**命名空间**

一个命名空间是一个逻辑的命名系统，用来组织庞大的系统类资源，使开发者使用起来结构清晰、层次分明、使用简单，同时，开发者可以使用自定义的命名空间以解决大型应用中可能出现的名称冲突。

(1) 定义命名空间

在C#中定义命名空间的语法格式如下：

namespace SpaceName

{

......

}

其中namespace为声明命名空间的关键字，SpaceName为命名空间的名称，在整个{}内的内容都属于名称为SpaceName的命名空间的范围。其中可以包含类、结构、枚举、委托和接口等可在程序中使用的类型。

(2) 嵌套命名空间

命名空间内包含的可以是一个类、结构、枚举、委托和接口，同时也可以在命名空间中嵌套其他命名空间，从而构成树状层次结构。

(3) using语句

当出现多层命名空间嵌套时，键入起来很繁琐，为此，要在文件的顶部列出类的命名空间，前面加上using关键字。在文件的其他地方，就可以使用其类型名称来引用命名空间中的类型了

**标识符与关键字**

(1) 标识符

标识符(identifier)名以字母、下划线(\_)等Unicode字符开头，但是不能以数字开头，不能包含空格，关键字不可以用作普通标识符，但可以用@前缀来避免这种冲突。

C#中标识符是区分大小写的，Myabc和myabc是两个完全不同的标识符。

(2) 关键字

关键字也称为保留字(reserved word)，在C#语言中有如下关键字：

Abstract、as、base、bool、break、byte、case、catch、char、checked、class、const、continue、decimal、default、delegate、do、double、else、enum、event、explicit、extern、false、finally、fixed、float、for、foreach、get、goto、if、implicit、in、int、interface、internal、is、lock、long、namespace、new、null、object、operator、out、override、params、private、protected、public、readonly、ref、return、sbyte、sealed、set、short、sizeof、stackalloc、static、string、struct、switch、this、throw、true、try、typeof、uint、ulong、unchecked、unsafe、using、value、virtual、volatile、while

**变量**

C#中声明变量的语法如下所示：

<type> <name>;

<type> <name1>,<name2>;

<type> <name = value>;

**数据类型**

1. 简单数据类型
2. 整型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 别名 | 有无符号 | 占据位数 | 允许值的范围 |
| sbyte | System.Sbyte | 是 | 1 | -128 到 127 |
| short | System.Int16 | 是 | 2 | -32768 到32767 |
| int | System.Int32 | 是 | 4 | -2147483648 到 2147483647 |
| long | System.Int64 | 是 | 8 | -9223372036854775808 到 9223372036854775807 |
| byte | System.Byte | 否 | 1 | 0 到 255 |
| ushort | System.Uint16 | 否 | 2 | 0 到 65535 |
| uint | ystem.UInt32 | 否 | 4 | 0 到 4294967295 |
| unlong | ystem.Uint64 | 否 | 8 | 0 到18446744073709551615 |

1. 布尔型

布尔类型是用来表示“真”和“假”这两个概念的。该类型变量只有两种取值：“真”和“假”。在C#中，分别采用true和false两个值来表示。

(3) 字符型

字符包括数字字符、英文字母、表达式符号等，C#提供的字符类型按照国际上公认的标准，采用Unicode字符集。一个Unicode的标准字符长度为16位，用它可以来表示世界上大多种语言。字符类型变量赋值形式有三种：

char chsomechar=“A”;

char chsomechar=“\x0065”; //十六进制

char chsomechar=“\u0065 ; //unicode表示法

转意字符，用来在程序中指代特殊的控制字符。

|  |  |  |  |
| --- | --- | --- | --- |
| 转义符 | 字符名 | 转义符 | 字符名 |
| \’ | 单引号 | \’’ | 双引号 |
| \\ | 反斜杠 | \0 | 空字符 |
| \a | 感叹号(Alert) | \b | 退格 |
| \f | 换页 | \n | 换行 |
| \r | 回车 | \t | 水平Tab |
| \v | 垂直Tab |  |  |

(4) 实数型

C#支持两种浮点类型：float和double。该类型与其他数据类型的比较如下表所示 ：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 别名 | 有无符号 | 占据位数 | 允许值的范围 |
| float | System.Single | 是 | 4 | 可能值从 ±1.5 x 10-45 到 ±3.4 x 1038 ，小数点后7位有效数字 |
| double | System.Double | 是 | 8 | 可能值从±5.0 x 10-324 to ±1.7 x 10308小数点后15到16位有效数字 |
| decimal | System.Decimal | 是 | 16 | 可能值从 ±1.0 x 10-28 到±7.9 x 1028 小数点后28到29位有效数字 |
| char | System.Char | N/A | 2 | 任何16位Unicode字符 |
| bool | System.Boolean | N/A | 1 / 2 | true 或者false |

2，结构类型

(1) 结构

结构类型是一种可以自己定义的数据类型，它是一个可以包含不同类型数据成员的一个结构，在结构类型中可以声明多个不同数据类型的组成部分，这些组成部分被称为结构体的成员或域。结构体允许嵌套。

结构类型包含的成员类型没有限制，结构体的成员也是没有限制的，任何合法的成员都可以包含在一个结构体内。

(2) 枚举

枚举enum实际上是为一组在逻辑上密不可分的整数值提供便于记忆的符号。如，我们声明一个代表星期的枚举类型的变量：

enum WeekDay{

Sunday，Monday，Tuesday，Wednesday，Thursday，Friday，Saturday

};

WeekDay day;

**引用类型**

|  |  |  |
| --- | --- | --- |
|  | **值类型** | **引用类型** |
| **变量中存放** | 真正的数据 | 指向数据的引用指针 |
| **内存空间分配** | 堆栈（Stack） | 托管堆（Managed Heap） |
| **内存需求** | 一般来说较少 | 较大 |
| **执行效能** | 较快 | 较慢 |
| **内存释放时间点** | 执行超过定义变量的作用域 | 由回收站负责回收 |
| **可以为null** | 不可 | 可以 |

(1) 类类型

类(class)是C#或者面向对象程序设计中最重要的组成部分，如果没有类，所有使用C#编写的程序都不能进行编译。由于类声明创建了新的引用类型，所以就生成了一个类类型(class types)。类类型中包含了数据、函数和嵌套类型，其中，数据中又可以包括常数、字段(fields)和事件；函数包括了方法、属性、索引器、操作符、构造器(constructors)以及析构器(destructors)。

在C#中，类类型只能单继承即一个对象的基类（父类）不能有多个，所以类只能从一个基类中派生出来，并具有它的部分或者全部属性。不过C#中一个类可以派生出多个接口。

(2) 接口类型

接口类型声明了一个抽象成员，而结构和类应用接口进行操作就必须获取这个抽象成员。接口中可以包含方法(methods)、属性(properties)、索引器(indexers)和事件(events)等成员。C#的接口只有署名，没有实现代码，接口能完成的事情只有名称，所以只能从接口衍生对象而不能对接口进行实例化。从面向对象的角度考虑，使用接口最大的好处就是，它使对象与对象之间的关系变为松耦合。对象之间可以通过接口进行调用，而不是直接通过函数。接口就相当于对象之间的协议一样，在调用接口时可以不关心接口的具体实现方法。这样某个对象进行改变时，其它对象不用进行任何修改还可正常运行。

(3) 代理类型

C#代码在托管状态下不支持指针操作，为了弥补去掉指针对语言灵活性带来的影响，C#引入了一个新的类型：代理（Delegate）。它相当于C++中指向函数的指针。与C++的指针不同，代理完全是面向对象的，它把一个对象实例和方法都进行封装，所以代理是类型安全的。

代理的一个有趣而又有用的特性是它不知道或不关心它引用的对象的类。只要方法的声明与代理的声明一致，任何对象都可以。这使得代理适合作匿名“调用”。

在声明代理时只需要指定代理指向的原型的类型，它不能有返回值也不能带有输出类型的参数。

(4) 数组类型

数组(array)是指同类数据组成的集合，它是数据最常用的存储方式之一。C#中的数组有矩形数组(rectangular arrays，含一维和多维数组)、锯齿形数组(jagged arrays)、嵌套数组。

**装箱与拆箱**

(1) 装箱

所谓装箱既是将一个值类型变量转化为一个引用类型的变量。引用类型可以为类类型也可以为接口类型等。装箱的过程首先创建一个引用类型的实例，然后将值类型变量的内容复制给该引用类型实例。装箱方式可以分为显式和隐式两种，分别如下：

int i = 123;

Object o = i; //隐式方式，先实例化一个Object对象o，然后将其值赋成i的值

Object o = Object(i); //显式方式

无论显式方式还是隐式方式实质都是一样的，首先实例化一个引用类型的对象o，然后将值类型变量i的值复制给o。

(2) 拆箱

拆箱与装箱在逻辑上是一对互逆的过程。拆箱转换是指将一个引用类型显式地转换成一个值类型，或是将一个接口类型显式地转换成一个执行该接口地值类型。需要指出的是装箱操作可以隐式进行但拆箱操作必须是显式的。拆箱过程分成两步：首先，检查这个对象实例，看它是否为给定的值类型的装箱值。然后，把这个实例的值拷贝给相应值类型的变量。比如：

int i = 123; //定义int型变量i，初值为123

object o = i; //执行装箱操作

int j = (int)o; //执行拆箱操作，将对象o的值赋于变量j

**数据类型转换**

(1) 隐式与显式类型转换

隐式转换是编译系统自动进行的，不需要加以声明。在该过程中，编译器无需对转换进行详细检查就能够安全的执行转换。隐式转换一般不会失败、不会出现致命隐患及造成信息丢失。

隐式转换无法完成由精度高的数据类型向精度低的类型转换，char类型数据也无法进行隐式类型转换。

显式类型转换，又称为强制类型转换。与隐式类型转换相对，该方式需要用户明确的指定转换的目标类型。

显式转换包含所有的隐式类型转换，即把任何编译器允许的隐式类型转换写成显式转换都是合法的。显式类型转换并不一定总是成功，且转换过程中会出现数据丢失。

(2) 数值字符串与数值之间的转换

数值到字符串：使用tostring()方法即可。

反过来，使用该数值类型对应类的Parse()函数。或使用System.Convert类的对应方法将数字字符串转化为相应的值。

(3) 字符串与字符数组之间的转化

字符串类String提供了一个ToCharArray()方法，该方法可以实现字符串到字符数组的转换；反之，可以使用String类的构造函数来实现字符数组到字符串的转换，String类实现了两个通过字符数组来构造字符串的构造函数，即String(char[]) 和String(char[], int, int) 。

其中，后者的后两个参数用来指定字符串的开始和结束字符在字符数组参数中的边界，若需要获取字符串中某个特定位置上的字符，可以使用直接索引字符串的方式。

**操作符与表达式**

1赋值操作符与表达式

C#中可以对变量进行连续赋值，这时赋值操作符是右关联的。这意味着从右向左操作符被分组，如a = b = c等价于 a = (b = c)。

2关系运算符与表达式

关系表达式的返回值总是布尔值。C#中关系操作符的优先级低于算术操作符，高于赋值操作符。

(1) 比较运算

C#中定义的比较操作符有：

== 等于

!= 不等于

< 等小于

> 等大于

<= 小于或等于

>= 大于或等于

(2) is操作符

is操作符被用于动态地检查运行时对象类型是否和给定的类型兼容。假设a是一个表达式或值，Type是一个类型，运算“a is Type”的结果，返回一个布尔值。它表示a是否能通过引用转换、装箱转换或拆箱转换，成功地转换于Type类型。

(3) as操作符

as运算符类似于强制转换操作；但是，如果转换不可行，as会返回null而不是引发异常。假设a为某一具体的值或表达式，Type为某一类型，形如a as Type的运算，若a可以转化为Type类型，则返回Type(a)否则返回null。且表达式a只被计算一次。

3逻辑运算符与表达式

C#语言提供了三种逻辑操作符：

&& (逻辑与)、||(逻辑或)、！(逻辑非)

4其他运算符与表达式

(1)自增和自减运算符

自增操作符“++”对变量的值加1，而自减操作符“--”对变量的值减1。它们适合于sbyte、byte、short、ushort、int、uint、long、ulong、char、float、double、decimal和任何enum类型。

自增和自减操作符的操作数必须是一个变量、一个属性访问器或一个索引指示器访问器，而不能是常量或者其它的表达式。如3++和x+y --都是非法的。如果操作数是一个访问器，那么这个访问器必须同时支持读和写。

自增和自减操作符又有前后缀之分，对于前缀操作符，遵循的原则是“先增减,后使用”，而后缀操作符则正好相反，是“先使用，后增减”。

(2) new运算符

new操作符用于创建一个新的类型实例，它有三种形式：

对象创建表达式，用于创建一个类类型或值类型的实例；

数组创建表达式，用于创建一个数组类型实例；

代表创建表达式，用于创建一个新的代表类型实例；

new操作符暗示一个类实例的创建，但不一定必须暗示动态内存分配，这和C++中对指针的操作不同。例如，下面三个式子分别创建了一个对象、一个数组和一个代表实例。

class A{}; A a = new A;

int[] intArr = new int[10];

delegate double DFunc(int x); DFunc f = new DFunc(5);

(3) typeof运算符

typeof操作符用于获取类型的 System.Type 对象.

(4) checked和unchecked操作符

checked和unchecked操作符用于整型算术运算时控制当前环境中的溢出检查。下列运算可能导致溢出，需要进行checked和unchecked检查：

当其操作数类型为整型时，执行++、–及-(负运算)等操作需要进行checked检查；

当两个操作数数据类型都是整型时，执行+、-、\*、／等操作需要进行checked检查；

当从一种整型到另一种整型地显式数据转换时，需要进行checked检查。

(5) 三元操作符

三元操作符“?:”，也称为条件操作符。

5运算符的优先级

|  |  |
| --- | --- |
| **类别** | **操作符** |
| 初级操作符 | () x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked |
| 一元操作符 | + - ！ ~ ++x --x (T)x |
| 乘除操作符 | \* ／ % |
| 加减操作符 | + - |
| 位移操作符 | << >> |
| 关系操作符 | < > <= >= is |
| 等于操作符 | == != |
| 逻辑与 | & |
| 逻辑异或 | ^ |
| 逻辑或 | | |
| 条件与 | && |
| 条件或 | || |
| 条件操作符 | ?: |
| 赋值操作符 | = \*= /= %= += -= <<= >>= &= ^= |= |

**选择结构程序设计**

1if语句

if语句是最常用的选择语句，它根据布尔表达式的值来判断是否执行后面的内嵌语句。if语句的基本语法格式为：

if (布尔表达式)

真值运算;

else //可选项

假值运算

2switch语句

switch开关语句处理多路分支的情形，使程序变得简洁，它的一般格式为：

switch(表达式)

{

case value1:

内嵌语句块1

case value2;

内嵌语句块2

default:

内嵌语句块N

}

switch语句的控制类型，即其中控制表达式(expression)的数据类型可以是sbyte、byte、short、ushort、uint、long、ulong、char、string或枚举类型(enum-type)。每个case标签中的常量表达式(value)必须属于或能隐式转换成控制类型，如果有两个或两个以上case标签中的常量表达式值相同，编译时将会报错。switch语句中最多只能有一个default标签。

(1) 不允许遍历

C和C++语言允许switch语句中case标签后不出现break语句，但C#不允许这样，它要求每个case标签项后使用break语句或跳转语句goto，即不允许从一个case自动遍历到其它case，否则编译时将报错。

(2) 允许任意排列switch语句中的case项而不会影响switch语句的执行结果。

(3) 每个switch项都以break、goto、case或goto default结束，事实上任何一种不导致“遍历”效果的语句都是允许的。如 throw 和 return 语句同样可以达到不“遍历”。

**循环结构程序设计**

1for循环语句

在事先知道循环次数的情况下，使用for循环语句是比较方便的。For循环语句的格式为：

for(初始条件;结束条件;循环变量迭代器)

{

循环体

}

其中“初始条件”、“结束条件”、“循环变量迭代器”这三项都是可选项。“初始条件”为循环控制变量做初始化，循环控制变量可以有一个或多个(用逗号隔开)。“结束条件”为循环控制条件，也可以有一个或多个语句。“循环变量迭代器”按规律改变循环控制变量的值。初始化、循环控制条件和循环控制都是可选的，如果忽略了条件，则产生一个死循环，要用跳转语句break或goto才能退出。

2foreach语句

表示遍历一个集合中的各元素，并针对各个元素执行内嵌语句。foreach语句的语法格式为：

foreach(type identifier in expression）embedded-statement

循环变量由类型type和标识符identifier声明，且表达式expression与集合相对应。循环变量代表循环正在处理的集合元素。在循环体内不能赋一个新值给循环变量，也不能把它当作ref或out参数。

3while语句

while语句有条件地将循环体内的语句执行0遍或若干遍。其语法格式为：

While (布尔表达式)

{

embedded-statement

}

4do-while语句

do-while语句与while语句框架上相似，不同之处在于，它将循环体内的语句至少执行一次。其语法格式为：

do{

embedded-statement

}while(布尔表达式)

5跳出循环

当循环条件不满足时循环自动就结束了，如在循环体内遇到特定的情况需要在循环条件满足的情况下终止循环，就需要用到中断循环语句。C#为此提供了4个命令。分别是：

Break：立即终止循环，并跳转到循环后面第一行代码执行

Continue：立即终止当次的循环(并不影响循环的继续执行)。

Return：用来返回到当前函数被调用的地方。如果return语句放在循环体内，当满足条件时执行return语句返回，循环自动结束 。

Goto：可以跳出循环，到已标记好的位置上。

**异常处理结构**

1异常的产生

C#在两种情况下会产生异常：

(1) 在C#语句和表达式的处理过程中激发了某个异常的条件，使得操作无法正常结束，引发异常。

(2) 使用throw语句手动抛掷异常

C#中的System.Exception类是所有异常的基本类型。这个类有几个值得注意的特性，这些特性由所有的异常类共享：Message是一个只读特性，它包含了人们可读的对该异常发生原因的描述。InnerException是一个只读特性，它包含了该异常的“内层异常”。如果它不是null，就表示当前异常是在响应另一个异常时发生的。引起当前异常的那个异常可以在InnerException特性中得到。

2处理异常

在代码中对异常进行处理，一般要使用三个代码块：

Try块的代码是程序中可能出现错误的操作部分。

catch块的代码是用来处理各种错误的部分（可以有多个）。必须正确排列捕获异常的catch子句,范围小的Exception放在前面的catch。即如果Exception之间存在继承关系，就应把子类的Exception放在前面的catch子句中。

Finally块的代码用来清理资源或执行要在try块末尾执行的其他操作（可以省略）。且无论是否产生异常，Finally块都会执行。

异常是由try\_catch\_finally语句块组合来处理的，执行异常处理过程通常有以下三种可能的形式：

(1) try-catch(s)

(2) try-finally

(3) try-catch(s)-finally

当异常发生时，系统将搜寻最近的能够处理该异常的catch从句，这由异常的运行期类型来决定。首先，在当前方法中搜寻封闭的try语句，按文本顺序考虑与该try语句相关的catch从句。如果无效，就搜寻调用try语句的方法和当前方法，按文本顺序考虑包含调用当前方法的位置的封闭try语句。继续这样的搜寻，直到找到一个能够处理当前异常的catch从句为止。该catch从句命名的异常类与当前异常发生时的运行期类型相同，或者是它的基类。没有命名异常类的catch从句可以处理任何异常。