Регулярные выражения

Класс Regex

Класс Regex и связанные с ним классы описаны в пространстве имен System.Text.RegularExpressions.

Конструктор и свойства

```
Regex(string pattern[, RegexOptions options]);
```

При создании экземпляра Regex регулярное выражение специальным образом обрабатывается, что в дальнейшем ускоряет его использование. Можно также использовать *статические* методы класса, не требующие создания экземпляра.

Для экземпляра класса Regex доступны два свойства (только для чтения): Options (типа RegexOptions) — набор опций поиска, переданных в конструкторе; RightToLeft (типа bool) — порядок поиска регулярного выражения (одна из опций).

Основные методы

bool IsMatch — найдено или нет требуемое выражение;

Match Match — возвращает первое найденное выражение;

MatchCollection Matches — возвращает все найденные выражения;

string[] Split — разбивает строку на фрагменты; разделители фрагментов определяются регулярным выражением;

string Replace — заменяет найденные выражения.

В списке параметров input — строка, в которой выполняется поиск, pattern (и replacement для Replace) — регулярное выражение, start — индекс первого символа, начиная с которого выполняется поиск, count — число возвращаемых фрагментов (Split) или число выполняемых замен (Replace).

Статические методы IsMatch, Match, Matches и Split:

```
(string input, string pattern[, RegexOptions options])
```

Статические методы Replace:

```
(string input, string pattern, MatchEvaluator
  evaluator[, RegexOptions options])
(string input, string pattern, string replacement[,
  RegexOptions options])
```

Экземплярные методы IsMatch, Match и Matches:

```
(string input[, int start])
```

Экземплярные методы Split:

```
(string input[, int count[, start]])
```

Экземплярные методы Replace:

```
(string input, MatchEvaluator evaluator[,
  int count[, int start]])
(string input, string replacement[, int count[, int
  start]])
```

Некоторые вспомогательные методы

static string Escape(string str) — возвращает вариант строки str, в котором экранированы все специальные символы, используемые в регулярных выражениях: \ * + ? | { [( ) ^ $. # и пробельные символы);

static string Unescape(string str) — восстанавливает строку, символы которой ранее были экранированы;

Вспомогательные классы

RegexOptions — перечисление, определяющее *опции поиска*. Основные члены (в скобках указывается значение этой опции, при ее задании непосредственно в регулярном выражении):

IgnoreCase (i) — игнорировать регистр при поиске;

Multiline (m) — *режим многострочного текста*, при котором символы ^ и $ соответствуют началу и концу каждой строчки текста (а не всей содержащей его строки типа string);

ExplicitCapture (n) — считать группами только те пары круглых скобок, которым явно присвоено имя или номер;

Singleline (s) — режим, при котором символ . (точка) соответствует любому символу (а не любому символу, кроме \n);

IgnorePatternWhitespaces (x) — игнорировать неэкранированные пробельные символы в регулярном выражении;

RightToLeft (r) — выполнять поиск справа налево;

None — дополнительных опций нет.

Несколько опций должны объединяться операцией |.

Group — класс, инкапсулирующий свойства *группы* регулярного выражения. Основные свойства (только для чтения):

bool Success — true, если группа найдена, иначе — false;

int Index — индекс начала найденной группы;

int Length — длина найденной группы;

string Value — значение найденной группы (если группа не найдена, то равно пустой строке).

Значение группы также возвращается методом ToString.

Match — класс (потомок Group), инкапсулирующий свойства *найденного вхождения* регулярного выражения. Имеет те же свойства, что и класс Group, которые в данном случае относятся не к группе, а ко всему найденному вхождению. Кроме того, имеет свойство только для чтения Groups типа GroupCollections — коллекцию всех групп, связанных с найденным вхождением; первый элемент этой коллекции (с индексом 0) соответствует *нулевой* группе, т. е. *всему* найденному вхождению.

Метод NextMatch() типа Match позволяет получить *следующее* найденное вхождение (если оно отсутствует, то свойство Success возвращенного объекта Match будет равно false).

GroupCollection — класс-коллекция групп, реализующий интерфейсы ICollection и IEnumerable. Имеет свойство Count типа int — количество групп (только для чтения) и два индексатора типа Group (только для чтения) с числовым или строковым параметром — номером (от 0) или именем группы.

MatchCollection — класс-коллекция найденных вхождений, реализующий интерфейсы ICollection и IEnumerable. Имеет свойство Count типа int — количество найденных (только для чтения) и индексатор типа Match (только для чтения) с числовым параметром — номером найденного вхождения (от 0).

MatchEvaluator — делегат с сигнатурой string (Match match), используемый в методе Replace и определяющий строку, на которую надо заменить найденное вхождение match.

Язык регулярных выражений

Некоторые специальные символы

\t — табуляция;

\r — возврат каретки;

\f — новая страница;

\n — новая строка;

\x*NN* — ASCII-символ в 16-ричной системе счисления;

\u*NNNN* — Unicode-символ в 16-ричной системе счисления.

Множества символов

[abcd] — один из символов в списке (отрицание: [^abcd]);

[a-d] — один из символов в диапазоне (отрицание: [^a-d]);

\d — десятичная цифра, т. е. [0-9] (отрицание: \D);

\w — словообразующий символ, например, для английского языка — [a-zA-Z_0-9] (отрицание: \W);

\s — пробельный символ, т. е. [\t\r\f\n] (отрицание: \S);

. — любой символ, кроме \n (в режиме SingleLine — любой символ).

Символы, указываемые в квадратных скобках, не должны экранироваться (за исключением символа]).

Квантификаторы

* — 0 или более соответствий;

+ — 1 или более соответствий;

? — 0 или 1 соответствие;

{*N*} — точно *N* соответствий;

{*N*,} — не менее *N* соответствий;

{*N*,*M*} — от *N* до *M* соответствий.

Примеры.

Ищется имя файла cv.doc, возможно, снабженное нумерацией. Обратите внимание на экранирование точки и на использование «буквального» режима в регулярном выражении:

```
Regex.IsMatch("cv12.doc", @"cv\d*\.doc") // true
```

Имя файла, которое оканчивается произвольным текстом:

```
Regex.IsMatch("cvnew.doc", @"cv.*\.doc") // true
```

Указанные квантификаторы являются «жадными». Добавление суффикса ? превращает квантификатор в «ленивый»:

```
Regex.Match("zz<i>A<i>B</i>C</i>zz", @"<i>.*</i>")
  // <i>A<i>B</i>C</i>
Regex.Match("zz<i>A<i>B</i>C</i>zz", @"<i>.*?</i>")
  // <i>A<i>B</i>
```

Директивы нулевой длины

^ — начало строки (в режиме Multiline — начало любой строчки многострочного текста);

$ — конец строки (в режиме Multiline — конец любой строчки многострочного текста);

\A — начало строки (в любом режиме);

\z — конец строки (в любом режиме);

\Z — конец строки или строчки многострочного текста;

\b — позиция на границе слова;

\B — позиция не на границе (т. е. внутри) слова;

(?=*expr*) — продолжать поиск, если для выражения *expr* есть соответствие справа (*положительный просмотр вперед*);

(?!*expr*) — продолжать поиск, если для выражения *expr* нет соответствия справа (*отрицательный просмотр вперед*);

(?<=*expr*) — продолжать поиск, если для выражения *expr* есть соответствие слева (*положительный просмотр назад*);

(?<!*expr*) — продолжать поиск, если для выражения *expr* нет соответствия слева (*отрицательный просмотр назад*);

Примеры.

```
Regex.Match("zz<i>A<i>B</i>C</i>zz", @".*(?=</i>)")
  // zz<i>A<i>B</i>C
Regex.Match("zz<i>A<i>B</i>C</i>zz",
  @"(?<=<i>).*(?=</i>)") // A<i>B</i>C
```

Распознавание концов строчек многострочного текста, которые в Windows помечаются двумя символами: \r\n.

```
foreach (Match m in Regex.Matches
  ("a.txt\r\nb.doc\r\nc.txt\r\nd.doc",
  @".*\.txt(?=\r?$)", RegexOptions.Multiline))
    Console.Write(m+" "); // a.txt c.txt
```

Подсчет числа пустых строк в исходном тексте text (квантификатор ? нужен для того, чтобы распознать последнюю пустую строку, после которой может отсутствовать символ \r):

```
Regex.Matches(text, @"^\r?$",
  RegexOptions.Multiline).Count
```

Границей слова считается позиция, в которой словообразующий символ \w соседствует либо с началом/концом строки, либо с символом, который не является словообразующим (\W).

Пример (поиск отдельных слов в строке s):

```
foreach (Match m in Regex.Matches("  aa   ss cc ",
  @"\b\w+\b"))
    Console.Write("("+m+")"); // (aa)(ss)(cc)
```

Группирование и ссылки на группы

(*expr*) — включить соответствие для выражения *expr* в нумерованную группу (группы нумеруются от 1 в соответствии с порядком следования их открывающих скобок; группа 0 соответствует всему найденному вхождению);

(?<*name*>*expr*) или (?'*name*'*expr*) — включить соответствие для выражения *expr* в именованную группу с именем *name*;

(?:*expr*) — группирующее выражение, не связываемое с нумерованной или именованной группой;

N — ссылка на ранее найденную группу с номером *N*;

\k<*name*> — ссылка на группу с именем *name*.

Пример. Чтобы выделить в регулярном выражении для телефонного номера \d{3}-\d{3}-\d{4} начальную группу из трех цифр (код региона) и завершающую группу из 7 цифр (собственно телефонный номер), достаточно заключить их в круглые скобки и воспользоваться свойством Groups:

```
Match m = Regex.Match("123-456-7890",
  @"(\d{3})-(\d{3}-\d{4})");
Console.WriteLine(m.Groups[0]); // 123-456-7890
Console.WriteLine(m.Groups[1]); // 123
Console.WriteLine(m.Groups[2]); // 456-7890
```

Найденные группы можно использовать при продолжении поиска. В следующем примере ищется слово, начинающееся и оканчивающееся на одну и ту же букву:

```
Regex.Match("push pop peek", @"\b(\w)\w*\1\b") // pop
```

Альтернативные варианты

a|b — подходит один из указанных вариантов; при прочих равных условиях предпочтение отдается левому варианту. Можно указывать более двух операндов.

Примеры.

```
Regex.Matches("10", "1|10") // 1
Regex.Matches("10", "10|1") // 10
Regex.Matches("10", "0|1|10") // 1 и 0
```

Комментарии

(?#*comment*) — комментарий;

#*comment* — комментарий до конца строки (только в режиме IgnorePatternWhitespaces).

Некоторые подстановки в выражениях замены

$$ — символ $;

$0 — все найденное вхождение;

$_ — вся исходная строка;

$*N* — найденная группа с номером *N* (или пустая строка, если группа не найдена);

${*name*} — найденная группа с именем *name* (или пустая строка, если группа не найдена).

Пример (заключаем все числа в угловые скобки):

```
Regex.Replace("10+2=12", @"\d+", "<$0>")
  // <10>+<2>=<12>
```

В случае сложных видов замены удобнее использовать вариант метода Replace с параметром-делегатом MatchEvaluator.

Пример (удваиваем все найденные числа):

```
Regex.Replace("10+2=12", @"\d+",
  m => (int.Parse(m.Value)*2).ToString()) // 20+4=24
```

Опции поиска

(?*opt*) — в качестве *opt* указывается буква соответствующей опции (см. описание класса RegexOptions). Любая опция, кроме (?r), может указываться в любом месте регулярного выражения и впоследствии может быть отменена директивой (?-*opt*). Опция (?r) должна быть указана в начале регулярного выражения и не может быть отменена. Опции можно объединять: (?i-ms) — опция i включена, опции m и s отключены.

Примеры:

```
Regex.Match("a", "A", RegexOptions.IgnoreCase) // a
Regex.Match("a", "(?i)A") // a
Regex.Match("BaAaAab", "(?i)A+") // aAaAa
Regex.Match("BaAAaab", "(?i)a(?-i)a") // Aa
```