字符串操作

字符串是 [String](https://msdn.microsoft.com/zh-cn/library/system.string.aspx) 类型的对象，它的值是文本。

在内部，文本被存储为 [Char](https://msdn.microsoft.com/zh-cn/library/system.char.aspx) 对象的顺序只读集合。

C# 字符串末尾没有以 null 结尾的字符；因此 C# 字符串可以包含任意数目的嵌入式 null 字符（“\0”）。

字符串的 [Length](https://msdn.microsoft.com/zh-cn/library/system.string.length.aspx) 属性代表它包含的 **Char** 对象的数量，而不是 Unicode 字符的数量。

若要访问字符串中的各个 Unicode 码位，请使用 [StringInfo](https://msdn.microsoft.com/zh-cn/library/system.globalization.stringinfo.aspx) 对象。

在 C# 中，**string** 关键字是 [String](https://msdn.microsoft.com/zh-cn/library/system.string.aspx) 的别名。因此，**String** 与 **string** 等效，您可以根据自己的喜好选择命名约定。

**String** 类提供了很多用于安全地创建、操作和比较字符串的方法。

除了在使用字符数组初始化字符串时以外，不要使用 [new](https://msdn.microsoft.com/zh-cn/library/fa0ab757.aspx) 运算符创建字符串对象。

字符串对象是不可变的：即它们创建之后就无法更改。所有看似修改字符串的 [String](https://msdn.microsoft.com/zh-cn/library/system.string.aspx) 方法和 C# 运算符实际上都以新字符串对象的形式返回结果。

加入双引号

string quote = @"Her name was ""Sara.""";

格式化字符串

s = System.String.Format("{0} times {1} = {2}", i, j, (i \* j));

子字符串

子字符串是包含在字符串中的任意字符序列。

访问各个字符

string s5 = "Printing backwards";

for (int i = 0; i < s5.Length; i++)

{

System.Console.Write(s5[s5.Length - i - 1]);

}

单独处理各字符串

[StringBuilder](https://msdn.microsoft.com/zh-cn/library/system.text.stringbuilder.aspx) 类创建了一个字符串缓冲区，用于在程序执行大量字符串操作时提供更好的性能。

string question = "hOW DOES mICROSOFT wORD DEAL WITH THE cAPS lOCK KEY?";

System.Text.StringBuilder sb = new System.Text.StringBuilder(question);

for (int j = 0; j < sb.Length; j++)

{

if (System.Char.IsLower(sb[j]) == true)

sb[j] = System.Char.ToUpper(sb[j]);

else if (System.Char.IsUpper(sb[j]) == true)

sb[j] = System.Char.ToLower(sb[j]);

}

// Store the new string.

string corrected = sb.ToString();

System.Console.WriteLine(corrected);

# 串联多个字符串

若要串联字符串变量，可以使用 + 或 += 运算符，也可以使用 [String.Concat](https://msdn.microsoft.com/zh-cn/library/system.string.concat.aspx)、[String.Format](https://msdn.microsoft.com/zh-cn/library/system.string.format.aspx) 或 [StringBuilder.Append](https://msdn.microsoft.com/zh-cn/library/system.text.stringbuilder.append.aspx) 方法。 + 运算符容易使用，且有利于提高代码的直观性

在字符串串联操作中，C# 编译器对 null 字符串和空字符串进行相同的处理，但它不转换原始 null 字符串的值。

但如果性能的优劣很重要，则应该总是使用 [StringBuilder](https://msdn.microsoft.com/zh-cn/library/system.text.stringbuilder.aspx) 类来串联字符串。下面的代码使用 [StringBuilder](https://msdn.microsoft.com/zh-cn/library/system.text.stringbuilder.aspx) 类的 [Append](https://msdn.microsoft.com/zh-cn/library/system.text.stringbuilder.append.aspx) 方法来串联字符串，因此不会有 **+** 运算符的链接作用产生。

# 修改字符串内容

## 替换某字符串

s = s.Replace("mountains", "peaks");

ReplaceSubstrings app = new ReplaceSubstrings();

app.searchFor = "the"; // A very simple regular expression.

app.replaceWith = "many";

s = Regex.Replace(s, app.searchFor, app.ReplaceMatchCase, RegexOptions.IgnoreCase);

## 移除某字符串

string temp = "many\_";

int i = s.IndexOf(temp);

if (i >= 0)

{

s = s.Remove(i, temp.Length);

}

去除前后多余空格

string s2 = " I'm wider than I need to be. ";

// Store the results in a new string variable.

temp = s2.Trim();

Console.WriteLine(temp);

// Output: I'm wider than I need to be.

## 数组表示法访问字符串中的各个字符

char[] chars = str.ToCharArray();

string str2 = new string(chars);

# 比较字符串

比较其值将不随用户计算机的区域设置的改变而变化的字符串

## 完全比较,包含大小写

bool result = root.Equals(root2, StringComparison.Ordinal);

## 不包含大小写比较

result = root.Equals(root2, StringComparison.OrdinalIgnoreCase);

## 比较内存地址

通过调用 [ReferenceEquals](https://msdn.microsoft.com/zh-cn/library/system.object.referenceequals.aspx) 方法，可以看到这两个字符串实际引用内存中的同一对象。

string a = "The computer ate my source code.";

string b = "The computer ate my source code.";

if (String.ReferenceEquals(a, b))

Console.WriteLine("a and b are interned.");

else

Console.WriteLine("a and b are not interned.");

string c = String.Copy(a);

if (String.ReferenceEquals(a, c))

Console.WriteLine("a and c are interned.");

else

Console.WriteLine("a and c are not interned.");

## ?通过首选方式比较字符串

## ?区分区域性的比较

# 拆分字符串

根据一个多个字符匹配拆分

char[] delimiterChars = { ' ', ',', '.', ':', '\t' };

string[] words = text.Split(delimiterChars);

# 使用字符串方法搜索字符串

string str = "Extension methods have all the capabilities of regular static methods.";

## 以...为开始的字符串

bool test1 = str.StartsWith("extension");

//不匹配大小写

bool test2 = str.StartsWith("extension", System.StringComparison.CurrentCultureIgnoreCase);

## 以...为结尾的字符串

bool test3 = str.EndsWith(".", System.StringComparison.CurrentCultureIgnoreCase);

## 获取...在字符串中的第一个开始的位置

int first = str.IndexOf("methods") + "methods".Length;

## 获取...在字符串中的最后一个开始的位置

int last = str.LastIndexOf("methods");

获取从m到n的子字符串

string str2 = str.Substring(first, last - first);

# 使用正则表达式搜索字符串

方法一:

不区分大小写的搜索某字符串

string sPattern = "code";

foreach (string s in sentences)

{

System.Console.Write("{0,24}", s);

if (System.Text.RegularExpressions.Regex.IsMatch(s, sPattern, System.Text.RegularExpressions.RegexOptions.IgnoreCase))

{

System.Console.WriteLine(" (match for '{0}' found)", sPattern);

}

else

{

System.Console.WriteLine();

}

}

//验证字符串格式

// "146-555-0122"

string sPattern = "^\\d{3}-\\d{3}-\\d{4}$";

if (System.Text.RegularExpressions.Regex.IsMatch(s, sPattern))

{

System.Console.WriteLine(" - valid");

}

方法二:

Regex r = new Regex(regstring, RegexOptions.IgnoreCase | RegexOptions.Singleline);

Match m = r.Match(txt);

if (m.Success)

{

String var1 = m.Groups[1].ToString();

}

# 字符串转换为数值

int i = 0;

string s = "108";

bool result = int.TryParse(s, out i); //i now = 108

如果字符串包含非数值字符或者所包含的数值对于指定的特定类型而言太大或太小，**TryParse** 都将返回 false 并将 out 参数设置为零。否则，它将返回 true，并且将 out 参数设置为字符串的数值。

# 字符串转换为DateTime

string date = "01/08/2008";

DateTime dt = Convert.ToDateTime(date);

Console.WriteLine("Year: {0}, Month: {1}, Day: {2}", dt.Year, dt.Month, dt.Day);

//指定日期格式

IFormatProvider culture = new System.Globalization.CultureInfo("fr-FR", true);

DateTime dt2 = DateTime.Parse(date, culture, System.Globalization.DateTimeStyles.AssumeLocal);

Console.WriteLine("Year: {0}, Month: {1}, Day {2}", dt2.Year, dt2.Month, dt2.Day);

# 旧式编码与Unicode转换

在 C# 中，内存中的所有字符串都是按 Unicode (UTF-16) 编码的。

将数据从存储器移动到 **string** 对象中后，数据将自动转换为 UTF-16。

如果数据仅包含从 0 到 127 的 ASCII 值，则此转换无需您执行任何额外的工作。

但若源文本包含扩展的 ASCII 字节值（128 到 255），则默认情况下，将根据当前代码页解释扩展字符。

若要指定应该根据其他某个代码页解释源文本，请使用 [System.Text.Encoding](https://msdn.microsoft.com/zh-cn/library/system.text.encoding.aspx) 类，

# 将rtf转换为纯文本