САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ

- **Сотавов Абакар Капланович**
- **Ассистент кафедры Информатики**(наб. канала Грибоедова, 30/32, ауд. 2038
- e-mail: sotavov@unecon.ru

- Материалы на сайте: http://de.unecon.ru/course/view.php?id=440

Символы и строки

Символы

Символьный тип char предназначен для хранения символов в кодировке Unicode. Символьный тип относится к встроенным типам данных C# и соответствует стандартному классу Char библиотеки .NET из пространства имен System. В этом классе определены статические методы, позволяющие задать вид и категорию символа, а также преобразовать символ в верхний или нижний регистр и в число. Некоторые методы приведены в

Некоторые методы класса System.Char

Метод	Описание
GetNumericValue	Возвращает числовое значение символа, если он является цифрой, и –1 в противном случае
IsControl	Возвращает true, если символ является управляющим
IsDigit	Возвращает true, если символ является десятичной цифрой
IsLetter	Возвращает true, если символ является буквой
IsLower	Возвращает true, если символ задан в нижнем регистре
IsUpper	Возвращает true, если символ записан в верхнем регистре
IsWhiteSpace	Возвращает true, если символ является пробельным (пробел, перевод строки и возврат каретки)
Parse	Преобразует строку в символ (строка должна состоять из одного символа)
ToLower	Преобразует символ в нижний регистр
MaxValue,MinValue	Возвращают символы с максимальным и минимальным кодами (эти символы не имеют видимого представления)

Листинг

Program.cs

Массивы символов

Массив символов, как и массив любого иного типа, построен на основе базового класса Array

Листинг

Program.cs

Строки типа string

Тип string предназначен для работы со строками символов в кодировке Unicode. Ему соответствует базовый класс System.String библиотеки .NET.

Создание строки:

```
1. string s;                              // инициализация отложена
2. string t = "qqq";   // инициализация строковым литералом
3. string u = new string(' ', 20);            // с пом. конструктора
4. string v = new string( a ); // создание из массива символов
// создание массива символов:  char[] a = { '0', '0', '0' };
```

Операции для строк

- присваивание (=);
- проверка на равенство (==);
- проверка на неравенство (!=);
- обращение по индексу ([]);
- сцепление (конкатенация) строк (+).

- Строки равны, если имеют одинаковое количество символов и совпадают посимвольно.
- Обращаться к отдельному элементу строки по индексу можно только для получения значения, но не для его изменения. Это связано с тем, что строки типа string относятся к неизменяемым типам данных.
- Методы, изменяющие содержимое строки, на самом деле создают новую копию строки. Неиспользуемые «старые» копии автоматически удаляются сборщиком мусора.

Некоторые элементы класса System.String

Название	Описание
Compare	Сравнение двух строк в алфавитном порядке. Разные реализации метода позволяют сравнивать строки и подстроки с учетом и без учета регистра и особенностей национального представления дат и т. д.
CompareOrdinal	Сравнение двух строк по кодам символов. Разные реализации метода позволяют сравнивать строки и подстроки
CompareTo	Сравнение текущего экземпляра строки с другой строкой
Concat	Конкатенация строк. Метод допускает сцепление произвольного числа строк
Copy	Создание копии строки

Format	Форматирование в соответствии с заданными спецификаторами формата
IndexOf, LastIndexOf,…	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора
Insert	Вставка подстроки в заданную позицию
Join	Слияние массива строк в единую строку. Между элементами массива вставляются разделители (см. далее)
Length	Длина строки (количество символов)
Remove	Удаление подстроки из заданной позиции
Replace	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
Split	Разделение строки на элементы, используя заданные разделители. Результаты помещаются в массив строк
Substring	Выделение подстроки, начиная с заданной позиции

Листинг

Program.cs

Пример форматирования строк

```
double a = 12.234;
int b = 29;
Console.WriteLine( " a = {0,6:C}  b = {1,2:X}", a, b );    //4
Console.WriteLine( " a = {0,6:0.##}  b = {1,5:0.# ' руб. '}", a, b); //5
Console.WriteLine(" a = {0:F3}  b = {1:D3}", a, b);
Console.WriteLine( " a = " + a.ToString("C"));
```

a = 12,23р. b = 1D

a = 12,23 b = 29 руб.

a = 12,234 b = 029

a = 12,23р.

{n[,m]**[:спецификатор_формата**[число]]**}**

Спецификаторы формата для строк

C или c	Вывод значений в денежном (currency) формате.
D или d	Вывод целых значений.
E или e	Вывод значений в экспоненциальном формате, то есть в виде d.ddd…E+ddd или d.ddd…e+ddd.
F или f	Вывод значений с фиксированной точностью.
G или g	Формат общего вида. Вывод значений с фиксированной точностью или в экспоненциальном формате, в зависимости от того, какой формат требует меньшего количества позиций.
N или n	Вывод значений в формате d,ddd,ddd.ddd. После спецификации можно задать целое число, определяющее длину дробной части
P или p	Вывод числа в процентном формате
R или r	Отмена округления числа при преобразовании в строку. Гарантирует, что при обратном преобразовании в значение того же типа получится то же самое
X или x	Вывод значений в шестнадцатеричном формате.

пользовательские шаблоны форматирования.

Число	Шаблон	Представление числа
1,243	00.00	01,24
1,243	#.##	1,24
0,1	00.00	00,10
0,1	#.##	,1

Пустые строки и строки null

•**Пустая строка** — экземпляр объекта System.String, содержащий 0 символов:

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

Для пустых строк можно вызывать методы.

•Строки со значениями **null**, напротив, не ссылаются на экземпляр объекта System.String, попытка вызвать метод для строки **null** вызовет исключение NullReferenceException. Однако строки **null** можно использовать в операциях объединения и сравнения с другими строками.

Строки типа StringBuilder

Класс StringBuilder определен в пространстве имен System.Text. Позволяет изменять значение своих экземпляров.

При создании экземпляра обязательно использовать операцию new и конструктор, например:

```
StringBuilder a = new StringBuilder();                    // 1
StringBuilder b = new StringBuilder( "qwerty" );          // 2
StringBuilder c = new StringBuilder( 100 );               // 3
StringBuilder d = new StringBuilder( "qwerty", 100 );     // 4
StringBuilder e = new StringBuilder( "qwerty", 1, 3, 100 );// 5
```

Конкатенация 50000 string ~ 1 мин., StringBuilder ~ 1 сек.

Основные элементы класса System.Text.StringBuilder

Append	Добавление в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки типа string
AppendFormat	Добавление форматированной строки в конец строки
Capacity	Получение или установка емкости буфера. Если устанавливаемое значение меньше текущей длины строки или больше максимального, генерируется исключение ArgumentOutOfRangeException
Insert	Вставка подстроки в заданную позицию
Length	Длина строки (количество символов)
MaxCapacity	Максимальный размер буфера
Remove	Удаление подстроки из заданной позиции
Replace	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
ToString	Преобразование в строку типа string

Листинг StringBuilder

Program.cs

Регулярные выражения

Регулярное выражение — шаблон (образец), по которому выполняется поиск соответствующего ему фрагмента текста.

тег html:

```
<[^>]+>
```

российский номер автомобиля:

```
[A-Z]\d{3}[A-Z]{2}\d\dRUS
```

IP-адрес:

```
\d\d?\d?\.\d\d?\d?\.\d\d?\d?\.\d\d?\d?
```

Регулярные выражения предназначены для обработки текстовой информации и обеспечивают:

эффективный **поиск** в тексте по заданному шаблону;

редактирование, замену и удаление подстрок;

формирование итоговых **отчетов** по результатам работы с текстом.

Язык описания регулярных выражений состоит из символов двух видов: обычных и метасимволов.

Обычный символ представляет в выражении сам себя.

Метасимвол:

класс символов (например, любая цифра **\d** или буква **\w**)

уточняющий символ (например, **^**).

повторитель (например, **+**).

Примеры:

выражение для поиска в тексте фрагмента «Вася» записывается с помощью четырех обычных символов «**Вася**»

выражение для поиска двух цифр, идущих подряд, состоит из двух метасимволов «**\d\d**»

выражение для поиска фрагментов вида «Вариант 1», «Вариант 2», …, «Вариант 9» имеет вид «**Вариант \d**»

выражение для поиска фрагментов вида «Вариант 1», «Вариант 23», «Вариант 719», …, имеет вид «**Вариант \d+**»

Метасимволы - классы символов

Класс символов	Описание	Пример
.	любой символ, кроме \n	c.t соответствует фрагментам cat, cut, c1t, c{t и т.д.
[]	любой одиночный символ из последовательности внутри скобок.	c[au1]t соответствует фрагментам cat, cut и c1t. c[a-z]t соответствует фрагментам cat, cbt, cct, cdt, …, czt
[^]	любой одиночный символ, не входящий в последовательность внутри скобок.	c[^au1]t соответствует фрагментам cbt, c2t, cXt и т.д. c[^a-zA-Z]t соответствует фрагментам cит, c1t, cЧt, c3t и т.д.
\w	любой алфавитно-цифровой символ, то есть символ из множества прописных и строчных букв и десятичных цифр	c\wt соответствует фрагментам cat, cut, c1t, cЮt и т.д. Не соответствует c{t, c;t и т.д.

\W	любой не алфавитно-цифровой символ, то есть символ, не входящий в множество прописных и строчных букв и десятичных цифр	c\Wt соответствует фрагментам c{t, c;t, c t и т.д. Не соответствует cat, cut, c1t, cЮt и т.д.
\s	любой пробельный символ, например, пробел, табуляция (\t, \v), перевод строки (\n, \r), новая страница (\f)	\s\w\w\w\s соответствует любому слову из трех букв, окруженному пробельными символами
\S	любой не пробельный символ, то есть символ, не входящий в множество пробельных	\s\S\S\s соответствует любым двум непробельным символам, окруженным пробельными.
\d	любая десятичная цифра	c\dt соответствует фрагментам c1t, c2t, …, c9t
\D	любой символ, не явдяющийся десятичной цифрой	c\Dt не соответствует фрагментам c1t, c2t, …, c9t.

Уточняющие (якорные) метасимволы

Мета-символ	фрагмент, совпадающий с регулярным выражением:
^	следует искать только в начале строки
$	следует искать только в конце строки
\A	следует искать только в начале многострочной строки
\Z	следует искать только в конце многострочной строки
\b	начинается или заканчивается на границе слова (т.е. между символами, соответствующими \w и \W)
\B	не должен встречаться на границе слова

Повторители

Мета-символ	Описание	Пример
*	0 или более повторений предыдущего элемента	ca*t соответствует фрагментам ct, cat, caat, caaaaaaaaaat и т.д.
+	1 или более повторений предыдущего элемента	ca+t соответствует фрагментам cat, caat, caaaaaaaaaat и т.д.
?	0 или 1 повторений предыдущего элемента	ca?t соответствует фрагментам ct и cat
{n}	ровно n повторений предыдущего элемента	ca{3}t соответствует фрагменту caaat. (cat){2} соответствует фрагменту catcat.
{n,}	по крайней мере n повторений предыдущего элемента	ca{3,}t соответствует фрагментам caaat, caaaat, caaaaaaaat и т.д.
{n,m}	от n до m повторений предыдущего элемента	ca{2,4}t соответствует фрагментам caat, caaat и caaaat

Примеры простых регулярных выражений

целое число (возможно, со знаком):

[-+]?**\d**+

вещественное число (может иметь знак и дробную часть, отделенную точкой):

[-+]?\d+**\.**?\d*

российский номер автомобиля (упрощенно):

[A-Z]**\d**{3}**[A-Z]**{2}\d\dRUS

ip-адрес (упрощенно):

(\d{1,3}**\.**){3}\d{1,3}

Для поддержки регулярных выражений в библиотеку .NET включены классы, объединенные в пространство имен System.Text.RegularExpressions.

Основной класс – **Regex**. Он реализует подсистему обработки регулярных выражений.

Подсистеме требуется предоставить:

Шаблон (регулярное выражение), соответствия которому требуется найти в тексте.

Текст, который требуется проанализировать с помощью шаблона.

См.:

http://msdn.microsoft.com/ru-ru/library/hs600312.aspx?ppud=4

Использование класса Regex

Обработчик регулярных выражений выполняет синтаксический **разбор** и **компиляцию** регулярного выражения, а также операции, **сопоставляющие** шаблон регулярного выражения с входной строкой.

Обработчик можно использовать одним из двух способов:

С помощью вызова статических методов класса Regex. Параметры метода содержат входную строку и шаблон регулярного выражения.

С помощью создания объекта Regex посредством передачи регулярного выражения в конструктор класса.

Методы класса Regex

позволяют выполнять следующие действия:

Определить, **встречается ли** во входном тексте шаблон регулярного выражения (метод IsMatch).

Извлечь из текста одно или все вхождения, соответствующие шаблону регулярного выражения (методы Match или Matches).

Заменить текст, соответствующий шаблону регулярного выражения (метод Replace).

Разделить строку на массив строк (метод Split).

Пример использования Regex.IsMatch

```
using System;
using System.Text.RegularExpressions;
public class Example
 { public static void Main()
   { string[] values = { "111-22-3333", "111-2-3333"};
     string pattern = @"^\d{3}-\d{2}-\d{4}$";
     foreach (string value in values)
      { if (Regex.IsMatch(value, pattern))
           Console.WriteLine("{0} is a valid SSN.", value);
        else Console.WriteLine("{0}: Invalid", value);
 } } }
 // Вывод:
 // 111-22-3333 is a valid SSN.
 // 111-2-3333: Invalid
```

Пример использования Regex.Matches

```
// совпадения со строкой "abc" во входной строке
using System;
using System.Text.RegularExpressions;
public class Example
 { public static void Main()
   { string pattern = "abc";
     string input = "abc123abc456abc789";
     foreach (Match match in Regex.Matches(input, pattern))
        Console.WriteLine("{0} found at position {1}.",
                          match.Value, match.Index);
 } }
 //Вывод:
 // abc found at position 0.
 // abc found at position 6.
 // abc found at position 12.
```

СПАСИБО ЗА ВНИМАНИЕ!

ИНЖЭКОН ENGECON

ФИНЭК
Санкт-Петербург

СПб ГУСЭ

ОБЪЕДИНЯЯ ЛУЧШЕЕ