**Регулярные выражения**

*System.Text.RegularExpressions.*

Классы пространства имен System.Text.RegularExpressions

|  |  |
| --- | --- |
| Capture | Представляет результаты из отдельной записи части выражения. Capture представляет одну подстроку для отдельной успешной записи. |
| CaptureCollection | Представляет последовательность подстрок записи. CaptureCollection возвращает набор записей, сделанных одной группой записи. |
| Group | Group представляет результаты отдельной группы записи. Группа записи может записать ноль, одну или более строк в одно совпадение из-за кванторов, таким образом Group предоставляет коллекцию объектов Capture. |
| GroupCollection | Представляет коллекцию групп записи. GroupCollection возвращает набор групп записи в одном совпадении. |
| Match | Представляет результаты из отдельного совпадения регулярного выражения. |
| MatchCollection | Представляет набор успешных совпадений, обнаруженных путем итеративного применения шаблона регулярного выражения к входной строке. |
| Regex | Представляет постоянное регулярное выражение. |
| RegexCompilationInfo | Представляет сведения о регулярном выражении, используемые для компиляции регулярного выражения в отдельную сборку. |
| RegexRunner | Класс RegexRunner является базовым классом для скомпилированных регулярных выражений. |
| RegexRunnerFactory | Создает класс RegexRunner |

Ключевой класс, который позволяет создавать регулярные выражения – **Regex** – имеет несколько конструкторов.

1) *new Regex(string pattern)*

*2) new Regex(string pattern, RegexOptions options)*

где: *pattern* – это маска регулярного выражения ;

*options* – дополнительные параметры, перечисление типа RegexOptions .

|  |  |
| --- | --- |
| *Структура перечисления RegexOptions* | |
| **Член** | **Описание** |
| CultureInvariant | Предписывает игнорировать национальные установки строки |
| ExplicitCapture | Модифицирует способ поиска соответствия, обеспечивая только буквальное соответствие |
| IgnoreCase | Игнорирует регистр символов во входной строке |
| IgnorePatternWhitespace | Удаляет из строки не защищенные управляющими символами пробелы и разрешает комментарии, начинающиеся со знака фунта или хеша |
| Multiline | Изменяет значение символов ^ и $ так, что они применяются к началу и концу каждой строки, а не только к началу и концу всего входного текста |
| RightToLeft | Предписывает читать входную строку справа налево вместо направления по умолчанию — слева направо (что удобно для некоторых азиатских и других языков, которые читаются в таком направлении) |
| Singleline | Специфицирует однострочный режим, в котором точка (.) символизирует соответствие любому символу |

|  |  |  |  |
| --- | --- | --- | --- |
| *Метасимволы, используемые в регулярных выражениях C#* | | | |
| **Символ** | **Значение** | **Пример** | **Соответствует** |
| Классы символов | | | |
| [...] | Любой из символов, указанных в скобках | [a-z] | В исходной строке может быть любой символ английского алфавита в нижнем регистре |
| [^...] | Любой из символов, не указанных в скобках | [^0-9] | В исходной строке может быть любой символ кроме цифр |
| . | Любой символ, кроме перевода строки или другого разделителя Unicode-строки |  |  |
| \w | Любой текстовый символ, не являющийся пробелом, символом табуляции и т.п. |  |  |
| \W | Любой символ, не являющийся текстовым символом |  |  |
| \s | Любой пробельный символ из набора Unicode |  |  |
| \S | Любой непробельный символ из набора Unicode. Обратите внимание, что символы \w и \S - это не одно и то же |  |  |
| \d | Любые ASCII-цифры. Эквивалентно [0-9] |  |  |
| \D | Любой символ, отличный от ASCII-цифр. Эквивалентно [^0-9] |  |  |
| **Символы повторения** | | | |
| {n,m} | Соответствует предшествующему шаблону, повторенному не менее n и не более m раз | s{2,4} | "Press", "ssl", "progressss" |
| {n,} | Соответствует предшествующему шаблону, повторенному n или более раз | s{1,} | "ssl" |
| {n} | Соответствует в точности n экземплярам предшествующего шаблона | s{2} | "Press", "ssl", но не "progressss" |
| ? | Соответствует нулю или одному экземпляру предшествующего шаблона; предшествующий шаблон является необязательным | Эквивалентно {0,1} |  |
| + | Соответствует одному или более экземплярам предшествующего шаблона | Эквивалентно {1,} |  |
| \* | Соответствует нулю или более экземплярам предшествующего шаблона | Эквивалентно {0,} |  |
| **Символы регулярных выражений выбора** | | | |
| | | Соответствует либо подвыражению слева, либо подвыражению справа (аналог логической операции ИЛИ). |  |  |
| (...) | Группировка. Группирует элементы в единое целое, которое может использоваться с символами \*, +, ?, | и т.п. Также запоминает символы, соответствующие этой группе для использования в последующих ссылках. |  |  |
| (?:...) | Только группировка. Группирует элементы в единое целое, но не запоминает символы, соответствующие этой группе. |  |  |
| **Якорные символы регулярных выражений** | | | |
| ^ | Соответствует началу строкового выражения или началу строки при многострочном поиске. | ^Hello | "Hello, world", но не "Ok, Hello world" т.к. в этой строке слово "Hello" находится не в начале |
| $ | Соответствует концу строкового выражения или концу строки при многострочном поиске. | Hello$ | "World, Hello" |
| \b | Соответствует границе слова, т.е. соответствует позиции между символом \w и символом \W или между символом \w и началом или концом строки. | \b(my)\b | В строке "Hello my world" выберет слово "my" |
| \B | Соответствует позиции, не являющейся границей слов. | \B(ld)\b | Соответствие найдется в слове "World", но не в слове "ld" |

**Пример 1**

using System;

using System.Text.RegularExpressions;

class Example

{

static void Main()

{

// Массив тестируемых строк

string[] test = {

"Wuck World", "Hello world", "My wonderful world"

};

// Проверим, содержится ли в исходных строках слово World

// при этом мы не укажем опции RegexOption

Regex regex = new Regex("World");

Console.WriteLine("Регистрозависимый поиск: ");

foreach (string str in test)

{

if (regex.IsMatch(str))

Console.WriteLine("В исходной строке: \"{0}\" есть совпадения!", str);

}

Console.WriteLine();

// Теперь укажем поиск не зависимый от регистра

regex = new Regex("World", RegexOptions.IgnoreCase);

Console.WriteLine("РегистроНЕзависимый поиск: ");

foreach (string str in test)

{

if (regex.IsMatch(str))

Console.WriteLine("В исходной строке: \"{0}\" есть совпадения!", str);

}

}

}

**Пример 2**

static void Main()

{

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");

}

}

**Пример 3**

class Program

{

static void Main()

{

string Out = "";

string text = "asdad asdfasfs ffhdfhd dfhgdfhgdfh dfds@gmail.com dsfgsgs sdgs@ee2fd.com dsgsg sdfsf dsfrrdve4@dfd.net sdsdgsg";

Regex regex = new Regex(@"\w+[a-zA-Z0-9\_.]+@+\w+[a-z0-9].[a-z]{2,3}");

Match match = regex.Match(text);

while (match.Success)

{

Out += match.Value + "\n";

match = match.NextMatch();

}

Console.Write(Out);

Console.ReadLine();

} }

**Пример 4**

using System;

using System.Text.RegularExpressions;

class Example

{

static void Main()

{

// Допустим в исходной строке нужно найти все числа,

// соответствующие стоимости продукта

string input = "Добро пожаловать в наш магазин, вот наши цены: " +

"1 кг. яблок - 20 руб. " +

"2 кг. апельсинов - 30 руб. " +

"0.5 кг. орехов - 50 руб.";

string pattern = @"\b(\d+\W?руб)";

Regex regex = new Regex(pattern);

// Получаем совпадения в экземпляре класса Match

Match match = regex.Match(input);

// отображаем все совпадения

while (match.Success)

{

// Т.к. мы выделили в шаблоне одну группу (одни круглые скобки),

// ссылаемся на найденное значение через свойство Groups класса Match

Console.WriteLine(match.Groups[1].Value);

// Переходим к следующему совпадению

match = match.NextMatch();

}

}

}

**Пример 5**

// Достигаем того же результата что и в предыдущем примере,

// используя метод Regex.Matches() возвращающий MatchCollection

foreach (Match match in regex.Matches(input))

{

Console.WriteLine(match.Groups[1].Value);

}

**Пример 6**

string pattern = @"(\d{3})-(\d{3}-\d{4})";

string input = "050-555-6666 097-932-1111 099-222-3333 063-888-9999";

MatchCollection matches = Regex.Matches(input, pattern);

foreach (Match match in matches)

{

Console.WriteLine("Код : {0}", match.Groups[1].Value);

Console.WriteLine("Номер телефона : {0}", match.Groups[2].Value);

Console.WriteLine();

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Метасимволы замены в регулярных выражениях C#* | | | | |
| **Символ** | **Описание** | **Пример шаблона** | **Пример шаблона замены** | **Результат (входная -> результирующая строки)** |
| $ number | Замещает часть строки, соответствующую группе number | \b(\w+)(\s)(\w+)\b | $3$2$1 | "один два" -> "два один" |
| $$ | Подставляет литерал "$" | \b(\d+)\s?USD | $$$1 | "103 USD" -> "$103" |
| $` | Замещает весь текст входной строки до соответствия | B+ | $` | "AABBCC" -> "AAAACC" |
| $' | Замещает весь текст входной строки после соответствия | B+ | $' | "AABBCC" -> "AACCCC" |
| $+ | Замещает последнюю захваченную группу | B+(C+) | $+ | "AABBCCDD" -> "AACCDD" |
| $\_ | Замещает всю входную строку | B+ | $\_ | "AABBCC" -> "AAAABBCCCC" |

**Пример 7**

using System;

using System.Text.RegularExpressions;

class Example

{

static void Main()

{

// Допустим в исходной строке нужно заменить "руб." на "$",

// а стоимость переместить после знака $

string input = "Добро пожаловать в наш магазин, вот наши цены: \n" +

"\t 1 кг. яблок - 20 руб. \n" +

"\t 2 кг. апельсинов - 30 руб. \n" +

"\t 0.5 кг. орехов - 50 руб. \n";

Console.WriteLine("Исходная строка:\n {0}", input);

// В шаблоне используются 2 группы

string pattern = @"\b(\d+)\W?(руб.)";

// Строка замены "руб." на "$"

string replacement1 = "$$$1"; // Перед первой группой ставится знак $,

// вторая группа удаляется без замены

input = Regex.Replace(input, pattern, replacement1);

Console.WriteLine("\nВидоизмененная строка: \n" + input);

}

}

**Пример 8**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

string myText = @"Сериализация представляет собой процесс сохранения объекта на диске. В другой части приложения или даже в совершенно отдельном приложении может производиться

десериализация объекта, возвращающая его в состояние, в котором он пребывал до сериализации.";

const string myReg = "со";

MatchCollection myMatch = Regex.Matches(myText, myReg);

Console.WriteLine("Все вхождения строки \"{0}\" в исходной строке: ", myReg);

foreach (Match i in myMatch)

Console.Write("\t" + i.Index);

// Усложним шаблон регулярного выражения

// введя в него специальные метасимволы

const string myReg1 = @"\b[с,д]\S\*ериализац\S\*";

MatchCollection match1 = Regex.Matches(myText, myReg1, RegexOptions.IgnoreCase);

findMyText(myText, match1);

Console.ReadLine();

}

static void findMyText(string text, MatchCollection myMatch)

{

Console.WriteLine("\n\nИсходная строка:\n\n{0}\n\nВидоизмененная строка:\n", text);

// Реализуем выделение ключевых слов в консоли другим цветом

for (int i = 0; i < text.Length; i++)

{

foreach (Match m in myMatch)

{

if ((i >= m.Index) && (i < m.Index + m.Length))

{

Console.BackgroundColor = ConsoleColor.Green;

Console.ForegroundColor = ConsoleColor.Black;

break;

}

else

{

Console.BackgroundColor = ConsoleColor.Black;

Console.ForegroundColor = ConsoleColor.White;

}

}

Console.Write(text[i]);

}

}

}

}

**Пример 9**

string pattern = @"((\d+)[\s+](\d+))";

string pattern1 = @"((\d+)[\s+](\w+)[\s+](\d+))";

string input = @"Обычная 1111 aa 222 строка 3333 444 . И еще просто 5555 666 набор цифр - как число 7777 888.";

int i = 1;

foreach (Match m in Regex.Matches(input, pattern))

{

Console.WriteLine("N groups {0} \tGroups {1} \n Group {2} \t Group {2} ", i, m.Groups[1].Value, m.Groups[2].Value, m.Groups[3].Value);

i++;

}

Console.WriteLine();

i = 1;

foreach (Match m in Regex.Matches(input, pattern1))

{

Console.WriteLine("N groups{0}\n Group {1} \t Group {2} \t Group {3}", i, m.Groups[2].Value, m.Groups[3].Value, m.Groups[4].Value);

i++;

}