**2\_4 Random有什么作用？**

用来产生随机数的。  
具体用法参考：<https://docs.microsoft.com/zh-cn/dotnet/api/system.random>

**2\_5 什么是字符串？C#中的字符串分为哪两类？**

* 字符串是用于表示文本的字符的有序集合。
* C#支持两种形式的字符串文字：常规字符串文本和逐字字符串文本。  
  逐字字符串是不需要转义的字符串，就像文件名一样：string myFileName = "C:\\myfolder\\myfile.txt";与string myFileName = @"C:\myfolder\myfile.txt";等价，@符号意味着从字面上读取该字符串，否则不要解释控制字符。  
  常规字符串文字由包含在双引号(如“hello”)中的零个或多个字符组成，并且可以包括简单转义序列(例如\t表示制表符字符)以及十六进制和Unicode转义序列。  
  逐字串文字由一个@字符、一个双引号字符、零个或多个字符和一个结束双引号字符组成。一个简单的例子是“Hello”。在逐字字符串文字中，分隔符之间的字符是逐字解释的，唯一的例外是引号-转义序列。特别是，简单转义序列以及十六进制和Unicode转义序列不使用逐字字符串文字处理。逐字串文字可能跨越多行。  
  换句话说，@“逐字字符串文字”中唯一的特殊字符是双引号。如果您希望编写包含双引号的逐字字符串，则必须编写两个双引号。所有其他字符都是按字面解释的。  
  甚至可以在逐字字符串文本中有文字新行。在常规字符串文本中，不能有文字新行。相反，您必须使用例如"\n".  
  逐字字符串文本通常用于在源代码中嵌入文件名和正则表达式，因为这些类型的字符串中的反斜杠很常见，如果使用正则字符串文字，则需要转义。

字符串参考：<https://docs.microsoft.com/zh-cn/dotnet/api/system.string>  
规则字符串和逐字字符串参考：<http://www.imooc.com/wenda/detail/571653>

**2\_6 String类的Concat()方法与StringBuffer类的Append()方法都可以连接两个字符串，它们之间有何不同？**

一般来讲，String类Concat方法的拼接速度远小于StringBuffer类的Append方法  
原因：

1. concat()方法：  
   concat源代码：

 public String concat(String str) {

        // 追加的字符串长度

        int otherLen = str.length();

        // 如果追加的字符串长度为0，则不做修改，直接返回原字符串

        if (otherLen == 0) {

            return this;

        }

        // 获取原字符串的字符数组value的长度

        int len = value.length;

        // 将原字符串的字符数组value放到buf字符数组中

        char buf[] = Arrays.copyOf(value, len + otherLen);

        // 将追加的字符串转化成字符数组，添加到buf中

        str.getChars(buf, len);

        // 产生一个新的字符串并返回

        return new String(buf, true);

    }

整体是一个数组的拷贝，虽然在内存中是处理都是原子性操作，速度非常快，但是，最后的return语句创建一个新String对象，也就是每次concat操作都会创建一个新的String对象，这也是限制concat方法速度的原因。

1. append()方法  
   append源代码：

public AbstractStringBuilder append(String str) {

        // 如果是null值，则把null作为字符串处理

        if (str == null)

            return appendNull();

        int len = str.length();

        // 追加后的字符数组长度是否超过当前值

        ensureCapacityInternal(count + len);

        // 字符串复制到目标数组

        str.getChars(0, len, value, count);

        count += len;

        return this;

    }

   private AbstractStringBuilder appendNull() {

        int c = count;

        ensureCapacityInternal(c + 4);

        final char[] value = this.value;

        value[c++] = 'n';

        value[c++] = 'u';

        value[c++] = 'l';

        value[c++] = 'l';

        count = c;

        return this;

    }

    private void ensureCapacityInternal(int minimumCapacity) {

        // overflow-conscious code

        if (minimumCapacity - value.length > 0)

            expandCapacity(minimumCapacity);  // 加长，并作数组拷贝

    }

整个append方法都在做字符数组的处理，加长，拷贝等，这些都是基本的数据处理，整个方法内并没有生成对象。只是最后toString返回一个对象而已。需要注意的是，append()方法返回的是一个StringBuilder(or StringBuffer)对象实例。

C# 字符串的三种拼接方法性能比较可参考：<https://blog.csdn.net/COCO56/article/details/102534600>

**3\_2 编写程序，接受用户输入的一个字符串和一个字符，把字符串中所有指定的字符删除后输出。**

using System;

namespace \_3\_2\_删除字符

{

    class Program

    {

        static void Main(string[] args)

        {

            string s = "asdasd";

            Console.Write("Before deletion: s=");

            Console.WriteLine(s);

            Console.WriteLine("\nInput the char that you want to delete in s.");

            string a = Console.ReadLine().ToString();

            Console.Write("\nAfter deletion: s=");

            s = s.Replace(a, "");

            Console.WriteLine(s);

            Console.WriteLine("\nPress any key to quit.");

            Console.ReadKey();

        }

    }

}

**3\_3 编程判断一个字符串是否是回文。**

using System;

using System.Collections.Generic;

namespace \_3\_3\_判断一个字符串是否是回文

{

    class Program

    {

        static void Main(string[] args)

        {

            //用栈和队列

            Console.WriteLine("请输入要进行判断的字符串");

            string str = Console.ReadLine();

            Stack<char> stack = new Stack<char>();

            Queue<char> queue = new Queue<char>();

            for (int i = 0; i < str.Length; i++)

            {

                stack.Push(str[i]);

                queue.Enqueue(str[i]);

            }

            bool isPlalindrome = true;//isPlalindrome初始化为真

            while (stack.Count > 0)

            {

                if (stack.Pop() != queue.Dequeue())//只要发现有一个不等，就把isPlalindrome设置为假

                {

                    isPlalindrome = false;

                    break;//发现有一个不等就退出循环

                }

            }

            Console.WriteLine("字符串是回文串：" + isPlalindrome);

            Console.ReadKey();

        }

    }

}

**3\_13 从键盘上输入10个整数，并放入一个一维数组中，然后将其前5个元素与后5个元素对换，即：第1个元素与第10个元素互换，第2个元素与第9个元素互换…第5个元素与第6个元素互换。分别输出数组原来各元素的值和对换后各元素的值。**

using System;

namespace \_3\_13\_元素对换

{

    class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Input 10 integers...");

            int[] a = new int[10];

            for (int i = 0; i < 10; i++)

                a[i] = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Before exchange :");

            foreach (var i in a)

                Console.Write(i + ",");

            for (int i = 0; i < 5; i++)

            {

                int t = a[i];

                a[i] = a[9 - i];

                a[9 - i] = t;

            }

            Console.WriteLine("\nAfter exchange :");

            foreach (var i in a)

                Console.Write(i + ",");

            Console.WriteLine("\nPress any key to quit.");

            Console.ReadKey();

        }

    }

}

/\*测试数据

1

2

3

4

5

6

7

8

9

0

\*/