

Лабораторная работа №4

Регулярные выражения

Цель работы: изучить работу с регулярными выражениями на языке C#.

Теоретическая часть

Язык регулярных выражений предназначен специально для обработки строк. Он включает два средства:

- набор управляющих кодов для идентификации специфических типов символов;
- система для группирования частей подстрок и промежуточных результатов таких действий.

Регулярные выражения поддерживаются множеством классов .NET из пространства имен `System.Text.RegularExpressions`. Они нужны для решения следующих задач:

- поиск определенных шаблонов символов;
- проверка текста на соответствие predetermined шаблону (например, адресу электронной почты);
- извлечение, изменение, замена или удаление текстовых подстрок;
- добавление извлеченных строк в коллекцию для создания отчета.

При работе с регулярными выражениями можно использовать специальные онлайн-сервисы, в которых можно проверить правильность разработанного шаблона:

- <https://regex101.com/>
- <https://regexr.com/>

В таблице приведены основные метасимволы, используемые для построения регулярных выражений.

Мета символ	Значение		
[...]	Любой из символов, указанных в скобках		
[^...]	Любой из символов, не указанных в скобках		
.	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]		
{n,m}	Соответствует предшествующему шаблону, повторенному не менее n и не более m раз		
{n,}	Соответствует предшествующему шаблону, повторенному n или более раз		
{n}	Соответствует в точности n экземплярам предшествующего шаблона		
?	Соответствует нулю или одному экземпляру предшествующего шаблона; предшествующий шаблон является необязательным		
+	Соответствует одному или более экземплярам предшествующего шаблона		
*	Соответствует нулю или более экземплярам предшествующего шаблона		
^	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"
	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.		

В C# для работы с регулярными выражениями используется класс **Regex**.

```

public static string Escape(string str);
public static bool IsMatch(string input, string pattern);
public static bool IsMatch(string input, string pattern, RegexOptions options);
public static bool IsMatch(string input, string pattern, RegexOptions options, TimeSpan matchTimeout);
public static Match Match(string input, string pattern);
public static Match Match(string input, string pattern, RegexOptions options);
public static Match Match(string input, string pattern, RegexOptions options, TimeSpan matchTimeout);
public static MatchCollection Matches(string input, string pattern, RegexOptions options, TimeSpan matchTimeout);
public static MatchCollection Matches(string input, string pattern, RegexOptions options);
public static string Replace(string input, string pattern, string replacement, RegexOptions options, TimeSpan matchTimeout);
public static string Replace(string input, string pattern, MatchEvaluator evaluator, RegexOptions options, TimeSpan matchTimeout);
public static string Replace(string input, string pattern, string replacement, RegexOptions options);
public static string Replace(string input, string pattern, MatchEvaluator evaluator, RegexOptions options);
public static string Replace(string input, string pattern, string replacement);
public static string Replace(string input, string pattern, MatchEvaluator evaluator);
public static string[] Split(string input, string pattern, RegexOptions options, TimeSpan matchTimeout);
public static string[] Split(string input, string pattern, RegexOptions options);
public static string[] Split(string input, string pattern);
public static string Unescape(string str);
protected internal static void ValidateMatchTimeout(TimeSpan matchTimeout);
public string[] GetGroupNames();
public int[] GetGroupNumbers();
public string GroupNameFromNumber(int i);
public int GroupNumberFromName(string name);
public bool IsMatch(string input);

```

Рисунок 1 – Класс Regex


Основные методы:

- **IsMatch** – проверка строки на соответствие шаблону, возвращает true или false;
- **Match** – возвращает совпадающее значение некоторой строки шаблону;
- **Matches** – возвращает коллекцию совпадений;
- **Replace** – заменяет подстроку в строке, если она соответствует шаблону;
- **Split** – разбивает строку на массив строк, в качестве разделителя используется шаблон.

Рассмотрим пример. Необходимо из текста получить номера студенческих групп, которые начинаются с цифры 9. Обратите внимание: в примерах ниже используются методы экземпляра класса. Можно использовать и статические методы, изображенные на рисунке 1.

```
// Создаем шаблон для поиска групп с номером, начинающимся с 9
string pattern = @"^b[9]\d+";
// Создаем экземпляр Regex
Regex rg = new Regex(pattern);
// Передаем исходную строку
string groups = "В конкурсе приняли участие студенты групп 940, 040, 945, 946 и 0714";
// Получаем все совпадения
MatchCollection matchedGroups = rg.Matches(groups);
/// Выводим все номера групп
for (int count = 0; count < matchedGroups.Count; count++)
    Console.WriteLine(matchedGroups[count].Value);
```

Рисунок 2 – Выбор групп по шаблону



```
940
945
946
```

Рисунок 3 – Результат

Также при проверке строки на соответствие шаблону можно использовать дополнительные опции `RegexOptions`. В следующем примере сравниваются результаты поиска в соответствии с регистром.

```
string[] source = {
    "Алясов и Дербуков пошли в буфет", "алясов съел сосиску в тесте", "дЕрБуКов допил чай аляСоВа"};
Regex regex = new Regex("Дербуков");
Console.WriteLine("Регистрозависимый поиск: ");
foreach (string str in source)
{
    if (regex.IsMatch(str))
        Console.WriteLine($"В исходной строке: \"{str}\" есть совпадения!");
}
Console.WriteLine();

regex = new Regex("алясов", RegexOptions.IgnoreCase);
Console.WriteLine("Регистронезависимый поиск: ");
foreach (string str in source)
{
    if (regex.IsMatch(str))
        Console.WriteLine($"В исходной строке: \"{str}\" есть совпадения!");
}
```

Рисунок 4 – Регистрозависимый и регистронезависимый поиск

```
Регистрозависимый поиск:  
В исходной строке: "Алясов и Дербуков пошли в буфет" есть совпадения!  
  
Регистронезависимый поиск:  
В исходной строке: "Алясов и Дербуков пошли в буфет" есть совпадения!  
В исходной строке: "алясов съел сосиску в тесте" есть совпадения!  
В исходной строке: "дЕрБуКоВ допил чай аЛяСоВа" есть совпадения!
```

Рисунок 5 – Результат

Практическая часть

- 1) Написать метод, который определяет количество входящих в заданную строку почтовых индексов (почтовый индекс состоит из 6 цифр).
- 2) Дана строка — предложение на русском языке. Поменять местами первую и последнюю буквы каждого слова.
- 3) Дана строка, содержащая помимо прочей информации номера телефонов в федеральном формате. Скрыть все цифры городской части номера кроме самой первой под символами **x**. Например, если в тексте имелся номер +7 (863) 297-51-11, то после преобразования он должен выглядеть как +7 (863) **2xx-xx-xx**. Считать, что код города может содержать от трёх до пяти цифр, а городской номер — от 7 до 5 цифр соответственно.
- 4) Выяснить, какими могут быть российские автомобильные номера (с кодом региона), составить соответствующее регулярное выражение и написать метод, который находит в строке все автомобильные номера и возвращает их в виде последовательности.
- 5) Дана строка. Сохранить в новую строку все содержащиеся в ней IPv4-адреса в десятичной записи с точками через разделитель.

Содержание отчета

1. Титульный лист
2. Цель работы
3. Задание

4. Код программы
5. Результат выполнения