**https://metanit.com/sharp/tutorial/7.2.php**

**Лекция 13**

**Работа со строками**

Довольно большое количество задач, которые могут встретиться при разработке приложений, так или иначе связано с обработкой строк. Например, парсинг веб-страниц, поиск в тексте, различные аналитические задачи, связанные с извлечением нужной информации из текста и т.д. Поэтому работе со строками уделяется особое внимание.

Что такое парсинг интернет-страниц?

Это процесс анализа содержимого сайтов и вычленения нужной информации. Фактически автоматизация процесса сбора информации на заданные темы и ее дальнейшая обработка.

Программы, осуществляющие парсинг (парсеры, чаще всего на JavaScript),

1. Проходятся по тысячам интернет-страниц.
2. Отделяют нужную человеку информацию от технических данных.
3. Отбрасывают ненужное, оставляя только то, что необходимо для заданной темы.
4. Производят упаковку данных в вид, нужный пользователю.

Результат этих поисков может быть оформлен в виде электронной таблицы или базы данных.

В дальнейшем итоговый результат нуждается в дополнительной обработке.

Читайте подробнее на FB.ru: <http://fb.ru/article/261908/parsing-chto-eto-takoe-i-kak-on-sozdaetsya>

В языке C# строковые значения представляет тип **string**, а вся функциональность работы с данным типом сосредоточена в классе **System.String**. Слово **string** является просто псевдонимом для класса **System.String**.

Объекты этого класса представляют текст как последовательность символов **Unicode**. Максимальный размер объекта String может составлять в памяти 2 ГБ (это около 1 миллиарда символов).

1. **Создание строк**

Создавать сроки можно, как используя *переменную* типа **string** и присваивая ей значение, так и применяя один из конструкторов класса **String**:

|  |  |
| --- | --- |
|  | **string s1 = "Группа 3ПКС-333";**  **string s2 = null;**    **string s3 = new String('a', 6); // результатом будет строка "aaaaaa"**  **string s4 = new String( new char[]{'w', 'o', 'r', 'l', 'd'} );** |

Конструктор **String** имеет различные версии (варианты перегруженных конструкторов).

Например, вызов конструктора **new String**('a', 6) создаст строку "aaaaaa". И так как строка представляет ссылочный тип, то она может хранить значение null.

1. **Строка как набор символов**

Так как строка хранит коллекцию символов, в ней определен индексатор для доступа к отдельным символам:

|  |  |
| --- | --- |
|  | **public char** this[ **int** index ] **{** get; **}** |

Применяя индексатор, мы можем обратиться к строке как к массиву символов и получить по индексу любой из ее символов:

|  |  |
| --- | --- |
|  | **string s1 = "Привет!";**  **char ch1 = s1[1]; // символ 'р'**  **Console.WriteLine(ch1);**  **Console.WriteLine(s1.Length);** |

Длину строки можно получить, используя свойство **Length** (как и в обычном массиве).

1. **Основные методы строк**

Основная функциональность **класса String** раскрывается через его методы, среди которых можно выделить следующие:

* **Compare**: сравнивает две строки с учетом текущей культуры пользователя (локали)
* **CompareOrdinal**: сравнивает две строки без учета локали
* **Contains**: определяет, содержится ли подстрока в строке
* **Concat**: соединяет строки
* **CopyTo**: копирует часть строки или всю строку в другую строку
* **EndsWith**: определяет, совпадает ли конец строки с подстрокой
* **Format**: форматирует строку
* **IndexOf**: находит индекс первого вхождения символа или подстроки в строке
* **Insert**: вставляет в строку подстроку
* **Join**: соединяет элементы массива строк
* **LastIndexOf**: находит индекс последнего вхождения символа или подстроки в строке
* **Replace**: замещает в строке символ или подстроку другим символом или подстрокой
* **Split**: разделяет одну строку на массив строк
* **Substring**: извлекает из строки подстроку, начиная с указанной позиции
* **ToLower**: переводит все символы строки в нижний регистр
* **ToUpper**: переводит все символы строки в верхний регистр
* **Trim**: удаляет начальные и конечные пробелы из строки

Приведем примеры использования этих методов.

1. **Конкатенация**

Конкатенация строк или объединение может производиться как с помощью операции **+**, так и с помощью метода **Concat (S1, S2)**:

|  |  |
| --- | --- |
|  | **string s1 = "Группа ";**  **string s2 = "3ПКС - 333";**  **string s3 = s1 + " " + s2; // результат: строка "Группа ПКС - 333"**  **string s4 = String.Concat(s3, "!!!"); // результат: строка " Группа ПКС - 123!!!"**    **Console.WriteLine(s4);** |

Метод **Concat**является статическим методом класса **String**, принимающим в качестве параметров **две** строки.

Имеются перегрузки метода, принимающие другое количество параметров.

Для объединения строк также может использоваться метод **Join**:

|  |  |
| --- | --- |
|  | **string s5 = "apple";**  **string s6 = "a day";**  **string s7 = "keeps";**  **string s8 = "a doctor";**  **string s9 = "away";**  **string[] values = new string[] { s5, s6, s7, s8, s9 };**    **String s10 = String.Join(" ", values);**  **// результат: строка "apple a day keeps a doctor away"** |

Использованная выше *версия метода* получает два параметра: строку-разделитель (в данном случае пробел) **и массив строк**, которые будут соединяться и разделяться разделителем.

1. **Сравнение строк**

Для расположения строк в алфавитном порядке (по первой букве строки) применяется **статический** метод **Compare (S1, S2)**. Приведенный ниже вариант метода принимает две строки и возвращает следующее число:

* Если первая строка стоит по алфавиту выше второй, то возвращается число меньше нуля.
* В противном случае возвращается число больше нуля.
* Если строки равны, то возвращается число 0.

|  |  |
| --- | --- |
|  | **string s1 = "Группа ";**  **string s2 = "3 ПКС - 333";**    **int result = String.Compare(s1, s2);**  **if (result<0)**  **Console.WriteLine("Строка s1 должна стоять перед строкой s2");**  **else if (result > 0)**  **Console.WriteLine("Строка s1 должна стоять после строки s2");**  **else**  **Console.WriteLine("Строки s1 и s2 необходимо сортировать по второй букве строки");**  **// Т.к. коды чисел лежат в диапазоне {48,…,57}, а коды русских букв {192 – 223, 224 - 255} результатом будет "Строка s2 перед строкой s1"** |

1. **Поиск в строке**

С помощью метода **IndexOf(substring)** мы можем определить **индекс первого вхождения отдельного символа или подстроки** в строке:

|  |  |
| --- | --- |
|  | **string s1 = "Привет, друзья!";**  **char ch = 'и';**  **int indexOfChar = s1.IndexOf(ch); // равно 2**  **Console.WriteLine(indexOfChar);**    **string subString = "вет";**  **int indexOfSubstring = s1.IndexOf(subString); // равно 3**  **Console.WriteLine(indexOfSubstring);** |

Подобным образом действует метод **LastIndexOf (subString)**, только находит индекс последнего вхождения символа или подстроки в строку.

Еще одна группа методов позволяет узнать начинается или заканчивается ли строка на определенную подстроку. Для этого предназначены методы  **StartsWith**(СТРОКА)и**EndsWith**(СТРОКА).

Например, у нас есть задача удалить из папки все файлы с расширением exe:

|  |  |
| --- | --- |
|  | **string path = @"C:\SomeDir";**    **string[] files = Directory.GetFiles(path);**    **for (int i = 0; i < files.Length; i++)**  **{**  **if(files[i].EndsWith(".exe"))**  **File.Delete(files[i]);**  **}** |

1. **Разделение строк**

С помощью функции **Split** можно разделить строку на массив подстрок. Существует несколько перегрузок метода.

Один из вариантов метода в качестве параметра функция **Split**(массив) принимает массив символов или строк, которые и будут служить разделителями.

Например, подсчитаем количество слов в сроке, разделив ее по пробельным символам:

|  |  |
| --- | --- |
|  | **string text = "И поэтому все так произошло";**    **string[] words = text.Split(new char[] { ' ' });**    **foreach (string s in words)**  **{**  **Console.WriteLine(s);**  **}** |

Это не лучший способ разделения по пробелам, так как во входной строке у нас могло бы быть несколько подряд идущих пробелов и в итоговый массив также бы попадали пробелы, поэтому лучше использовать другую версию метода – с двумя параметрами:

|  |  |
| --- | --- |
|  | **string[] words = text.Split(new char[] { ' ' },**  **StringSplitOptions.RemoveEmptyEntries);** |

Второй параметр **StringSplitOptions.RemoveEmptyEntries** говорит, что при проверке нужно удалять все пустые подстроки.

**Ещё примеры по поводу разделения строк:**

Пусть требуется разделить строку на подстроки, отделенные друг от друга заданными символами-разделителями.

В примере ниже задается строка символов, в которой присутствует несколько символов-разделителей. При помощи функции **Split** данная строка разделяется на подстроки, которые затем выводятся на экран каждая в отдельной строке.

Для задания символов-разделителей используется массив символов.

В этом примере также применяется функция **Trim**, необходимая в данном случае для того, чтобы убедиться в том, что заданная строка не состоит из одних лишь пробелов. Исходный код примера.

**string words = "строка, содержащая несколько слов, а также**

**знаков пунктуации: таких как двоеточие и точка.";**

**string [] split = words.Split(new Char [] {' ', ',', '.', ':'});**

**foreach (string s in split)**

**{**

**if (s.Trim() != "")**

**Response.Write(s+"</br>");**

**}**

Итогом работы данной программы будет следующий результат, изображенный на рис. 1.

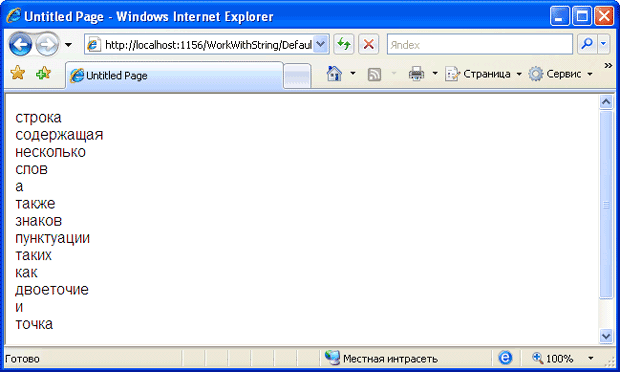


Рис. 1.Результат работы программы с использованием функции **Split**

**Обрезка строки**

Для обрезки начальных или концевых символов используется функция **Trim**:

|  |  |
| --- | --- |
|  | **string** text = " \*Группа 3ПКС-333. ";    text = text.**Trim**(); // результат "\*Группа 3ПКС-333."  text = text.**Trim**(new char[] { '\*', '.' }); // результат "Группа 3ПКС-333" |

Функция Trim без параметров обрезает начальные и конечные пробелы и возвращает обрезанную строку. Чтобы явным образом указать, какие начальные и конечные символы следует обрезать, мы передаём в функцию массив этих символов.

Эта функция имеет частичные аналоги: функция **TrimStart** обрезает начальные символы, а функция **TrimEnd** обрезает конечные символы.

Выделить определенную часть строки позволяет функция **Substring**. Одна из версий этой функции применяет два параметра:

* индекс начала подстроки и
* длину возвращаемой части строки.

|  |  |
| --- | --- |
|  | string text = "Хороший день";  // просто обрезаем начиная с третьего символа  text = text.Substring(2); // результат "роший день"  Console.WriteLine(text);  // Выделяем нужную подстроку  text = text.Substring(0, text.Length - 2);  // результат "роший де"   Console.WriteLine(text); |

Функция **Substring** также возвращает обрезанную строку. В качестве параметра первая использованная версия применяет индекс, начиная с которого надо обрезать строку.

**Вставка**

Для вставки одной строки в другую применяется функция Insert:

|  |  |
| --- | --- |
|  | string text = "Хороший день";  string subString = "замечательный ";    text = text.Insert(8, subString);  Console.WriteLine(text); |

Первым параметром в функции Insert является индекс, по которому надо вставлять подстроку, а второй параметр - собственно подстрока.

**Удаление строк**

Удалить часть строки помогает метод Remove:

|  |  |
| --- | --- |
|  | string text = "Хороший день";  // индекс последнего символа  int ind = text.Length - 1;  // вырезаем последний символ  text = text.Remove(ind);  Console.WriteLine(text);    // вырезаем первые два символа  text = text.Remove(0, 2); |

Первая версия метода Remove принимает индекс в строке, начиная с которого надо удалить все символы. Вторая версия принимает еще один параметр - сколько символов надо удалить.

**Замена**

Чтобы заменить один символ или подстроку на другую, применяется метод **Replace**:

|  |  |
| --- | --- |
|  | string text = "хороший день";    text = text.Replace("хороший", "плохой");  Console.WriteLine(text);    text = text.Replace("о", "");  Console.WriteLine(text); |

Во втором случае применения функции Replace строка из одного символа "о" заменяется на пустую строку, то есть фактически удаляется из текста. Подобным способом легко удалять какой-то определенный текст в строках.

**Смена регистра**

Для приведения строки к верхнему и нижнему регистру используются соответственно функции **ToUpper()** и **ToLower()**:

|  |  |
| --- | --- |
|  | string hello = "Hello world!";    Console.WriteLine(hello.ToLower()); // hello world!  Console.WriteLine(hello.ToUpper()); // HELLO WORLD! |

### 