В.В. Подбельский

Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

06 Часть 2

Введение в Регулярные Выражения

Благодарность Д.Д. Сагалову за обновление формата презентации

Общие Сведения

Обозначение в DOS всех файлов с одним именем:
«Итоговая_ведомость.*»

Традиционные задачи:

- Проверка наличия подстрок.
- Проверка соответствия строки.
- Выбор из строки подстрок.
- Замены подстрок.

Регулярные выражения в .NET находятся в отдельном пространстве имён:
using System.Text.RegularExpressions;

Язык Регулярных Выражений

Элементы регулярного выражения:

- **Терминальные символы** (объекты формальной грамматики, имеющие в нём конкретное неизменяемое значение и являющиеся элементом построения слов данного языка, обобщение понятия “буквы”);
- **Метасимволы** (объекты, обозначающий какую-либо сущность языка и не имеющие конкретного символьного значения).

Прототип метода поиска подстроки:

```
public static Match Match(string str, string reg)
```

- `str` - исследуемая строка;
- `reg` - регулярное выражение.

Свойства Класса Match

Свойство	Значение
Index	Самая левая позиция в строке, начиная с которой размещена найденная подстрока.
Length	Длина найденной подстроки.
Value	Найденная подстрока.
Success	Логическое значение, обозначающее успех поиска.

Match: Примеры

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

Match result = Regex.Match("together", @"th");
Console.WriteLine(result.Success);
Console.WriteLine(result.Value);
Console.WriteLine(result.Index);
Console.WriteLine(result.Length);
Console.WriteLine(result.ToString());
// \u0066 = 'f', \u006F = 'o', \u0072 = 'r'.
Console.WriteLine(Regex.Match("format", @"\u0066\u006F\u0072"));
```

Вывод:

```
True
th
4
2
th
for
```

Экранирование Метасимволов

\	➔	\\	*	➔	*
+	➔	\+	?	➔	\?
|	➔	\\|	{	➔	\{
}	➔	\}	[	➔	\[
]	➔	\]	(	➔	\(
)	➔	\)	^	➔	\^
$	➔	\$	.	➔	\.
#	➔	\#			

Подмножества Символов

`.` – любой одиночный символ

Примеры:

```
Console.WriteLine(Regex.Match("шило и мыло", @"..ло"));      // шило

Console.WriteLine(Regex.Match("11.22.33.44.55.66", @"4\.5")); // 4.5
```

Метод Matches

Прототип:

```
public static MatchCollection Matches(string str, string reg)
```

Пример:

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

foreach (Match mat in
          Regex.Matches("11.22.33.44.55.66", @".\.."))
{
    Console.Write(mat.Value + "  ");
}
```

Вывод:
1.2 2.3 3.4 4.5 5.6

Метасимвол Альтернативного Выбора

MatchCollection – коллекция элементов типа Match

Пример подмножества шестнадцатеричных цифр:

[0-9a-fA-F]

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

// \u0020 - [пробел]
MatchCollection group = Regex.Matches
        ("кот кит кат кто ком", @"\u0020к[ио].");
foreach (Match mat in group)
{
    Console.WriteLine(mat.Value);
}
```

Вывод:

кит

ком

Обозначения Подмножеств

\d - десятичная цифра ➔ [0-9]
\w - символ идентификатора, ➔ [a-zA-Z_0-9]
\s - пробельный символ ➔ [\n\r\t\f]

\p{*категория*}.

Категории Символов

\p{L} – любые буквы любого языка,
\p{Lu} – буквы в верхнем регистре (прописные),
\p{Ll} – буквы в нижнем регистре (строчные),
\p{N} – цифры,
\p{P} – любые пунктуационные знаки,
\p{M} – диакритические знаки (ë, î, ö),
\p{S} – символы отличные от пробельных,
\p{Z} – пробелы, невидимые разделители,
\p{C} – управляющие символы.

«Отрицательные» Подмножества

Пример:
[^0-4] – запрещены цифры от 0 до 4

\D ➔ [^0-9]
\W ➔ [^a-zA-Z_0-9]
\S ➔ [^\n\r\t\f]
\P{*категория*} - любой символ, кроме символа, принадлежащего категории.

Директивы Нулевой Ширины

^ - Соответствие в начале строки.
$ - Соответствие в конце строки или перед символом **\n**.

\A - Соответствие в начале строки.
\z - Соответствие в конце строки.
\G - Соответствие в конце другого соответствия.
\b - Соответствие на границе между **\w** и **\W** .
\B - Соответствие не на границе **\b**.

Примеры

```
MatchCollection res1 = Regex.Matches("кот кит кат кто ком",
                                     @"к[^та].$");
foreach (Match mat in res1)
{
    Console.Write(mat.Value + "  ");
}
```

Вывод:
ком

```
MatchCollection res2 = Regex.Matches("кот кит кат кто ком",
                                     @"^к[^та].");
foreach (Match mat in res2)
{
    Console.Write(mat.Value + "  ");
}
```

Вывод:
кот

Квантификаторы

"zo*"	соответствует "z", "zo" и "zooo".
"zo+"	соответствует "zo" и "zooo" (не "z").
"do(es)?"	соответствует "do" и "does" (не "doeses").
{n}	"o{2}" соответствует "oo" (не "o" / "ooo").
{n,}	"o{2,}" соответствует "ooo" (не "o"). "o{1,}" == "o+", "o{0,}" == "o*".
{n,m}	m и n — неотрицательные целые числа, где n <= m. "o{0,1}" == "o?".

Пример – Выделение Слов из Текста

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

string line = " 38 попугаев и другие    персонажи";
foreach (Match mt in Regex.Matches(line, @"\b\w+\b"))
{
    Console.WriteLine(mt.Value);
}
```

Вывод:
38
попугаев
и
другие
персонажи

Пример с Объектом Класса Regex

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

Regex key = new Regex(@"\w{6,}");
string citato = "Conditio sine qua non – непременное условие";
MatchCollection words = key.Matches(citato);
foreach (Match mt in words)
{
    Console.WriteLine($"Слово: {mt.Value}, длина: {mt.Length}, " +
                      $"позиция: {mt.Index}");
}
```

Вывод:

Слово: Conditio, длина: 8, позиция: 0
Слово: непременное, длина: 11, позиция: 24
Слово: условие, длина: 7, позиция: 36

«Жадные» и «Ленивые» Квантификаторы

```
// * – жадный квантификатор:
Console.WriteLine(Regex.Match("сорок сороков", @".*к"));
```

Результат поиска: «сорок сорок»

```
// *? – ленивый квантификатор:
Console.WriteLine(Regex.Match("сорок сороков", @".*?к"));
```

Результат поиска: «сорок»

```
string text = "рациональная <b>дробь</b> - " +
    "это упорядоченная пара <b>целых</b> чисел";
Regex.Match(text, @"<b>.*?</b>");
```

Результат поиска: «дробь»

Для регулярного выражения ,@".*" **результат поиска – длинная строка**: «дробь – это упорядоченная пара целых»

Поиск с Просмотром

```
string URL = @"http:\\www.ecom.ru\tricks\tric01\nature.htm";
Console.WriteLine(Regex.Match(URL, @"\\\\.*?\\"));
// «Неудачный» результат:  \\www.ecom.ru\
```

Метасимволы просмотра:
(?<=*выражение*) – просмотр назад
(?=*выражение*) – просмотр вперед
(?<!*выражение*) – просмотр назад с отрицанием
(?!*выражение*) – просмотр вперед с отрицанием

```
Console.WriteLine(Regex.Match(URL,    @"(?<=\\\\).*?(?=\\)"));
// Результат: www.ecom.ru
```

Группы Регулярного Выражения

Пример анализа телефонного номера:

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

string phone = @"101-421-232-44-57";
string reg = @"(\d\d\d)-(\d\d\d)-(\d\d\d-\d\d-\d\d)";
Match memb = Regex.Match(phone, reg);
Console.WriteLine("Полный номер: " + memb.Groups[0]);
Console.WriteLine("Код страны: " + memb.Groups[1]);
Console.WriteLine("Код города: " + memb.Groups[2]);
Console.WriteLine("Номер телефона: " + memb.Groups[3]);
```

Вывод:
Полный номер: 101-421-232-44-57
Код страны: 101
Код города: 421
Номер телефона: 232-44-57

Именованные Группы

Именование:
(?<*имя_группы*>....)
(?'*имя_группы*'....)

Обращение:
\k<*имя_группы*>
\k'*имя_группы*'

Пример с Обратной Ссылкой

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

string sentence = @"Penny saved is a Penny got.";
string pattern = @"(?'first'\w+)(.*)\k'first'";
Match result = Regex.Match(sentence, pattern);
Console.WriteLine(result.Length);
Console.WriteLine(result.Groups[0]);
Console.WriteLine(result.Groups[1]);
Console.WriteLine(result.Groups["first"]);
```

Вывод:

22
Penny saved is a Penny
 saved is a
Penny

Выделить HTML-Элемент

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

string regFind = @"<(?'tag'\w+).*?>(?'text'.*?)</\k'tag'>";
string str = @"<script
language=""JavaScript"">Content!!!</script>";

Match m = Regex.Match(str, regFind);
Console.WriteLine(m.Groups[0]);
Console.WriteLine(m.Groups["tag"]);
Console.WriteLine(m.Groups["text"]);
```

Вывод:

<script language="JavaScript">Content!!!</script>
script
Content!!!

Замены в Тексте

Прототип (заголовок) метода:

```
public static string Replace(string input, string pattern, string replacement)
```

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

string find = @"h3>";
string change = @"h5>";
Console.WriteLine(Regex.Replace(
     "Начало <h3>Оглавление</h3> Конец", find, change));
```

Вывод:
Начало <h5>Оглавление</h5> Конец

Обращения к Группам в Шаблоне Подстановки

$i где i=0,1,2,… или **${*имя_группы*}**

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

string line = "кот кит кат кто ком";
Console.WriteLine(Regex.Replace(line, @"к[ои]т", @"'$0'"));
```

Вывод:
'кот' 'кит' кат кто ком

Делегат-параметр Метода Replace

Прототип (заголовок) метода:

```
public static string Replace(string input, string pattern,
                             MatchEvaluator evaluator)
```

Объявление (сигнатура) делегата:

```
public delegate string MatchEvaluator( Match match )
```

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

Console.WriteLine(Regex.Replace("3 1 0 6 2",
@"\d",
    r => (r.Value == "0" ? "нуль" :
          r.Value == "1" ? "один" : "NaN")));
```

Вывод:
NaN один нуль NaN NaN

Деление Текста

Прототип метода:

```
public static string[] Split(string input, string pattern)
```

Выделить слова предложения:

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

string line = "Veni,  vidi,  vici.";
Console.WriteLine(line);
foreach (string s in Regex.Split(line, @"[,.]\s*"))
{
    Console.Write(s + " ");
}
```

Вывод:

Veni vidi vici