## 1.2什么是.NET Framework

C#编写的程序需要编译和运行的，.NET Framework就是应用程序运行时的执行环境，为应用程序提供以下几种服务。

* 全面的类库 .NET Framework提供了丰富的类库以供调用，这使我们不必再去编写大量的代码去处理常见的操作。
* 内存管理 在许多其他的编程语言中，开发人员需要负责内存的分配和释放，以及对象生命周期的处理等任务。.NET Framework为了减轻开发人员的工作，提供了内存管理服务。
* 通用类型系统（CTS） CTS定义了可以在中间语言中使用的预定义数据类型。
* 开发结构和技术 .NET Framework提供了开发特定应用程序所需要的库，如Web应用程序的ASP.NET技术。
* 语言互操作性 面向.NET Fremework的语言编译器都提供了生成中间语言代码的机制，这种机制使得不同语言之间进行互操作成为可能。

### 1.2.1 .NET Framework的组成

.NET Framework包括公共语言进行时（CLR）和.NET Framework类库（FCL）。

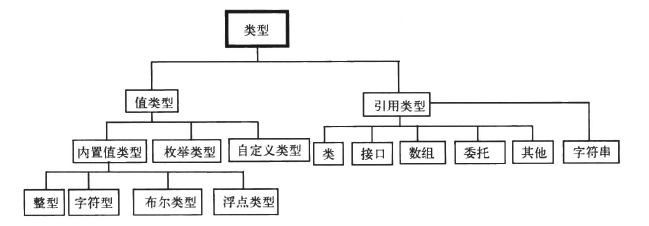
#### 1.CLR

公共语言进行时是.NET Framework的核心基础。我们可以将CLR看成一个在执行时管理代码的代理，它提供了内存管理、线程管理和异常处理等服务，而且还负责对代码实施严格的类型安全检查，保证代码的正确性。我们将手CLR管理的代码成为托管代码，将不受CLR管理的代码成为非托管代码。

* CTS

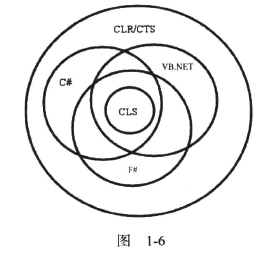
CTS用于解决不用语言之间数据类型不同的问题，如C#中整形是int，而VB.NET中的整形确实Integer，通过CTS可以吧这两个类型变成通用类型Int32。所有.NET语言都共享CTS这一类型系统，从而使他们之间能够实现无缝互操作。

CTS类型主要分为两大类：引用类型和值类型。



* CLS

CLS是一种最底的语言标准，它制定了以.NET平台为目标的语言所必须支持的最小特征，以及某一语言与其他.NET语言之间实现互操作所需要的完备特征。凡是遵守这个标准的语言，在.NET框架下都可以实现互相调用；不符合公共语言规范的代码就不能被其他语言调用，继而不能与其他语言进行互操作。

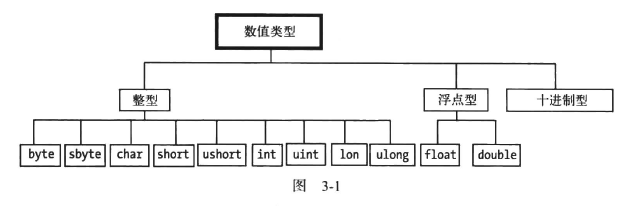


#### 2. .NET Framework

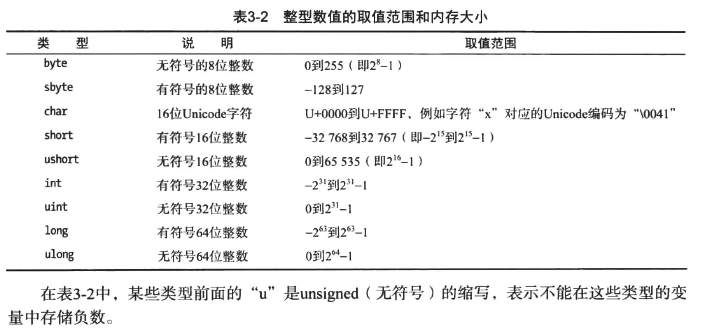
.NET Framework类库就是一组DLL程序集的集合，其中包含了大量定义好的类型，这些类型都公开了一些功能。

由于FCL中包含了数量极多的类型，因此有必要将相关的一组类型放到一个单独的命名空间中加以区别，例如System.IO命名空间中就包含了用于执行I/O操作的类型。因此，在使用FCL中某个类时，还必须知道该类所在的命名空间。

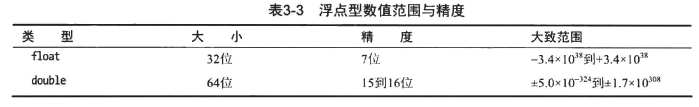
### 3.2.1 数值类型



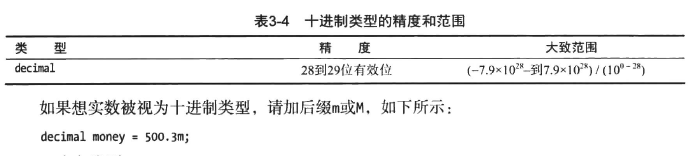
#### 1.整型



#### 2.浮点型



#### 3.十进制



## 4.5 类与结构体的区别

* 语法上的区别在于，定义类要使用关键字class，而定义结构体则使用关键字struct。
* 结构体中不可对声明字段进行初始化，但类可以。
* 如果没有为类显示地定义构造函数，C#编译器会自动生成一个无参数的实例构造函数，我们称之为隐式构造函数；而一旦我们为类显示地定义了一个构造函数，C#编译器就不会再自动生成隐式构造函数了。与此不同的是，在结构体中，无论你是否显式地定义了构造函数，隐式构造函数都是一直存在的。
* 结构体重不能显式地定义无参数的构造函数，这也说明无参构造函数是一直存在的，所以不能在显式地为结构体添加一个无参的构造函数；而类中则可以显式地定义一个无参数的构造函数。
* 在结构体的构造函数中，必须要为结构体中的所有字段赋值。
* 创建结构体对象可以不使用new关键字，但此时结构体对象中的字段是没有初始值的；而类必须使用new关键字来创建对象。
* 结构体不能继承结构或者类，但可以实现接口；而类可以继承类但不能继承结构，它也可以实现接口。
* 类是引用类型，而结构体是值类型。
* 结构体不能定义析构函数，而类可以有析构函数。
* 不能用abstract和sealed关键字修饰结构体，而类可以。

### 5.2.3 子类的初始化顺序

子类的初始化顺序：

1. 初始化类的实例字段；
2. 调用基类的构造函数，如果没有指明基类，则调用System.Object的构造函数；
3. 调用子类的构造函数。

## 5.3 多态

由于可以继承基类的所有成员，子类就都有了相同的行为，但是有时子类的某些行为需要相互区别，子类需要覆写父类的方法来实现子类特有的行为，这样的技术在面向对象的编程中就是多态。

### 5.3.1使用virtual和override关键字实现方法重写

### 5.3.2 阻止派生类重写虚成员

用sealed关键字可以防止一个类被其他类继承。同样，也可以用sealed关键字来阻止派生类重写虚成员。

### 5.3.3 使用新成员隐藏基类成员

如果在派生类中定义了与基类成员同名的成员，可以使用new关键字把基类成员隐藏起来。

## 5.4所有类的父类：System.Object

所有类都派生自System.Object类。

## 6.1 什么是接口

接口可以理解为对一组方法声明进行的统一命名，但这些方法没有提供任何实现。

也就是说，把一组方法声明在一个接口中，然后继承与该接口的类都需要实现这些方法。

通过接口，你可以对方法进行统一管理，避免在每种类型中重复定义这些方法。

6.2.1接口的定义

在接口中定义方法不能添加任何访问修饰符，因为接口中的方法默认为public，如果显示地指定了修饰符，则会出现编译时错误。

在接口中出了可以定义方法外，还可以包含属性、事件、索引器、或者这四类成员（包括方法）类型的任意组合；但接口类型不能包含字段、运算符重载、实力构造函数和析构函数。

## 6.3显示接口实现方式

* 采用隐式接口实现时，类和接口都可以访问接口中的方法；而采用显式接口实现方式，接口方法只能通过接口来完成访问，因为此时接口方法默认为私有。
* 当类实现单个接口时，通常使用隐式接口来实现方式，这样累的对象可以直接去访问接口方法。
* 当类实现了多个接口，并且接口中包含相同的方法名称、参数和返回类型时，则应使用显式接口实现方式。即使没有相同的方法签名，在实现多个接口时，仍推荐使用显式的方式，因为这样可以标识出哪个方法属于哪个接口。

## 6.4接口与抽象类

* 抽象类使用abstract关键字进行定义，而接口使用interface进行定义；他们都不能进行实例化。
* 抽象类中包含虚方法、费抽象方法和静态成员；但接口中不能包含虚方法和任何静态成员，并且接口总只能定义方法，不能有具体实现，方法的具体实现由实现类完成。
* 抽象类不能实现多继承，接口则支持多继承。注意，从严格意义上说，类接口接口应该成为类实现接口。
* 抽象类是对一类对象的抽象，继承于抽象类的类于抽象类为属于的关系；而类实现接口只是代表类具有接口声明的方法，是一种can-do的关系。所以一般接口后都带有able字段，标识“我能做”的意思，例如微软库中的IComparable接口和ICloneable接口等。

## 7.3教你看懂IL代码

### 7.3.1 IL基本类型

|  |  |
| --- | --- |
| IL基本类型 | C#基本类型 |
| Bool | System.Boolean |
| Char | System.Char |
| Int8 | System.SByte |
| Int16 | System.Int16 |
| Int32 | System.int32 |
| Int64 | System.int64 |
| Float32 | System.Single |
| Float64 | System.Double |
| object | System.Object |
| string | System.String |
| Array | System.Array |

### 7.3.2 变量的声明

在C#语言中，变量的声明格式为“数据类型+变量名称”。而对于IL代码，.locals指令代表变量的声明，声明语句放在IL\_标记前面。例如：.locals init([0] string helloString)就声明了一个名为helloString的变量，其类型为string。

### 7.3.3 基本运算

和C#语言一样，IL中也提供了大量的运算指令来完成基本的运算操作。

* 算术运算。包括加法指令add、乘法指令sub、除法指令div，以及求余指令rem等。
* 位运算。包括一元指令not、与指令and、或指令or等，结果以1和0分别表示真、加，运算结果压入评估栈栈顶。

### 7.3.4 IL中的流程控制

流程控制语句是语言执行的基础。在C#语言中，流程控制语句有if-else语句、while语句和for语句等，同样，IL中也有流程控制指令。

## 8.1 C#委托是什么

C#中的委托可以理解为函数的一个包装，它使得C#中的函数可以作为参数来杯传递，这在作用上相当于C++中的函数指针。C++用函数指针获取函数的入口地址，然后通过这个指针来实现对函数的操作。

被委托包装的函数必须满足以下规则：

* 方法的签名必须与委托一致，方法签名包含参数的个数、类型和顺序；
* 方法的返回类型要和委托一致，注意，方法的返回类型不属于方法签名的一部分。

9.1 什么是事件

事件涉及两类角色-----事件发布者和事件订阅者。当某个事件发生后，事件分布者会发布消息；事件订阅者会接受到事件已发生的通知，并做出相应处理。其中，触发事件的对象称为事件发布者，捕获事件并对其做出处理的对象称为事件订阅者。事件的触发可能源于用户与系统的交互（例如按下键盘或者单击鼠标），也可能源于事件发布者的某种程序逻辑。

9.2.1 如何定义事件

Public event EventHandler birthday

事件定义的结构为：访问修饰符 event 委托类型 事件名；

这里，访问修饰符一般定义为public，因为事件的订阅者需要对事件进行订阅与取消操作，定义为公共类型可视事件对其他类可见。事件定义中还包括委托类型，它既可以是自定义的委托类型，也可以是.NET类库中预定义的委托类型EventHandler。

9.2.2 订阅和取消事件

使用“+=”运算符来订阅事件，使用“-=”云算法来取消事件订阅。