Строки и регулярные выражения

Строки в C#

string

StringBuilder

символы (тип char) - самостоятельно

Строки типа string

Тип string предназначен для работы со строками символов в кодировке Unicode. Ему соответствует базовый класс System.String библиотеки .NET.

Создание строки:

1. `string s;` // инициализация отложена
2. `string t = "qqq";` // инициализация строковым литералом
3. `string u = new string(' ', 20);` // с пом. конструктора
4. `string v = new string( a );` // создание из массива символов

// создание массива символов: `char[] a = { '0', '0', '0' };`

Операции для строк

- присваивание (=);
- проверка на равенство (==);
- проверка на неравенство (!=);
- обращение по индексу ([]);
- сцепление (конкатенация) строк (+).

- Строки равны, если имеют одинаковое количество символов и совпадают посимвольно.
- Обращаться к отдельному элементу строки по индексу можно только для получения значения, но не для его изменения. Это связано с тем, что строки типа string относятся к неизменяемым типам данных.
- Методы, изменяющие содержимое строки, на самом деле создают новую копию строки. Неиспользуемые «старые» копии автоматически удаляются сборщиком мусора.

Некоторые элементы класса System.String

Название	Описание
Compare	Сравнение двух строк в алфавитном порядке. Разные реализации метода позволяют сравнивать строки и подстроки с учетом и без учета регистра и особенностей национального представления дат и т. д.
CompareOrdinal	Сравнение двух строк по кодам символов. Разные реализации метода позволяют сравнивать строки и подстроки
CompareTo	Сравнение текущего экземпляра строки с другой строкой
Concat	Конкатенация строк. Метод допускает сцепление произвольного числа строк
Copy	Создание копии строки

Format	Форматирование в соответствии с заданными спецификаторами формата
IndexOf, LastIndexOf, …	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора
Insert	Вставка подстроки в заданную позицию
Join	Слияние массива строк в единую строку. Между элементами массива вставляются разделители (см. далее)
Length	Длина строки (количество символов)
Remove	Удаление подстроки из заданной позиции
Replace	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
Split	Разделение строки на элементы, используя заданные разделители. Результаты помещаются в массив строк
Substring	Выделение подстроки, начиная с заданной позиции

Пример

```
string s = "пре красная королева";
        Console.WriteLine( s );
        string sub = s.Substring( 3 ).Remove( 12, 2 );        // 1
        Console.WriteLine( sub );

        string[] mas = s.Split(' ');                          // 2
        string joined = string.Join( "! ", mas );
        Console.WriteLine( joined );
```

прекрасная королева
красная корова
прекрасная! королева

Пример: разбиение текста на слова

```
StreamReader inputFile = new StreamReader("example.txt");
string text = inputFile.ReadToEnd();

char[] delims = ".,;:!?\n\xD\xA\" ".ToCharArray();
string[] words = text.Split(delims,
                            StringSplitOptions.RemoveEmptyEntries);
foreach (string word in words) Console.WriteLine(word);

Console.WriteLine("Слов в тексте: " + words.Length);

// слова, оканчивающиеся на «a»:
foreach (string word in words)
    if (word[word.Length-1] == 'a') Console.WriteLine(word);
```

Пример форматирования строк

```
double a = 12.234;
int b = 29;
Console.WriteLine( " a = {0,6:C}  b = {1,2:X}", a, b );
Console.WriteLine( " a = {0,6:0.##}  b = {1,5:0.# ' руб. '}", a, b);
Console.WriteLine(" a = {0:F3}  b = {1:D3}", a, b);
Console.WriteLine( " a = " + a.ToString("C"));
```

a = 12,23р. b = 1D

a = 12,23 b = 29 руб.

a = 12,234 b = 029

a = 12,23р.

{n[,m][:спецификатор_формата[число]]}

Спецификаторы формата для строк

C или c	Вывод значений в денежном (currency) формате.
D или d	Вывод целых значений.
E или e	Вывод значений в экспоненциальном формате, то есть в виде d.ddd…E+ddd или d.ddd…e+ddd.
F или f	Вывод значений с фиксированной точностью.
G или g	Формат общего вида. Вывод значений с фиксированной точностью или в экспоненциальном формате, в зависимости от того, какой формат требует меньшего количества позиций.
N или n	Вывод значений в формате d,ddd,ddd.ddd. После спецификации можно задать целое число, определяющее длину дробной части
P или p	Вывод числа в процентном формате
R или r	Отмена округления числа при преобразовании в строку. Гарантирует, что при обратном преобразовании в значение того же типа получится то же самое
X или x	Вывод значений в шестнадцатеричном формате.

Примеры пользовательских шаблонов

Число	Шаблон	Представление числа
1,243	00.00	01,24
1,243	#.##	1,24
0,1	00.00	00,10
0,1	#.##	,1

Пустые строки и строки null

- **Пустая строка** — экземпляр объекта System.String, содержащий 0 символов:

```
string s = "";
```

Для пустых строк можно вызывать методы.

- Строки со значениями **null**, напротив, не ссылаются на экземпляр объекта System.String, попытка вызвать метод для строки **null** вызовет исключение NullReferenceException. Однако строки **null** можно использовать в операциях объединения и сравнения с другими строками.

Строки типа StringBuilder

Работа с копиями копий строк
может в конце концов надоесть.
Э. Троелсен

Класс StringBuilder определен в пространстве имен System.Text. Позволяет изменять значение своих экземпляров.

При создании экземпляра обязательно использовать операцию new и конструктор, например:

- StringBuilder a = new StringBuilder(); // 1
- StringBuilder b = new StringBuilder("qwerty"); // 2
- StringBuilder c = new StringBuilder(100); // 3
- StringBuilder d = new StringBuilder("qwerty", 100); // 4
- StringBuilder e = new StringBuilder("qwerty", 1, 3, 100);// 5

Конкатенация 50000 string ~ 1 мин., StringBuilder ~ 1 сек.

Основные элементы класса System.Text.StringBuilder

Append	Добавление в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки типа `string`
AppendFormat	Добавление форматированной строки в конец строки
Capacity	Получение или установка емкости буфера. Если устанавливаемое значение меньше текущей длины строки или больше максимального, генерируется исключение `ArgumentOutOfRangeException`
Insert	Вставка подстроки в заданную позицию
Length	Длина строки (количество символов)
MaxCapacity	Максимальный размер буфера
Remove	Удаление подстроки из заданной позиции
Replace	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
ToString	Преобразование в строку типа `string`

Пример использования StringBuilder

```
Console.Write( "Введите зарплату: " );
double   salary = double.Parse( Console.ReadLine() );
StringBuilder a = new StringBuilder();
a.Append( "зарплата " );
a.AppendFormat( "{0, 6:C} - в год {1, 6:C}",
                        salary, salary * 12 );
Console.WriteLine( a );
a.Replace( "р.", "тыс.$" );
Console.WriteLine( "А лучше было бы: " + a );
```

```
Введите зарплату: 3500
зарплата 3 500,00р. - в год 42 000,00р.
А лучше было бы: зарплата 3 500,00тыс.$ - в год 42 000,00тыс.$
```

Регулярные выражения

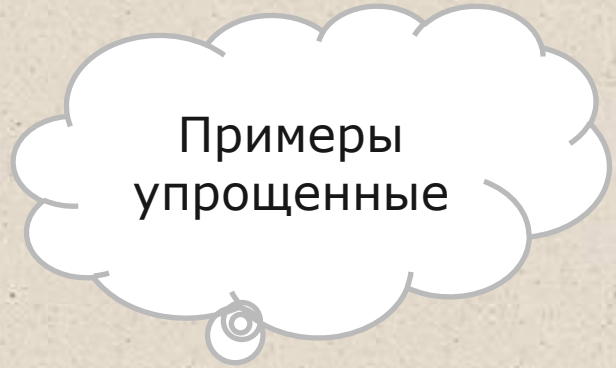
Регулярное выражение — шаблон (образец), по которому выполняется поиск соответствующего ему фрагмента текста.

- тег html:

```
<[^>]+>
```

- российский номер автомобиля:

```
[A-Z]\d{3}[A-Z]{2}\d\dRUS
```



- IP-адрес:

```
\d\d?\d?\.\d\d?\d?\.\d\d?\d?\.\d\d?\d?
```

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.

Язык описания регулярных выражений

Язык описания регулярных выражений состоит из символов двух видов: обычных и метасимволов.

- **Обычный символ** представляет в выражении сам себя.
- **Метасимвол**:
  - **класс символов** (например, любая цифра **\d** или буква **\w**)
  - **уточняющий символ** (например, **^**).
  - **повторитель** (например, **+**).

Примеры:

- выражение для поиска в тексте фрагмента «Вася» записывается с помощью четырех обычных символов «**Вася**»
- выражение для поиска двух цифр, идущих подряд, состоит из двух метасимволов «**\d\d**»
- выражение для поиска фрагментов вида «Вариант 1», «Вариант 2», …, «Вариант 9» имеет вид «**Вариант \d**»
- выражение для поиска фрагментов вида «Вариант 1», «Вариант 23», «Вариант 719», …, имеет вид «**Вариант \d+**»

Метасимволы - классы символов

Класс символов	Описание	Пример
`.`	любой символ, кроме `\n`	**`c.t`** соответствует фрагментам `cat`, `cut`, `c1t`, `c{t` и т.д.
`[ ]`	любой одиночный символ из последовательности внутри скобок.	**`c[au1]t`** соответствует фрагментам `cat`, `cut` и `c1t`. **`c[a-z]t`** соответствует фрагментам `cat`, `cbt`, `cct`, `cdt`, …, `czt`
`[^]`	любой одиночный символ, не входящий в последовательность внутри скобок.	**`c[^au1]t`** соответствует фрагментам `cbt`, `c2t`, `cXt` и т.д. **`c[^a-zA-Z]t`** соответствует 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	**`c\wt`** соответствует фрагментам `cat`, `cut`, `c1t`, `cЮt` и т.д. Не соответствует `c{t`, `c;t` и т.д.

продолжение таблицы

`\W`	любой **не** алфавитно-цифровой символ, то есть символ, не входящий в множество прописных и строчных букв и десятичных цифр	`c\Wt` соответствует фрагментам `c{t`, `c;t`, `c t` и т.д. Не соответствует `cat`, `cut`, `c1t`, `cЮt` и т.д.
`\s`	любой пробельный символ, например, пробел, табуляция (`\t`, `\v`), перевод строки (`\n`, `\r`), новая страница (`\f`)	`\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	`\s\S\S\s` соответствует любым двум непробельным символам, окруженным пробельными.
`\d`	любая десятичная цифра	`c\dt` соответствует фрагментам `c1t`, `c2t`, …, `c9t`
`\D`	любой символ, не явдяющийся десятичной цифрой	`c\Dt` не соответствует фрагментам `c1t`, `c2t`, …, `c9t`.

Уточняющие (якорные) метасимволы

Мета-символ	фрагмент, совпадающий с регулярным выражением:
^	следует искать только в начале строки
$	следует искать только в конце строки
\A	следует искать только в начале многострочной строки
\Z	следует искать только в конце многострочной строки
\b	начинается или заканчивается на границе слова (т.е. между символами, соответствующими \w и \W)
\B	не должен встречаться на границе слова

^cat$

^$

^cat

cat$

Повторители

Мета-символ	Описание	Пример
*	0 или более повторений предыдущего элемента	**ca*t** соответствует фрагментам ct, cat, caat, caaaaaaaaaaat и т.д.
+	1 или более повторений предыдущего элемента	**ca+t** соответствует фрагментам cat, caat, caaaaaaaaaaat и т.д.
?	0 или 1 повторений предыдущего элемента	**ca?t** соответствует фрагментам ct и cat
{n}	ровно n повторений предыдущего элемента	**ca{3}t** соответствует фрагменту caaat. (cat){2} соответствует фрагменту catcat.
{n,}	по крайней мере n повторений предыдущего элемента	**ca{3,}t** соответствует фрагментам caaat, caaaat, caaaaaaaaaaat и т.д.
{n,m}	от n до m повторений предыдущего элемента	**ca{2,4}t** соответствует фрагментам caat, caaat и caaaat

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`

- ip-адрес (упрощенно):

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

Поддержка регулярных выражений в .NET

- Для поддержки регулярных выражений в библиотеку .NET включены классы, объединенные в пространство имен System.Text.RegularExpressions.
- Основной класс – **Regex**. Он реализует подсистему обработки регулярных выражений.
- Подсистеме требуется предоставить:
  - **Шаблон** (регулярное выражение), соответствия которому требуется найти в тексте.
  - **Текст**, который требуется проанализировать с помощью шаблона.

См.:

http://msdn.microsoft.com/ru-ru/library/hs600312.aspx?ppud=4

Использование класса Regex

Обработчик регулярных выражений выполняет синтаксический **разбор** и **компиляцию** регулярного выражения, а также операции, **сопоставляющие** шаблон регулярного выражения с входной строкой.

Обработчик можно использовать одним из двух способов:

- С помощью вызова статических методов класса Regex. Параметры метода содержат входную строку и шаблон регулярного выражения.
- С помощью создания объекта Regex посредством передачи регулярного выражения в конструктор класса.

Методы класса Regex

позволяют выполнять следующие действия:

- Определить, **встречается ли** во входном тексте шаблон регулярного выражения (метод IsMatch).
- **Извлечь** из текста одно или все вхождения, соответствующие шаблону регулярного выражения (методы Match или Matches).
- **Заменить** текст, соответствующий шаблону регулярного выражения (метод Replace).
- **Разделить** строку на массив строк (метод Split).

Пример использования Regex.IsMatch

```
using System;
using System.Text.RegularExpressions;
public class Example
 { public static void Main()
    { string[] values = { "111-22-3333", "111-2-3333"};
      string pattern = @"^\d{3}-\d{2}-\d{4}$";
      foreach (string value in values)
       { if (Regex.IsMatch(value, pattern))
            Console.WriteLine("{0} is a valid SSN.", value);
         else Console.WriteLine("{0}: Invalid", value);
 } } }
 // Вывод:
 // 111-22-3333 is a valid SSN.
 // 111-2-3333: Invalid
```

Пример использования Regex.Matches

```
// совпадения со строкой "abc" во входной строке
using System;
using System.Text.RegularExpressions;
public class Example
 { public static void Main()
    { string pattern = "abc";
      string input = "abc123abc456abc789";
      foreach (Match match in Regex.Matches(input, pattern))
          Console.WriteLine("{0} found at position {1}.",
                                match.Value, match.Index);
 } }
 //Вывод:
 // abc found at position 0.
 // abc found at position 6.
 // abc found at position 12.
```

Или:

```
using System;
using System.Text.RegularExpressions;
public class Example
 { public static void Main()
    { string pattern = "abc";
      string input = "abc123abc456abc789";
      Match match = Regex.Match(input, pattern);
      while (match.Success)
        { Console.WriteLine("{0} found at position {1}.",
                                    match.Value, match.Index);
          match = match.NextMatch();
 } } }
 // создание экземпляра класса Match
```

Еще пример использования Regex.Matches

```
string pattern = @"\b91*9*\b";
string input = "99 95 919 929 9119 9219 999 9919 91119";
foreach (Match match in Regex.Matches(input, pattern))
      Console.WriteLine("'{0}' found at position {1}.",
                              match.Value, match.Index);

 // Вывод:
 // '99' found at position 0.
 // '919' found at position 6.
 // '9119' found at position 14.
 // '999' found at position 24.
 // '91119' found at position 33.
```

Пример использования Regex.Replace

```
// Добавить $ перед десятичной цифрой:
using System;
using System.Text.RegularExpressions;
public class Example
 { public static void Main()
    { string pattern = @"\b\d+\.\d{2}\b";
      string replacement = "$$$&";
      string input = "Total Cost: 103.64";
      Console.WriteLine(Regex.Replace(input, pattern,
                                      replacement));
 } }
 // Вывод:
 // Total Cost: $103.64
```

$$ - символ доллара ($).
$& - вся сопоставленная подстрока.

Пример использования Regex.Split

```
// помещает элементы нумерованного списка в массив строк:
using System;
using System.Text.RegularExpressions;
public class Example
 { public static void Main()
    { string input = "1. Eggs 2. Bread 3. Milk 4. Coffee 5. Tea";
      string pattern = @"\b\d{1,2}\.\s";
      foreach (string item in Regex.Split(input, pattern))
         { if (! String.IsNullOrEmpty(item))
                Console.WriteLine(item);
         }
    }
 }
```

```
// Вывод:
 // Eggs
 // Bread
 // Milk
 // Coffee
 // Tea
```

Разбиение строки на слова (метод Split)

```
public static void Main()
   {
      string text        = "Салат -  $4, борщ -$3, одеколон - $10.";
      string pattern     = "[- ,.]+";
      Regex  r           = new Regex( pattern );
      string [] words = r.Split(text);
      foreach ( string word in words ) Console.WriteLine( word );
   }
```

Результат:
Салат
$4
Борщ
$3
Одеколон
$10

Пример РВ: удаление символов

Метод CleanInput используется для удаления потенциально опасных символов, введенных в текстовое поле пользователем (не являющихся алфавитно-цифровыми, за исключением **@ - .**.

```
using System;
using System.Text.RegularExpressions;
public class Example
{
   static string CleanInput(string strIn)
      {   return Regex.Replace(strIn, @"[^\w\.@-]", ""); }
}
```

шаблон регулярного выражения **[^\w\.@-\\%]** также разрешает знак процента и обратную косую черту во входной строке.

Группирование

Группирование (с помощью круглых скобок) применяется во многих случаях:

- требуется задать повторитель не для отдельного символа, а для последовательности;
- для запоминания фрагмента, совпавшего с выражением, заключенным в скобки, в некоторой переменной. Имя переменной задается в угловых скобках или апострофах:

  (**?<имя_переменной>**фрагмент_выражения)

  например: номера телефонов в виде **nnn-nn-nn** запоминаются в переменной num:

  (**?<num>**\d\d\d-\d\d-\d\d)
- для формирования обратных ссылок:

  (\w)\1 – поиск сдвоенных символов

  (?<z>\w+) (\k<z>) – поиск повторяющихся слов, разделенных пробелом

Поиск повторяющихся слов в строке

```
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 s1 = "Oh, oh! Give me more!";
    if ( r.IsMatch( tst1 ) ) Console.WriteLine( " s1 yes" );
    else                     Console.WriteLine( " s1 no"  );

    string s2 = "Oh give me, give me more!";
    if ( r.IsMatch( tst2 ) ) Console.WriteLine( " s2 yes" );
    else                     Console.WriteLine( " s2 no"  );
  }
}
```

Результат работы программы после замены \s на .

```
tst1 yes
tst2 yes
```

Запоминание найденных фрагментов

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

Пример РВ: поиск href

поиск и печать всех значений href="..." и их позиций во входной строке.

```
private static void DumpHRefs(string inputString)
{ Match m;
   string HRefPattern =
    "href\\s*=\\s*(?:\"(?<1>[^\"]*)\"|(?<1>\\S+))";
   m = Regex.Match(inputString, HRefPattern,
    RegexOptions.IgnoreCase | RegexOptions.Compiled);
   while (m.Success)
   {   Console.WriteLine(
            "Found href " + m.Groups[1] + " at " +
             m.Groups[1].Index);
    m = m.NextMatch();
    }
}
```

вызов метода DumpHRefs

```
public static void Main()
 { string inputString = "My favorite web sites include:</P>" +
   "<A HREF=\"http://msdn2.microsoft.com\">" + "MSDN Home
   Page</A></P>" + "<A
   HREF=\"http://www.microsoft.com\">" + "Microsoft
   Corporation Home Page</A></P>" + "<A
   HREF=\"http://blogs.msdn.com/bclteam\">" + ".NET Base
   Class Library blog</A></P>";
   DumpHRefs(inputString);
}
 // Output:
 // Found href http://msdn2.microsoft.com at 43
 // Found href http://www.microsoft.com at 102
 // Found href http://blogs.msdn.com/bclteam at 176
```

href\s*=\s*(?:"(?<1>[^""]*)"|(?<1>\S+))

href	Совпадение с литеральной строкой "href"
\s*	Сопоставить нулю или нескольким символам пробела.
=	Сопоставить знаку равенства.
\s*	Сопоставить нулю или нескольким символам пробела.
(?:"(?<1>[^""]*)"|(?<1>\S+))	Сопоставить одному из следующих без назначения результата захваченной группе: Одиночные кавычки, за которыми следует ноль или несколько вхождений любого символа, отличающегося от одиночных кавычек, за которыми следуют одиночные кавычки. В этот шаблон включена группа с именем 1. Один или более символов, отличных от пробела. В этот шаблон включена группа с именем 1.
(?<1>[^"]*)	Присвоить ноль или несколько вхождений любого символа, отличного от одиночной кавычки, захваченной группе с именем 1.
"(?<1>\S+)	Присвоить один или более символов, отличных от пробела, захваченной группе с именем 1.

Извлечение протокола и номера порта из URL-адреса

```
using System;
using System.Text.RegularExpressions;
public class Example
{ public static void Main()
  { string url =
    "http://www.contoso.com:8080/letters/readme.html";
    Regex r = new
    Regex(@"^(?<proto>\w+)://[^/]+?(?<port>:\d+)?/");
    Match m = r.Match(url);
    if (m.Success)
    Console.WriteLine(r.Match(url).Result("${proto}${port}"));
 }
}
 // output:
 // http:8080
```

^(?<proto>\w+)://[^/]+?(?<port>:\d+)?/

^	Соответствие должно обнаруживаться в начале строки.
(?<proto>\w+)	Совпадение с одним или несколькими символами слова. Эта группа должна получить имя proto.
://	Соответствует двоеточию, за которым следуют две косые черты.
[^/]+?	Соответствует одному или нескольким вхождениям (но как можно меньшему числу) любого символа, отличного от косой черты.
(?<port>:\d+)?	Соответствует вхождениям в количестве 0 или 1 двоеточия, за которым следует одна или несколько цифр. Эта группа должна получить имя port.
/	Соответствует косой черте.

Пример РВ: допустимый e-mail адрес

Метод IsValidEmail возвращает значение true, если строка содержит допустимый адрес электронной почты, и значение false, если нет.

```
using System;
using System.Text.RegularExpressions;
public class RegexUtilities
{   public static bool IsValidEmail(string strIn)
    {   return Regex.IsMatch(strIn,
   @"^(?(""")(""".+?""@)|(([0-9a-zA-Z]((\.(?!\.))|[-!#\$%&'\*\+/
   =\?\^`\{\}\|~\w])*)(?<=[0-9a-zA-Z])@))" +
   @"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-zA-Z][-\w]*[0-9
   a-zA-Z]\.)+[a-zA-Z]{2,6}))$");
    }
}
```

@"^(?(""")(""".+?""@)|(([0-9a-zA-Z]((\.(?!\.))

Шаблон	Описание
^	Соответствие должно обнаруживаться **в начале строки**.
(?("")	Определение, является ли первый символ кавычкой. (?("") является началом конструкции изменения.
((?("")(".+?""@)	Если первый символ **является кавычкой**, имеется соответствие открывающей кавычки, после которой следует как минимум одно вхождение любого символа с последующей закрывающей кавычкой. Строка должна заканчиваться знаком @.
|(([0-9a-zA-Z]	Если первый символ не является кавычкой, имеется соответствие любой буквы с A до Z или любой цифре от 0 до 9.
(\.(?! \.))	Если следующим символом является точка, имеется соответствие. Если этот символ не является точкой, выполняется поиск вперед к следующему символу и продолжается поиск соответствия. **(?!\.)** является утверждением отрицательного поиска вперед нулевой ширины, предотвращающим отображение двух последовательных точек в локальной части адреса электронной почты.

[-!#\$%&'*\+/=\?\^\`\{\}|~\w]	Если следующий символ не является точкой, имеется соответствие любого символа слова или одного из следующих символов: -!#$%'*+=?^`{}|~.
((\.(?! \.))[-!#\$%'*\+/=\?\^\`\{\}|~\w])*	Соответствие шаблону изменения (точка, после которой следует символ, отличный от точки, или одна цифра) 0 или несколько раз.
@	Соответствие символу @.
(?<=[0-9a-zA-Z])	Продолжение поиска соответствия, если символ, предшествующий символу @, является буквой от A до Z, от a до z или цифрой от 0 до 9. Конструкция (?<=[0-9a-zA-Z]) определяет утверждение положительного поиска вперед нулевой ширины.
(?(\[)	Проверка, является ли символ, следующий после символа @, открывающей круглой скобкой.
(\[(\d{1,3}\.){3}\d{1,3}\])	Если этот символ является открывающей круглой скобкой, имеется соответствие открытой круглой скобки, после которой идет IP-адрес (четыре группы из одной-трех цифр, разделенные точкой) и закрывающая круглая скобка.

|(([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA-Z]{2,6})

Если символ, следующий за @, не является открывающей круглой скобкой, имеется соответствие одного буквенно-цифрового символа со значением A-Z, a-z или 0-9 с последующими 0 или несколькими вхождениями символа слова или дефиса, за которыми следует буквенно-цифровой символ со значением A-Z, a-z или 0-9 с последующей точкой. Этот шаблон можно повторить один или несколько раз, после него должны следовать от двух до шести буквенно-цифровых (a-z, A-Z) символов. Эта часть регулярного выражения предусмотрена для захвата имени домена.

```
public class Application
{ public static void Main()
 { string[] emailAddresses = {
    "david.jones@proseware.com",
    "d.j@server1.proseware.com",
    "jones@ms1.proseware.com",
    "j.@server1.proseware.com",
     "j@proseware.com9",
     "js#internal@proseware.com",
    "j_9@[129.126.118.1]",
     "j..s@proseware.com",
     "js*@proseware.com",
     "js@proseware..com",
     "js@proseware.com9",
     "j.s@server1.proseware.com" };
foreach (string emailAddress in emailAddresses)
  { if (RegexUtilities.IsValidEmail(emailAddress))
    Console.WriteLine("Valid: {0}", emailAddress);
     else Console.WriteLine("Invalid: {0}", emailAddress);
    } } }
```

```
// output:
 // Valid: david.jones@proseware.com
 // Valid: d.j@server1.proseware.com
 // Valid: jones@ms1.proseware.com
 // Invalid: j.@server1.proseware.com
 // Invalid: j@proseware.com9
 // Valid: js#internal@proseware.com
// Valid: j_9@[129.126.118.1]
 // Invalid: j..s@proseware.com
 // Invalid: js*@proseware.com
 // Invalid: js@proseware..com
 // Invalid: js@proseware.com9
 // Valid: j.s@server1.proseware.com
```

Пример PB: замена формата даты

метод Regex.Replace заменяет даты в форме мм/дд/гг на даты в форме дд-мм-гг.

```
static string MDYToDMY(string input)
{ return Regex.Replace(input,
   "\\b(?<month>\\d{1,2})/(?<day>\\d{1,2})/(?<ye
   ar>\\d{2,4})\\b", "${day}-${month}-${year}"); }
```

```
"\\b(?<month>\\d{1,2})/(?<day>\\d{1,2})/(?<year>\\d{2,4})\\b", "${day}-${month}-${year}
```

\b	Совпадение должно начинаться на границе слова.
(?<month>\d{1,2})	Совпадение с одной или двумя десятичными цифрами. Это записанная группа month.
/	Совпадение с косой чертой.
(?<day>\d{1,2})	Совпадение с одной или двумя десятичными цифрами. Это записанная группа day.
/	Совпадение с косой чертой.
(?<year>\d{2,4})	Совпадение двумя–четырьмя десятичными цифрами. Это записанная группа year.
\b	Совпадение должно заканчиваться на границе слова.

Шаблон **${day}-${month}-${year}** задает строку замены, представленную в таблице:

$(day)	Добавляет строку, которая записана группой записи day.
-	Добавляет знак дефиса.
$(month)	Добавляет строку, которая записана группой записи month.
-	Добавляет знак дефиса.
$(year)	Добавляет строку, которая записана группой записи year.

Пример вызова метода MDYToDMY

```
using System;
using System.Globalization;
using System.Text.RegularExpressions;
public class Class1
 { public static void Main()
 { string dateString = DateTime.Today.ToString("d",

    DateTimeFormatInfo.InvariantInfo);
    string resultString = MDYToDMY(dateString);
   Console.WriteLine("Converted {0} to {1}.", dateString,
                                resultString);
 }
// Вывод, если было запущено 8/21/2007:
 // Converted 08/21/2007 to 21-08-2007.
```

Механизм НКА

Регулярное выражение должно:
- находить то, что надо
- не находить то, чего не надо

- Формальной моделью алгоритма распознавания лексем, обозначаемых регулярным выражением, является конечный автомат.
- НКА - недетерминированный конечный автомат (допускает более одного перехода из каждого состояния)
- Общие правила разбора:
  - предпочтение отдается совпадению, ближайшему к началу строки (левее)
  - модификаторы – жадные
- В НКА выполняется возврат к предыдущему сохраненному состоянию по принципу LIFO

Жадные и нежадные повторители

Жадный квантор	Ленивый квантор	Описание
*	*?	Выделить ноль или несколько раз.
+	+?	Выделить один или несколько.
?	??	Выделить ноль или один раз.
{}n	{}?n	Соответствие ровно n раз.
{,}n	{,}?n	Выделяет как минимум n раз.
{n,m}	{n,m}?	Выделяет от n до m раз.

Примеры жадности

say "yes" instead of "yeah", please

".*" => "yes" instead of "yeah"

"[^"]*" => "yes"

in 1991 there were 2 056 items

.*(\d\d) => 56

.*(\d+) => 6

```
//Выделить ноль или несколько раз (ленивое совпадение): *?
string pattern = @"\b\w*?oo\w*?\b";
 string input = "woof root root rob oof woo woe";
 foreach (Match match in Regex.Matches(input, pattern,
    RegexOptions.IgnoreCase))
      Console.WriteLine("'{0}' found at position {1}.",
                                match.Value, match.Index);
 //Вывод:
 // 'woof' found at position 0.
 // 'root' found at position 5.
 // 'root' found at position 10.
 // 'oof' found at position 19.
 // 'woo' found at position 23.
```

Минимальные (не жадные) повторители

Мета-символ	Описание	Пример
`*?`	0 или более повторений предыдущего элемента	`ca*t` соотв **caaatcaata** `ca*?t` соотв **caaatcaata**
`+?`	1 или более повторений предыдущего элемента	
`??`	0 или 1 повторений предыдущего элемента	
`{n}?`	ровно `n` повторений предыдущего элемента	
`{n,}?`	по крайней мере `n` повторений предыдущего элемента	
`{n,m}?`	от `n` до `m` повторений предыдущего элемента	