**字符串（C# 编程指南）**

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。

**字符串与System.String**

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。此外，C# 语言还重载某些运算符来简化常见的字符串操作。有关关键字的更多信息，请参见[string（C# 参考）](https://msdn.microsoft.com/zh-CN/library/362314fe.aspx)。有关类型及其方法的更多信息，请参见 [String](https://msdn.microsoft.com/zh-CN/library/system.string.aspx)。

**声明和初始化字符串**

可以通过各种方式来声明和初始化字符串，如下面的示例所示：

// Declare without initializing.

string message1;

// Initialize to null.

string message2 = null;

// Initialize as an empty string.

// Use the Empty constant instead of the literal "".

string message3 = System.String.Empty;

//Initialize with a regular string literal.

string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";

// Initialize with a verbatim string literal.

string newPath = @"c:\Program Files\Microsoft Visual Studio 9.0";

// Use System.String if you prefer.

System.String greeting = "Hello World!";

// In local variables (i.e. within a method body)

// you can use implicit typing.

var temp = "I'm still a strongly-typed System.String!";

// Use a const string to prevent 'message4' from

// being used to store another string value.

const string message4 = "You can't get rid of me!";

// Use the String constructor only when creating

// a string from a char\*, char[], or sbyte\*. See

// System.String documentation for details.

char[] letters = { 'A', 'B', 'C' };

string alphabet = new string(letters);

注意，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。

使用 [Empty](https://msdn.microsoft.com/zh-CN/library/system.string.empty.aspx) 常量值初始化字符串可新建字符串长度为零的 [String](https://msdn.microsoft.com/zh-CN/library/system.string.aspx) 对象。零长度字符串的字符串表示形式为 ""。使用 [Empty](https://msdn.microsoft.com/zh-CN/library/system.string.empty.aspx) 值（而不是 [null](https://msdn.microsoft.com/zh-CN/library/edakx9da.aspx)）初始化字符串可以降低发生 [NullReferenceException](https://msdn.microsoft.com/zh-CN/library/system.nullreferenceexception.aspx) 的可能性。请在尝试访问字符串之前使用静态 [IsNullOrEmpty(String)](https://msdn.microsoft.com/zh-CN/library/system.string.isnullorempty.aspx) 方法验证字符串的值。

**字符串对象的不可变性**

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。在下面的示例中，当连接 s1 和 s2 的内容以形成一个字符串时，不会修改两个原始字符串。 **+=** 运算符会创建一个包含组合内容的新字符串。这个新对象赋给变量 s1，而最初赋给 s1 的对象由于没有其他任何变量包含对它的引用而释放，用于垃圾回收。

string s1 = "A string is more ";

string s2 = "than the sum of its chars.";

// Concatenate s1 and s2\. This actually creates a new

// string object and stores it in s1, releasing the

// reference to the original object.

s1 += s2;

System.Console.WriteLine(s1);

// Output: A string is more than the sum of its chars.

由于“修改”字符串实际上是创建新字符串，因此创建对字符串的引用时必须谨慎。如果创建了对字符串的引用，然后“修改”原始字符串，则该引用指向的仍是原始对象，而不是修改字符串时创建的新对象。下面的代码说明了这种行为：

string s1 = "Hello ";

string s2 = s1;

s1 += "World";

System.Console.WriteLine(s2);

//Output: Hello

有关如何创建基于修改（例如搜索和替换原始字符串的操作）的新字符串的更多信息，请参见[如何：修改字符串内容（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms228599.aspx)。

**正则字符串和原义字符串**

如果必须嵌入 C# 提供的转义符，则应使用正则字符串，如下面的示例所示：

string columns = "Column 1\tColumn 2\tColumn 3";

//Output: Column 1 Column 2 Column 3

string rows = "Row 1\r\nRow 2\r\nRow 3";

/\* Output:

Row 1

Row 2

Row 3

\*/

string title = "\"The \u00C6olean Harp\", by Samuel Taylor Coleridge";

//Output: "The Æolean Harp", by Samuel Taylor Coleridge

如果字符串文本包含反斜杠字符（例如在文件路径中），为方便起见和提高可读性，应使用原义字符串。由于原义字符串保留换行符作为字符串文本的一部分，因此可用于初始化多行字符串。在原义字符串中嵌入引号时请使用双引号。下面的示例演示原义字符串的一些常见用途：

string filePath = @"C:\Users\scoleridge\Documents\";

//Output: C:\Users\scoleridge\Documents\

string text = @"My pensive SARA ! thy soft cheek reclined

Thus on mine arm, most soothing sweet it is

To sit beside our Cot,...";

/\* Output:

My pensive SARA ! thy soft cheek reclined

Thus on mine arm, most soothing sweet it is

To sit beside our Cot,...

\*/

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

//Output: Her name was "Sara."

**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 |
| \f | 换页 | 0x000C |
| \n | 换行 | 0x000A |
| \r | 回车 | 0x000D |
| \t | 水平制表符 | 0x0009 |
| \U | 代理项对的 Unicode 转义序列。 | \Unnnnnnnn |
| \u | Unicode 转义序列 | \u0041 = "A" |
| \v | 垂直制表符 | 0x000B |
| \x | Unicode 转义序列类似于“\u”，只是长度可变。 | \x0041 = "A" |

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 编译时，原义字符串转换为所有转义序列均保持不变的普通字符串。因而，如果在调试器监视窗口中查看原义字符串，则看到的将是编译器添加的转义字符，而不是源代码中的原义版本。例如，原义字符串 @"C:\files.txt" 在监视窗口中将显示为 "C:\files.txt"。 |

**格式字符串**

格式字符串是内容可以在运行时动态确定的一种字符串。采用以下方式创建格式字符串：使用静态 [Format](https://msdn.microsoft.com/zh-CN/library/1ksz8yb7.aspx) 方法并在大括号中嵌入占位符，这些占位符将在运行时替换为其他值。下面的示例使用格式字符串输出循环中每个迭代的结果：

class FormatString

{

static void Main()

{

// Get user input.

System.Console.WriteLine("Enter a number");

string input = System.Console.ReadLine();

// Convert the input string to an int.

int j;

System.Int32.TryParse(input, out j);

// Write a different string each iteration.

string s;

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

{

// A simple format string with no alignment formatting.

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

System.Console.WriteLine(s);

}

//Keep the console window open in debug mode.

System.Console.ReadKey();

}

}

[WriteLine](https://msdn.microsoft.com/zh-CN/library/zdf6yhx5.aspx) 方法的一个重载将格式字符串用作参数。因此，可以只嵌入格式字符串，而无需显式调用该方法。但若使用 [WriteLine](https://msdn.microsoft.com/zh-CN/library/65cydf9y.aspx) 方法在 Visual Studio**“输出”**窗口中显示调试输出，则必须显式调用 [Format](https://msdn.microsoft.com/zh-CN/library/1ksz8yb7.aspx) 方法，因为 [WriteLine](https://msdn.microsoft.com/zh-CN/library/65cydf9y.aspx) 只接受字符串，而不接受格式字符串。有关格式字符串的更多信息，请参见[.NET Framework 中的格式化类型](https://msdn.microsoft.com/zh-CN/library/26etazsy.aspx)。

**子字符串**

子字符串是包含在字符串中的任意字符序列。使用 [Substring](https://msdn.microsoft.com/zh-CN/library/hxthx5h6.aspx) 方法可以基于原始字符串的一部分创建新字符串。可以使用 [IndexOf](https://msdn.microsoft.com/zh-CN/library/kwb0bwyd.aspx) 方法搜索子字符串的一个或多个匹配项。使用 [Replace](https://msdn.microsoft.com/zh-CN/library/czx8s9ts.aspx) 方法可将指定子字符串的所有匹配项替换为一个新字符串。与 [Substring](https://msdn.microsoft.com/zh-CN/library/hxthx5h6.aspx) 方法一样，[Replace](https://msdn.microsoft.com/zh-CN/library/czx8s9ts.aspx) 实际上返回的也是新字符串，而不修改原始字符串。有关更多信息，请参见[如何：使用字符串方法搜索字符串（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms228630.aspx)和[如何：修改字符串内容（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms228599.aspx)。

string s3 = "Visual C# Express";

System.Console.WriteLine(s3.Substring(7, 2));

// Output: "C#"

System.Console.WriteLine(s3.Replace("C#", "Basic"));

// Output: "Visual Basic Express"

// Index values are zero-based

int index = s3.IndexOf("C");

// index = 7

**访问各个字符**

可以使用带索引值的数组表示法获取对各个字符的只读访问，如下面的示例所示：

string s5 = "Printing backwards";

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

{

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

}

// Output: "sdrawkcab gnitnirP"

如果 [String](https://msdn.microsoft.com/zh-CN/library/system.string.aspx) 方法不提供修改字符串中的各个字符所必须具有的功能，则您可以使用 [StringBuilder](https://msdn.microsoft.com/zh-CN/library/system.text.stringbuilder.aspx) 对象“就地”修改各个字符，然后使用 [StringBuilder](https://msdn.microsoft.com/zh-CN/library/system.text.stringbuilder.aspx) 方法创建一个新字符串来存储结果。在下面的示例中，假设您必须以特定方式修改原始字符串，然后存储结果以备将来使用：

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);

// Output: How does Microsoft Word deal with the Caps Lock key?

**Null 字符串和空字符串**

空字符串是不包含字符的 [System.String](https://msdn.microsoft.com/zh-CN/library/system.string.aspx) 对象的实例。在各种编程方案中经常会使用空字符串表示空白文本字段。可以对空字符串调用方法，因为它们是有效的 [System.String](https://msdn.microsoft.com/zh-CN/library/system.string.aspx) 对象。空字符串可按如下方式初始化：

string s = String.Empty;

空字符串具有字符串的使用属性和方法,可以像普通字符串一样使用,他的长度是0

相反，null 字符串并不引用 [System.String](https://msdn.microsoft.com/zh-CN/library/system.string.aspx) 对象的实例，任何对 null 字符串调用方法的尝试都会生成 [NullReferenceException](https://msdn.microsoft.com/zh-CN/library/system.nullreferenceexception.aspx)。但是，可以在串联和比较操作中将 null 字符串与其他字符串一起使用。下面的示例阐释了引用 null 字符串导致引发异常的情形以及并不导致引发异常的情形：

null字符串不能使用字符串的所有属性和方法,会抛异常,但是null字符串可以被赋值为普通字符串.也可以和普通字符串相加也就是连接起来,连接后,它就表示null字符串了.

static void Main()

{

string str = "hello";

string nullStr = null;

string emptyStr = String.Empty;

string tempStr = str + nullStr;

// Output of the following line: hello

Console.WriteLine(tempStr);

bool b = (emptyStr == nullStr);

// Output of the following line: False

Console.WriteLine(b);

// The following line creates a new empty string.

string newStr = emptyStr + nullStr;

// Null strings and empty strings behave differently. The following

// two lines display 0.

Console.WriteLine(emptyStr.Length);

Console.WriteLine(newStr.Length);

// The following line raises a NullReferenceException.

//Console.WriteLine(nullStr.Length);

// The null character can be displayed and counted, like other chars.

string s1 = "\x0" + "abc";

string s2 = "abc" + "\x0";

// Output of the following line: \* abc\*

Console.WriteLine("\*" + s1 + "\*");

// Output of the following line: \*abc \*

Console.WriteLine("\*" + s2 + "\*");

// Output of the following line: 4

Console.WriteLine(s2.Length);

}

**使用 StringBuilder 快速创建字符串**

.NET 中的字符串操作已高度优化，大多数情况下不会显著影响性能。但在某些应用场景中，例如在执行好几百甚至好几千次的紧凑循环中，字符串操作会影响性能。 [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。 [StringBuilder](https://msdn.microsoft.com/zh-CN/library/system.text.stringbuilder.aspx) 字符串还使您能够重新分配个别字符（内置字符串数据类型所不支持的字符）。例如，此代码在不创建新字符串的情况下更改了一个字符串的内容：

System.Text.StringBuilder sb = new System.Text.StringBuilder("Rat: the ideal pet");

sb[0] = 'C';

System.Console.WriteLine(sb.ToString());

System.Console.ReadLine();

//Outputs Cat: the ideal pet

在本示例中，[StringBuilder](https://msdn.microsoft.com/zh-CN/library/system.text.stringbuilder.aspx) 对象用于从一组数值类型中创建字符串：

class TestStringBuilder

{

static void Main()

{

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

// Create a string composed of numbers 0 - 9

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

{

sb.Append(i.ToString());

}

System.Console.WriteLine(sb); // displays 0123456789

// Copy one character of the string (not possible with a System.String)

sb[0] = sb[9];

System.Console.WriteLine(sb); // displays 9123456789

}

}

**字符串、扩展方法和 LINQ**

由于 [String](https://msdn.microsoft.com/zh-CN/library/system.string.aspx) 类型实现 [IEnumerable<T>](https://msdn.microsoft.com/zh-CN/library/9eekhta0.aspx)，因此可以对字符串使用 [Enumerable](https://msdn.microsoft.com/zh-CN/library/system.linq.enumerable.aspx) 类中定义的扩展方法。对于 [String](https://msdn.microsoft.com/zh-CN/library/system.string.aspx) 类型，为了避免视觉上的混乱，从 IntelliSense 中排除了这些方法，但这些方法仍然可用。此外，还可以对字符串使用 LINQ 查询表达式。有关更多信息，请参见 [LINQ and Strings](https://msdn.microsoft.com/zh-CN/library/bb397915.aspx)。