Регулярные выражения

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.

С помощью регулярных выражений удобно обрабатывать, например, файлы в формате HTML, файлы журналов или длинные текстовые файлы. Для поддержки регулярных выражений в библиотеку .NET включены классы, объединенные в пространство имен `System.Text.RegularExpressions`.

Метасимволы

Регулярное выражение — это шаблон (образец), по которому выполняется поиск соответствующего ему фрагмента текста. Язык описания регулярных выражений состоит из символов двух видов: обычных и *метасимволов*. Обычный символ представляет в выражении сам себя, а метасимвол — некоторый класс символов, например любую цифру или букву.

Например, регулярное выражение для поиска в тексте фрагмента «Вася» записывается с помощью четырех обычных символов `Вася`, а выражение для поиска двух цифр, идущих подряд, состоит из двух метасимволов `\d\d`.

С помощью комбинаций метасимволов можно описывать сложные шаблоны для поиска. Например, можно описать шаблон для IP-адреса, адреса электронной почты, различных форматов даты, заголовков определенного вида и т. д.

ПРИМЕЧАНИЕ

Синтаксис регулярных выражений .NET в основном позаимствован из языка Perl 5. Неподготовленного человека вид сложного регулярного выражения может привести в замешательство, но при вдумчивом изучении он обязательно почувствует его красоту и очарование. Пожалуй, регулярные выражения более всего напоминают заклинания, по которым волшебным образом преобразуется текст. Ошибка всего в одном символе делает заклинание бессильным, зато, верно составленное, оно творит чудеса!

В табл. 15.2 описаны наиболее употребительные метасимволы, представляющие собой классы символов.

Метасимволы, перечисленные в табл. 15.3, уточняют позицию в строке, в которой следует искать совпадение с регулярным выражением, например, только в начале или в конце строки. Эти метасимволы являются мнимыми, то есть в тексте им не соответствует никакой реальный символ.

Таблица 15.2. Классы символов

Класс символов	Описание	Пример
.	Любой символ, кроме \n	Выражение c.t соответствует фрагментам cat, cut, c1t, c{t и т. д.
[]	Любой одиночный символ из последовательности, записанной внутри скобок. Допускается использование диапазонов символов	Выражение c[aul]t соответствует фрагментам cat, cut и c1t, а выражение c[a-z]t — фрагментам cat, cbt, cct, cdt, ..., czt
[^]	Любой одиночный символ, не входящий в последовательность, записанную внутри скобок. Допускается использование диапазонов символов	Выражение c[^aul]t соответствует фрагментам cbt, c2t, cXt и т. д., а выражение c[^a-zA-Z]t — 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	Выражение c\wt соответствует фрагментам cat, cut, c1t, c0t и т. д., но не соответствует фрагментам 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	Выражение c\Wt соответствует фрагментам c{t, c;t, c t и т. д., но не соответствует фрагментам cat, cut, c1t, c0t и т. д.
\s	Любой пробельный символ, например символ пробела, табуляции (\t, \v), перевода строки (\n, \r), новой страницы (\f)	Выражение \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	Выражение \s\S\S\s соответствует любым двум непробельным символам, окруженным пробельными
\d	Любая десятичная цифра	Выражение c\dt соответствует фрагментам c1t, c2t, ..., c9t
\D	Любой символ, не являющийся десятичной цифрой	Выражение c\Dt не соответствует фрагментам c1t, c2t, ..., c9t

Таблица 15.3. Уточняющие метасимволы

Метасимвол	Описание
^	Фрагмент, совпадающий с регулярным выражением, следует искать только в начале строки
$	Фрагмент, совпадающий с регулярным выражением, следует искать только в конце строки
\A	Фрагмент, совпадающий с регулярным выражением, следует искать только в начале многострочной строки

Метасимвол	Описание
\Z	Фрагмент, совпадающий с регулярным выражением, следует искать только в конце многострочной строки
\b	Фрагмент, совпадающий с регулярным выражением, начинается или заканчивается на границе слова (то есть между символами, соответствующими метасимволам \w и \W)
\B	Фрагмент, совпадающий с регулярным выражением, не должен встречаться на границе слова

Например, выражение ^cat соответствует символам cat, встречающимся в начале строки, выражение cat$ — символам cat, встречающимся в конце строки (то есть за ними идет символ перевода строки), а выражение ^$ представляет пустую строку, то есть начало строки, за которым сразу же следует ее конец.

В регулярных выражениях часто используют повторители. *Повторители* — это метасимволы, которые располагаются непосредственно после обычного символа или класса символов и задают количество его повторений в выражении. Например, если требуется записать выражение для поиска в тексте пяти идущих подряд цифр, вместо метасимволов \d\d\d\d\d можно записать \d{5}. Такому выражению будут соответствовать фрагменты 11111, 12345, 53332 и т. д.

Наиболее употребительные повторители перечислены в табл. 15.4.

Таблица 15.4. Повторители

Метасимвол	Описание	Пример
*	Ноль или более повторений предыдущего элемента	Выражение ca*t соответствует фрагментам ct, cat, caat, caaaaaaaaaat и т. д.
+	Одно или более повторений предыдущего элемента	Выражение ca+t соответствует фрагментам cat, caat, caaaaaaaaaat и т. д.
?	Ни одного или одно повторение предыдущего элемента	Выражение ca?t соответствует фрагментам ct и cat
{n}	Ровно n повторений предыдущего элемента	Выражение ca{3}t соответствует фрагменту caaat, а выражение (cat){2} — фрагменту catcat[1]
{n,}	По крайней мере n повторений предыдущего элемента	Выражение ca{3,}t соответствует фрагментам caaat, caaaat, caaaaaaaaaaat и т. д.
{n,m}	От n до m повторений предыдущего элемента	Выражение ca{2,4}t соответствует фрагментам caat, caaat и caaaat

Помимо рассмотренных элементов регулярных выражений можно использовать конструкцию *выбора* из нескольких элементов. Варианты выбора перечисляются

[1] Круглые скобки служат для группировки символов.

через вертикальную черту. Например, если требуется определить, присутствует ли в тексте хотя бы один элемент из списка «cat», «dog» и «horse», можно использовать выражение

```
cat|dog|horse
```

При поиске используется так называемый «ленивый» алгоритм, по которому поиск прекращается при нахождении самого короткого из возможных фрагментов, совпадающих с регулярным выражением.

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
```

ВНИМАНИЕ

Если требуется описать в выражении обычный символ, совпадающий с каким-либо метасимволом, его предваряют обратной косой чертой. Так, для поиска в тексте символа точки следует записать \., а для поиска косой черты — \\.

Например, для поиска в тексте имени файла cat.doc следует использовать регулярное выражение `cat\.doc`. Символ «точка» *экранируется* обратной косой чертой для того, чтобы он воспринимался не как метасимвол «любой символ» (в том числе и точка!), а непосредственно[1].

Для группирования элементов выражения используются круглые скобки. *Группирование* применяется во многих случаях, например, если требуется задать повторитель не для отдельного символа, а для последовательности (это использовано в предыдущей таблице). Кроме того, группирование служит для запоминания в некоторой переменной фрагмента текста, совпавшего с выражением, заключенным в скобки. Имя переменной задается в угловых скобках или апострофах:

```
(?<имя_переменной>фрагмент_выражения)
```

Фрагмент текста, совпавший при поиске с фрагментом регулярного выражения, заносится в переменную с заданным именем. Пусть, например, требуется выделить из текста номера телефонов, записанных в виде nnn-nn-nn. Регулярное выражение для поиска номера можно записать так:

```
(?<num>\d\d\d-\d\d-\d\d)
```

[1] Обратите внимание на то, что метасимволы регулярных выражений не совпадают с метасимволами, которые используются в шаблонах имен файлов, таких как *.doc.

При анализе текста в переменную с именем num будут последовательно записываться найденные номера телефонов.

Рассмотрим еще один вариант применения группирования — для формирования *обратных ссылок*. Все конструкции, заключенные в круглые скобки, автоматически нумеруются, начиная с 1. Эти номера, предваренные обратной косой чертой, можно использовать для ссылок на соответствующую конструкцию. Например, выражение (\w)\1 используется для поиска сдвоенных символов в словах (wa*ll*, ma*ss*, co*o*perate)[1].

Круглые скобки могут быть вложенными, при этом номер конструкции определяется порядком открывающей скобки в выражении. Примеры:

```
(Вася)\s+(должен)\s+(?<sum>\d+)\sруб\.\s+Ну что же ты, \1
```

В этом выражении три подвыражения, заключенных в скобки. Ссылка на первое из них выполняется в конце выражения. С этим выражением совпадут, например, фрагменты

```
Вася должен 5 руб. Ну что же ты, Вася
Вася    должен    53459 руб.    Ну что же ты, Вася
```

Выражение, задающее IP-адрес:

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

Адрес состоит из четырех групп цифр, разделенных точками. Каждая группа может включать от одной до трех цифр. Примеры IP-адресов: 212.46.197.69, 212.194.5.106, 209.122.173.160. Первая группа, заключенная в скобки, задает весь адрес. Ей присваивается номер 1. В нее вложены вторые скобки, определяющие границы для повторителя {3}.

Переменную, имя которой задается внутри выражения в угловых скобках, также можно использовать для обратных ссылок в последующей части выражения. Например, поиск двойных символов в словах можно выполнить с помощью выражения (?<s>\w)\k<s>, где s — имя переменной, в которой запоминается символ, \k — элемент синтаксиса.

В регулярное выражение можно помещать *комментарии*. Поскольку выражения обычно проще писать, чем читать, это — очень полезная возможность. Комментарий либо помещается внутрь конструкции (?#), либо располагается, начиная от символа # до конца строки[2].

Классы библиотеки .NET для работы с регулярными выражениями

Классы библиотеки .NET для работы с регулярными выражениями объединены в пространство имен System.Text.RegularExpressions.

---

[1] Если написать просто \w\w, будут найдены все пары алфавитно-цифровых символов.

[2] Для распознавания этого вида комментария должен быть включен режим x RegexOptions.IgnorePatternWhitespace.

Начнем с класса Regex, представляющего собственно регулярное выражение. Класс является неизменяемым, то есть после создания экземпляра его корректировка не допускается. Для описания регулярного выражения в классе определено несколько перегруженных *конструкторов*:

- Regex() — создает пустое выражение;
- Regex(String) — создает заданное выражение;
- Regex(String, RegexOptions) — создает заданное выражение и задает параметры для его обработки с помощью элементов перечисления RegexOptions (например, различать или не различать прописные и строчные буквы).

Пример конструктора, задающего выражение для поиска в тексте повторяющихся слов, расположенных подряд и разделенных произвольным количеством пробелов, независимо от регистра:

```
Regex rx = new Regex( @"\b(?<word>\w+)\s+(\k<word>)\b",
                      RegexOptions.IgnoreCase );
```

Поиск фрагментов строки, соответствующих заданному выражению, выполняется с помощью методов IsMatch, Match и Matches.

Метод IsMatch возвращает true, если фрагмент, соответствующий выражению, в заданной строке найден, и false в противном случае. В листинге 15.3 приведен пример поиска повторяющихся слов в двух тестовых строках. В регулярное выражение, приведенное ранее, добавлен фрагмент, позволяющий распознавать знаки препинания.

Листинг 15.3. Поиск в строке дублированных слов (методом IsMatch)

```
using System;
using System.Text.RegularExpressions;

public class Test
{   public static void Main()
    {
        Regex r = new Regex( @"\b(?<word>\w+)[.,:;!? ]\s*(\k<word>)\b",
                             RegexOptions.IgnoreCase );

        string tst1 = "Oh, oh! Give me more!";
        if ( r.IsMatch( tst1 ) ) Console.WriteLine( " tst1 yes" );
        else                     Console.WriteLine( " tst1 no"  );

        string tst2 = "Oh give me, give me more!";
        if ( r.IsMatch( tst2 ) ) Console.WriteLine( " tst2 yes" );
        else                     Console.WriteLine( " tst2 no"  );
    }
}
```

Результат работы программы:

```
tst1 yes
tst2 no
```

Повторяющиеся слова в строке tst2 располагаются не подряд, поэтому она не соответствует регулярному выражению. Для поиска повторяющихся слов, расположенных в произвольных позициях строки, в регулярном выражении нужно всего-навсего заменить пробел (\s) «любым символом» (.):

```
Regex r = new Regex( @"\b(?<word>\w+)[..:;!? ].*(\k<word>)\b",
                     RegexOptions.IgnoreCase );
```

Метод Match класса Regex, в отличие от метода IsMatch, не просто определяет, произошло ли совпадение, а возвращает объект класса Match — очередной фрагмент, совпавший с образцом. Рассмотрим листинг 15.4, в котором используется этот метод.

Листинг 15.4. Выделение из строки слов и чисел (методом Match)

```
using System;
using System.Text.RegularExpressions;
public class Test
{
    public static void Main()
    {
        string text    = "Салат - $4, борщ - $3, одеколон - $10.";
        string pattern = @"(\w+) - \$(\d+)[..]";
        Regex  r       = new Regex( pattern );
        Match  m       = r.Match( text );
        int    total   = 0;
        while ( m.Success )
        {
            Console.WriteLine( m );
            total += int.Parse( m.Groups[2].ToString() );
            m = m.NextMatch();
        }
        Console.WriteLine( "Итого: $" + total );
    }
}
```

Результат работы программы:

```
Салат - $4.
борщ - $3.
одеколон - $10.
Итого: $17
```

При первом обращении к методу Match возвращается первый фрагмент строки, совпавший с регулярным выражением pattern. В классе Match определено свойство Groups, возвращающее коллекцию фрагментов, совпавших с подвыражениями в круглых скобках. Нулевой элемент коллекции содержит весь фрагмент, первый элемент — фрагмент, совпавший с подвыражением в первых скобках, второй элемент — фрагмент, совпавший с подвыражением во вторых скобках, и т. д. Если при определении выражения задать фрагментам имена, как это было

сделано в предыдущем листинге, можно будет обратиться к ним по этим именам, например:

```
string pattern = @"(?'name'\w+) - \$(?'price'\d+)[.,]";
...
total += int.Parse( m.Groups["price"].ToString() );
```

ПРИМЕЧАНИЕ

Метод NextMatch класса Match продолжает поиск в строке с того места, на котором закончился предыдущий поиск.

Метод Matches класса Regex возвращает объект класса MatchCollection — коллекцию всех фрагментов заданной строки, совпавших с образцом.

Рассмотрим теперь пример применения метода Split класса Regex. Этот метод разбивает заданную строку на фрагменты в соответствии с разделителями, заданными с помощью регулярного выражения, и возвращает эти фрагменты в массиве строк. В листинге 15.5 строка из листинга 15.4 разбивается на отдельные слова.

Листинг 15.5. Разбиение строки на слова (методом Split)

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
public class Test
{   public static void Main()
    {
        string text        = "Салат -  $4, борщ -$3, одеколон - $10.";
        string pattern     = "[- ..]+";
        Regex  r           = new Regex( pattern );
        List<string> words = new List<string>( r.Split( text ) );
        foreach ( string word in words ) Console.WriteLine( word );
    }
}
```

Результат работы программы:

```
Салат
$4
борщ
$3
одеколон
$10
```

Метод Replace класса Regex позволяет выполнять замену фрагментов текста. Определено несколько перегруженных версий этого метода. Вот как выглядит пример простейшего применения метода в его статическом варианте, заменяющего все вхождения символа $ символами у.е.:

```
string text  = "Салат -  $4, борщ -$3, одеколон - $10.";
string text1 = Regex.Replace( text, @"\$", "у.е." );
```

Другие версии метода позволяют задавать любые действия по замене с помощью делегата MatchEvaluator, который вызывается для каждого вхождения фрагмента, совпавшего с заданным регулярным выражением.

ПРИМЕЧАНИЕ

Помимо классов Regex и Match в пространстве имен System.Text.RegularExpressions определены вспомогательные классы, например, класс Capture — фрагмент, совпавший с подвыражением в круглых скобках; класс CaptureCollection — коллекция фрагментов, совпавших со всеми подвыражениями в текущей группе; класс Group содержит коллекцию Capture для текущего совпадения с регулярным выражением и т. д.

В качестве более реального примера применения регулярных выражений рассмотрим программу анализа файла журнала веб-сервера. Это текстовый файл, каждая строка которого содержит информацию об одном соединении с сервером. Четыре строки файла приведены ниже:

```
ppp-48.pool-113.spbnit.ru - - [31/May/2002:02:08:32 +0400] "GET / HTTP/1.1" 200
2434 "http://www.price.ru/bin/price/firminfo_f?fid=10922&where=01&base=2"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
81.24.130.7 - - [31/May/2002:08:13:17 +0400] "GET /swf/menu.swf HTTP/1.1" 200
4682 "-" "Mozilla/4.0 (compatible; MSIE 5.01; Windows 98)"
81.24.130.7 - - [31/May/2002:08:13:17 +0400] "GET /swf/header.swf HTTP/1.1" 200
21244 "-" "Mozilla/4.0 (compatible; MSIE 5.01; Windows 98)"
gate.solvo.ru - - [31/May/2002:10:43:03 +0400] "GET / HTTP/1.0" 200 2422
"http://www.price.ru/bin/price/firminfo_f?fid=10922&where=01&base=1"
"Mozilla/4.0 (compatible; MSIE 5.0; Windows NT; DigExt)"
```

Подобные файлы могут иметь весьма значительный объем, поэтому составление итогового отчета в удобном для восприятия формате имеет важное значение. Если рассматривать каждую строку файла как совокупность полей, разделенных пробелами, то поле номер 0 содержит адрес, с которого выполнялось соединение с сервером, поле номер 5 — операцию (GET при загрузке информации), поле 8 — признак успешности выполнения операции (200 — успешно) и, наконец, поле 9 — количество переданных байтов.

Приведенная в листинге 15.6 программа формирует в формате HTML итоговый отчет, содержащий таблицу адресов, с которых выполнялось обращение к серверу, и суммарное количество переданных байтов для каждого адреса.

Листинг 15.6. Анализ журнала веб-сервера

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text.RegularExpressions;

public class Test
{    public static void Main()
     {
```

продолжение

Листинг 15.6 (*продолжение*)

```
        StreamReader f   = new StreamReader( "access_log" );
        StreamWriter w   = new StreamWriter( "report.htm" );
        Regex        get = new Regex( "GET" );
        Regex        r   = new Regex( " " );
        string s, entry;
        int value;
        string[] items   = new string[40];
        Dictionary<string, int> table = new Dictionary<string, int>();

        while ( (s = f.ReadLine() ) != null )
        {
            items = r.Split( s );
            if ( get.IsMatch( items[5] ) && items[8] == "200" )
            {
                entry = items[0];
                value = int.Parse( items[9] );
                if ( table.ContainsKey( entry ) ) table[entry] += value;
                else                              table[entry]  = value;
            }
        }
        f.Close();
        w.Write( "<html><head><title> Report </title></head><body>" +
                 "<table border =1 <tr><td> Computer <td> Bytes </tr>" );
        foreach ( string item in table.Keys )
            w.Write( "<tr><td>{0}<td>{1}</tr>", item, table[item] );
        w.Write( "</table></body></html>" );
        w.Close();
    }
}
```

Фрагмент результата работы программы показан на рис. 15.1.

Computer	Bytes
ppp-48.pool-113.spbnit.ru	107039
test223.sovam.com	2422
210.82.124.83	20052
ipblock209-209-002.octetgroup.net	162781
81.24.130.7	74756
gate.solvo.ru	113692
212.113.108.164	261199

Рис. 15.1. Фрагмент журнала веб-сервера

Файл access_log считывается построчно, каждая строка разбивается на поля, которые заносятся в массив items. Затем, если загрузка прошла успешно, о чем свидетельствуют значения GET и 200 полей 5 и 8, количество переданных байтов

(поле 9) преобразуется в целое и заносится в хеш таблицу по ключу, которым служит адрес, хранящийся в поле 0.

Для формирования HTML файла report.htm используются соответствующие теги. Файл можно просмотреть, например, с помощью Internet Explorer. Как ви дите, программа получилась весьма компактной за счет использования стандарт ных классов библиотеки .NET[1].