参和实参的差别就是，在方法声明时候出现的参数是形式参数。在方法进行实例化的时候出现的参数是实际参数。

实际参数和形式参数在方法执行不同阶段的值。

是不是说，这个参数主要是指的是方法的参数列表呢？

使用引用类型的参数，在方法的声明和方法的调用的时候，都需要使用这个ref关键字。

引用参数，无论你是传递值类型的变量还是传递引用类型的变量都是会直接传递引用，也就是会更改原来的方法的数值。

5.9 参数

形式参数——声明在方法的参数列表中。

形式参数是变量，有自己的类型和名称，并且可以被读取和写入。

实际参数——位于方法调用时候的参数列表中。

* 值参数
* 引用参数
* 值参数传递值类型和引用类型的不同
* 引用参数的声明和调用时候都要使用ref
* 引用参数传递值类型和引用类型的不同



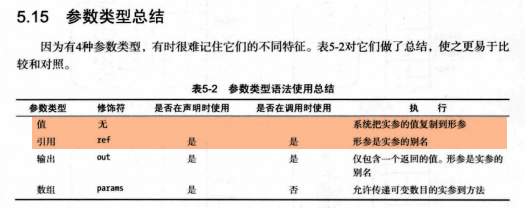
引用类型对象——指的是说一个引用类型的实例化对象，比如说是class对象。

可以将引用类型的对象作为值参数或者引用参数进行传递。

形式参数就是在方法声明时候，出现在参数列表的对象，实际参数就是在方法进行实例化的时候，出现在方法参数列表的对象。

引用类型作为值参数——方法结束后，实际参数指向自己原来的对象。方法所操作的东西都是形式参数所操作的东西。

引用类型作为引用参数——方法结束之后，实际参数指向的是方法内创建的新对象。



类型 进行实例化之后产生对象，对象应该是变量的别名，变量可以作为参数传递到方法中。