

Регулярные выражения (RegEx)

Вступление

Регулярные выражения - удобный способ описывать шаблоны текстов.

С помощью регулярных выражений вы можете проверять пользовательский ввод, искать некоторые шаблоны, такие как электронные письма телефонных номеров на веб-страницах или в некоторых документах и так далее.

Ниже приведена исчерпывающая шпаргалка по регулярным выражениям всего на одной странице.

Символы

Простые совпадения

Серия символов соответствует этой серии символов во входной строке.

RegEx	Находит
foobar	foobar

Непечатные символы (escape-коды)

Для представления непечатаемого символа в регулярном выражении используется \x с шестнадцатеричным кодом. Если код длиннее 2 цифр (более U+00FF), то он обрамляется в фигурные скобки.

RegEx	Находит
\xAB	символ с 2-значным шестнадцатеричным кодом AB
\x{AB20}	символ с 1-4 значным шестнадцатеричным кодом AB20
foo\x20bar	foo bar (обратите внимание на пробел в середине)

Существует ряд predefined escape-кодов для непечатных символов, как в языке C :

RegEx	Находит
\t	tab (HT/TAB), тоже что \x09
\n	символ новой строки (LF), то же что \x0a
\r	возврат каретки (CR), тоже что \x0d
\f	form feed (FF), то же что \x0c
\a	звонок (BEL), тоже что \x07
\e	escape (ESC), то же что \x1b

RegEx	Находит
\cA ... \cZ	chr(0) по chr(25). Например \cI соответствует табуляции. Также поддерживаются буквы в нижнем регистре «a»...»z».

Эскейпинг

Для представления спецсимволов (.,*?|\(\)[\{\}^\$), перед ними надо поставить \. Чтобы вставить сам обратный слэш его надо удвоить.

RegEx	Находит
^FooBarPtr	^FooBarPtr здесь ^ не означает начало строки
\[a\]	[a] это не класс символов

Классы символов

Пользовательские классы

Символьный класс - это список символов внутри []. Класс соответствует любому **одному** символу, указанному в этом классе.

RegEx	Находит
foob[aeiou]r	foobar, foobar и т. д., но не foobbr, foobar и т. д.

Вы можете инвертировать класс - если первый символ после [является ^, то класс соответствует любому символу, **кроме** символов, перечисленных в классе.

RegEx	Находит
foob[^aeiou]r	foobbr, foobar и т. д., но не foobar, foobar и т. д.

Внутри списка символ - используется для указания диапазона, так что a-z представляет все символы между a и z включительно.

Если вы хотите, чтобы - сам был членом класса, поместите его в начало или конец списка или предварите его обратной косой чертой ([escape](#)).

Если вы хотите буквально использовать символ] поместите его в начало списка или [escape](#) обратной косой чертой.

RegEx	Находит
[-az]	a, z и -
[az-]	a, z и -
[A\ -z]	a, z и -
[a-z]	символы от a до z
[\n-\x0D]	символы от #10 до #13

 v: latest ▾

Dot Meta-Char

Meta-char `.` (dot) by default matches any character. But if you turn **off** the modifier `/s`, then it won't match line-break characters.

The `.` does not act as meta-class inside `user character classes`. `[.]` means a literal `«.»`.

Метаклассы

Существует ряд predefined классов символов, которые делают регулярные выражения более компактными. Их называют метаклассы:

RegEx	Находит
<code>\w</code>	буквенно-цифровой символ (включая <code>_</code>)
<code>\W</code>	не буквенно-цифровой
<code>\d</code>	числовой символ (тоже, что <code>[0-9]</code>)
<code>\D</code>	не числовой
<code>\s</code>	любой пробел (такой же как <code>[\t\n\r\f]</code>)
<code>\S</code>	не пробел
<code>\h</code>	горизонтальный разделитель. Табуляция, пробел и все символы в Unicode категории «разделители» (space separator Unicode category)
<code>\H</code>	не горизонтальный разделитель
<code>\v</code>	вертикальные разделители. новая строка и все символы «разделители строк» в Unicode
<code>\V</code>	не вертикальный разделитель
<code>\R</code>	unicode line break: LF, pair CR LF, CR, FF (form feed), VT (vertical tab), U+0085, U+2028, U+2029

Все указанные в таблице метаклассы можно использовать внутри `пользовательских классов`.

RegEx	Находит
<code>foob\dr</code>	<code>foob1r</code> , <code>foob6r</code> и т. д., но не <code>foobar</code> , <code>foobbr</code> и т. д.
<code>foob[\w\s]r</code>	<code>foobar</code> , <code>foob r</code> , <code>foobbr</code> и т. д., но не <code>foob1r</code> , <code>foob=r</code> и т. д.

Примечание: [TRegExpr](#)

Свойства `SpaceChars` и `WordChars` определяют, какие символы входят в классы `\w`, `\W`, `\s`, `\S`.

Таким образом, вы можете переопределить эти классы.

 [v: latest](#) ▾

Разделители

Разделители строк

Метасимвол	Находит
<code>^</code>	совпадение нулевой длины в начале строки
<code>\$</code>	совпадение нулевой длины в конце строки
<code>\A</code>	совпадение нулевой длины в начале строки
<code>\z</code>	совпадение нулевой длины в конце строки
<code>\Z</code>	похож на <code>\z</code> но совпадает перед разделителем строки, а не сразу после него, как <code>\z</code>
<code>\G</code>	zero-length match at the end pos of the previous match

Примеры:

RegEx	Находит
<code>^foobar</code>	foobar только если он находится в начале строки
<code>foobar\$</code>	foobar, только если он в конце строки
<code>^foobar\$</code>	foobar только если это единственная строка в строке
<code>foob.r</code>	foobar, foobbr, foob1r и так далее

Метасимвол `^` совпадает с точкой начала строки (нулевой длины). `$` - в конце строки. Если включен `modifier /m`, они совпадают с началами или концами строк внутри текста.

Обратите внимание, что в последовательности `\x0D\x0A` нет пустой строки.


Примечание: TRegExpr

Если вы используете **Unicode версию**, то `^/$` также соответствует `\x2028`, `\x2029`, `\x0B`, `\x0C` или `\x85`.

Метасимвол `\A` совпадает с точкой нулевой длины в начале строки, `\z` - в конце (после символов завершения строки). Модификатор `modifier /m` на них не влияет. `\Z` тоже самое что `\z` но совпадает с точкой перед символами завершения строки (LF and CR LF).

Обратите внимание, что выражение `^.*$` не соответствует точке между `\x0D\x0A`, потому что это неразрывный разделитель строк. Но оно соответствует пустой строке в последовательности `\x0A\x0D`, поэтому из-за неправильного порядка кодов он не воспринимается как разделитель строк и считается просто двумя символами.

Примечание: TRegExpr

Многострочная обработка может быть настроена с помощью свойств `LineSeparator` и `LinePairedSeparator`.  [v: latest](#)

Таким образом, вы можете использовать разделители стиля Unix `\n` или стиль DOS / Windows `\r\n` или смешивать их вместе (как описано выше по умолчанию).

Если вы предпочитаете математически правильное описание, вы можете найти его на сайте www.unicode.org.

Разделители слов

Regex	Находит
<code>\b</code>	разделитель слов
<code>\B</code>	разделитель с не -словом

Граница слова `\b` - это точка между двумя символами, у которой `\w` с одной стороны от нее и `\W` с другой стороны (в любом порядке).

Повторы

Повтор

За любым элементом регулярного выражения может следовать допустимое число повторений элемента.



Regex	Находит
<code>{n}</code>	ровно n раз
<code>{n,}</code>	по крайней мере n раз
<code>{,m}</code>	not more than m times (only with AllowBraceWithoutMin)
<code>{n,m}</code>	по крайней мере n , но не более чем m раз
<code>*</code>	ноль или более, аналогично <code>{0,}</code>
<code>+</code>	один или несколько, похоже на <code>{1,}</code>
<code>?</code>	ноль или единица, похожая на <code>{0,1}</code>

То есть цифры в фигурных скобках `{n,m}` определяют минимальное n и максимальное m количество повторов (совпадений во входном тексте).

`{n}` эквивалентно `{n,n}` и означает точно n раз. `{n,}` совпадает n или более раз.

The variant `{,m}` is only supported if the property AllowBraceWithoutMin is set.

Теоретически значение n и m не ограничены (можно использовать максимальное значение для 32-х битного числа).

Using `{` without a correct range will give an error. This behaviour can be changed by setting the property AllowLiteralBraceWithoutRange, which will accept `{` as a literal char, if not followed by a range. A range with a low value bigger  [v: latest](#)  value will always give an error.

Regex	Находит
foob.*r	foobar, foobalkjdfldkj9r и foobr
foob.+r	foobar, foobalkjdfldkj9r, но не foobr
foob.?r	foobar, foobbr и foobr, но не foobalkj9r
fooba{2}r	foobaar
fooba{2}r	foobaar, foobaaar, foobaaaar и т. д.
fooba{2,3}r	foobaar, или foobaaar, но не foobaaaar
(foobar){8,10}	8, 9 или 10 экземпляров foobar (()) это Группа

Жадность

Повторы в жадном режиме захватывают как можно больше из входного текста, в не жадном режиме - как можно меньше.

По умолчанию все повторы являются жадными. Используйте ? Чтобы сделать любой повтор не жадным.

Для строки abbbbs:

Regex	Находит
b+	bbbb
b+?	b
b*?	пустую строку
b{2,3}?	bb
b{2,3}	bbb

Вы можете переключить все повторы в не жадный режим (**modifier** /g, ниже мы используем **in-line** модификатор change).

Regex	Находит
(?-g)b+	b

Сверхжадные повторы (Possessive Quantifier)

Синтаксис: a++, a*+, a?+, a{2,4}+. В настоящее время реализован только для простых групп и не будет работать для сложных, как например (foo|bar){3,5}+.

Полное описание (на английском) Вкратце, сверхжадный повтор ускоряет работу в сложных случаях.

Альтернативы

Выражения в списке альтернатив разделяются |.

Таким образом, `fee|fie|foe` будет соответствовать любому из `fee`, `fie` или `foe` (также как и `f(e|i|o)e`).

Первое выражение включает в себя все от последнего разделителя шаблона (`(`, `[` или начало шаблона) до первого `|`, а последнее выражение содержит все от последнего `|` к следующему разделителю шаблона.

Звучит сложно, поэтому обычной практикой является заключение списка альтернатив в скобки, чтобы минимизировать путаницу относительно того, где он начинается и заканчивается.

Выражения в списке альтернатив пробуются слева направо, принимается первое же совпадение.

Например, регулярное выражение `foo|foot` в строке `barefoot` будет соответствовать `foo` - первое же совпадение.

Также помните, что `|` в квадратных скобках воспринимается просто как символ, поэтому, если вы напишете `[fee|fie|foe]`, это тоже самое что `[feio|]`.

Regex	Находит
<code>foo(bar foo)</code>	<code>foobar</code> или <code>foofoo</code>

Группы (подвыражения)

Скобки `(...)` также могут использоваться для определения групп (подвыражений) регулярного выражения.

Примечание: [TRegExpr](#)

Позиция, длина и фактические значения подвыражений будут в [MatchPos](#), [MatchLen](#) и [Match](#).

Вы можете заменить их с помощью функции [Substitute](#).

Подвыражения нумеруются слева направо по открывающим их скобкам (включая вложенные группы (подвыражения)). У первой группы номер 1. У выражения в целом - 0.

Например, для входной строки `foobar` регулярное выражение `(foo(bar))` найдет:

Группы (подвыражения)	значение
0	<code>foobar</code>
1	<code>foobar</code>
2	<code>bar</code>

Ссылки на группы (Backreferences)

 [v: latest](#) ▾

Meta-chars \1 through \9 are interpreted as backreferences to capture groups. They match the previously found group with the specified index.

The meta char \g followed by a number is also interpreted as backreferences to capture groups. It can be followed by a multi-digit number.

RegEx	Находит
(.)\1+	aaaa и cc
(+)\1+	также abab и 123123
(.)\g1+	aaaa и cc

(["']?)(\d+)\1 соответствует "13" (в двойных кавычках) или '4' (в одинарных кавычках) или 77 (без кавычек) и т. д.

Именованные группы (подвыражения) и ссылки на НИХ

To make some group named, use this syntax: (?P<name>expr). Also Perl syntax is supported: (?'name'expr). And further: (?<name>expr)

Имя группы должно начинаться с буквы или _, далее следуют буквы, цифры или _. Именованные и не именованные группы имеют общую нумерацию от 1 до 9.

Backreferences to named groups are (?P=name), the numbers \1 to \9 can also be used. As well as the example \g and \k in the table below.

Supported syntax are

(?P=name) \g{name} \k{name} \k<name> \k'name' =====

Example

RegEx	Находит
(?P<qq>["'])\w+(?P=qq)	"word" и 'word'

Модификаторы

Модификаторы предназначены для изменения поведения регулярных выражений.

Вы можете установить модификаторы глобально в вашей системе или изменить их внутри регулярного выражения, используя (?imsxr-imsxr).

Примечание: TRegExpr

Для изменения модификаторов используйте `ModifierStr` или соответствующее свойство `Модификатор *`.

Значения по умолчанию определены в [глобальных переменных](#). Скажем, глобальная переменная `RegExprModifierX` определяет значение по умолчанию для свойства `ModifierX`.

i, без учета регистра

Регистро-независимые сравнения. Использует установленные в вашей системе языковые настройки, см. также [InvertCase](#).

m, многострочные строки

Обрабатывать строку как несколько строк. Таким образом, `^` и `$` соответствуют началу или концу любой строки в любом месте строки.

Смотрите также [Разделители строк](#).

s, одиночные строки

Обрабатывать строку как одну строку. Так что `.` соответствует любому символу, даже разделителям строк.

Смотрите также [Разделители строк](#), которые обычно не совпадают.

г, жадность

Примечание: Специфичный для [TRegExpr](#) модификатор.

Отключив его `Off`, вы переключите [повторитель](#) в [не-жадный](#) режим.

Итак, если модификатор `/g` имеет значение `Off`, то `+` работает как `+`, `*` как `*` и так далее.

По умолчанию этот модификатор имеет значение `Вкл.`

x, расширенный синтаксис

Позволяет комментировать регулярные выражения и разбивать их на несколько строк.

Если модификатор включен, мы игнорируем все пробелы, которые не заэскейплены обратной косой чертой, и не включены в класс символов.

Также символ `#` отделяет комментарии.

Обратите внимание, что вы можете использовать пустые строки для форматирования регулярного выражения для лучшей читаемости:

 [v: latest](#) ▾

```
(
(abc) # комментарий 1
#
(efg) # комментарий 2
)
```

Это также означает, что если вам нужно вставить пробел или символ # в шаблон (вне класса символов, где они не затрагиваются /x), вам придется либо эскейпить их, либо кодировать, используя шестнадцатеричный код.

г, русские диапазоны

Примечание: Специфичный для TRegExpr модификатор.

В русской таблице ASCII символы ё / Ё размещаются отдельно от других.

Большие и маленькие русские символы находятся в отдельных диапазонах, это не отличается от ситуации с английскими символами, но, тем не менее, я хотел иметь краткую форму.

С этим модификатором вместо [а-яА-ЯёЁ] вы можете написать [а-Я], если вам нужны все русские символы.

Когда модификатор включен:

RegEx	Находит
а-я	символы от а до я и ё
А-Я	символы от А до Я и Ё
а-Я	все русские символы

Модификатор по умолчанию установлен на Вкл.

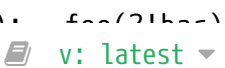
Проверки или заглядывания вперед и назад (Assertions)

Заглядывание вперед (lookahead assertion) `foo(?=bar)` совпадает «foo» только перед «bar», при этом сама строка «bar» не войдет в найденный текст.

Отрицательное заглядывание вперед (negative lookahead assertion): `foo(?!bar)` совпадает «foo» только если после этой строки не следует «bar».

Ретроспективная проверка (lookbehind assertion): `(?<=foo)bar` совпадает «bar» только после «foo», при этом сама строка «foo» не войдет в найденный текст.

Отрицательное заглядывание назад (negative lookbehind assertion) `foo(?!bar)` совпадает «bar» только если перед этой строкой нет «foo».



Ограничения:

- Variable length lookbehind are not allowed to contain capture groups. This can be allowed by setting the property AllowUnsafeLookBehind. If this is enabled and there is more than one match in the text that the group might capture, then the wrong match may be captured. This does not affect the correctness of the overall assertion. (I.e., the lookbehind will correctly return if the text before matched the pattern).
- Variable length lookbehind may be slow to execute, if they do not match.

Не захватываемые группы (подвыражения)

Синтаксис: `(?:subexpression)`.

У этих групп (подвыражений) нет номера, их нельзя указать в ссылке на группу. Эти группы используют чтобы за счет группировки сделать регулярное выражение более читаемым, но нет необходимости расходовать ресурсы на то, чтобы реально отдельно захватывать то, с чем такие группы совпадут:

RegExp	Находит
<code>(https? ftp)://([^\r\n]+)</code>	в <code>https://sorokin.engineer</code> захватит подвыражения <code>https</code> и <code>sorokin.engineer</code>
<code>(?:https? ftp)://([^\r\n]+)</code>	в <code>https://sorokin.engineer</code> захватит только <code>sorokin.engineer</code>

Атомарные группы

Синтаксис: `(?>expr|expr|...)`.



Атомарные группы это специальный случай незахватывающих групп. [Подробнее](#)

Модификаторы

Синтаксис для одного модификатора: `(?i)` чтобы включить, и `(?-i)` чтобы выключить. Для большого числа модификаторов используется синтаксис: `(?msgxr-imsgr)`.

Можно использовать внутри регулярного выражения. Это может быть особенно удобно, поскольку оно имеет локальную область видимости. Оно влияет только на ту часть регулярного выражения, которая следует за оператором `(?imsgr-imsgr)`.

И если оно находится внутри подвыражения, оно будет влиять только на это подвыражение, а именно на ту часть подвыражения, которая следует за оператором. Таким образом, в `((?i)Saint)-Petersburg` это влияет только на подвыражение `((?i)Saint)`, поэтому оно будет соответствовать `saint-Petersburg`, но не `saint-petersburg`.

Inline modifiers can also be given as part of a non-capturing group: `( v: latest `

RegEx	Находит
(?i)Saint-Petersburg	Saint-petersburg и Saint-Petersburg
(?i)Saint-(?-i)Petersburg	Saint-Petersburg, но не Saint-petersburg
(?i)(Saint-)?Petersburg	Saint-petersburg и saint-petersburg
((?i)Saint-)?Petersburg	saint-Petersburg, но не saint-petersburg

Комментарии

Синтаксис: (?#text). Все, что внутри скобок, игнорируется.

Обратите внимание, что комментарий закрывается ближайшим), поэтому нет способа вставить литерал) в комментарий.

Рекурсия

Синтаксис (?R), синоним (?0).

Выражение a(?R)?z совпадает с одним или более символом «a» за которым следует точно такое же число символов «z».

Основное назначение рекурсии - сбалансировать обрамление вложенного текста. Общий вид b(?:m|(?R))*e где «b» это то что начинает обрамляемый текст, «m» это собственно текст, и «e» это то, что завершает обрамление.

Если же обрамляемый текст также может встречаться без обрамления то выражение будет b(?R)*e|m.

Вызовы подвыражений

Нумерованные группы (подвыражения) обозначают (?1) ... (?90) (максимальное число групп определяется константой в TRegExpr).


Синтаксис для именованных групп : (?P>name). Поддерживается также Perl вариант синтаксиса: (?&name).

Supported syntax are

(?number) (?P>name) (?&name) \g<name> \g'name' =====

Это похоже на рекурсию, но повторяет только указанную группу (подвыражение).

Unicode категории (category)

В стандарте Unicode есть именованные категории символов (Unicode  v: latest ▾ Категория обозначается одной буквой, и еще одна добавляется, чтобы указать

подкатегорию. Например «L» это буква в любом регистре, «Lu» - буквы в верхнем регистре, «Ll» - в нижнем.

- Cc - Control
- Cf - Формат
- Co - Частное использование
- Cs - Заменитель (Surrogate)
- Ll - Буква нижнего регистра
- Lm - Буква-модификатор
- Lo - Прочие буквы
- Lt - Titlecase Letter
- Lu - Буква в верхнем регистре
- Mc - Разделитель
- Me - Закрывающий знак (Enclosing Mark)
- Mn - Несамостоятельный символ, как умляут над буквой (Nonspacing Mark)
- Nd - Десятичная цифра
- Nl - Буквенная цифра - например, китайская, римская, руническая и т.д. (Letter Number)
- No - Другие цифры
- Pc - Connector Punctuation
- Pd - Dash Punctuation
- Pe - Close Punctuation
- Pf - Final Punctuation
- Pi - Initial Punctuation
- Po - Other Punctuation
- Ps - Open Punctuation
- Sc - Currency Symbol
- Sk - Modifier Symbol
- Sm - Математический символ
- So - Прочие символы
- Zl - Разделитель строк
- Zp - Разделитель параграфов
- Zs - Space Separator

Meta-character `\p` denotes one Unicode char of specified category. Syntax: `\pL` and `\p{L}` for 1-letter name, `\p{Lu}` for 2-letter names.

Метасимвол `\P` это символ **не** из Unicode категории (category).

These meta-characters are supported within character classes too.

Послесловие

В этой [древней статье](#) из [прошлого века](#) есть примеры использования регулярных выражений.