

字符串与时间

王忠民

内容简介

1. String 类
2. StringBuilder 类
3. 正则表达式
4. 日期和时间处理

1 字符串

字符串是 C# 编程语言中最常用到的数据类型，很多操作都需要使用字符串来完成。

1.1 String 类的构造函数

1.2 比较字符串

1.3 提取字符串

1.4 拆分字符串

1.5 定位与查找字符串

1.6 格式化字符串

1.7 String 类中的其它常用方法

1.1 String 类的构造函数

字符串是 Unicode **字符的有序集合**，用于表示文本。String 对象是 System.Char 对象的有序集合。System.String 是一个类，专门用于存储字符串，它允许对字符串进行多种操作。

String 类的 3 种构造函数，分别按不同的方式对字符串进行初始化

名称	说明
String(Char[])	将 String 类新实例初始化为由 Unicode 字符数组指示的值
String(Char,Int32)	将 String 类的新实例初始化为由重复指定次数的指定 Unicode 字符指示的值
String(Char[],Int32,Int32)	将 String 类的新实例初始化为由 Unicode 字符数组、该数组内的起始字符位置和一个长度指示的值

例：声明一个字符数组和字符串，然后再声明 4 个 String 类的对象，分别为 str0、str1、str2 和 str3。

```
static void Main(string[] args)
{
    char[] charSet = {'计','算','机','软','件'};
    string str = "好好学习天天向上";

    string str0 = str;
    string str1 = new string(charSet); // 获取由 charStr 字符数组指示的值
    string str3 = new string('F',6); // 获取字符 F，并且将字符 F 重复 6 次
    string str2 = new string(charSet, 2, 3); // 获取 charStr 字符数组中从 2
    开始的 3 个字符

}
```

◆ + 运算符用于连接字符串。例：`string a = "good " + "morning";` 这将创建一个包含“good morning”的字符串对象。

◆ 字符串是不可变的，即：字符串对象在创建后，尽管从语法上看您似乎可以更改其内容，但事实上并不可行。例如下面的代码，编译器实际上会创建一个新字符串对象来保存新的字符序列，且新对象将赋给 `b`。然后字符串“he”将适宜于垃圾回收。

```
string b = "he";
```

```
b += "llo";
```

◆ 相等运算符 (`==` 和 `!=`) 是为了比较 `string` 对象 (而不是引用) 的值。这使得对字符串相等性的测试更为直观。例如：

```
string a = "hello"; string b = "he";
```

```
b += "llo";
```

```
Console.WriteLine(a == b);
```

```
Console.WriteLine((object)a == (object)b);
```

这将先显示“True”，然后显示“False”，因为字符串的内容是相同的，但是 `a` 和 `b` 引用的不是同一个字符串实例。

◆ [] 运算符可以用于对 string 的各个字符的只读访问。例：

```
string str = "test";
```

```
char x = str[2];      // x = 's'; 对 str[2] 赋值会引起编译错误
```

◆ 字符串文本可包含任何字符。包括转义序列。

- 以反斜线“\”开头，后跟一个或几个字符，具有特定含义。如‘\n’表示换行
- 消除紧随其后的字符的原有含义，故称“转义”字符。
- 主要用来表示那些用一般字符不便于表示的控制代码、不可见的字符

字符串转义序列

转义序列	字符名称	Unicode 编码
\'	单引号	0x0027
\"	双引号	0x0022
\\	反斜杠	0x005C
\0	Null	0x0000
\a	警报	0x0007
\b	Backspace	0x0008
\n	换行	0x000A
\r	回车	0x000D
\t	水平制表符	0x0009
\u	Unicode 转义序列	\u0041 = "A"

◆ 转义序列 \\ 表示反斜杠， \u0066 表示字母 f。

```
String a = "\\u0066";
```

```
Console.WriteLine(a);    // 输出结果： \f
```


原义字符串以 @ 开头并且也用双引号引起来。

优势在于不处理转义序列，因此很容易写入，例如完全限定的文件名就是原义字符串：

@ "c:\Docs\Source\a.txt"

等效于：

"c:\\Docs\\Source\\a.txt"

1.2 比较字符串

比较字符串是根据字符串中的字母顺序来处理的。可以使用“==”符号比较两个字符串是否相等，也可以使用 String 提供的方法进行比较。

1.Compare() 方法和 CompareTo() 方法

Compare() 方法和 CompareTo() 方法都可以用来比较字符串，但是比较方式不同。**Compare 方法**是 String 类的静态方法，包括 6 种重载方法。CompareTo 不是静态方法，可以通过一个 String 对象调用。

```
str1.CompareTo(str2);
```

```
string.Compare(str1, str2);
```

Compare() 和 CompareTo() 的返回值类型都是 int，有 3 种情况：

- 当 $str1 > str2$ 时，返回 1
- 当 $str1 == str2$ 时，返回 0
- 当 $str1 < str2$ 时，返回 -1

■ Compare() 的重载：

Compare(str1,str2)

Compare(str1,str2,boolean)

Compare(str1,index1,str2,index2,length)

Compare(str1,index1,str2,index2,length,boolean)

boolean 表示是否忽略比较字符的大小写。 index1 和 index2 表示整数偏移量， length 表示要比较的字符的数量

```
static void Main(string[] args)
{
    string str0 = " 好好学习天天向上 ";
    string str1 = " 我学习英语 OK ";
    string str2 = " 我学习英语 ok ";
    Console.WriteLine(string.Compare(str0, str1));
    Console.WriteLine(string.Compare(str0,2, str1,1,2));
    Console.WriteLine(string.Compare(str1, str2, false));
    Console.Read();
}
```

2.Equals()

== 和 Equals 都可以用来比较字符串的值是否相等。返回值是 bool 类型。如果相等返回 true ，否则返回 false 。

例：使用 Equals() 方法判断用户输入的用户名和密码是否相等。

```
if (username.Equals(userpass) && username == userpass)
    Console.WriteLine(" 恭喜您，登录信息成功！ ");
else
    Console.WriteLine(" 很抱歉，您的用户名和密码有误 ");
```

3.Contains()

String.Contains() 方法用于确定某个字符串中是否包含另一个字符串，该方法返回一个 bool 类型，如果包含则返回 true，否则返回 false。此方法区分大小写，从字符串的第一个字符位置开始，一直搜索到最后一个字符位置。

例：声明并初始化两个字符串变量 myColor 和 color，然后使用 Contains() 方法判断 myColor 变量中是否包含 color 变量的内容。

```
string myColor = "我最喜欢的颜色是红色、绿色和白色。";  
string color = "白色";  
if (myColor.Contains(color))  
{  
    Console.WriteLine("返回结果为 true");  
}  
else  
{  
    Console.WriteLine("返回结果为 false");  
}
```

1.3 提取字符串

提取字符串是指从字符串中获取指定的字符串。

```
string hobby = " My name is Lily "
```

假设只想得到字符串“ Lily”，这时候需要用到 String 类中的 Substring() 方法。该方法有两个重载：

- String.Substring(int index1) 从指定位置开始返回要提取的字符串
- String.Substring(int index1,int length) 第一个参数指定子字符串开始提取的位置；第二个参数指定字符串中的字符数

```
static void Main(string[] args)
{
    string mysub = " 柏拉图说：人生短短几十年，不要给自己留下了什么遗憾... 人生的苦闷有二，一是没有被满足，二是它得到了满足。 ";
    string sub1 = mysub.Substring(5);
    String sub2 = mysub.Substring(7, 5);
    Console.WriteLine(" 整段文字是： " + mysub+"\n");
    Console.WriteLine(" 提取的第一句话是： " + sub1+"\n");
    Console.WriteLine(" 提取的第二句话是： " + sub2);
    Console.ReadLine();
}
```

1.4 拆分字符串

拆分字符串是指一个字符串被指定的分隔字符或字符串分隔为多个字符串，返回由子字符串组成的字符串数组。实现拆分字符串的功能需要使用 String 类的 Split() 方法，该方法有多种重载形式：

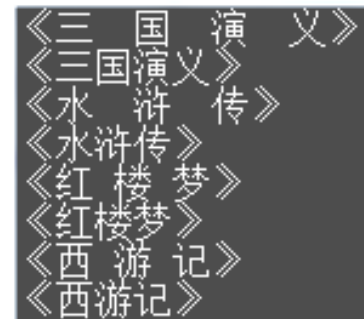
- string[] Split(char[] separator)
- string[] Split(char[] separator, int count)
- string[] Split(char[] separator, StringSplitOptions options)
- string[] Split(char[] separator, int count, StringSplitOptions options)
- string[] Split(string[] separator, StringSplitOptions options)
- string[] Split(string[] separator, int count, StringSplitOptions options)

separator 参数表示分隔字符或字符串数组；count 表示要返回的字符串的最大数量；options 表示字符串分隔选项，是一个枚举类型，主要有两个值：

- System.StringSplitOptions.RemoveEmptyEntries 省略返回数组中的空元素
- System.StringSplitOptions.None 要包含返回的数组中的空元素

例：mysub 存储四大名著的名称（含空格），中间用逗号分隔。用 Split() 方法拆分：

```
string mysub = " 《三 国 演 义》 ， 《水 浒 传》 ，  
                《红 楼 梦》 ， 《西 游 记》 ";  
string[] subarray = mysub.Split(' ' , ',');  
foreach (string sub in subarray)  
{ Console.WriteLine(sub);  
    string[] subchar = sub.Split(' ');  
    foreach (string item in subchar)  
        Console.WriteLine(item);  
    Console.WriteLine();  
}  
Console.Read();
```



```
《三 国 演 义》  
《三 国 演 义》  
《水 浒 传》  
《水 浒 传》  
《红 楼 梦》  
《红 楼 梦》  
《西 游 记》  
《西 游 记》
```

```
string mysub = " 《三 国 演 义》 ，第二 《水 浒 传》 ， 《红 楼 梦》 ， 《西 游 记》  
string[] sep = { "》", " 《" };  
string[] subarray = mysub.Split(sep, System.StringSplitOptions.None);  
foreach (string sub in subarray)  
{ Console.WriteLine(sub); }
```


1.5 定位与查找字符串

定位与查找字符串是指从字符串中查找指定的字符或字符串。

方法	说明
IndexOf()	返回指定 字符 或 字符串 第一次出现的索引位置 (从 0 开始计算)。如果没有找到，则返回 -1
IndexOfAny()	返回字符数组中元素第一次出现的索引位置
LastIndexOf()	返回指定字符或字符串的最后一个索引位置。
LastIndexOfAny()	返回字符数组中元素最后出现的一个位置。
StartsWith()	判断字符串是否以指定子字符串开始，返回 true 或 false
EndsWith()	判断字符串是否以指定子字符串结束，返回 true 或 false

```
string str = " 我喜欢看的是那些伟大的著作 ";  
Console.WriteLine("' 第一次出现的位置 : " + str.IndexOf(' '));  
Console.WriteLine("' 那些 \" 最后出现的位置是 : " + str.LastIndexOf(" 那些 "));  
Console.WriteLine(" 是否以 \" 著作 \" 结尾 " + str.EndsWith(" 著作 "));
```

1.6 格式化字符串

格式化字符串是指将字符、数字或日期等转化为指定格式的字符串。主要使用 `Format()` 方法，它是 **String 类的静态方法**。多种重载形式：

- `string Format(string format, object arg0)`
将指定字符串中的一个或多个格式项替换为对象 `arg0` 的字符串表示形式
- `string Format(string format, params object[] args)`
将数组中的对象格式化为 `format` 表示的形式
- `string Format(string format, object arg0, object arg1)`
将指定的对象 `arg0` 和 `arg1` 格式化为 `format` 表示的形式
- `string Format(string format, object arg0, object arg1, object arg2)`
将指定的对象 `arg0`、`arg1` 和 `arg2` 格式化为 `format` 表示的形式

```
string s1 = " 北京 "; string s2=" 上海 ";  
String s = string.Format(" 两个城市是： {0} 和 {1}", s1, s2);
```

```
string[] s0 = {" 北京 ", " 上海 ", " 广州 "};  
String s = string.Format(" 城市名是： {0}{1}{2}", s0);
```

日期和时间的格式化

Format() 方法除了可以格式化字符串外，还可以格式化日期和时间。 .NET Framework 提供了一种被称为“标准日期和时间格式字符串”的格式字符串。常用的日期和时间格式化说明符主要包括 8 类：

- d 短日期模式， 2010/10/1
- D 长日期模式， 2010 年 10 月 1 日
- f 长日期和短时间模式的组合，由空格分隔
- F 长日期和长时间模式的组合，由空格分隔
- M 或 m 月日模式
- t 短时间模式，“HH:mm”
- T 长时间模式，“HH:mm:ss”
- Y 或 y 年月模式

```
Time now = DateTime.Now;  
sole.WriteLine(now);  
sole.WriteLine();  
g ss = String.Format("{0:d}", now); // 短日期  
sole.WriteLine(ss);  
String.Format("{0:D}", now); // 长日期  
sole.WriteLine(ss);  
String.Format("{0:m}", now); // 月日模式 ( M )  
sole.WriteLine(ss);  
String.Format("{0:y}", now); // 年月模式 ( Y )  
sole.WriteLine(ss);  
String.Format("{0:t}", now); // 短时间  
sole.WriteLine(ss);  
String.Format("{0:T}", now); // 长时间  
sole.WriteLine(ss);  
String.Format("{0:f}", now); // 长日期短时间  
sole.WriteLine(ss);  
String.Format("{0:F}", now); // 长日期长时间  
sole.WriteLine(ss);
```

2014/11/12 18:16:37

2014/11/12

2014年11月12日

11月12日

2014年11月

18:16

18:16:37

2014年11月12日 18:16

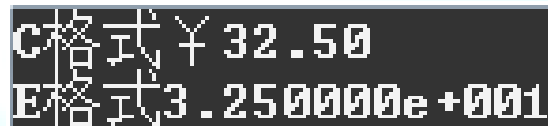
2014年11月12日 18:16:37

数字格式化

使用 Format() 方法除了可以格式化字符串和日期外，还可以格式化数字。常用的数字格式化说明符主要包括 8 类：

- C 或 c 将数字转换为表示货币金额的字符串
- D 或 d 将整数数字转换为十进制数字的字符串
- E 或 e 将数字转换为科学计数法的字符串，即 "-d.ddd...E+dd d" 或 "-d.ddd...e+ddd" 形式
- F 或 f 将数字转换为表示浮点数的字符串，即 "-ddd.ddd..." 形式
- G 或 g 根据数字类型以及是否存在精度说明符，数字会转换为定点或科学计数法的最紧凑形式
- N 或 n 用逗号分割千位的数字，比如 1234 将会被变成 1,234
- P 或 p 将数字转换为表示百分比的字符串
- X 或 x 将整数数字转换为十六进制数字的字符串

```
double d = 32.5 ;  
Console.WriteLine("C 格式 {0:c}", d);  
Console.WriteLine("E 格式 {0:e}", d);
```



```
C格式 ¥32.50  
E格式 3.250000e+001
```

1.7 String 类中的其它常用方法

String 类中还有一些常用的方法，例如

ToUpper()、ToLower()、Copy()、CopyTo()、Replace()、Trim()、Contact() 以及 Join() 等等。

1.ToUpper() 方法和 ToLower() 方法

ToUpper() 方法表示将字符串全部转换为大写；ToLower() 方法表示将字符串全部转换为小写。

```
string intro = "Hello,My name is Jim Green,I'm a student.";
string upper = intro.ToUpper();
Console.WriteLine(" 原句为：{0}", intro);
Console.WriteLine(" 全部转换为大写：{0}", upper);
Console.WriteLine(" 全部转换为小写：{0}", intro.ToLower());
```

```
原句为: Hello,My name is Jim Green,I' m a student.
全部转换为大写: HELLO,MY NAME IS JIM GREEN,I' M A STUDENT.
全部转换为小写: hello,my name is jim green,i' m a student.
```

1.7 String 类中的其它常用方法

2.Copy() 方法和 CopyTo() 方法

Copy() 将一个字符串的内容完整的复制到一个新的字符串中，在此过程中，会创建一个与指定的 String 类具有相同值的 String 类的新实例

CopyTo() 从字符串中复制指定数量的字符到另外一个字符数组。

SourceString.CopyTo (SourceStart, DestCharArray, DestStart, count)

```
string oldStr = " 我十分高兴！ ";
char[] charArray = { '晴', '朗', '的', '一', '天' };
Console.Write(" 原始字符串是： " + oldStr+"\r\n 原始字符数组");
foreach (char c in charArray)
    Console.Write(c);
Console.WriteLine();
string newStr = String.Copy(oldStr);
Console.WriteLine("Copy 的结果： "+newStr);
oldStr.CopyTo(2, charArray, 1,3);
Console.Write("CopyTo 的结果： ");
foreach (char c in charArray)
```

```
原始字符串是： 我十分高兴！
原始字符数组是： 晴朗的一天
Copy的结果： 我十分高兴！
CopyTo的结果： 晴分高兴天
```

3.Replace() 方法

将字符串中指定的字符替换为新的字符，或者将指定的字符串替换为新的字符串。Replace() 方法有两种重载形式：

- string Replace(char oldChar,char newChar) **替换字符**

- string Replace(string oldValue,string newValue) **替换字符串**

oldChar 参数表示被替换的字符，newChar 参数表示替换后的字符；

oldValue 参数表示被替换的字符串，newValue 参数表示替换后的字符串。

```
string oldstr = "人们都说我的生活好快乐，我也认为自己真的快乐。";  
string newstr = oldstr.Replace("快乐","happy");  
Console.WriteLine("替换前的句子： " + oldstr);  
Console.WriteLine("替换后的句子： " + newstr);
```

替换前的句子： 人们都说我的生活好快乐，我也认为自己真的快乐。

替换后的句子： 人们都说我的生活好happy，我也认为自己真的happy。

4.Concat() 和 Join() 都表示连接字符串

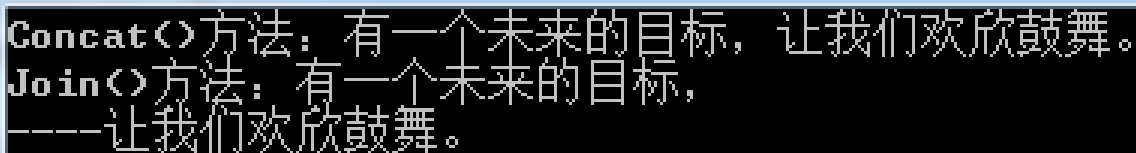
Concat() 将一个或多个字符串对象连接为一个新的字符串；

Join() 将指定的字符串**数组**中的所有字符串连接为一个新的字符串，而且被连接的各个字符串被指定分隔字符串分隔。Join 的格式：

string.Join(进行连接的分隔符，要连接的字符串数组)

例： 声明两个字符串，分别用 Concat() 和 Join() 将它们连接起来：

```
string stringFirst = " 有一个未来的目标， ";  
string stringSecond = " 让我们欢欣鼓舞。 ";  
string concatstring = string.Concat(stringFirst, stringSecond);  
string joinstring = string.Join("\n---", new string[] { stringFirst, stringSecond });  
Console.WriteLine("Concat() 方法： " + concatstring);  
Console.WriteLine("Join() 方法： " + joinstring);
```



```
Concat()方法： 有一个未来的目标， 让我们欢欣鼓舞。  
Join()方法： 有一个未来的目标，  
----让我们欢欣鼓舞。
```

5.Trim() 方法

Trim() 删除最前和最后的空格，中间部分的空格不变。

```
String previousString = " I am a string! ";  
String currentString = previousString.Trim();
```

“ I’m a string ! ” → “I’m a string!”

2 StringBuilder 类

.NET Framework 中提供了另外一个类 **StringBuilder**，它和 **String** 类相反，表示**动态字符串**，使用这个类可以动态的插入字符串、追加字符串以及删除字符串。

2.1 追加字符串

2.2 插入字符串

2.3 替换字符串

2.4 移除字符串

2.1 追加字符串

将指定的字符或字符串追加到字符串的末尾。3 种方法：

- Append() 方法 将指定的字符或字符串追加到字符串的末尾
- AppendLine() 方法 追加指定的字符串完成后，还追加一个换行符号
- AppendFormat() 方法 先格式化被追加的字符串，然后将其追加到末尾

Append() 方法几乎可以将任意值追加到字符串后面，如 int、char、char[]、decimal、long、bool 和 short 等等。常用的几种重载形式：

方法	含义
Append(bool value)	此实例的结尾追加指定的布尔值的字符串表示形式
Append(char value)	此实例的结尾追加指定 Unicode 字符的字符串形式
Append(double value)	实例的结尾追加指定的双精度浮点数的字符串形式
Append(int value)	实例的结尾追加指定的 32 位有符号整数字符串形式
Append(string value)	在此实例的结尾追加指定字符串的副本
Append(string value,int index,int count)	在此实例的结尾追加指定子字符串的副本

```
String s1 = "This is the first string.";
String s2 = "The second is better.";
StringBuilder sb = new StringBuilder();
sb.Append(s1);
sb.Append(s2);
Console.WriteLine(sb);
```

A screenshot of a console window with a black background and white text. The text displayed is "This is the first string.The second is better." followed by a newline character. The window has a standard title bar at the top.

This is the first string.The second is better.

2.2 插入字符串

插入字符串可以使用 Insert() 方法，几乎可以将任意类型的值插入到字符串的指定位置。Insert() 方法常用的重载形式如表所示。

方法	含义
Insert(int index,double value)	将双精度浮点数的字符串形式插入到此实例的指定位置
Insert(int index,string value)	将字符串插入到此实例中的指定字符位置
Insert(int index,char value)	将指定 Unicode 字符的字符串形式插入到指定位置
Insert(int index,bool value)	将布尔值的表现形式插入到此实例中的指定字符位置
Insert(int index,int vlaue)	将 32 位带符号整数的字符串形式插入到指定字符位置
Insert(int index,string value,int count)	将指定字符串的一个或更多副本插入到此实例中的指定字符位置

例：在歌词开头处添加歌名和演唱者，并且在结尾处再添加一句歌词。

```
StringBuilder sbuilder = new StringBuilder(" 昨天所有的荣誉，已变成遥远的回忆。 \n");
sbuilder.Insert(0, " 从头再来 ( 刘欢 )\n");
sbuilder.Append(" 勤勤恳恳已度过半生，今夜重又走入风雨。 ");
Console.WriteLine(sbuilder);
```

```
从头再来<刘欢>
昨天所有的荣誉，已变成遥远的回忆。
勤勤恳恳已度过半生，今夜重又走入风雨。
```

2.3 替换字符串

前面已经介绍过 String 类的 Replace() 方法，它用来表示替换字符或字符串。同样地，在 StringBuilder 类中也提供了 Replace() 方法

。

```
sbuilder.Replace("刘欢", "liuhuan"); // 将 sbuilder 中的“刘欢”都换成 liuhuan
```

2.4 移除字符串

从字符串中指定位置开始移除其后指定数量的字符。StringBuilder 类中提供了 Remove() 方法，该方法中有两个参数，第一个参数表示开始移除字符的位置，第二个参数表示要移除的字符数量。

```
StringBuilder strbuilder = new StringBuilder();  
strbuilder.Insert(0, " 冬天来了，春天还会远吗？");  
strbuilder.Remove(4,6);    // 从 4 号字符移除 6 个  
Console.WriteLine(strbuilder.ToString());
```


3 正则表达式

正则表达式是指由普通字符（例如字符 A 到 Z）以及特殊字符（称为元字符）组成的文字模式。

它可以看作是一个模板，将某个字符模式与所搜索的字符串进行匹配。

正则表达式中有一套语法匹配规则，包括字符匹配、重复匹配、字符定位、转义匹配、字符分组、字符替换和字符决策。

Visual Studio 中提供了很多用于处理正则表达式的类，如 Regex 类。在使用的时候，需要引入 `System.Text.RegularExpressions` 命名空间。

Regex 是正则表达式的基本类，它提供了 `IsMatch()`、`Replace()`、`Match()` 和 `Split()` 等方法。

Regex 类

1. 字符匹配 IsMatch

该方法返回一个 bool 类型，匹配成功返回 true，否则返回 false。

例：验证输入的电话号码是否合法：北京区号 010 后面紧跟 8 位数字。

```
using System.Text.RegularExpressions ;  
string Regextext = "010\\d{8}";  
string tel = Console.ReadLine();  
if (Regex.IsMatch(tel, Regextext))  
    Console.WriteLine(" 恭喜您，您的电话号码有效。 ");
```

```
if ( Regex.IsMatch(tel, "010[3-6]")) // 010 后面跟一位 3~6 之间的数字  
if ( Regex.IsMatch(str, "\\D"))      // 一个非数字  
if ( Regex.IsMatch(addr, "w{3}") )   // 包含 www
```

正则表达式——字符匹配

字符匹配表示一个范围内的字符是否匹配。例如，[A-Z] 表示匹配字母 A-Z 之间的任意字符。它主要用于检查一个字符串中是否含有某种子字符串、将匹配的子字符串做替换或从字符串中取出符合某个条件的子字符串等。

字符语法	语法解释	语法例子
\d	匹配 0-9 之间的数字	\d 可以匹配 1 ，不可以匹配 10
\D	匹配非数字	\D 匹配 F ，不匹配 3
\w	单个字母 / 数字	\w\w\w 匹配 Ac3 ，不匹配 @#A
\W	非字母 / 数字	\W\W 匹配 @# ，不匹配 da
\s	匹配空白字符	\D\s\d 匹配 D 3 ，不匹配 33
\S	匹配非空字符	\S\S\S 匹配 abc ，不匹配 a(空格)b
.	匹配任意字符	... 匹配 C\$3 ，但是不匹配换行
[...]	匹配括号中的任意字符	[3-5] 匹配 3 、 4 和 5 ，不匹配 6
[^...]	匹配非括号中的字符	[^3-5] 匹配 6 ，不匹配 3~5 的字符

正则表达式——重复匹配

在多数情况下可能要匹配一个单词或一组数字。重复匹配就是用来确定前面的内容重复出现的次数。

字符语法	语法解释	语法例子
$\{n\}$	n 是非负整数，匹配 n 次字符	$\backslash c\{3\}$ 表示匹配 $\backslash c\backslash c\backslash c$ ，不匹配 $\backslash c\backslash c$ 或 $\backslash c\backslash c\backslash c\backslash c$
$\{n, \}$	n 是非负整数，匹配 n 次和 n 次以上	$\backslash c\{2, \}$ 表示匹配 $\backslash c\backslash c$ 和 $\backslash c\backslash c\backslash c$ 以上，不匹配 $\backslash c$
$\{n, m\}$	n 是非负整数，匹配 n 次以上和 m 次以下	$\backslash c\{1, 3\}$ 匹配 $\backslash c$ 、 $\backslash c\backslash c$ 和 $\backslash c\backslash c\backslash c$ ，不匹配 $\backslash c\backslash c\backslash c\backslash c$
$?$	重复匹配 0 次或 1 次	$3?$ 匹配 3 或 0，不匹配非 3 和 0，等价于 $\{0, 1\}$
$+$	匹配 1 次或多次	$\backslash s^+$ 匹配一个以上的 $\backslash s$ ，等价于 $\{1, \}$
$*$	匹配 0 次以上	$\backslash s^*$ 匹配 0 以上 $\backslash s$ ，不匹配非 $N^*\backslash s$ ，等价于 $\{0, \}$

正则表达式——字符定位

符号	描述	例	匹配
^	从字符串的开头开始。	^\d{3}	"567-777-" 中的 "567"
\$	匹配必须出现在字符串的末尾。	-\d{4}\$	"8-12-2012" 中的 "-2012"
\b	单词的开头或结尾，也就是单词的分界处，只匹配一个位置	\ba\w*\b	匹配以字母 a 开头的单词——先是某个单词开始处 (\b)，然后是字母 a，然后是任意数量的字母 (\w*)，最后是单词结束处 (\b)

例： 判断是否是数字

```
string s = Console.ReadLine();  
if (Regex.IsMatch(s, "^[0-9]*$")) // 定位在开头，匹配 0 次以上必须出现在字符串末尾
```

```
    Console.WriteLine("ok");
```

前面判断北京电话号码的例子，如何确保：输入的全是数字，区号 010 之前和 8 位号码之后都没有其它内容？

Regex 类

2.Replace() 方法

在正则表达式中，Replace() 方法实现字符串的替换。

例： 将电子邮件地址中的“@”替换为“WO”

```
string test = Regex.Replace(email, "@", "WO");
```


Regex 类

3.Split() 方法

根据匹配正则表达式进行拆分，储存在字符串数组中。两种重载形式：

- Regex.Split(string input,string pattern)
- Regex.Split(string input,string pattern,RegexOptions options)

```
string addr = "12345@qq.com&&67890@sina.com";  
string[] list = Regex.Split(addr, "\\W+");  
foreach (string str in list)  
{  
    Console.WriteLine(str);    // 循环输出各个字符串  
}
```



```
12345  
qq  
com  
67890  
sina  
com
```


4 日期和时间处理

4.1 DateTime 结构

4.2 TimeSpan 结构

4.3 日期和时间的常用操作

4.1 DateTime 结构

DateTime 结构是一个值类型，表示时间上的一刻，通常以日期和当天的时间表示。它的时间值以 **100 纳秒为单位**（该单位称为刻度 tick）进行计算。

1. 字段

DateTime 结构中只有两个**静态的只读**字段：MaxValue 和 MinValue。MaxValue 表示 DateTime 的最大可能值，MinValue 表示 DateTime 的最小可能值。

例如分别使用 MaxValue 和 MinValue 获得 DateTime 的最大值和最小值，然后在控制台输出，具体代码如下：

```
DateTime maxtime = DateTime.MaxValue;  
DateTime mintime = DateTime.MinValue;  
Console.WriteLine(maxtime + "*****" + mintime);
```

运行上述代码，控制台输出的结果是：

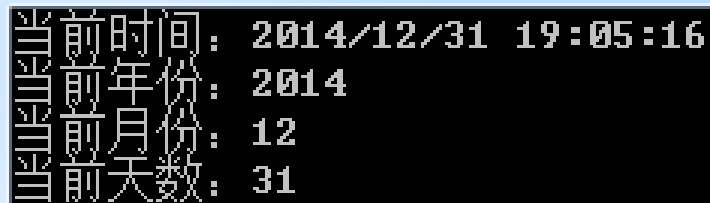
9999-12-31 23:59:59*****0001-1-1 00:00:00

2. 属性

DateTime 结构中不仅包括字段，还包括属性。在 DateTime 结构中包含 16 个只读属性，其中 3 个静态属性，13 个实例属性。通过这些属性我们可以获取系统的当前时间、年份、月份、分钟等等。

属性	说明
Now	静态属性，获取计算机上的当前时间
Today	静态属性，获取当前日期
UtcNow	静态属性，获取计算机上的当前时间，世界时间 (UTC)
Day	获取此实例所表示的日期为该月中的第几天
DayOfWeek	获取此实例所表示的日期是星期几
DayOfYear	获取此实例所表示的日期是该年中的第几天
Year	获取年份部分
Month	获取日期的月份部分
Date	获取日期部分
Hour	获取日期的小时部分
Minute	获取日期的分钟部分
Second	获取日期的秒部分
Millisecond	获取日期的毫秒部分
Ticks	获取日期和时间的刻度数（计时周期数）
TimeOfDay	获取当天时间，即当天自午夜以来已经过时间的部分

```
DateTime nowtime = DateTime.Now;  
Console.WriteLine(" 当前时间 : " + nowtime);  
Console.WriteLine(" 当前年份 : " + DateTime.Now.Year);  
Console.WriteLine(" 当前月份 : " + DateTime.Now.Month);  
Console.WriteLine(" 当前天数 : " + DateTime.Now.Day);
```



```
当前时间: 2014/12/31 19:05:16  
当前年份: 2014  
当前月份: 12  
当前天数: 31
```

3. 方法

DateTime 结构中的方法包括**静态方法**和**实例方法**，其中静态方法 14 个，实例方法 28 个。下表列出了一些**静态方法**。

方法	说明
Compare()	比较两个 DateTime 实例，返回一个指示第一个实例是早于、等于还是晚于第二个实例
DaysInMonth()	返回指定年和月份中的天数
Equals()	比较两个 DateTime 结构是否相等
IsLeapYear()	判断指定的年份是否为闰年。如果是返回 true，否则返回 false
Parse()	将字符串转换为等效的 DateTime
ParseExact()	将字符串转换为等效的 DateTime。其格式必须与指定格式匹配
TryParse()	将字符串转换为等效的 DateTime。成功返回 true，否则 false
TryParseExact()	将字符串转换为等效的 DateTime。格式必须与指定格式匹配

4.2 TimeSpan 结构

TimeSpan 表示时间间隔或持续时间，按正负天数、小时数、分钟数、秒数以及秒的小数部分进行度量。用于度量持续时间的最大时间单位是天，时间间隔以天为单位来度量。

TimeSpan 的值是等于所表示时间间隔的刻度数。一个刻度等于 100 纳秒，TimeSpan 对象的值的范围在 MinValue 和 MaxValue 之间。

TimeSpan 值可以表示为：

[-]d.hh:mm:ss.ff

其中减号是可选的，它指示负时间间隔，d 分量表示天，hh 表示小时（24 小时制），mm 表示分钟，ss 表示秒，而 ff 为秒的小数部分。即时间间隔包括整的正负天数、天数和剩余的不足一天的时长，或者只包含不足一天的时长。

例：创建 TimeSpan 结构的对象 ts0，指定该对象的小时部分为 12、分钟部分为 32、秒部分为 40。最终的对象值为 12:32:40。代码如下：

```
TimeSpan ts0 = new TimeSpan(12, 32, 40);
```

1. 字段

TimeSpan 结构中包含 8 个静态字段，其中 3 个只读字段，5 个常数字段。3 个只读字段分别是：MaxValue 表示最大的 TimeSpan 值，MinValue 表示最小的 TimeSpan 值，Zero 指定零 TimeSpan 值。5 个常数字段是：

- TicksPerDay 一天中的刻度数
- TicksPerHour 1 小时的刻度数
- TicksPerMillisecond 1 毫秒的刻度数
- TicksPerMinute 1 分钟的刻度数
- TicksPerSecond 1 秒的刻度数

例如使用 MaxValue、MinValue 和 TicksPerHour 字段获得 TimeSpan 结构的最大值、最小值和 1 小时的刻度数并且将结果输出。代码如下。

```
Console.WriteLine("TimeSpan 的最大值 : "+TimeSpan.MaxValue);  
Console.WriteLine("TimeSpan 的最小值 : " +  
TimeSpan.MinValue);  
Console.WriteLine("1 小时的刻度数 : "+TimeSpan.TicksPerHour);
```

2. 属性

TimeSpan 结构主要包含 11 个实例属性，均为只读属性：

属性	说明
Days	获取 TimeSpan 结构所表示的时间间隔的天数部分
Hours	获取 TimeSpan 结构所表示的时间间隔的小时数
Milliseconds	获取 TimeSpan 结构所表示的时间间隔的毫秒数
Minutes	获取 TimeSpan 结构所表示的时间间隔的分钟
Seconds	获取 TimeSpan 结构所表示的时间间隔的秒数
Ticks	表示当前 TimeSpan 结构的值的刻度数
TotalDays	获取以整天数和天的小数部分表示的当前 TimeSpan 的值
TotalHours	获取以整小时数和小时的小数部分表示的当前 TimeSpan 结构的值
TotalMinutes	获取以整分钟数和分钟的小数部分表示的当前 TimeSpan 结构的值
TotalSeconds	获取以整秒数和秒的小数部分表示的当前 TimeSpan 结构的值
TotalMillisecond	获取以整毫秒数和毫秒的小数部分表示的当前 TimeSpan 结构的值

3. 方法

和 DateTime 结构一样， TimeSpan 结构中也包含静态方法和实例方法。 TimeSpan 结构中的静态方法：

方法	说明
Compare()	比较两个 TimeSpan 的值，它的返回值为 -1、0 和 1
Equals()	判断两个 TimeSpan 结构的实例是否相等。相等返回 true
FromDays()	根据指定的天数，创建一个 TimeSpan 结构的实例
FromHours()	根据指定的小时数，创建一个 TimeSpan 结构的实例
FromMinutes()	根据指定的分钟数，创建一个 TimeSpan 结构的实例
FromSeconds()	根据指定的秒数，创建一个 TimeSpan 结构的实例
FromMilliseconds()	根据指定的毫秒数，创建一个 TimeSpan 结构的实例
FromTicks()	根据指定的刻度数，创建一个 TimeSpan 结构的实例
Parse()	将时间间隔的字符串转换为相应的 TimeSpan 结构
ParseExact()	将时间间隔的字符串转换为相应的 TimeSpan 结构。该字符串的格式必须与指定的格式完全匹配
ReferenceEquals()	确定指定的 System.Object 实例是否是相同的实例
TryParse()	将时间间隔的字符串转换为相应的 TimeSpan 结构。
TryParseExact()	将时间间隔的字符串转换为相应的 TimeSpan 结构。返回一个指示是否成功的值

4.3 日期和时间的常用操作

1. 追加时间

追加时间是指将指定的时间追加到一个时间上，获得一个新的时间。以 Add 开头的方法都可以实现追加时间的功能。主要包括 Add() 方法、AddYears() 方法、AddMonths()、AddDays()、AddHours()、AddMinutes()、AddSeconds()、AddMilliseconds() 和 AddTicks() 方法。

例：实例方法 AddYears()、AddMonths() 等方法将时间追加到 dtm 对象，并获得一个新的时间。

```
DateTime dtm = DateTime.Now;
```

```
DateTime newTime= dtm.AddYears(1).AddDays(1).AddYears(3).AddMonths(2)
```

2. Subtract() 计算时间差

Subtract() 方法可以计算时间差。它的参数可以是 DateTime 类型，也可以是 TimeSpan 类型。如果参数为 DateTime 类型返回值为 TimeSpan 类型；如果参数为 TimeSpan 类型返回值为 DateTime 类型
例：

```
DateTime dttime = Convert.ToDateTime("2012-1-1 10:29:23");  
TimeSpan subtime = DateTime.Now.Subtract(dttime);  
Console.WriteLine(subtime);  
TimeSpan tspan = new TimeSpan(12, 34, 53);  
DateTime spantime = DateTime.Now.Subtract(tspan);  
Console.WriteLine(spantime);
```

3. 格式化

DateTime 结构提供了多个方法，主要以 To 开头，使用这些方法可以将 DateTime 对象转换为不同的字符串格式。这些方法包括 ToString()、ToLongDateString()、ToLongTimeString()、ToShortDateString() 和 ToShortTimeString()。

```
string str1 = DateTime.Now.ToString();  
string str2 = DateTime.Now.ToShortTimeString();
```