Регулярные выражения

Материал из Викиучебника — открытых книг для открытого мира

Заглавная страница
Каталог учебников
Кулинарная книга
Случайная статья

Участие
Справка
Форум
Свежие правки
Новые страницы
Пожертвовать

Инструменты
Ссылки сюда
Связанные правки
Спецстраницы
Постоянная ссылка
Сведения о странице
Элемент Викиданных
Цитировать

В Википедии имеется статья по теме **«Регулярные выражения»**

Регуля́рные выраже́ния (англ. *regular expressions*, жарг. ***регэ́кспы*** или ***ре́гексы***) — система обработки текста, основанная на специальной системе записи образцов для поиска. Образец (англ. *pattern*), задающий правило поиска, по-русски также иногда называют «шаблоном», «маской».

Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Многие языки программирования уже поддерживают регулярные выражения для работы со строками. Например, Perl и Tcl имеют встроенный в их синтаксис механизм обработки регулярных выражений. Набор утилит (включая редактор sed и фильтр grep), поставляемых в дистрибутивах Unix, одним из первых способствовал популяризации понятия регулярных выражений.

Содержание

страницу

Печать/экспорт

Создать книгу
Скачать как PDF
Версия для печати

В других проектах

Викисклад
Википедия

На других языках

English
日本語
Português
中文

Править ссылки

Базовые понятия [править]

Регулярные выражения используются для сжатого описания некоторого множества строк с помощью шаблонов, без необходимости перечисления всех элементов этого множества. При составлении шаблонов используется специальный синтаксис, поддерживающий, обычно, следующие операции:

Перечисление

Вертикальная черта разделяет допустимые варианты. Например, «gray|grey» соответствует *gray* или *grey*.

Группировка

Круглые скобки используются для определения области действия и приоритета операторов. Например, «gray|grey» и «gr(a|e)y» являются разными образцами, но они оба описывают множество, содержащее *gray* и *grey*.

Квантификация

Квантификатор после символа или группы определяет, сколько раз предшествующее выражение может встречаться.

{n,m}

общее выражение, повторений может быть **от n до m включительно**.

{n,}

общее выражение, **n и более повторений**.

{,m}

общее выражение, **не более m повторений**.

{n}

общее выражение, **ровно n повторений**

?

Знак вопроса означает **0 или 1** раз, то же самое, что и **{0,1}**. Например, «colou?r» соответствует и *color*, и *colour*.

*****

Звёздочка означает **0, 1 или любое число** раз (**{0,}**). Например, «go*gle» соответствует *ggle*, *gogle*, *google* и др.

+

Плюс означает **хотя бы 1** раз (**{1,}**). Например, «go+gle» соответствует *gogle*, *google* и т. д. (но не *ggle*).

Конкретный синтаксис регулярных выражений зависит от реализации.

В теории формальных языков [править]

Регулярные выражения состоят из констант и операторов, которые определяют множества строк и множества операций на них соответственно. На данном конечном алфавите Σ определены следующие константы:

- (*пустое множество*) ∅ обозначает ∅

- (*пустая строка*) ε обозначает множество {ε}
- (*строка*) *a* в Σ обозначает множество {*a*}

и следующие операции:

- (*связь*, *конкатенация*) *RS* обозначает множество { αβ | α из *R* и β из *S* }. Пример: {"ab", "c"}{"d", "ef"} = {"abd", "abef", "cd", "cef"}.
- (*перечисление*) *R*|*S* обозначает объединение *R* и *S*.
- (*замыкание Клини, звезда Клини*) *R** обозначает минимальное надмножество из *R*, которое содержит ε и закрыто связью строк. Это есть множество всех строк, которые могут быть получены связью нуля или более строк из *R*. Например, {"ab", "c"}* = {ε, "ab", "c", "abab", "abc", "cab", "cc", "ababab", … }.

Многие книги используют символы ∪, + или ∨ для перечисления вместо вертикальной черты.

Синтаксис [править]

Традиционные регулярные выражения в Unix [править]

Синтаксис «базовых» регулярных выражений Unix на данный момент определён POSIX как устаревший, но он до сих пор широко распространён из соображений обратной совместимости. Многие Unix-утилиты используют такие регулярные выражения по умолчанию.

В этом синтаксисе большинство символов соответствуют сами себе («a» соответствует «a» и т. д.). Исключения из этого правила называются метасимволами:

.	Соответствует любому единичному символу.

[ ]	Соответствует любому единичному символу из числа заключённых в скобки. Символ «-» интерпретируется буквально только в том случае, если он расположен непосредственно после открывающей или перед закрывающей скобкой: [abc-] или [-abc]. В противном случае, он обозначает интервал символов. Например, [abc] соответствует «a», «b» или «c». [a-z] соответствует буквам нижнего регистра латинского алфавита. Эти обозначения могут и сочетаться: [abcq-z] соответствует a, b, c, q, r, s, t, u, v, w, x, y, z. Чтобы установить соответствие символам «[» или «]», достаточно, чтобы закрывающая скобка была первым символом после открывающей: [][ab] соответствует «]», «[», «a» или «b».
[^]	Соответствует единичному символу из числа тех, которых нет в скобках. Например, [^abc] соответствует любому символу, кроме «a», «b» или «c». [^a-z] соответствует любому символу, кроме символов нижнего регистра в латинском алфавите.
^	Соответствует началу текста (или началу любой строки в мультистроковом режиме).Ищет с начала текста.
$	Соответствует концу текста (или концу любой строки в мультистроковом режиме).
\(\)	Объявляет «отмеченное подвыражение», которое может быть использовано позже (см. следующий элемент: *n*). «Отмеченное подвыражение» также является «блоком». В отличие от других операторов, этот (в традиционном синтаксисе) требует бэкслеша.
n	Где *n* — это цифра от 1 до 9; соответствует *n*-му отмеченному подвыражению. Эта конструкция теоретически **нерегулярна**, она не была принята в расширенном синтаксисе регулярных выражений.

*	• Звёздочка после выражения, соответствующего единичному символу, соответствует нулю или более копий этого выражения. Например, «[xyz]*» соответствует пустой строке, «x», «y», «zx», «zyx», и т. д. • \n*, где *n* — это цифра от 1 до 9, соответствует нулю или более вхождений для соответствия *n*-го отмеченного подвыражения. Например, «\(a.\)c\1*» соответствует «abcab» и «abcaba», но не «abcac». • Выражение, заключённое в «\(» и «\)» и сопровождаемое «*», следует считать неправильным. В некоторых случаях, оно соответствует нулю или более вхождений строки, которая была заключена в скобки. В других, оно соответствует выражению, заключённому в скобки, учитывая символ «*».
\{*x*,*y*\}	Соответствует последнему блоку, встречающемуся не менее *x* и не более *y* раз. Например, «a\{3,5\}» соответствует «aaa», «aaaa» или «aaaaa». В отличие от других операторов, этот (в традиционном синтаксисе) требует бэкслеша.

Различные реализации регулярных выражений интерпретируют обратную косую черту перед метасимволами по-разному. Например, egrep и Perl интерпретируют скобки и вертикальную черту как метасимволы, если перед ними *нет* обратной косой черты и воспринимают их как обычные символы, если черта есть.

Многие диапазоны символов зависят от выбранных настроек локализации. POSIX стандартизовал объявление некоторых классов и категорий символов, как показано в следующей таблице:

POSIX класс	подобно	Perl	означает
[:upper:]	[A-Z]		символы верхнего регистра

[:lower:]	[a-z]		символы нижнего регистра
[:alpha:]	[A-Za-z]		символы верхнего и нижнего регистра
[:alnum:]	[A-Za-z0-9]		цифры, символы верхнего и нижнего регистра
	[A-Za-z0-9_]	\w	цифры, символы верхнего, нижнего регистра и "_"
	[^A-Za-z0-9_]	\W	не цифры, символы верхнего, нижнего регистра и "_"
[:digit:]	[0-9]	\d	цифры
	[^0-9]	\D	не цифры
[:xdigit:]	[0-9A-Fa-f]		шестнадцатеричные цифры
[:punct:]	[.,!?:…]		знаки пунктуации
[:blank:]	[\t]		пробел и TAB
[:space:]	[\t\n\r\f\v]	\s	символы пробелов(пропуска)
	[^ \t\n\r\f\v]	\S	не символы пробелов(пропуска)
[:cntrl:]	[\x00-\x1F\x7F]		символы управления
[:graph:]	[:alnum:] ∪ [:punct:]		символы печати
[:print:]	[\x20-\x7E]		символы печати и символы пропуска(видимые символы и пробелы)

Защита метасимволов [править]

Способ представить сами метасимволы `.`, `-` `[` `]` и другие в регулярных выражениях без интерпретации, то есть, в качестве простых (не специальных) символов — предварить их обратной косой чертой: \. Например, чтобы представить сам символ «точка» (просто точка,

и ничего более), надо написать `\.` (обратная косая черта, а за ней - точка). Чтобы представить символ открывающей квадратной скобки `[`, надо написать `\[` (обратная косая черта и следом за ней скобка `[`) и т.д. Сам метасимвол `\` тоже может быть защищен, то есть представлен как `\\` (две обратных косых черты), и тогда интерпретатор регулярных выражений воспримет его как простой символ обратной косой черты `\`.

«Жадные» выражения [править]

Квантификаторам в регулярных выражениях соответствует максимально длинная строка из возможных (квантификаторы являются «жадными», англ. *greedy*). Это может оказаться значительной проблемой. Например, часто ожидают, что выражение `(<.*>)` найдёт в тексте теги HTML. Однако этому выражению соответствует целиком строка

```
<p><b>Википедия</b> — свободная энциклопедия, в которой <i>каждый</i> может изменить или дополнить любую статью</p>
```

.

Эту проблему можно решить двумя способами. Первый состоит в том, что в регулярном выражении учитываются символы, *не* соответствующие желаемому образцу (`<[^>]*>` для вышеописанного случая). Второй заключается в определении квантификатора как нежадного (ленивого, англ. *lazy*)— большинство реализаций позволяют это сделать, добавив после него знак вопроса.

Например, выражению `(<.*?>)` соответствует не вся показанная выше строка, а отдельные теги (выделены цветом):

```
<p><b>Википедия</b> — свободная энциклопедия, в которой <i>каждый</i> может изменить или дополнить любую статью</p>
```

- `*?` - «не жадный» («ленивый») эквивалент `*`
- `+?` - «не жадный» («ленивый») эквивалент `+`
- `{n,}?` - «не жадный» («ленивый») эквивалент `{n,}`

Использование «ленивых» квантификаторов, правда, может повлечь за собой обратную проблему, когда выражению соответствует слишком короткая строка.

Также существуют квантификаторы повышения жадности, то, что захвачено ими однажды, назад уже не отдается. Сверхжадные квантификаторы (possessive quantifiers)

- `*+` - «сверхжадный» эквивалент `*`
- `++` - «сверхжадный» эквивалент `+`
- `{n,}+` - «сверхжадный» эквивалент `{n,}`

Современные (расширенные) регулярные выражения в POSIX [править]

Регулярные выражения в POSIX аналогичны традиционному Unix-синтаксису, но с добавлением некоторых метасимволов:

+ Указывает на то, что предыдущий символ или группа может повторяться один или несколько раз. В отличие от звёздочки, хотя бы одно повторение обязательно.

? Делает предыдущий символ или группу необязательной. Другими словами, в соответствующей строке она может отсутствовать, либо присутствовать ровно один раз.

| Разделяет альтернативные варианты регулярных выражений. Один символ задаёт две альтернативы, но их может быть и больше, достаточно использовать больше вертикальных чёрточек. Необходимо помнить, что этот оператор использует максимально возможную часть выражения. По этой причине, оператор альтернативы

чаще всего используется внутри скобок.

Также было отменено использование обратной косой черты: \{…\} становится {…} и \(…\) становится (…).

Perl-совместимые регулярные выражения (PCRE) [править]

Регулярные выражения в Perl имеют более богатый и в то же время предсказуемый синтаксис, чем даже в POSIX. По этой причине очень многие приложения используют именно Perl-совместимый синтаксис регулярных выражений.

Группы [править]

( )

Простая группа с захватом.

(?:)

Группа без захвата. То же самое, но заключённое в скобках выражение не добавляется к списку захваченных фрагментов. Например, если требуется найти или «здравствуйте», или «здрасте», но не важно, какое именно приветствие найдено, можно воспользоваться выражением `здра(?:сте|вствуйте)`.

(?=)

Группа с положительной опережающей проверкой (positive lookahead assertion). Продолжает поиск только если справа от текущей позиции в тексте находится заключённое в скобки выражение. При этом само выражение не захватывается. Например, `говор(?=ить)` найдёт «говор» в «говорить», но не в «говорит». Иными словами ищет в строке говор после которого сразу идут символы ить - если находит выдает истину, иначе ложь (FALSE).

(?!)

Группа с негативной опережающей проверкой (negative lookahead assertion). Продолжает поиск только если справа от текущей позиции в тексте не находится заключённое в скобки выражение. При этом само выражение не захватывается. Например, `говор(?!ить)` найдёт «говор» в «говорит», но не в «говорить».

(?<=)

Группа с положительной ретроспективной проверкой (positive lookbehind assertion). Продолжает поиск только если слева от текущей позиции в тексте находится заключённое в скобки выражение. При этом само выражение не захватывается. Например, `(?<=об)говорить` найдёт «говорить» в «обговорить», но не в «уговорить».

(?<!)

Группа с отрицательной ретроспективной проверкой (negative lookbehind assertion). Продолжает поиск только если слева от текущей позиции в тексте не находится заключённое в скобки выражение. При этом само выражение не захватывается. Например, `(?<!об)говорить` найдёт «говорить» в «уговорить», но не в «обговорить».

...

Реализации [править]

- NFA (Nondeterministic Finite State Machine; Недетерминированные Конечные Автоматы) используют «жадный» алгоритм отката, проверяя все возможные расширения регулярного выражения в определённом порядке и выбирая первое подходящее значение. NFA может обрабатывать подвыражения и обратные ссылки. Но из-за алгоритма отката традиционный NFA может проверять одно и то же место несколько раз, что отрицательно сказывается на скорости работы. Поскольку традиционный NFA принимает первое найденное соответствие, он может и не найти самое длинное из

вхождений (этого требует стандарт POSIX, и существуют модификации NFA выполняющие это требование — GNU sed). Именно такой механизм регулярных выражений используется, например, в Perl, Tcl и .NET.

- DFA (Deterministic Finite-state Automaton; Детерминированные Конечные Автоматы) работают линейно по времени, поскольку не используют откаты и никогда не проверяют какую-либо часть текста дважды. Они могут гарантированно найти самую длинную строку из возможных. DFA содержит только конечное состояние, следовательно, не обрабатывает обратных ссылок, а также не поддерживает конструкций с явным расширением, то есть не способен обработать и подвыражения. DFA используется, например, в lex и egrep.

Литература [править]

- *Билл Смит* **Методы и алгоритмы вычислений на строках (regexp)** = Computing Patterns in Strings. — М.: «Вильямс», 2006. — С. 496. — ISBN 0-201-39839-7
- Фридл Дж. Регулярные выражения. Библиотека программиста. — СПб.: Питер, 2001. 352 с. ISBN 5-318-00056-8.
- *Бен Форта* **Освой самостоятельно регулярные выражения (regexp). PHP, Perl, JavaScript, Java, C#(си шарп), Visual Basic, ASP.NET,JSP, MySQL, Unix, Linux** = Sams Teach Yourself Regular Expressions in 10 Minutes. — М.: «Вильямс», 2004. — С. 192. — ISBN 0-672-32566-7
- *Царьков В. Б.* **Теория и методика построения регулярных выражений. Проблема самообразования**. — 2010.
- Ян Гойвертс, Стивен Левитан Регулярные выражения. Сборник рецептов — СПб.: Символ-Плюс, 2010.352 с. ISBN 978-5-93286-181-3, ISBN 978-0-596-52068-7 (англ.)

Ссылки [править]

- Regular-Expressions.info — Regex Tutorial, Examples and Reference - Regexp Patterns
- Regex Guru Blog
- A JavaScript and regular expression centric blog
- The one of a kind JavaScript regular expression library
- A JavaScript regular expression tester
- Онлайн генератор регулярных выражений
- Онлайн сервис по работе с регулярными выражениями
- PCRE.RU — Регулярные выражения, примеры, документация и шаблоны в perl, php, javascript, apache
- Online regex tester and debugger: JavaScript, Python, PHP, and PCRE
- Regular expression tester
- RegexPlanet — Online Regular Expression (Regex) Testing and Cookbook
- Regex Tutorial: From Regex 101 to Advanced Regex
- Регулярные выражения в JavaScript
- Регулярные выражения в C#

Категории: Программирование | Требуется внимание (все учебники)