Лекция Регулярные выражения

Стандартный *класс string* позволяет выполнять над строками различные *операции*, в том числе *поиск*, замену, вставку и удаление подстрок. Тем не менее, есть классы задач по обработке символьной информации, где стандартных возможностей явно не хватает. Чтобы облегчить решение подобных задач, в *Net* Framework встроен более мощный аппарат работы со строками, основанный на регулярных выражениях.

Регулярные выражения предназначены для обработки текстовой информации и обеспечивают:

1. Эффективный поиск в тексте по заданному шаблону;
2. Редактирование текста;
3. Формирование итоговых отчетов по результатам работы с текстом.

Подробно рассмотрим первые два аспекта применения регулярных выражений.

Метасимволы в регулярных выражениях

Регулярное выражение - это шаблон, по которому выполняется поиск соответствующего фрагмента текста. Язык описания регулярных выражений состоит из символов двух видов: обычных символов и метасимволов. Обычный символ представляет в выражении сам себя, а метасимвол - некоторый *класс символов*.

Рассмотрим наиболее употребительные метасимволы:

Класс символов	Описание	Пример
.	Любой символ, кроме `\n`.	Выражение `c. t` соответствует фрагментам: `cat`, `cut`, `c#t`, `c{t` и т.д.
`[]`	Любой одиночный символ из последовательности, записанной внутри скобок. Допускается использование *диапазонов символов*.	Выражение `c[aui]t` соответствует фрагментам: `cat`, `cut`, *`cit`*. Выражение `c[a-c]t` соответствует фрагментам: `cat`, *`cbt`*, `cct`.
`[^]`	Любой одиночный символ, не входящий в последовательность, записанную внутри скобок. Допускается использование *диапазонов символов*.	Выражение `c[^aui]t` соответствует фрагментам: *`cbt`*, `cct`, `c2t` и т.д. Выражение `c[^a-c]t` соответствует фрагментам: `cdt`, *`cet`*, `c%t` и т.д.
`\w`	Любой алфавитно – цифровой символ.	Выражение `c\wt` соответствует фрагментам: *`cbt`*, `cct`, `c2t` и т.д., но не соответствует фрагментам `c%t`, `c{t` и т.д.
`\W`	Любой не алфавитно – цифровой символ.	Выражение `c\Wt` соответствует фрагментам: `c%t`, `c{t`, `c. t` и т.д., но не соответствует фрагментам *`cbt`*, `cct`, `c2t` и т.д.
`\s`	Любой пробельный символ.	Выражение `\s\w\w\w\s` соответствует любому слову из трех букв, окруженному пробельными символами.
`\S`	Любой не пробельный символ.	Выражение `\s\S\S\S\s` соответствует любым трем непробельным символам, окруженным пробельными.
`\d`	Любая десятичная цифра	Выражение `c\dt` соответствует фрагментам: `c1t`, `c2t`, `c3t` и т.д.
`\D`	Любой символ, не являющийся десятичной цифрой	Выражение `c\Dt` не соответствует фрагментам: `c1t`, `c2t`, `c3t` и т.д.

Кроме метасимволов, обозначающие классы символов, могут применяться уточняющие метасимволы:

Уточняющие символы	Описание
^	Фрагмент, совпадающий с регулярными выражениями, следует искать только в начале строки
$	Фрагмент, совпадающий с регулярными выражениями, следует искать только в конце строки
\A	Фрагмент, совпадающий с регулярными выражениями, следует искать только в начале многострочной строки
\Z	Фрагмент, совпадающий с регулярными выражениями, следует искать только в конце многострочной строки
\b	Фрагмент, совпадающий с регулярными выражениями, начинается или заканчивается на границе слова, т.е. между символами, соответствующими метасимволам \w и \W
\B	Фрагмент, совпадающий с регулярными выражениями, не должен встречаться на границе слов

В регулярных выражениях часто используются повторители - метасимволы, которые располагаются непосредственно после обычного символа или группы символов и задают количество его повторений в выражении.

Повторители	Описание	Пример
*	Ноль или более повторений предыдущего элемента	Выражение ca*t соответствует фрагментам: ct, cat, caat, caaat и т.д.
+	Одно или более повторений предыдущего элемента	Выражение ca+t соответствует фрагментам: cat, caat, caaat и т.д.
?	Не более одного повторения предыдущего элемента	Выражение ca?t соответствует фрагментам: ct, cat.
{n}	Ровно n повторений предыдущего элемента	Выражение ca{3}t соответствует фрагменту: caaat. Выражение (cat){2} соответствует фрагменту: c а tcat.
{n,}	По крайней мере n повторений предыдущего элемента	Выражение ca{3,}t соответствует фрагментам: c ааа t, caaaat, caaaaaaat и т.д. Выражение (cat){2,} соответствует фрагментам: catcat, catcatcat и т.д.
{n, m}	От n до m повторений предыдущего элемента	Выражение ca{2,4}t соответствует фрагментам: c аа t, caaat, caaaat.

Регулярное выражение записывается в виде строкового литерала, причем перед строкой необходимо ставить символ @, который говорит о том, что строку нужно будет рассматривать и в том случае, если она будет занимать несколько строчек на экране. Однако символ @ можно не ставить, если в качестве шаблона используется шаблон без метасимволов.

Замечание. Если нужно найти какой-то символ, который является метасимволом, например, точку, можно это сделать защитив ее обратным слэшем. Т.е. просто точка означает любой одиночный символ, а \. означает просто точку.

Примеры регулярных выражений:

1. слово rus - @"rus" или "rus"
2. номер телефона в формате xxx-xx-xx - @"\d\d\d-\d\d-\d\d" или @"\d{3}(-\d\d){2}"
3. номер автомобиля - @"[A-Z]\d{3}[A-Z]{2}\d{2,3}RUS"

Задания. Запишите регулярное выражение, соответствующее:

1. дате в формате дд.мм.гг или дд.мм.гггг
2. времени в формате чч.мм или чч:мм
3. целому числу (со знаком и без)
4. вещественному числу (со знаком и без, с дробной частью и без, с целой частью и без)

Поиск в тексте по шаблону

Пространство имен библиотеки базовых классов `System.Text.RegularExpressions` содержит все объекты платформы .NET Framework, имеющие отношение к регулярным выражениям. Важнейшим классом, поддерживающим регулярные выражения, является класс `Regex`, который представляет неизменяемые откомпилированные регулярные выражения. Для описания регулярного выражения в классе определено несколько перегруженных конструкторов:

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

Поиск фрагментов строки, соответствующих заданному выражению, выполняется с помощью методов `IsMatch`, `Match`, `Matches` класса `Regex`.

Метод `IsMatch` возвращает `true`, если фрагмент, соответствующий выражению, в заданной строке найден, и `false` в противном случае. Например, попытаемся определить, встречается ли в заданном тексте слово `собака`:

```
static void Main()
{
  Regex r = new Regex("собака",RegexOptions.IgnoreCase);
  string text1 = "Кот в доме, собака в конуре.";
  string text2 = "Котик в доме, собачка в конуре.";
  Console.WriteLine(r.IsMatch(text1));
  Console.WriteLine(r.IsMatch(text2));
}
```

Замечание. `RegexOptions.IgnoreCase` - означает, что регулярное выражение применяется без учета регистра символов.

Можно использовать конструкцию выбора из нескольких элементов. Варианты выбора перечисляются через вертикальную черту. Например, попытаемся определить, встречается ли в заданном тексте слов `собака` или `кот`:

```
static void Main(string[] args)
{
  Regex r = new Regex("собака|кот",RegexOptions.IgnoreCase);
  string text1 = "Кот в доме, собака в конуре.";
  string text2 = "Котик в доме, собачка в конуре.";
  Console.WriteLine(r.IsMatch(text1));
  Console.WriteLine(r.IsMatch(text2));
}
```

Попытаемся определить, есть ли в заданных строках номера телефона в формате `xx-xx-xx` или `xxx-xx-xx`:

```
static void Main()
{
  Regex r = new Regex(@"\d{2,3}(-\d\d){2}");
  string text1 = "tel:123-45-67";
  string text2 = "tel:no";
  string text3 = "tel:12-34-56";
  Console.WriteLine(r.IsMatch(text1));
  Console.WriteLine(r.IsMatch(text2));
  Console.WriteLine(r.IsMatch(text3));
}
```

Задание. Измените программу так, чтобы можно было определить, содержится в тексте дата в формате дд.мм.гг.

Метод `Match` класса `Regex` не просто определяет, содержится ли текст, соответствующий шаблону, а возвращает объект класса `Match` - последовательность фрагментов текста, совпавших с шаблоном. Следующий пример позволяет найти все номера телефонов в указанном фрагменте текста:

```
static void Main()
{
  Regex r = new Regex(@"\d{2,3}(-\d\d){2}");
  string text = @"Контакты в Москве tel:123-45-67, 123-34-56; fax:123-56-45
                  Контакты в Саратове tel:12-34-56; fax:12-56-45";
  Match tel = r.Match(text);
```

```
  while (tel.Success)
  {
    Console.WriteLine(tel);
    tel = tel.NextMatch();
  }
}
```

Следующий пример позволяет подсчитать сумму целых чисел, встречающихся в тексте:

```
static void Main()
{
  Regex r = new Regex(@"[-+]?\d+");
  string text = @"5*10=50 -80/40=-2";
  Match teg = r.Match(text);
  int sum = 0;
  while (teg.Success)
  {
    Console.WriteLine(teg);
    sum += int.Parse(teg.ToString());
    teg = teg.NextMatch();
  }
  Console.WriteLine("sum=" + sum);
}
```

Задание. Измените программу так, чтобы на экран дополнительно выводилось количество найденных чисел.

Метод `Matches` класса `Regex` возвращает объект класса `MatchCollection` - коллекцию всех фрагментов заданной строки, совпавших с шаблоном. При этом метод `Matches` многократно запускает метод `Match`, каждый раз начиная поиск с того места, на котором закончился предыдущий поиск.

```
static void Main(string[] args)
  {
    string text = @"5*10=50 -80/40=-2";
    Regex theReg = new Regex(@"[-+]?\d+");
    MatchCollection theMatches = theReg.Matches(text);
    foreach (Match theMatch in theMatches)
    {
      Console.Write("{0} ", theMatch.ToString());
    }
    Console.WriteLine();
  }
}
```

Редактирование текста

Регулярные выражения могут эффективно использоваться для редактирования текста. Например, метод `Replace` класса `Regex` позволяет выполнять замену одного фрагмента текста другим или удаление фрагментов текста:

Пример *1*. Изменение номеров телефонов:

```
static void Main(string[] args)
 {
 string text = @"Контакты в Москве tel:123-45-67, 123-34-56; fax:123-56-45.
Контакты в Саратове tel:12-34-56; fax:11-56-45";
 Console.WriteLine("Старые данные\n"+text);
 string newText=Regex.Replace(text, "123-", "890-");
 Console.WriteLine("Новые данные\n" + newText);
 }
```

Задание. Измените программу так, чтобы шестизначные номера заменялись на семизначные добавлением 0 после первых двух цифр. Например, номер `12-34-56` заменился бы на `120-34-56`.

Пример 2. Удаление всех номеров телефонов из текста:

```
static void Main()
  {
    string text = @"Контакты в Москве tel:123-45-67, 123-34-56; fax:123-56-45.
                Контакты в Саратове tel:12-34-56; fax:12-56-45";
    Console.WriteLine("Старые данные\n"+text);
    string newText=Regex.Replace(text, @"\d{2,3}(-\d\d){2}", "");
    Console.WriteLine("Новые данные\n" + newText);
  }
}
```

Задание. Измените программу так, чтобы из текста удалялись слова `tel` и `fax` (если после данных слов стоят двоеточия, то их тоже следует удалить).

Пример 3. Разбиение исходного текста на фрагменты:

```
static void Main()
{
  string text = @"Контакты в Москве tel:123-45-67, 123-34-56; fax:123-56-45.
            Контакты в Саратове tel:12-34-56; fax:12-56-45";
  string []newText=Regex.Split(text,"[ ,.:;]+");
  foreach( string a in newText)
  Console.WriteLine(a);
}
```