# 1. 变量

1）作用：在计算机中存储数据

2）属性：数据类型、名称、值

3）使用之前必须先声明

4）声明：

int a;

int b, c;

5）初始化--赋值

int d = 1;

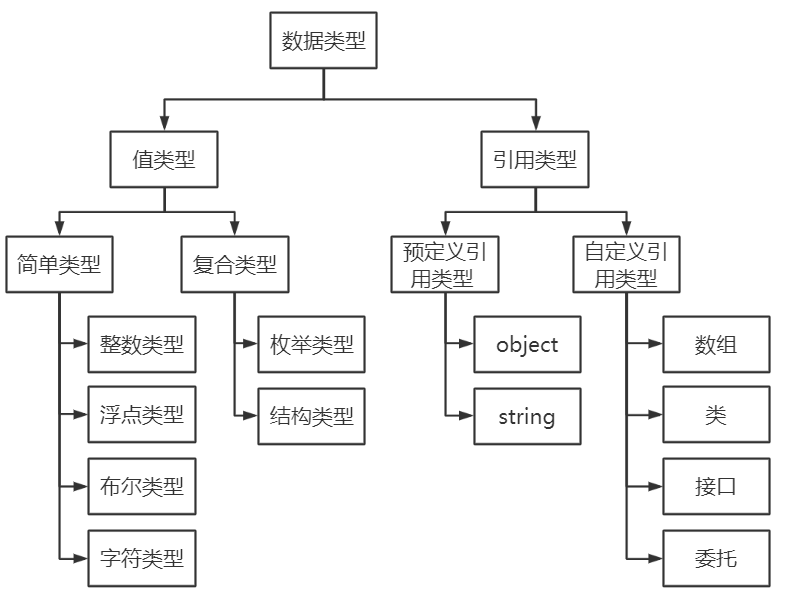
int e;

e = 1;

b = c = a = 2;

注意：赋值的时候赋值符号两边的数据类型要一致

# 2. 数据类型：值类型（整数类型、浮点类型、布尔类型、字符类型、枚举类型、结构类型）和引用类型



1）值类型：存储在栈区域中的，效率较高

a 整数类型

声明有符号整数的关键字：sbyte(1) short(2) int(4) long(8)

声明无符号整数的关键字：byte(1) ushort(2) uint(4) ulong(8)

在给整数变量赋值的时候可以赋十进制、八进制、十六进制的值

int f = 10;

int g = 020; 赋值八进制的数，需要数据前面加 零

int h = 0x30; 赋值十六进制的数，需要数据前面加 零x

原码、反码、补码：计算机中存储和计算的都是补码

正数：原码、反码、补码是一样的。

负数：原码是数据转换成二进制的初始形式（符号位为1），反码等于原码除符号位以外各位取反，补码等于反码加1

b 浮点类型

声明的关键字：float(4) double(8) decimal(16)

double i = 2.3;

float j = 2.3F; 使用字符后缀说明数据类型

float k = 2.3f;

double l = 2.3d;

double m = 2.3D;

decimal n = 2.3m;

c 字符类型: 需要用单引号将字符包裹起来

声明的关键字：char Char（2）

字符类型常用方法的使用方式：Char.方法名（参数）

转义字符：为了直接输出那些特殊的符号 \， { 要用 { 来转义。

Char类型常用方法

|  |  |
| --- | --- |
| **方法** | **说明** |
| **IsDigit** | 指示某个Unicode字符是否属于十进制数字类别 |
| **IsLetter** | 指示某个Unicode字符是否属于字母类别 |
| **IsNumber** | 指示某个Unicode字符是否属于数字类别 |
| **IsPunctuation** | 指示某个Unicode字符是否属于标点符号 |
| **IsSeparator** | 指示某个Unicode字符是否属于分隔符类别 |
| **IsUpper** | 指示某个Unicode字符是否属于大写字母 |
| **IsLower** | 指示某个Unicode字符是否属于小写字母 |
| **IsWhiteSpace** | 指示某个Unicode字符是否是空格符 |
| **Parse** | 将指定字符串的值转换为它的等效Unicode字符 |
| **ToLower** | 将Unicode字符的值转换为它的小写等效项 |
| **ToString** | 将字符的值转换为其等效的字符串 |
| **ToUpper** | 将Unicode字符的值转换为它的大写等效项 |
| **TryParse** | 将指定字符串的值转换为它的等效Unicode字符 |

常用转义字符表

|  |  |
| --- | --- |
| **转义字符** | **说明** |
| **\n** | 回转换行 |
| **\t** | 横向跳到下一制表位 |
| **\”** | 显示双引号 |
| **\r** | 退格符 |
| **\b** | 回车符 |
| **\f** | 换页符 |
| **\\** | 显示反斜线 |
| **\’** | 显示单引号 |
| **\uxxxx** | 4位十六进制所表示的字符，如\u0052 |

d 布尔类型

声明关键字：bool

取值范围：true false

e 枚举类型：enum

使用步骤

/\*

1. 创建一个枚举类型

enum Directin { UP, DOWN, LEFT, RIGHT }

2. 声明一个枚举类型的变量

Direction d;

3. 给变量赋值

d = Directin.UP

\*/

f 结构类型

/\*

1. 创建一个结构类型

struct Route {

Direction dir;

double dis;

}

2. 声明一个结构类型的变量

Route route;

3. 给结构类型变量赋值

route.dir = Direction.UP;

route.dis = 2.3;

\*/

2）引用类型：真实数据存储在堆区域，栈区域只是存储的相应堆区域的首地址。

# 3. 变量的作用域：变量起作用的代码区域

成员变量：定义在类中的变量，作用域为整个类。

局部变量：定义在方法中，作用域为单个方法。

# 4.类型转换

1）隐式类型转换：C#在运行过程中以安全的方式自动进行转换。

转换条件：数据类型要兼容；目标类型大于源类型（指存储空间）

转换特点：数据永远不会溢出

2）显示类型转换（强制类型转换）：通过指定类型说明符来进行转换

转换条件：数据类型要兼容

转换特点：数据有可能会溢出造成转换出错

int o = (int)2.3;

3）Convert方法：用于强制转换

转换条件：任意转任意

Convert类提供的若干用于数据类型转换的方法（更多方法及方法的用法参考C#中文官方文档）

|  |  |
| --- | --- |
| **方法** | **描述** |
| **ToBoolean** | 如果可能的话，把类型转换为布尔型。 |
| **ToByte** | 把类型转换为字节类型。 |
| **ToChar** | 如果可能的话，把类型转换为单个 Unicode 字符类型。 |
| **ToDateTime** | 把类型（整数或字符串类型）转换为日期-时间结构。 |
| **ToDecimal** | 把浮点型或整数类型转换为十进制类型。 |
| **ToDouble** | 把类型转换为双精度浮点型。 |
| **ToInt16** | 把类型转换为16位整数类型。 |
| **ToInt32** | 把类型转换为32位整数类型。 |
| **ToInt64** | 把类型转换为 64 位整数类型。 |
| **ToSbyte** | 把类型转换为有符号字节类型。 |
| **ToSingle** | 把类型转换为小浮点数类型。 |
| **ToString** | 把类型转换为字符串类型。 |
| **ToType** | 把类型转换为指定类型。 |
| **ToUInt16** | 把类型转换为 16 位无符号整数类型。 |
| **ToUInt32** | 把类型转换为 32 位无符号整数类型。 |
| **ToUInt64** | 把类型转换为 64 位无符号整数类型。 |

# 5. 常量: const 常量 readonly常量

1）const：静态常量

初始化的时候必须赋初值

类型必须属于值类型，但是不能是struct

内存消耗：无

特点：不灵活；性能高、节约内存。

2）readonly常量：动态常量

初始化的时候可以赋值，在构造函数中也可以赋值

数据类型无限制

内存消耗：有

特点：灵活；性能低、占用内存。

# 6. 运算符：单目运算符 双目运算符 三目运算符（?:）

1）算术运算符：+ - \* / %

2）自增自减： ++ --

前缀式: 先计算，再赋值

int p = 1;

int q;

q = ++p; ==> p=p+1;q=p;

后缀式：先赋值，再计算

q = p++; ==> q = p; p = p+1;

注意：不能对常量和数值进行自增自减

3）赋值运算符：简单赋值和复合赋值

注意：左操作不能是常量或数值，右操作数没有限制；左操作必须大于有操作数。

|  |  |  |
| --- | --- | --- |
| **赋值运算符** | **表达式示例** | **含义** |
| **=** | x = 10 | 将10赋值给变量x |
| **+=** | x += y | x = x + y |
| **-=** | x -= y | x = x – y |
| **\*=** | x \*= y | x = x \* y |
| **/=** | x /= y | x = x / y |
| **%=** | x %= y | x = x % y |
| **>>=** | x >>= y | x = x >> y |
| **<<=** | x << = y | x = x << y |
| **&=** | x &= y | x = x & y |
| **|=** | x |= y | x = x | y |
| **^=** | x ^= y | x = x ^ y |

4）关系运算符：对运算符两边的数据进行关系比较

返回的结果是一个布尔类型的值，满足为true，否则为false。

5）逻辑运算符：用于连接一个或多个条件，判断这些条件是否成立。

注意：参与逻辑运算的数据类型都必须是布尔类型，计算结果也是一个布尔类型。

短路运算符：在逻辑表达式中从左端的表达式可推断出整个表达式的值称为“短路”。

&& ||

6）位运算符：对二进制中的每一位进行一定的操作，完成这种操作的运算符叫做位运算符。

&: 将两个运算对象按位进行与运算，如果两个操作数的精度不同，则结果的精度与高精度的操作数相同。

|  |  |  |
| --- | --- | --- |
| **数值1** | **数值2** | **结果** |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

| ：将两个运算对象按位进行或运算，如果两个操作数的精度不同，则结果的精度与高精度的操作数相同。

|  |  |  |
| --- | --- | --- |
| **数值1** | **数值2** | **结果** |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

~：位取反

|  |  |
| --- | --- |
| **数值1** | **结果** |
| ~1 | 0 |
| ~0 | 1 |

^：将两个运算对象按位进行异或运算，如果两个操作数的精度不同，则结果的精度与高精度。

|  |  |  |
| --- | --- | --- |
| **数值1** | **数值2** | **结果** |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

>>

/\*

将一个二进制操作数向右移动指定的位数

右边（低位端）溢出的位被丢弃，左边（高位端）的空位再补齐与符号位相同的数据。

相当于除以2的n次幂

\*/

<<

/\*

将一个二进制操作数向左移动指定的位数

左边（高位端）溢出的位被丢弃，右边（低位端）的空位用0补齐

相当于乘以2的n次幂

\*/

7）三目运算符： 表达式一?表达式二:表达式三

运算逻辑：如果表达式一为true，则整个表达式的结果为表达式二的结果；如果表达式一为false，则整个表达式的结果为表达式三的结果。

int r;

r = 5 > 9 ? 1 : 2; r=2

8）运算符的优先级

9）运算符的结合性：从左往右计算还是从右往左计算