静态构造函数的特点：

1. 静态构造函数没有修饰符修饰(public,private),因为静态构造函数不是我们程序员调用的，是由.net 框架在合适的时机调用的。
2. 静态构造函数没有参数，因为框架不可能知道我们需要在函数中添加什么参数，所以规定不能使用参数。
3. 静态构造函数前面必须是static 关键字。如果不加这个关键字，那就是普通的构造函数了。
4. 静态构造函数中不能实例化实例变量。（变量可以分为类级别和实例级别的变量，其中类级别的有static关键字修饰）。
5. 静态函数的调用时机，是在类被实例化或者静态成员被调用的时候进行调用，并且是由.net框架来调用静态构造函数来初始化静态成员变量。
6. 一个类中只能有一个静态构造函数。
7. 无参数的静态构造函数和无参数的构造函数是可以并存的。因为他们一个属于类级别，一个属于实例级别，并不冲突。
8. 静态构造函数只会被执行一次。并且是在特点5中的调用时机中进行调用。
9. 就像如果没有在类中写构造函数，那么框架会为我们生成一个构造函数，那么如果我们在类中定义了静态变量，但是又没有定义静态构造函数，那么框架也会帮助我们来生成一个静态构造函数来让框架自身来调用。

例：程序static\_structure\_test1

public class A

{

public static string textA;

public string strText;

// 静态构造函数只能有1个

// 静态构造函数不能有参数

// 静态构造函数不能有访问修饰符

// 静态构造函数由.net框架调用

// 静态构造函数只会调用1次

static A()

{

*Console*.*WriteLine*("static constructor A");

textA = "AAA";

}

public A()

{

*Console*.*WriteLine*("common constructor A");

strText = "AAAAAAAA";

}

}

public class B : A

{

public static string textB;

static B()

{

*Console*.*WriteLine*("static constructor B");

textA = "BBB";

textB = "BBBAAA";

}

public B()

{

*Console*.*WriteLine*("common constructor B");

strText = "BBBBBBBB";

}

}

class Program

{

static void Main(string[] args)

{

*Console*.*WriteLine*(B.textA);

*Console*.*WriteLine*(A.textA);

*Console*.*WriteLine*(B.textB);

*Console*.*WriteLine*(A.textA);

}

}

输出为：

static constructor A

AAA

AAA

static constructor B

BBBAAA

BBB

从上述结果可以看出：

当执行B.textA时，尽管是通过类B去调用的，B继承自A，由于textA是类A的静态变量，所以调用的是类A的静态构造函数，而且也不会继续调用类B的静态构造函数。之后执行A.textA时，由于之前已经调用了类A的静态构造函数，所以不再重新调用了。两个输出均为AAA。

继续执行B.textB时，会调用类B的静态构造函数，因为textB是B的静态变量，之后执行A.textA时，在B的静态构造函数中，textA被修改为BBB，所以输出为BBB。

例：程序static\_structure\_test2

public class A

{

public static string textA;

public string strText;

// 静态构造函数只能有1个

// 静态构造函数不能有参数

// 静态构造函数不能有访问修饰符

// 静态构造函数由.net框架调用

// 静态构造函数只会调用1次

static A()

{

*Console*.*WriteLine*("static constructor A");

textA = "AAA";

}

public A()

{

*Console*.*WriteLine*("common constructor A");

strText = "AAAAAAAA";

}

}

public class B : A

{

public static string textB;

static B()

{

*Console*.*WriteLine*("static constructor B");

textA = "BBB";

textB = "BBBAAA";

}

public B()

{

*Console*.*WriteLine*("common constructor B");

strText = "BBBBBBBB";

}

}

class Program

{

static void Main(string[] args)

{

A a = new A();

A a1 = new A();

*Console*.*WriteLine*(A.textA);

*Console*.*WriteLine*();

B b = new B();

*Console*.*WriteLine*(A.textA);

*Console*.*WriteLine*(B.textA);

}

}

输出为：

static constructor A

common constructor A

common constructor A

AAA

static constructor B

common constructor A

common constructor B

BBB

BBB

从上述输出可以看出：

执行A a = new A()和A a1 = new A()时，静态构造函数执行了一次，而普通构造函数执行了两次，可以看出在构造A的实例对象时，会首先调用类A的静态构造函数。

执行B b = new B()时，类B的静态构造函数执行了1次，其访问了类A的静态变量textA，按理会执行类A的静态构造函数，但由于之前类A的静态构造函数已执行，就不会重复调用了。由于执行了类B的静态构造函数，所以A.textA和B.textA都输出BBB。

例：程序static\_structure\_test3

public class A

{

static A()

{

*Console*.*WriteLine*("static constructor A");

}

public A()

{

*Console*.*WriteLine*("common constructor A");

}

}

public class B : A

{

static B()

{

*Console*.*WriteLine*("static constructor B");

}

public B()

{

*Console*.*WriteLine*("common constructor B");

}

}

class Program

{

static void Main(string[] args)

{

B b = new B();

}

}

如果只执行B b = new B()，并且不访问静态变量，会有如下输出：

static constructor B

static constructor A

common constructor A

common constructor B

可以看出，构造实例对象b时，会首先调用类B的静态构造函数，构造b的类A的部分时，会先调用类A的静态够函数，然后调用类A的普通构造函数，最后调用类B的普通构造函数。

总结来说，继承链中，类的静态构造函数的访问时机与类的普通构造函数并不一样。继承类构造时，一定会先调用基类的普通构造函数，然后调用自身的构造函数。继承类在构造时，会先调用自身的静态构造函数（如果之前没有调用过），如果其中访问了基类的静态变量，则又会去调用基类的静态构造函数（如果之前没有调用过）。如果没有访问，则会等继承类的静态构造函数执行完后，再去调用基类的静态构造函数（如果之前没有调用过）。