

Строки

Класс *string*

Класс *string* предназначен для хранения строк с символами из набора Unicode, причем размер строк на практике ограничивается только размерами доступной оперативной памяти.

Особенности инициализации. Неизменяемость объектов *string*

Инициализация с помощью литеральной строки:

```
string s = "ABCD";
```

Конструкторы.

```
string(char value, int count);
```

```
string(char[] value);
```

Пример:

```
string s = new string('A', 5);
```

После создания и размещения в памяти объект типа *string* *нельзя изменить*. Класс *string* является *ссылочным* типом.

Разбор строки

```
static числовой_тип Parse(string value[,  
    IFormatProvider p]);
```

В качестве параметра *p* можно использовать объект типа *System.Globalization.CultureInfo*, это позволяет учитывать различные *региональные настройки*. Например: *new CultureInfo("en-US")* (или *"ru-RU"*). Изменение текущих настроек:

```
System.Threading.Thread.CurrentThread.CurrentCulture  
    = new System.Globalization.CultureInfo("en-US").
```

Управляющие последовательности и буквальныe строки

`\\` — символ «\»;

`\'` — символ «'»;

`\"` — символ «"»;

`\0` — символ с кодом 0;

`\t` — символ табуляции (код 9);

`\n` — символ «переход на новую строку» (код 10);

`\r` — символ «возврат каретки» (код 13);

`\uN` — символ Unicode с шестнадцатеричным кодом *N* (*N* должен состоять из 4 цифр и может меняться от 0000 до FFFF).

Например, символ `'\u0041'` обозначает латинскую букву «А».

Пример:

```
string name = "C:\\Windows\\System32\\Notepad.exe";
```

Буквальные строки (verbatim strings):

```
string name = @"C:\Windows\System32\Notepad.exe";
```

Поля и свойства

```
static readonly string Empty;
```

```
int Length { get; }
```

```
char this [int index] { get; }
```

Пример: `"ABCD"[2]` вернет символ «C».

Сравнение строк

В языке C# сравнение строк на равенство/неравенство может проводиться с помощью операций `==` и `!=` (при этом учитывается регистр и не учитываются региональные настройки); операции `<`, `<=`, `>`, `>=` для строк *не определены*.

```
static int Compare(string strA, string strB[, bool  
    ignoreCase[, CultureInfo culture]]);
```

```
static int Compare(string strA, int startA, string  
    strB, int startB, int count[, bool ignoreCase[,  
    CultureInfo culture]]);
```

```
static int CompareOrdinal(string strA, string strB);
```

```
static int CompareOrdinal(string strA, int startA,  
    string strB, int startB, int count);
```

```
int CompareTo(string str);
```

```
bool StartsWith(string str);
```

```
bool EndsWith(string str);
```

Поиск в строке

С учетом регистра и без учета региональных настроек.

```
int IndexOf(char value[, int start[, int count]]);
```

```
int IndexOf(string value[, int start[, int count]]);
```

```
int LastIndexOf(char value[, int start[, int count]]);
```

```
int LastIndexOf(string value[, int start[, int count]]);
```

```
int IndexOfAny(char[] values[, int start[, int count]]);
```

```
int LastIndexOfAny(char[] values[, int start[, int count]]);
```

Преобразование строки

```
string Insert(int start, string str);
```

```
string Remove(int start[, int count]);
```

```
string Replace(char value, char newValue);
```

```
string Replace(string value, string newValue);
```

```
string Substring(int start[, int count]);
```

```
string PadLeft(int total[, char padChar]);
```

```
string PadRight(int total[, char padChar]);
```

```
string Trim([params char[] trimChars]);
```

```
string TrimStart([params char[] trimChars]);
```

```
string TrimEnd([params char[] trimChars]);
```

```
string ToLower([CultureInfo culture]);
```

```
string ToUpper([CultureInfo culture]);
```

Объединение и разбиение строк

Для *сцепления (конкатенации)* нескольких строк в языке C# предусмотрена операция `+`. При сцеплении строки *s* и объекта *x* *другого типа* в языке C# автоматически выполняется преобразование объекта *x* к его строковому представлению (то есть для *x* вызывается метод `ToString`).

```
static string Concat(params object[] a);
```

```
static string Join(string sep, string[] a[,
```

```
    int start, int count]);
```

```
char[] ToCharArray([int start, int count]);
```

```
void CopyTo(int start, char[] array, int arrayStart,  
    int count);
```

```
string[] Split([params char[] sep]);
```

```
string[] Split(char[] sep, int count);
```

```
string[] Split(char[] sep[, int count],
```

```
    StringSplitOptions opt);
```

```
string[] Split(string[] sep[, int count],
```

```
    StringSplitOptions opt);
```

Параметр *opt* может принимать одно из двух значений перечислимого типа *StringSplitOptions*: *None* и *RemoveEmptyEntries*. Если массив разделителей равен `null` или метод `Split` вызван без параметров, то разделителем считается любой *пробельный символ*. В случае строкового массива разделителей важным является *порядок* его элементов, поскольку при поиске разделителей элементы массива *sep* перебираются по возрастанию индексов.

Класс *System.Text.StringBuilder*

Основной особенностью объектов класса *StringBuilder* является *возможность их изменения*, в том числе *посимвольного*.

Свойства и конструкторы

```
int Capacity { get; set; }
```

```
int MaxCapacity { get; }
```

```
int Length { get; set; }
```

```
char this [int index] { get; set; }
```

```
StringBuilder([int capacity[, int maxCapacity]]);
```

```
StringBuilder(string value[, int start, int count],
```

```
    int capacity);
```

Совместимость объектов string и StringBuilder

Объекты string и StringBuilder не совместимы по присваиванию. Для преобразования «обычной» строки (типа string) к объекту StringBuilder необходимо воспользоваться соответствующим конструктором класса StringBuilder.

```
string ToString([int start, int count]);
```

Для повышения эффективности программы желательно не изменять строку StringBuilder после вызова ее метода ToString.

Преобразование строки типа StringBuilder

```
StringBuilder Append(object value);
StringBuilder Append(bool value);
StringBuilder Append(числовой_тип value);
StringBuilder Append(char value[, int charCount]);
StringBuilder Append(char[] value[, int start, int count]);
StringBuilder Append(string value[, int start, int count]);
StringBuilder AppendLine([string value]);
StringBuilder Insert(int index, object value);
StringBuilder Insert(int index, bool value);
StringBuilder Insert(int index, числовой_тип value);
StringBuilder Insert(int index, char value);
StringBuilder Insert(int index, char[] value[, int
    start, int count]);
StringBuilder Insert(int index, string value[, int count]);
StringBuilder Remove(int start, int count);
StringBuilder Replace(char value, char newValue[,
    int start, int count]);
StringBuilder Replace(string value, string newValue[,
    int start, int count]);
```

Дополнительные методы класса StringBuilder

```
void CopyTo(int start, char[] array, int arrayStart,
    int count);
int EnsureCapacity(int capacity);
```

Форматирование данных

Форматирование по умолчанию и явное форматирование

Механизм явного форматирования основан на использовании двух настроечных параметров: *форматной строки* и *регионального стандарта*: (интерфейс IFormattable):

```
string ToString(string fmt, IFormatProvider p);
```

Спецификаторы формата

Спецификатор формата может быть дополнен *спецификатором точности* (например, "D6").

C или c — *денежный формат* (currency).

D или d — *десятичный целочисленный формат* (decimal).

E или e — *экспоненциальный числовой формат* (exponential).

F или f — *числовой с фиксированной точкой* (fixed-point).

P или p — *процентный формат* (percent).

X или x — *16-ричный целочисленный формат* (hexadecimal).

G или g — *общий* (general), используется более краткое представление.

	en-US	ru-RU
C	\$246.00	246,00р.
C1	\$246.0	246,0р.
C0	\$246	246р.
D	246	246
D2	246	246
D4	0246	0246
E	2.460000E+002	2,460000E+002
e1	2.5e+002	2,5e+002
E0	2E+002	2E+002

	en-US	ru-RU
F	246.00	246,00
f1	246.0	246,0
F0	246	246
P	24,600.00 %	24 600,00%
P0	24,600 %	24 600%
X	F6	F6
x2	f6	f6
X4	00F6	00F6

Спецификаторы формата для *перечислимых типов* (потомков класса Enum): G (отображение имени перечислимой константы), D (отображение десятичного числа, соответствующего перечислимой константе), X (отображение шестнадцатеричного числа, соответствующего перечислимой константе).

Одновременное форматирование нескольких объектов: метод Format

```
static string Format([IFormatProvider p,] string fmt,
    params object[] args);
```

В *форматной строке* fmt можно указывать обычный текст и *форматные настройки* для каждого из форматируемых параметров. Эти настройки имеют вид

```
{ind[, width][:spec]}
```

ind — целое число, которое определяет индекс форматируемого элемента в массиве args (индексация ведется от 0; данный атрибут является обязательным);

width — целое число, модуль которого задает минимальную *ширину поля вывода* (то есть минимальное число позиций, отводимое для форматируемого элемента), а знак определяет *способ выравнивания* элемента в пределах поля вывода;

spec — строка, которая задает *спецификатор формата* для данного элемента (эта строка может также содержать формат, явно определяемый с помощью *символов-заполнителей*). Между двоеточием и строкой spec не должно быть пробелов.

Пример.

```
string.Format
    ("Формат D:{0, 6:D4}, формат X:{0, 6:X4}", 246)
// Формат D: 0246, формат X: 00F6
```

```
StringBuilder.AppendFormat([IFormatProvider p,]
    string fmt, params object[] args);
```

Возможность форматирования обеспечивается также в методах, связанных с выводом данных в текстовые потоки (методы Write и WriteLine классов StreamWriter и Console).

Кодирование и декодирование символьных данных

Кодирование данных по умолчанию: формат UTF-8

По умолчанию при записи символьных данных в файлы применяется формат кодирования UTF-8. В UTF-8 символы ASCII кодируются одним байтом. Символы Unicode с кодами от 128 до 2047 (содержащие символы различных европейских и среднеазиатских языков) преобразуются в 2 байта.

Если требуется прочесть или сохранить символьные данные в другом формате, то формат кодирования необходимо явно указать в конструкторах соответствующих потоков-оболочек.

Явная установка формата кодирования: класс System.Text.Encoding

```
static Encoding UTF8 { get; }
static Encoding Unicode { get; } // UTF-16
static Encoding Default { get; }
static Encoding GetEncoding(int codepage);
```

Возвращает формат кодирования, соответствующий кодовой странице 8-битной кодировки с номером codepage. Например, для *Windows-кодировки кириллицы* «Cyrillic (Windows)» параметр codepage надо положить равным 1251.