字符串是值为文本的String类型对象。文本在内部存储为Char对象的次序只读集合。在C#字符串末尾没有null终止字符。因此，一个C#字符串可以包含任何数量的嵌入的null字符（’\0’,在C语言中，这是字符串截止的标志）.字符串的Length属性表示其包含的Char对象数量，而非Unicode字符数。若要访问字符串中各个Unicode码位，请使用StringInfo（https://docs.microsoft.com/zh-cn/dotnet/api/system.globalization.stringinfo）对象。

1. String 和 System.String在C#中，string关键字是String的别名。因此，String和string是等效的。String 类提供了安全创建、操作和比较字符串的多种方法。 此外，C# 语言重载了部分运算符，以简化常见字符串操作。 有关关键字的详细信息，请参阅 [string](https://docs.microsoft.com/zh-cn/dotnet/csharp/language-reference/keywords/string)。 有关类型及其方法的详细信息，请参阅 [String](https://docs.microsoft.com/zh-cn/dotnet/api/system.string)。
2. 创建字符串的方法

// 创建字符串的方法

string message1;

string message2 = null;

string message3 = System.String.Empty;

string oldPath = "C:\\abc\\ddd";

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

System.String greeting = "Hello World!";

// 在方法体类，使用var声明，非明确类型的变量

var temp = "implicit type";

// 字符串常量

const string constString = "不可变";

// 可以说从C继承来的声明方法

// See System.String documentation for details.

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

string alphabet = new string( letters );

Console.WriteLine( alphabet );

请注意，不要使用 [new](https://docs.microsoft.com/zh-cn/dotnet/csharp/language-reference/keywords/new-operator) 运算符创建字符串对象，除非使用字符数组初始化字符串。

使用 [Empty](https://docs.microsoft.com/zh-cn/dotnet/api/system.string.empty) 常量值初始化字符串，以新建字符串长度为零的 [String](https://docs.microsoft.com/zh-cn/dotnet/api/system.string) 对象。 长度为零的字符串文本表示法是“”。 通过使用 [Empty](https://docs.microsoft.com/zh-cn/dotnet/api/system.string.empty) 值（而不是 [null](https://docs.microsoft.com/zh-cn/dotnet/csharp/language-reference/keywords/null)）初始化字符串，可以减少 [NullReferenceException](https://docs.microsoft.com/zh-cn/dotnet/api/system.nullreferenceexception) 发生的可能性。 尝试访问字符串前，先使用静态 [IsNullOrEmpty(String)](https://docs.microsoft.com/zh-cn/dotnet/api/system.string.isnullorempty#System_String_IsNullOrEmpty_System_String_) 方法验证字符串的值

1. 字符串对象的不可变性。

字符串对象是“不可变的”：它们在创建后无法更改。 看起来是在修改字符串的所有 [String](https://docs.microsoft.com/zh-cn/dotnet/api/system.string) 方法和 C# 运算符实际上都是在新的字符串对象中返回结果。 在下面的示例中，当 s1 和 s2 的内容被串联在一起以形成单个字符串时，两个原始字符串没有被修改。 += 运算符创建一个新的字符串，其中包含组合的内容。 这个新对象被分配给变量 s1，而分配给 s1 的原始对象被释放，以供垃圾回收，因为没有任何其他变量包含对它的引用。

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.

由于字符串“modification”实际上是一个新创建的字符串，因此，必须在创建对字符串的引用时使用警告。 如果创建了字符串的引用，然后“修改”了原始字符串，则该引用将继续指向原始对象，而非指向修改字符串时所创建的新对象

1. 常规和逐字字符串文本

常规字符使用转义字符,eg:

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

当字符串文本包含反斜杠字符（例如在文件路径中）时，出于便捷性和更强的可读性的考虑，使用逐字字符串。 由于逐字字符串将新的行字符作为字符串文本的一部分保留，因此可将其用于初始化多行字符串。 使用双引号在逐字字符串内部嵌入引号。 下面的示例演示逐字字符串的一些常见用法：

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."

1. 格式化字符串

格式字符串是可以在运行时以动态方式确定其内容的字符串。 使用静态 Syetem.String.[Format](https://docs.microsoft.com/zh-cn/dotnet/api/system.string.format) 方法，并在大括号中嵌入将在运行时被其他值替换的占位符，从而创建格式字符串。 下面的示例使用格式字符串来输出每个循环迭代的结果

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.

// 输入的字符串转化为数字

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();

}

}

1. 访问单个字符串

可以使用包含索引值的数组表示法来获取对单个字符的只读访问权限

string test = @"hello,worldd

这是第四行了！";

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

Console.WriteLine( $"第{i}个字符：{test[i]}" );

}

如果 [String](https://docs.microsoft.com/zh-cn/dotnet/api/system.string) 方法不提供修改字符串中的各个字符所需的功能，可以使用 [StringBuilder](https://docs.microsoft.com/zh-cn/dotnet/api/system.text.stringbuilder) 对象“就地”修改各个字符，再新建字符串来使用 [StringBuilder](https://docs.microsoft.com/zh-cn/dotnet/api/system.text.stringbuilder) 方法存储结果。 在下面的示例中，假定必须以特定方式修改原始字符串，然后存储结果以供未来使用：

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?

1. Null字符串调用任何方法都会报[NullReferenceException](https://docs.microsoft.com/zh-cn/dotnet/api/system.nullreferenceexception)错误，但是可以用于一些合并操作：

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);

1. 使用StringBuilder（System.Text）快速创建字符串

.NET 中的字符串操作进行了高度的优化，在大多数情况下不会显著影响性能。 但是，在某些情况下（例如，执行数百次或数千次的紧密循环），字符串操作可能影响性能。 [StringBuilder](https://docs.microsoft.com/zh-cn/dotnet/api/system.text.stringbuilder) 类创建字符串缓冲区，用于在程序执行多个字符串操控时提升性能。 使用 [StringBuilder](https://docs.microsoft.com/zh-cn/dotnet/api/system.text.stringbuilder) 字符串，还可以重新分配各个字符，而内置字符串数据类型则不支持这样做。

StringBuilder str = new StringBuilder( "" );

str.Append( "hello" );

str.Append( " Bill" );

Console.WriteLine( str );