**Регулярные выражения**

**Последнее обновление: 31.10.2015**

Классы StringBuilder и String предоставляют достаточную функциональность для работы со строками. Однако .NET предлагает еще один мощный инструмент - регулярные выражения. Регулярные выражения представляют эффективный и гибкий метод по обработке больших текстов, позволяя в то же время существенно уменьшить объемы кода по сравнению с использованием стандартных операций со строками.

Основная функциональность регулярных выражений в .NET сосредоточена в пространстве имен **System.Text.RegularExpressions**. А центральным классом при работе с регулярными выражениями является класс **Regex**. Например, у нас есть некоторый текст и нам надо найти в нем все словоформы какого-нибудь слова. С классом Regex это сделать очень просто:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | string s = "Бык тупогуб, тупогубенький бычок, у быка губа бела была тупа";  Regex regex = new Regex(@"туп(\w\*)");  MatchCollection matches = regex.Matches(s);  if (matches.Count > 0)  {      foreach (Match match in matches)          Console.WriteLine(match.Value);  }  else  {      Console.WriteLine("Совпадений не найдено");  } |

Здесь мы находим в искомой строке все словоформы слова "туп". В конструктор объекта Regex передается регулярное выражение для поиска. Далее мы разберем некоторые элементы синтаксиса регулярных выражений, а пока достаточно знать, что выражение туп(\w\*) обозначает, найти все слова, которые имеют корень "туп" и после которого может стоять различное количество символов. Выражение \w означает алфавитно-цифровой символ, а звездочка после выражения указывает на неопределенное их количество - их может быть один, два, три или вообще не быть.

Метод Matches класса Regex принимает строку, к которой надо применить регулярные выражения, и возвращает коллекцию найденных совпадений.

Каждый элемент такой коллекции представляет объект **Match**. Его свойство Value возвращает найденное совпадение.

**Параметр RegexOptions**

Класс Regex имеет ряд конструкторов, позволяющих выполнить начальную инициализацию объекта. Две версии конструкторов в качестве одного из параметров принимают перечисление RegexOptions. Некоторые из значений, принимаемых данным перечислением:

* **Compiled**: при установке этого значения регулярное выражение компилируется в сборку, что обеспечивает более быстрое выполнение
* **CultureInvariant**: при установке этого значения будут игнорироваться региональные различия
* **IgnoreCase**: при установке этого значения будет игнорироваться регистр
* **IgnorePatternWhitespace**: удаляет из строки пробелы и разрешает комментарии, начинающиеся со знака #
* **Multiline**: указывает, что текст надо рассматривать в многострочном режиме. При таком режиме символы "^" и "$" совпадают, соответственно, с началом и концом любой строки, а не с началом и концом всего текста
* **RightToLeft**: приписывает читать строку справа налево
* **Singleline**: устанавливает однострочный режим, а весь текст рассматривается как одна строка

Например:

|  |  |
| --- | --- |
| 1 | Regex regex = new Regex(@"туп(\w\*)", RegexOptions.IgnoreCase); |

При необходимости можно установить несколько параметров:

|  |  |
| --- | --- |
| 1 | Regex regex = new Regex(@"туп(\w\*)", RegexOptions.Compiled | RegexOptions.IgnoreCase); |

**Синтаксис регулярных выражений**

Рассмотрим вкратце некоторые элементы синтаксиса регулярных выражений:

* **^**: соответствие должно начинаться в начале строки (например, выражение @"^пр\w\*" соответствует слову "привет" в строке "привет мир")
* **$**: конец строки (например, выражение @"\w\*ир$" соответствует слову "мир" в строке "привет мир", так как часть "ир" находится в самом конце)
* **.**: знак точки определяет любой одиночный символ (например, выражение "м.р" соответствует слову "мир" или "мор")
* **\***: предыдущий символ повторяется 0 и более раз
* **+**: предыдущий символ повторяется 1 и более раз
* **?**: предыдущий символ повторяется 0 или 1 раз
* **\s**: соответствует любому пробельному символу
* **\S**: соответствует любому символу, не являющемуся пробелом
* **\w**: соответствует любому алфавитно-цифровому символу
* **\W**: соответствует любому не алфавитно-цифровому символу
* **\d**: соответствует любой десятичной цифре