Лекция №.
Работа с регулярными выражениями

Основные понятия

Регулярные выражения представляют эффективный и гибкий метод по обработке больших текстов, позволяя в то же время существенно уменьшить объемы кода по сравнению с использованием стандартных операций со строками.

Основная функциональность регулярных выражений в .NET сосредоточена в пространстве имен **System.Text.RegularExpressions**. А центральным классом при работе с регулярными выражениями является класс **Regex**.

Регулярные выражения

Регуля́рные выраже́ния (англ. regular expressions) — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters).

Основные понятия

Например, у нас есть некоторый текст и нам надо найти в нем все словоформы какого-нибудь слова. С классом Regex это сделать очень просто:

```
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");
}
```

Основные понятия

Здесь мы находим в искомой строке все словоформы слова "туп". В конструктор объекта Regex передается регулярное выражение для поиска. Далее мы разберем некоторые элементы синтаксиса регулярных выражений, а пока достаточно знать, что выражение туп(\w*) обозначает, найти все слова, которые имеют корень "туп" и после которого может стоять различное количество символов. Выражение \w означает алфавитно-цифровой символ, а звездочка после выражения указывает на неопределенное их количество - их может быть один, два, три или вообще не быть.

Метод Matches класса Regex принимает строку, к которой надо применить регулярные выражения, и возвращает коллекцию найденных совпадений.

Каждый элемент такой коллекции представляет объект **Match**. Его свойство Value возвращает найденное совпадение.

Параметр RegexOptions

Класс Regex имеет ряд конструкторов, позволяющих выполнить начальную инициализацию объекта. Две версии конструкторов в качестве одного из параметров принимают перечисление RegexOptions. Некоторые из значений, принимаемых данным перечислением:

Compiled: при установке этого значения регулярное выражение компилируется в сборку, что обеспечивает более быстрое выполнение

CultureInvariant: при установке этого значения будут игнорироваться региональные различия

IgnoreCase: при установке этого значения будет игнорироваться регистр

IgnorePatternWhitespace: удаляет из строки пробелы и разрешает комментарии, начинающиеся со знака #

Параметр RegexOptions

Multiline: указывает, что текст надо рассматривать в многострочном режиме. При таком режиме символы "^" и "$" совпадают, соответственно, с началом и концом любой строки, а не с началом и концом всего текста

RightToLeft: приписывает читать строку справа налево

Singleline: устанавливает однострочный режим, а весь текст рассматривается как одна строка

Regex regex = new Regex(@"туп(\w*)", RegexOptions.IgnoreCase);

При необходимости можно установить несколько параметров:

Regex regex = new Regex(@"туп(\w*)", RegexOptions.Compiled | RegexOptions.IgnoreCase);

Параметр RegexOptions

Multiline: указывает, что текст надо рассматривать в многострочном режиме. При таком режиме символы "^" и "$" совпадают, соответственно, с началом и концом любой строки, а не с началом и концом всего текста

RightToLeft: приписывает читать строку справа налево

Singleline: устанавливает однострочный режим, а весь текст рассматривается как одна строка

Regex regex = new Regex(@"туп(\w*)", RegexOptions.IgnoreCase);

При необходимости можно установить несколько параметров:

Regex regex = new Regex(@"туп(\w*)", RegexOptions.Compiled | RegexOptions.IgnoreCase);

Синтаксис регулярных выражений

Рассмотрим вкратце некоторые элементы синтаксиса регулярных выражений:

^: соответствие должно начинаться в начале строки (например, выражение @"^пр\w*" соответствует слову "привет" в строке "привет мир")

**$**: конец строки (например, выражение @"\w*ир$" соответствует слову "мир" в строке "привет мир", так как часть "ир" находится в самом конце)

.: знак точки определяет любой одиночный символ (например, выражение "м.р" соответствует слову "мир" или "мор")

***:** предыдущий символ повторяется 0 и более раз

+: предыдущий символ повторяется 1 и более раз

?: предыдущий символ повторяется 0 или 1 раз

\s: соответствует любому пробельному символу

Синтаксис регулярных выражений

\S: соответствует любому символу, не являющемуся пробелом

\w: соответствует любому алфавитно-цифровому символу

\W: соответствует любому не алфавитно-цифровому символу

\d: соответствует любой десятичной цифре

\D : соответствует любому символу, не являющемуся десятичной цифрой

Регулярные выражения

Спецсимволы

() подмаска, вложенное выражение;
[] групповой символ;
{a,b} количество вхождений от «a» до «b»
| логическое «или», в случае с односимвольными альтернативами используйте [];
\ экранирование спец символа;

Регулярные выражения

Позиция внутри строки

^	^a	**a**aa aaa	начало строки
$	a$	aaa aa**a**	конец строки
\A	\Aa	**a**aa aaa	
		aaa aaa	начало текста
\z	a\z	aaa aaa	
		aaa aa**a**	конец текста

Позиция внутри строки

Якоря

Якоря в регулярных выражениях указывают на начало или конец чего-либо.

Например, строки или слова. Они представлены определенными символами.

К примеру, шаблон, соответствующий строке, начинающейся с цифры, должен иметь следующий вид:

^[0-9]+

Синтаксис регулярных выражений

Теперь посмотрим на некоторые примеры использования. Возьмем первый пример с скороговоркой "Бык тупогуб, тупогубенький бычок, у быка губа бела была тупа" и найдем в ней все слова, где встречается корень "губ":

string s = "Бык тупогуб, тупогубенький бычок, у быка губа бела была тупа";

*Regex regex = new Regex(@"\w*губ\w*");*

Так как выражение \w* соответствует любой последовательности алфавитно-цифровых символов любой длины, то данное выражение найдет все слова, содержащие корень "губ".

Синтаксис регулярных выражений

Второй простенький пример - нахождение телефонного номера в формате 111-111-1111:

```
string s = "456-435-2318";
Regex regex = new Regex(@"\d{3}-\d{3}-\d{4}");
```

Если мы точно знаем, сколько определенных символов должно быть, то мы можем явным образом указать их количество в фигурных скобках: \d{3} - то есть в данном случае три цифры.

Мы можем не только задать поиск по определенным типам символов - пробелы, цифры, но и задать конкретные символы, которые должны входить в регулярное выражение. Например, перепишем пример с номером телефона и явно укажем, какие символы там должны быть:

```
string s = "456-435-2318";
Regex regex = new Regex("[0-9]{3}-[0-9]{3}-[0-9]{4}");
```

Синтаксис регулярных выражений

В квадратных скобках задается диапазон символов, которые должны в данном месте встречаться. В итоге данный и предыдущий шаблоны телефонного номера будут эквивалентны.

Также можно задать диапазон для алфавитных символов: *Regex regex = new Regex("[a-v]{5}");* - данное выражение будет соответствовать любому сочетанию пяти символов, в котором все символы находятся в диапазоне от a до v.

Можно также указать отдельные значения: *Regex regex = new Regex(@"[2]*-[0-9]{3}-\d{4}");.* Это выражение будет соответствовать, например, такому номеру телефона "222-222-2222" (так как первые числа двойки)

С помощью операции | можно задать альтернативные символы: *Regex regex = new Regex(@"[2\|3]{3}-[0-9]{3}-\d{4}");.* То есть первые три цифры могут содержать только двойки или тройки. Такой шаблон будет соответствовать, например, строкам "222-222-2222" и "323-435-2318". А вот строка "235-435-2318" уже не подпадает под шаблон, так как одной из трех первых цифр является цифра 5.

Синтаксис регулярных выражений

Итак, у нас такие символы, как *, + и ряд других используются в качестве специальных символов. И возникает вопрос, а что делать, если нам надо найти, строки, где содержится точка, звездочка или какой-то другой специальный символ? В этом случае нам надо просто экранировать эти символы слешем:

```
Regex regex = new Regex(@"[2\|3]{3}\.[0-9]{3}\.\d{4}");
// этому выражению будет соответствовать строка "222.222.2222"
```

Синтаксис регулярных выражений

Проверка на соответствие строки формату

Нередко возникает задача проверить корректность данных, введенных пользователем. Это может быть проверка электронного адреса, номера телефона, Класс Regex предоставляет статический метод **IsMatch**, который позволяет проверить входную строку с шаблоном на соответствие:

```
string pattern = @"^(?("")(""[^""]+?""@)|(([0-9a-z]((\.(?!\.))|[-!#\$%&'\*\+/=\?\^`\{\}\|~\w])*)(?<=[0-9a-z])@))" +
        @"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-z][-\w]*[0-9a-z]*\.)+[a-z0-9]{2,17}))$";
while (true)
{
    Console.WriteLine("Введите адрес электронной почты");
    string email = Console.ReadLine();
```

Синтаксис регулярных выражений

Проверка на соответствие строки формату

```
...
    if (Regex.IsMatch(email, pattern, RegexOptions.IgnoreCase))
    {
        Console.WriteLine("Email подтвержден");
        break;
    }
    else
    {
        Console.WriteLine("Некорректный email");
    }
}
```

Синтаксис регулярных выражений

Проверка на соответствие строки формату

Переменная pattern задает регулярное выражение для проверки адреса электронной почты. Данное выражение предлагает нам Microsoft на страницах msdn.

Для проверки соответствия строки шаблону используется метод IsMatch: Regex.IsMatch(email, pattern, RegexOptions.IgnoreCase). Последний параметр указывает, что регистр можно игнорировать. И если введенная строка соответствует шаблону, то метод возвращает true.

Синтаксис регулярных выражений

Замена и метод Replace

Класс Regex имеет метод Replace, который позволяет заменить строку, соответствующую регулярному выражению, другой строкой:

```
string s = "Мама  мыла  раму. ";
string pattern = @"\s+";
string target = " ";
Regex regex = new Regex(pattern);
string result = regex.Replace(s, target);
```

Данная версия метода Replace принимает два параметра: строку с текстом, где надо выполнить замену, и сама строка замены. Так как в качестве шаблона выбрано выражение "\s+ (то есть наличие одного и более пробелов), метод Replace проходит по всему тексту и заменяет несколько подряд идущих пробелов ординарными.

Синтаксис регулярных выражений

Замена и метод Replace

Класс Regex имеет метод Replace, который позволяет заменить строку, соответствующую регулярному выражению, другой строкой:

```
string s = "Мама  мыла  раму. ";
string pattern = @"\s+";
string target = " ";
Regex regex = new Regex(pattern);
string result = regex.Replace(s, target);
```

Данная версия метода Replace принимает два параметра: строку с текстом, где надо выполнить замену, и сама строка замены. Так как в качестве шаблона выбрано выражение "\s+ (то есть наличие одного и более пробелов), метод Replace проходит по всему тексту и заменяет несколько подряд идущих пробелов ординарными.

Регулярные выражения

Конкретное задание из автоинспекции

Вашей задачей является разработка двух библиотек, которые позволят Вам использовать их в следующих сессиях при выполнении конкурсного задания. При создании библиотек Вы должны использовать только те именования, которые указаны в задании. Если названия или типы данных будут отличны от указанных, Ваша работа не сможет быть проверенной.
Обращаем Ваше внимание, проверка библиотек автоматическая!
Во время выполнения конкурсного задания вам необходимо использовать предоставленные ресурсы.
Библиотеки не должны работать с базой данных. Все данные должны храниться только в рамках библиотеки.

Регулярные выражения

Конкретное задание из автоинспекции

Теперь мы должны исключить символы знаки запрещенные буквы
В VIN разрешено использовать только следующие буквы латинского алфавита и арабские цифры:
0 1 2 3 4 5 6 7 8 9 A B C D E F G H J K L M N P R S T U V W X Y Z
Использовать буквы I, O, Q запрещено, так как I, O, Q сходны по начертанию с цифрами 1, 0, а также между собой
VIN состоит из 3 частей:
1. WMI (World Manufacturers Identification) — всемирный индекс изготовителя
2. VDS (Vehicle Description Section) — описательная часть
3. VIS (Vehicle Identification Section) — отличительная часть

Регулярные выражения

Конкретное задание из автоинспекции

Рассмотрим первый вариант, когда мы используем только стандартные условия и циклы:

```
If(vin[0] = “1,2…”)
{
If(vin[1]=“!/…”)
// И так куча условий
}
```

Конкретное задание из автоинспекции

А теперь такую же библиотеку опишем, но уже с использованием регулярных переменных:

```
public static bool CheckVIN(string vin)
    {
        bool result = false;

        //Регулярное выражение ВИН
        Regex regex =new Regex(@"[A-HJ-NPR-Z0-9]{13}[0-9]{4}");
       MatchCollection matches = regex.Matches(vin);

        //Номера состоящие из 17 чисел и цифр
        if (vin.Length == 17 && matches.Count>0)
        {
            result = true;
        }
        return result;
    }
```