7/12

笔记本: 暑期实习
创建时间: 2021/7/12 10:30 **更新时间:** 2021/7/13 13:16
作者: 134exetj717
URL: http://wiki.suncaper.net/pages/viewpage.action?pageId=39855813

- 元组

```
C#7.0提供了元组 （tuple），允许在一个语句中完成所有变量的赋值。例如：(string country,
string capital, double gdpPerCapital) = ("Malawi", "Lilongwe", 226.50);
```

- 方法
  - 可以向调用方返回一个特定的值
  - 值参数

    1. 在栈中为形参分配空间
    2. 计算实参的值，并把该值复制给形参
    3. 调用值参数的方法不会修改内存中对应实参的值，所以使用值参数时可以保证实参的安全性

  - 引用参数：方法的形参中以ref修饰符声明的参数属引用参数，（注意，C++中是用&

    1. 不会为这类形参在栈上分配空间
    2. 形参的参数名将作为实参变量的别名，指向相同的内存位置

```
public void method(ref SampleClass3 s1)    //定义公共方法method
    {
        s1.num = 20;
        s1 = new SampleClass3();
        s1.num = 30;
    }
```

  - 输出参数：以 out 修饰符声明的参数属输出参数。与引用参数类似，输出参数也不开辟新的区域内容。

    1. 输出参数和引用参数的区别：
       1. 输出参数在调用之前不需要初始化对象，而且调用的方法中不能读取该参数，同时在返回之前必须要返回该参数；
       2. 引用参数在调用之前必须要初始化对象，而且调用的方法可以读取该参数。

```
public void addnum(int num1, out int num2)    //定义公共方法addnum()
    { num2 = num + num1; }
```

  - 参数数组：以 params 修饰符声明的参数为参数数组，用于处理相同数据类型而且参数个数可变的情况

    1. 方法声明中的params关键字之后不允许任何其他参数，并且只允许一个params关键字。参数数组不能再有ref和out修饰符

```
声明方式：

public void addnum(ref int sum, params int[] b)
```

```
    {
        sum = 0;
        foreach (int item in b)
            sum += item;
    }
```

调用方式：

```
addnum(ref x)              //零个实参
addnum(ref x,1)             /1个实参
addnum(ref x,1,2)            //2个实参
addnum(ref x,1,2,3,4,5)         //5个实参
```

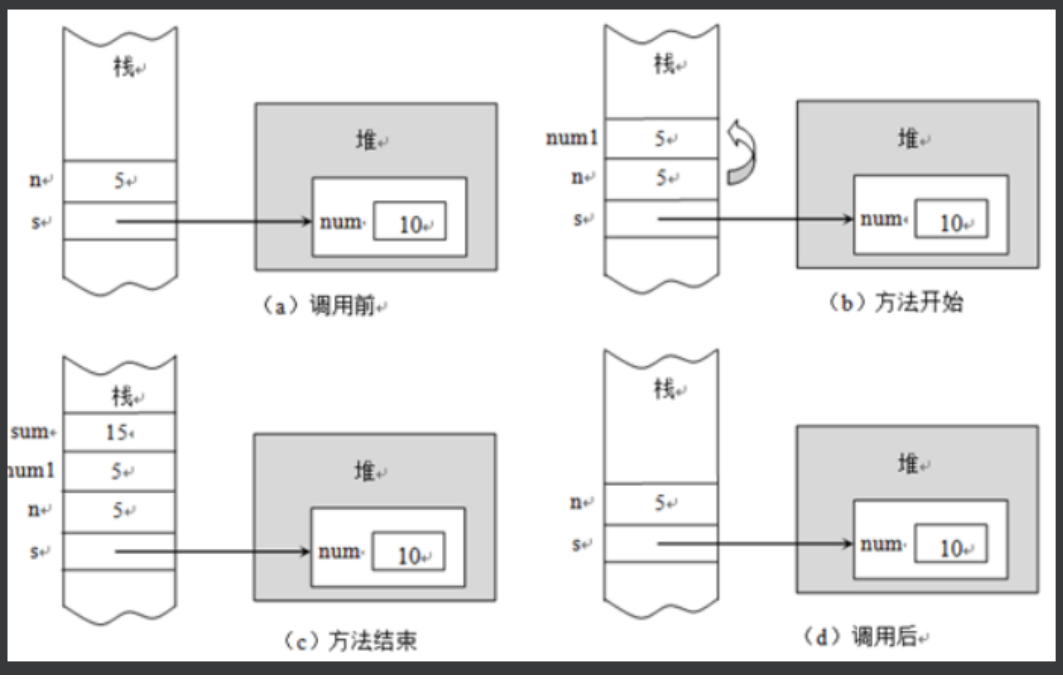
- 可选参数：为了表名每个参数是可选的，需要在方法定义时为她提供参数默认值

1. 如果一个方法包括必填参数、可填参数和params参数，那么　必填参数>可填参数>params参数

```
class MyClass
{
    public int add(int a, int b = 1, int c = 2)
    { return a + b + c; }
}
```

```
static void Main()
{
    SampleClass1 s = new SampleClass1();
    int n = 5;
    Console.WriteLine(s.addnum(n));    //输出15
}
```

值参数:



- 拓展，栈和堆的区别：

1. 申请方式：栈由系统自动分配；堆由人为申请开辟

2. 申请大小：栈小堆大
3. 申请效率：栈快堆慢
4. 底层不同：栈是连续空间，但堆的不连续的空间
5. 存储内容不同：栈在函数调用时，函数调用语句的下一条执行语句首先入栈，然后是各个参数进栈，其中静态变量是不进栈的（另分配有内存）；而堆中具体内容是人为安排

- this 关键字：在类中使用，是对当前实例的引用
- 运算符重载：同名运算符可用于运算不同类型的数据，目的是让使用类对象像使用基本数据对象类型一样自然合理

```
public static 返回类型 operator 运算符(参数列表)
   {
      ......
   }
```

- 浅拷贝和深拷贝的区别：
  - 若拷贝的对象中没有引用，那么浅拷贝和深拷贝是一样的；
  - 前者，引用直接复制，因此浅拷贝后的内容里改变引用的值，也会改变原被引用值
  - 后者，引用在复制的过程中，把被引用的内容复制一份，然后复制引用，因此深拷贝后的内容里改变引用的值，并不会改变原被引用值

比如，用 ref b 引用了一个整型 a=2 ，浅拷贝令b=5时，a会变为5；但深拷贝令b=5，a 依然 =2

- 嵌套类的使用
  - 嵌套类可以使用包含类，即里面的类可以使用外面的类
  - 只需要给A定义一个函数，函数里面包含创建一个B类，同时调用B类的方法；同时B类中重载构造函数，使其能够继承A类中的内容，如下：

```
using System;
namespace Proj
{
    class A                    //声明包含类A
    {
        string stra = "A"; //类A的私有字段
        public void funa1()    //定义类A的公有方法funa1()
        {
            B b = new B(this);     //将this即类A的对象传递给类B的构造函数
            b.funb1();
        }
        public void funb2()    //定义类A的公有方法funa2()
        {
            B b = new B(this);    //将this即类A的对象传递给类B的构造函数
            b.funb2();
        }
        class B                        //声明嵌套类B
        {
            private A m_parent;          //类B的私有字段
            string strb = "B";        //类B的私有字段
            public B() { }
            public B(A parent)            //定义类B的公有方法funb1()
            { m_parent = parent; }
            public void funb1()             //定义类B的公有方法funb1()
            { Console.WriteLine(m_parent.stra); }
            public void funb2()
            { Console.WriteLine(strb); }
        }
    }
    class Program
    {
        static void Main()
        {
            A a = new A();
```

```
            a.funa1();
            a.funb2();
        }
    }
}
```

- 索引器的使用
    - 什么是索引器？：索引器提供了一种访问类或结构的方法，即允许按照与数组相同的方式对类、结构或接口进行索引，比如，一个大学名称类University

```
un[0] = "清华大学";
un[1] = "北京大学";
un[3] = "武汉大学";
```

    - 声明索引器：要声明类或结构上的索引器，需使用this关键字

```
public int this[int index]    //索引器声明
{
    // get和set访问器
}

public class University
{
    const int MAX = 5;
    private string[] name = new string[MAX];
    public string this[int index]  //索引器
    {
        get
        {
            if (index >= 0 && index < MAX)
                return name[index];
            else
                return name[0];
        }
        set
        {
            if (index >= 0 && index < MAX)
                name[index] = value;
        }
    }
}
```

    - 使用其他非整数的索引类型：就好像是python中的字典一样，也可以用字符串的形式来实现索引
- 委托的使用（delegate）
    - 什么是委托？：允许在运行时选择要调用的函数，类似于C++、C中的函数指针
    - 委托类型的声明

```
[修饰符] delegate 返回类型 委托类型名(参数列表);
private delegate void mydelegate(int n);
```

    - 定义与实例化委托

```
委托类型名 委托对象名;
mydelegate p; // 没有实例化

class MyDeClass
{
    public void fun1(int n)
    {
        Console.WriteLine("{0}的2倍={1}", n, 2 * n);
    }
    public void fun2(int n)
    {
```

```
        Console.WriteLine("{0}的3倍={1}", n, 3 * n);
    }
}
MyDeClass obj = new MyDeClass();
mydelegate p = new mydelegate(obj.fun1);
```

- 委托的调用

```
委托对象名(实参列表);
p(100);
```

- 委托与匿名方法关联

```
delegate 返回类型 委托类型名(参数列表);
委托类型名 委托对象名=返回类型(参数列表) { /*匿名方法代码*/ };
委托对象名(实参列表)

elegate void mydelegate(string mystr);  //声明委托类型
class Program
{
    static void Main(string[] args)
    {
        mydelegate p = delegate (string mystr)
        {
            Console.WriteLine(mystr);
        };
        p("String");   //输出：String
    }
}
```

- 委托与Lambda（I）表达式

```
delegate void mydelegate(string mystr);      //声明委托类型
class Program
{
    static void Main(string[] args)
    {
        mydelegate p = mystr => Console.WriteLine(mystr);    //左边是参数，当只有一个参数时可以不要小括号（），右边是函数体，一般需要大括号{}
        //或mydelegate p = (mystr=>Console.WriteLine(mystr));
        p("String");                  //输出：String
    }
}
```

- 事件的应用
  - 什么是事件？：某一事件发生时伴随着一系列委托的出现。例如，老师宣布开始上课，然后同学们开始读书、写作业等一系列行为
  - 创建事件：事件相当于委托队列

```
public delegate void delegateType();             //声明委托类型

[修饰符] event  委托类型名  事件名;
public event delegateType  ClassEvent;     //声明一个上课事件
```

- 订阅事件，将委托加入队列中

```
事件类对象名.事件名+=new 委托类型名(事件处理方法);

例如，以下语句是订阅者s1（s1是Student类对象）向事件源t（Teacher类对象）订阅ClassEvent事件，其中事件处理方法是Student类的Listener方法：
t.ClassEvent += new Teacher.delegateType(s1.Listener);
```

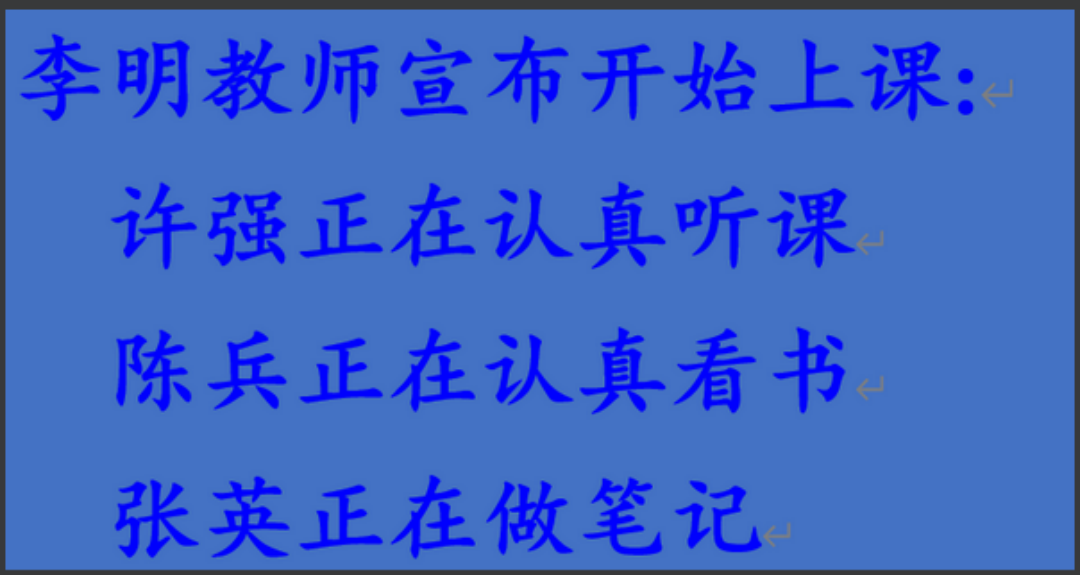
- 引发事件

```
public void Start()             //定义引发事件的方法
{
    Console.WriteLine(tname + "教师宣布开始上课:");
    if (ClassEvent != null)
        ClassEvent();    //当事件不空时引发该事件
}
```

- 演示

```
using System;
using System.Collections;
namespace aaa
{
    public class Teacher                    //教师类，事件源类
    {
        private string tname;           //教师姓名
        public delegate void delegateType();       //声明委托类型
        public event delegateType ClassEvent;//声明一个上课事件
        public Teacher(string name)      //构造函数
        { this.tname = name; }
        public void Start()               //定义引发事件的方法
        {
            Console.WriteLine(tname + "教师宣布开始上课:");
            if (ClassEvent != null)
                ClassEvent();             //当事件不空时引发该事件
        }
    }
    public class Student              //学生类,订阅者类
    {
        private string sname;     //学生姓名
        public Student(string name)    //构造函数
        { this.sname = name; }
        public void Listener()        //听课方法
        { Console.WriteLine("  学生" + sname + "正在认真听课"); }
        public void Record()  //做笔记方法
        { Console.WriteLine("  学生" + sname + "正在做笔记"); }
        public void Reading() //看书方法
        { Console.WriteLine("  学生" + sname + "正在认真看书"); }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Teacher t = new Teacher("李明");
            Student s1 = new Student("许强");
            Student s2 = new Student("陈兵");
            Student s3 = new Student("张英");
            //以下是3个学生订阅同一个事件
            t.ClassEvent += new Teacher.delegateType(s1.Listener);
            t.ClassEvent += new Teacher.delegateType(s2.Reading);
            t.ClassEvent += new Teacher.delegateType(s3.Record);
            t.Start();        //引发事件
        }
    }
}
```

该程序的执行结果如下：



- 类的使用：采用new的格式，由于C#中默认不支持指针，因此我们new的时候系统也是自动给我们分配内存而已，此时用.进行引用，而不是->
  - 对于二维数组来说，每一个元素我们都需要new，同时二维数组的地址我们也需要new，所以就是要new 两次
- datetime：日期与时间
  - 格式为：2021-7-12 7:08:59
  - 两种创建方式，一种是（年份，月份，天数，小时，分钟，秒）；另一种是用字符串转换，采用convert.todatetime（）的方式或者用.parse（很多类型都可以用这个转化
  - 自加：addDay（数量）、addSeconds（数量）等，注意，这些函数都是返回一个datetime类型，因此我们必须用 datetime= addDay（1）这种格式
  - 求现在的时间：加上 .now
  - 各类函数

    1. DayOfWeek：返回星期几，从周日算起
    2. Date：返回当前的日期

  - timespan，时间差：可以用来计算时间的差值
- get和set函数的使用（只读，只写）：

```
class gorr
        {
            public int num
            {
                get { return num; }
            }
        }
```

```
gorr e = new gorr();
            e.num = 3;        //此时会报错，显示e.num 是只读的
```

```
class getScore
    {
        class Course
        {
            public int rate;
            public string name;
            public Course(string name, int rate)
            {
                this.name = name;
                this.rate = rate;
            }
        }
        class Student
        {
            public int id;
            public string name;
            public Student(string nameb, int id)
            {
                this.name = name;
                this.id = id;
            }
        }
        class Score
        {
            public string cname;
            public string sname;
            public int score;
            public Score(string cname, string sname, int Myscore)
            {
                this.cname = cname;
                this.sname = sname;
                this.score = Myscore;
            }
        }
        public void getMyScore()
        {
            Course[] myCourse = new Course[5] { new Course("课程1", 4), new Course("课
程2", 3), new Course("课程3", 2), new Course("课程4", 6), new Course("课程5", 3) };
            Student students = new Student("李华", 1);
            Score[] myscore = new Score[5];
            int[] scores = { 92, 80, 98, 70, 89 };
            for (int i = 0; i < myCourse.Length; i++)
            {
                myscore[i] = new Score(myCourse[i].name, students.name, scores[i]);
            }
            double s1 = 0, s2 = 0, s3 = 0;
            for (int i = 0; i < myscore.Length; i++)
            {
                switch (myscore[i].score / 10)
                {
                    case 10:
                    case 9: s1 += 4 * myCourse[i].rate; break;
                    case 8: s1 += 3 * myCourse[i].rate; break;
                    case 7: s1 += 2 * myCourse[i].rate; break;
                    case 6: s1 += 1 * myCourse[i].rate; break;
                    default: break;
                }
                s2 += myscore[i].score * myCourse[i].rate;
                s3 += myCourse[i].rate;
            }
            s1 = s1 / s3;
            s2 = s2 * 4 / s3 / 100;
            Console.WriteLine($"学号：{students.id} 姓名：{students.name}\n");
            Console.WriteLine($"\t课程名\t学分\t分数\n");
            for (int i = 0; i < myscore.Length; i++)
```

```
            {
                 Console.WriteLine($"\t{myscore[i].cname}\t{myCourse[i].rate}\t{myscore[i].score}\n");
            }
            Console.WriteLine($"常见算法GPA={s1}，标准算法GPA={s2}");
        }
    }
class exam
    {
        class student
        {
            public DateTime startTime;
            public TimeSpan needTime;
            public string name;
            public delegate void finish(string name);
            public event finish finishEvent;
            public student(TimeSpan needTime,string name)
            {
                this.needTime = needTime;
                this.name = name;
            }
            public void startExam()
            {
                Console.WriteLine($"学生{name}在{startTime+needTime}时开始答题...\n");
            }
            public void endExam()
            {
                Console.WriteLine($"经过一段时间...\n");
                if (finishEvent != null)
                {
                    finishEvent(name);
                }
            }
        }
        class teacher
        {
            public DateTime startTime;
            public delegate void beginExam();
            public event beginExam beginEvent;
            public teacher(DateTime startTime)
            {
                this.startTime = startTime;
            }
            public void declareStartExam()
            {
                Console.WriteLine($"教师宣布开始考试\n");
                if (beginEvent != null)
                {
                    beginEvent();
                }
            }
            public void declareEndExam(string name)
            {
                Console.WriteLine($"\t学生{name}完成考试，老师收卷\n");
            }
        }
        public void enterExam()
        {
            student[] stu = new student[5];
            stu[0] = new student(TimeSpan.Parse("1:01:00"), "陈华");
            stu[1] = new student(TimeSpan.Parse("1:03:00"), "王丽");
            stu[2] = new student(TimeSpan.Parse("1:05:00"), "刘畅");
            stu[3] = new student(TimeSpan.Parse("1:06:00"), "张军");
            stu[4] = new student(TimeSpan.Parse("1:08:00"), "许源");
            teacher Myteacher = new teacher(DateTime.Parse("2021-7-12 7:08:59"));
            for(int i = 0; i < stu.Length; i++)
            {
                Myteacher.beginEvent += new teacher.beginExam(stu[i].startExam);
            }
            int n = 0;
            TimeSpan sumTime = TimeSpan.Parse("0:00:00");
            Myteacher.declareStartExam();
            while (true)
            {
```

```
                if (n == 5) break;
                for(int i = 0; i < stu.Length; i++)
                {
                    if (sumTime.Equals(stu[i].needTime))
                    {
                        n++;
                        stu[i].finishEvent +=
new  student.finish(Myteacher.declareEndExam);
                        stu[i].endExam();
                    }
                }
                sumTime = sumTime.Add(TimeSpan.Parse("0:00:01"));
            }
        }
    }
    public class Getsalary
    {
        class Coder
        {
            string name;
            string id;
            int salary;
            public Coder(string name,string id,int salary)
            {
                this.name = name;
                this.id = id;
                this.salary = salary;
            }
            public void intro()
            {
                Console.WriteLine($"程序员姓名：{name}\n工号：{id}\n");
            }
            public void showSalary()
            {
                Console.WriteLine($"基本工资为：{salary}，奖金：无\n");
            }
            public void work()
            {
                Console.WriteLine("正在努力写代码......\n");
            }
        }
        class Manager
        {
            string name;
            string id;
            int salary;
            public Manager(string name, string id, int salary)
            {
                this.name = name;
                this.id = id;
                this.salary = salary;
            }
            public void intro()
            {
                Console.WriteLine($"经理姓名：{name}\n工号：{id}\n");
            }
            public void showSalary()
            {
                Console.WriteLine($"基本工资为：{salary}，奖金：{salary*0.2}\n");
            }
            public void work()
            {
                Console.WriteLine("正在努力的做着管理工作，分配任务，检查程序员提交上来的代
码.....\n=================\n");
            }
        }
        public void printBasic()
        {
            Coder coder = new Coder("Kobe", "0025", 10000);
            Manager manager = new Manager("James", "9527", 15000);
            manager.intro();
            manager.showSalary();
            manager.work();
```

```
            coder.intro();
            coder.showSalary();
            coder.work();
        }
    }
static void Main(string[] args)
        {
            getScore a = new getScore();
            a.getMyScore();
            clare b = new clare();
            b.declare();
            Getsalary c = new Getsalary();
            c.printBasic();
        }
```