Строки в C#

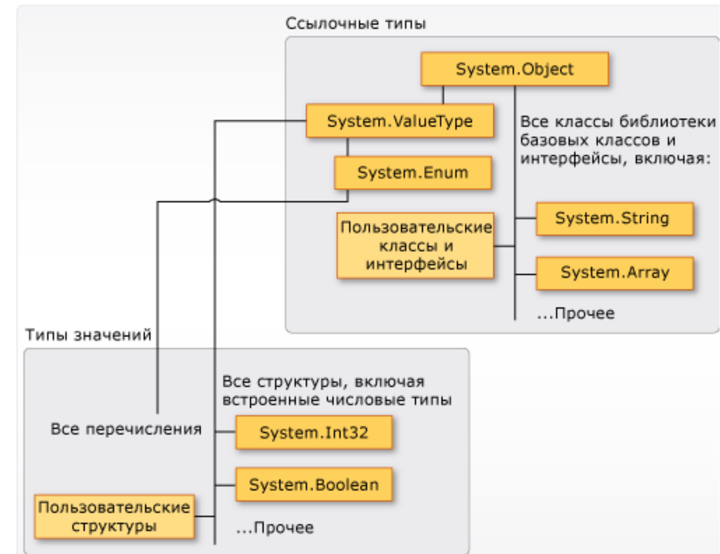
Класс char

char - одиночные символы. Размер 16 бит.

Тип **char**, как и все типы *C#*, является классом.

char, основанный на классе **System.Char** и использующий двухбайтную кодировку *Unicode*. Этот класс наследует свойства и методы класса **Object** и имеет большое число собственных методов.

Тип	Диапазон	Размер	Тип платформы .NET Framework
char	от U+0000 до U+FFFF	16-разрядный символ Юникода	System.Char



Unicode

Стандарт **Unicode** предложен в 1991 году некоммерческой организацией «Консорциум Юникода» (Unicode Consortium, Unicode Inc.). Применение этого стандарта позволяет закодировать очень большое число символов из разных письменностей: в документах Unicode могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, при этом становится ненужным переключение кодовых страниц

Стандарт состоит из двух основных разделов: универсальный набор символов (*UCS, universal character set*) и семейство кодировок (*UTF, Unicode transformation format*).

Универсальный набор символов задаёт однозначное соответствие символов кодам — элементам кодового пространства, представляющим неотрицательные целые числа.

Семейство кодировок определяет машинное представление последовательности кодов UCS.

Таким образом, первая версия Юникода представляла собой кодировку с фиксированным размером символа в 16 бит, то есть общее число кодов было 2^{16} (65 536). Отсюда происходит практика обозначения символов четырьмя шестнадцатеричными цифрами.

Кодовое пространство разбито на 17 *плоскостей* (*planes*) по 2^{16} (65 536) символов. Нулевая плоскость (*plane 0*) называется *базовой* (*basic*) и содержит символы наиболее употребительных письменностей. Остальные плоскости — дополнительные (*supplementary*). Первая плоскость (*plane 1*) используется, в основном, для исторических письменностей, вторая (*plane 2*) — для редко используемых иероглифов китайского письма (ККЯ), третья (*plane 3*) зарезервирована для архаичных китайских иероглифов. Плоскости 15 и 16 выделены для частного употребления

Большинство современных операционных систем в той или иной степени обеспечивают поддержку Юникода. В операционных системах семейства Windows NT для внутреннего представления имён файлов и других системных строк используется двухбайтовая кодировка UTF-16LE. Порядок байт — от младшего к старшему (Little-Endian, интеловский x86).

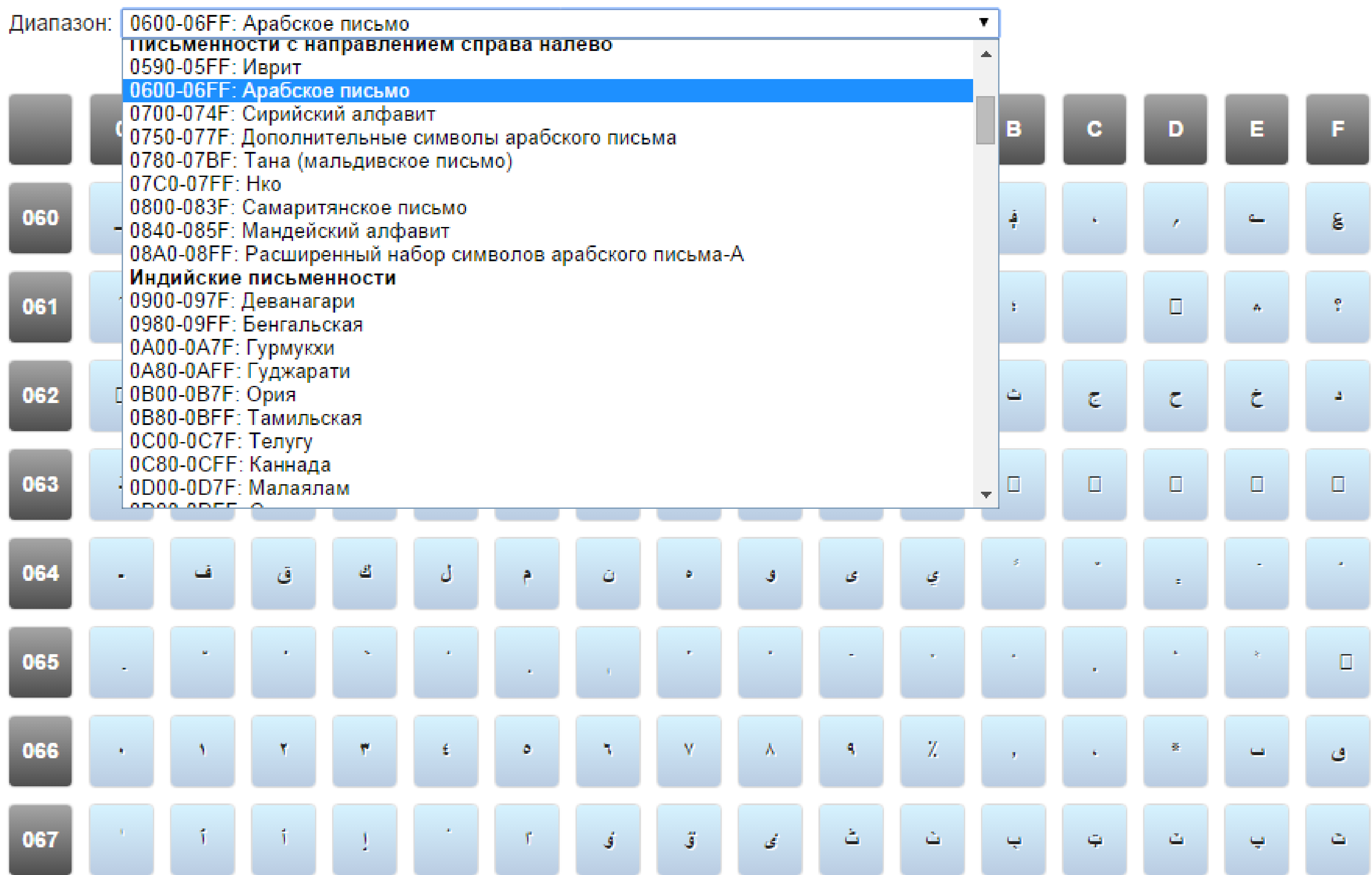
UNIX-подобные операционные системы, в том числе GNU/Linux, BSD, Mac OS X, используют для представления Юникода кодировку UTF-8.

Коды в стандарте Юникод разделены на несколько областей.

Область с кодами от U+0000 до U+007F содержит символы набора ASCII с соответствующими кодами.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000																
001																
002		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
006	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Далее расположены области знаков различных письменностей, знаки пунктуации и технические символы.



Часть кодов зарезервирована для использования в будущем.

Под символы кириллицы выделены области знаков с кодами от U+0400 до U+052F, от U+2DE0 до U+2DFF, от U+A640 до U+A69F

Работа по доработке стандарта продолжается. Новые версии выпускаются по мере изменения и пополнения таблиц символов.

Первый стандарт выпущен в 1991 году, последний — в 2014, следующий ожидается летом 2015 года

Для обозначения символов Unicode используется запись вида «U+*xxxx*» (для кодов 0…FFFF), Например, символ «я» (U+044F) имеет код $044F_{16}$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
040	Ѐ	Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ	Ѝ	Ў	Џ
041	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
042	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
043	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
044	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
045	ѐ	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	ѝ	ў	џ
046	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ
047	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ
048	Ҁ	ҁ	҂	◌҃	◌҄	◌҅	◌҆	◌҇	◌҈	◌҉	Ҋ	ҋ	Ҍ	ҍ	Ҏ	ҏ
049	Ґ	ґ	Ғ	ғ	Ҕ	ҕ	Җ	җ	Ҙ	ҙ	Қ	қ	Ҝ	ҝ	Ҟ	ҟ
04A	Ҡ	ҡ	Ң	ң	Ҥ	ҥ	Ҧ	ҧ	Ҩ	ҩ	Ҫ	ҫ	Ҭ	ҭ	Ү	ү
04B	Ұ	ұ	Ҳ	ҳ	Ҵ	ҵ	Ҷ	ҷ	Ҹ	ҹ	Һ	һ	Ҽ	ҽ	Ҿ	ҿ
04C	Ӏ	Ӂ	ӂ	Ӄ	ӄ	Ӆ	ӆ	Ӈ	ӈ	Ӊ	ӊ	Ӌ	ӌ	Ӎ	ӎ	ӏ
04D	Ӑ	ӑ	Ӓ	ӓ	Ӕ	ӕ	Ӗ	ӗ	Ә	ә	Ӛ	ӛ	Ӝ	ӝ	Ӟ	ӟ
04E	Ӡ	ӡ	Ӣ	ӣ	Ӥ	ӥ	Ӧ	ӧ	Ө	ө	Ӫ	ӫ	Ӭ	ӭ	Ӯ	ӯ
04F	Ӱ	ӱ	Ӳ	ӳ	Ӵ	ӵ	Ӷ	ӷ	Ӹ	ӹ	Ӻ	ӻ	Ӽ	ӽ	Ӿ	ӿ

Символьную константу можно задавать:

- \- символом, заключенным в одинарные кавычки;
- \- escape - последовательностью, задающей код символа;
- \- Unicode - последовательностью, задающей **Unicode**-код символа.

```
char[] chars = new char[4];

chars[0] = 'X';        // Character literal
chars[1] = '\x0058'; // Hexadecimal
chars[2] = (char)88; // Cast from integral type
chars[3] = '\u0058'; // Unicode

foreach (char c in chars)
{
   Console.Write(c + " ");
 }
// Output: X X X X
```

```
public void TestChar()
{
        char ch1 = 'A', ch2 = '\x5A', ch3 = ‚\u0058';
        char ch = new Char();
        ch = ch1;
        int code;
        code = ch;                          //преобразование символьного типа в тип int
        ch1 = (char) (code + 1);
        //преобразование символьного типа в строку s = ch;
        string s;
        s = ch1.ToString() + ch2.ToString() + ch3.ToString();
        Console.WriteLine("s= {0}, ch= {1}, code = {2}", s, ch, code);
}
```

Явные или неявные преобразования между классами char и string отсутствуют, но, благодаря методу ToString, переменные типа char стандартным образом преобразуются в тип string.

Можно неявно преобразовать **char** в **int**, **uint**, **long**, **ulong**, **float**, **double** и **decimal**.

Обратные преобразования целочисленных типов в тип **char** уже явные.

Приведем две процедуры, выполняющие взаимно-обратные операции

- получение по коду символа и
- получение символа по его коду:

```
public int SayCode(char sym)
 {
        return sym;
  }
```

Преобразование к целому типу выполняется неявно

```
public char SaySym(object code)
 {
        return (char)((int)code);
 }
```

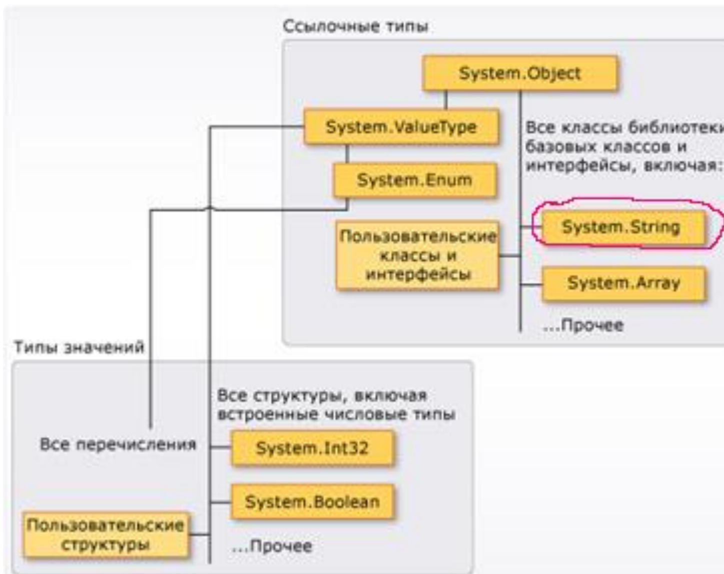
Преобразование явное. Формальный параметр имеет тип **Object,** что позволяет передавать ей в качестве аргумента код, заданный любым целочисленным типом. Платой за это является необходимость выполнять два явных преобразования.

Класс **Char,** как и все классы в **C#**, наследует свойства и методы родительского класса **Object.** Но у него есть и собственные методы и свойства

Метод	Описание
GetNumericValue	Возвращает численное значение символа, если он является цифрой, и (-1) в противном случае
GetUnicodeCategory	Все символы разделены на категории. Метод возвращает Unicode категорию символа.
IsControl	Возвращает true, если символ является управляющим
IsDigit	Возвращает true, если символ является десятичной цифрой
IsLetter	Возвращает true, если символ является буквой
IsLetterOrDigit	Возвращает true, если символ является буквой или цифрой
IsLower	Возвращает true, если символ задан в нижнем регистре
IsNumber	Возвращает true, если символ является числом (десятичной или шестнадцатиричной цифрой)
IsPunctuation	Возвращает true, если символ является знаком препинания
IsSeparator	Возвращает true, если символ является разделителем
IsSurrogate	Некоторые символы Unicode с кодом в интервале [0x1000, 0x10FFF] представляются двумя 16-битными "суррогатными" символами. Метод возвращает true, если символ является суррогатным
IsUpper	Возвращает true, если символ задан в верхнем регистре
IsWhiteSpace	Возвращает true, если символ является "белым пробелом". К белым пробелам, помимо пробела, относятся и другие символы, например, символ конца строки и символ перевода каретки
Parse	Преобразует строку в символ. Естественно, строка должна состоять из одного символа, иначе возникнет ошибка
ToLower	Приводит символ к нижнему регистру
ToUpper	Приводит символ к верхнему регистру
MaxValue, MinValue	Свойства, возвращающие символы с максимальным и минимальным кодом. Возвращаемые символы не имеют видимого образа

Класс String

С точки зрения регулярного программирования строковый **тип данных** string относится к числу одному из важных в *C#*. Этот тип определяет и поддерживает символьные строки. В целом ряде других языков программирования строка представляет собой *массив символов*. А в *C#* строки являются *объектами*. Следовательно, тип string относится к числу ***ссылочных***.



Класс **string,** основан на классе **System.String**

string - строки переменной длины

char - отдельные символы

Массив символов char[ ] - строки постоянной длины

Объект типа string можно создать из массива типа char.

Например:

```
char[ ] chararray = {'e', 'x', 'a', 'm', 'p', 'l', 'e'};
string str = new string(chararray);
```

Строковые литералы имеют тип **string** и могут быть написаны в двух формах: в кавычках и в кавычках с @.

```
string str = "Пример строки";
```

Если строка начинается с @, escape-последовательности *не* обрабатывается, благодаря чему можно удобно написать, например, полное имя и путь файла:

```
@"c:\Docs\Source\a.txt" // rather than "c:\\Docs\\Source\\a.txt"
```

Конструкторы:

- String(char ch, int iCount);
- String(char[ ] ach);
- String(char[ ] ach, int iStartIndex, int iCount);

```
public void TestDeclStrings()
{
   string world = "Мир";
   //string s1 = new string("Мир"); // - ошибка
   //string s2 = new string();          // - ошибка
   string s3 = new string('s', 5);
   char[ ] yes = "Yes".ToCharArray();
   string stryes = new string(yes);
   string strye = new string(yes,0,2);
}
```

Операции над строками

- присваивание (=);
- две операции проверки эквивалентности (==) и (!=);
- конкатенация или сцепление строк (+);
- взятие индекса ([ ]).

```
public void TestOpers()
{
   string s1 =“good ", s2 =“morning";
   string s3 = s1+s2;
   bool b1 = (s1 == s2);
   char ch1 = s1[0],  ch2=s2[0];
   s2 = s1;
   b1 = (s1 != s2);
   ch2 = s2[1];
}
```

Методы и свойства класса String

В *System.String* предоставляется набор методов для определения длины символьных данных, поиска подстроки в текущей строке, преобразования символов из верхнего регистра в нижний и наоборот, и т.д.

Метод	Назначение
Compare()	Статический метод, который позволяет сравнить две строки
CompareOrdinal()	То же, что Compare, но без учета локальных установок
Concat()	Комбинирует отдельные экземпляры строк в одну строку (конкатенация)
Contains()	Метод, который позволяет определить, содержится ли в строке определенная подстрока
CopyTo()	Копирует определенное число символов, начиная с определенной позиции в новый экземпляр массива
Equals()	Метод, который позволяет проверить, содержатся ли в двух строковых объектах идентичные символьные данные
Format()	Статический метод, позволяющий сформатировать строку с использованием других элементарных типов данных (например, числовых данных или других строк) и обозначений типа {0}
IndexOf()	Находит первое вхождение заданной подстроки или символа в строке
IndexOfAny()	Находит первое вхождение в строку любого символа из набора
Insert()	Метод, который позволяет вставить строку внутрь другой определенной строки

Метод	Назначение
Join()	Строит новую строку, комбинируя содержимое массива строк
LastIndexOf()	То же, что IndexOf, но находит последнее вхождение
LastIndexOfAny()	То же, что IndexOfAny, но находит последнее вхождение
PadLeft() PadRight()	Методы, которые позволяют дополнить строку какими-то символами, соответственно, справа или слева
Remove() Replace()	Методы, которые позволяют получить копию строки с соответствующими изменениями (удалением или заменой символов)
Split()	Метод, возвращающий массив string с присутствующими в данном экземпляре подстроками внутри, которые отделяются друг от друга элементами из указанного массива char или string
Substring()	Извлекает подстроку, начиная с определенной позиции строки
ToUpper () ToLower()	Методы, которые позволяют создавать копию текущей строки в формате, соответственно, верхнего или нижнего регистра
Trim()	Метод, который позволяет удалять все вхождения определенного набора символов с начала и конца текущей строки

Пример следующей программы использует несколько из вышеуказанных методов:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
  class Program
   {
      static void Main(string[] args)
             {
              // Сравним первые две строки

              string s1 = "это строка";
              string s2 = "это текст, а это строка" ;

              if (String.CompareOrdinal(s1, s2) != 0)
                          Console.WriteLine( "Строки s1 и s2 не равны" );

             if (String.Compare(s1, 0, s2, 13, 10, true) == 0)
                          Console.WriteLine( "При этом в них есть одинаковый текст" );

              // Конкатенация строк

              Console.WriteLine(String.Concat("\n" + "Один, два " , "три, четыре" ));
```

```
// Поиск в строке

// Первое вхождение подстроки
  if (s1.IndexOf("это") != -1)
    Console.WriteLine("Слово \"это\" найдено в строке, оно" + "находится на: {0} позиции" , s1.IndexOf("это"));

  // Последнее вхождение подстроки

  if (s2.LastIndexOf("это") != -1)
  Console.WriteLine("Последнее вхождение слова \"это\" находится" + "на {0} позиции", s2.LastIndexOf("это"));

  // Поиск из массива символов

  char[] myCh = {'Ы','х','т'};
  if (s2.IndexOfAny(myCh) != -1)
            Console.WriteLine( "Один из символов из массива ch " +  "найден в текущей строке на позиции {0}" ,
  s2.IndexOfAny(myCh));

  // Определяем начинается ли строка с заданной подстроки

  if (s2.StartsWith("это текст") == true)
        Console.WriteLine("Подстрока найдена!");

  // Определяем содержится ли в строке подстрока  на примере определения ОС пользователя

    string myOS = Environment.OSVersion.ToString();
    if (myOS.Contains("NT 5.1"))
          Console.WriteLine( "Ваша операционная система Windows XP" );
    Console.ReadLine(); }
 }
}
```

Постоянство строк

- Содержимое объекта типа string ***не подлежит изменению***. Это означает, что однажды созданную последовательность символов изменить нельзя.
- Но данное ограничение способствует более эффективной реализации символьных строк. Поэтому этот, на первый взгляд, очевидный недостаток на самом деле превращается в преимущество.
- Так, если требуется строка в качестве разновидности уже имеющейся строки, то для этой цели следует создать новую строку, содержащую все необходимые изменения. А поскольку неиспользуемые строковые объекты автоматически собираются в "мусор", то о дальнейшей судьбе ненужных строк можно даже не беспокоиться.

```
static void addNewString()
  {
      string s = "This is my stroke";
      s = "This is new stroke";
      // s1[0]='L'; ошибка
   }
```

Преобразования между char и string

Явные или неявные преобразования между классами char и string отсутствуют, но благодаря методу ToString, переменные типа char стандартным образом преобразуются в тип string

```
static void addNewString()
  {
      string s = "This is my stroke";
      s = "This is new stroke";
      // s1[0]='L'; ошибка
   }
```

```
public void TestDeclStrings()
{
   char[ ] yes = "Yes".ToCharArray();
   string stryes = new string(yes);
   string strye = new string(yes,0,2);
}
```