

## Регулярные выражения

Материал из Википедии — свободной энциклопедии

Эта версия страницы ожидает проверки и может отличаться от последней подтверждённой, проверенной 27 июня 2011.



Перейти к: навигация, поиск

Регуля́рные выраже́ния (англ. regular expressions, сокр. RegExp, RegEx, жарг. регэ́кспы или ре́гексы) — это формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters). По сути это строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска.

Регулярные выражения произвели прорыв в электронной обработке текстов в конце XX века. Набор утилит (включая редактор sed и фильтр grep), поставляемых в дистрибутивах UNIX, одним из первых способствовал популяризации регулярных выражений для обработки текстов. Многие современные языки программирования имеют встроенную поддержку регулярных выражений. Среди них Perl, Java[1], PHP, JavaScript, языки платформы .NET Framework[2], Python, Tcl, Ruby, Lua и др.

Регулярные выражения используются некоторыми текстовыми редакторами и утилитами для поиска и подстановки текста. Например, при помощи регулярных выражений можно задать шаблоны, позволяющие:

- найти все последовательности символов «кот» в любом контексте, как то: «кот», «котлета», «терракотовый»;
- найти отдельно стоящее слово «кот» и заменить его на «кошка»;
- найти слово «кот», которому предшествует слово «персидский» или «чеширский»;
- убрать из текста все предложения, в которых упоминается слово кот или кошка.

Регулярные выражения позволяют задавать и гораздо более [сложные](#) шаблоны поиска или замены.

### Содержание

- 1 Историяnht
- 2 В теории формальных языков
- 3 Синтаксис
  - 3.1 Представление символов
    - 3.1.1 Обычные символы (литералы) и специальные символы (метасимволы)
    - 3.1.2 Любой символ
    - 3.1.3 Символьные классы (наборы символов)
  - 3.2 Позиция внутри строки
  - 3.3 Квантификация (поиск последовательностей)
    - 3.3.1 Жадная и ленивая квантификация
    - 3.3.2 Ревнивая квантификация (Сверхжадная)
  - 3.4 Группировка
    - 3.4.1 Обозначение группы
    - 3.4.2 Обратная связь
    - 3.4.3 Группировка без обратной связи
    - 3.4.4 Атомарная группировка
    - 3.4.5 Модификаторы
    - 3.4.6 Комментарии
  - 3.5 Перечисление
  - 3.6 Просмотр вперёд и назад
  - 3.7 Поиск по условию
- 4 Разновидности регулярных выражений
  - 4.1 Базовые регулярные выражения POSIX
  - 4.2 Расширенные регулярные выражения POSIX
  - 4.3 Регулярные выражения, совместимые с Perl
- 5 Нечёткие регулярные выражения

- 6 Реализации
- 7 Примечания
- 8 Литература
- 9 См. также
- 10 Ссылки

[править] История

Истоки регулярных выражений лежат в теории автоматов, теории формальных языков и классификации формальных грамматик по Хомскому[3]. Эти области изучают вычислительные модели (автоматы) и способы описания и классификации формальных языков. В 1940-х гг. Уоррен Маккалок и Уолтер Питтс описали нервную систему, используя простой автомат в качестве модели нейрона. Математик Стивен Клини позже описал эти модели, используя свою систему математических обозначений, названную «регулярные множества». Кен Томпсон встроил их в редактор QED, а затем в редактор ed под UNIX. С этого времени регулярные выражения стали широко использоваться в UNIX и UNIX-подобных утилитах, например: expr, awk, Emacs, vi, lex и Perl.

Регулярные выражения в Perl и Tcl происходят от реализации, написанной Генри Спенсером. Филип Хейзел разработал библиотеку PCRE (англ. Perl-compatible regular expressions — Perl-совместимые регулярные выражения), которая используется во многих современных инструментах, таких как PHP и Apache.

[править] В теории формальных языков

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

- (пустое множество)  $\square$
- (пустая строка)  $\epsilon$  обозначает строку, не содержащую ни одного символа. Эквивалентно «».
- (символьный литерал) «a», где a — символ алфавита  $\Sigma$ .

и следующие операции:

- (сцепление, конкатенация)  $RS$  обозначает множество  $\{a\beta \mid a \in R \ \& \ \beta \in S\}$ . Например,  $\{\text{"boy"}, \text{"girl"}\}\{\text{"friend"}, \text{"cott"}\} = \{\text{"boyfriend"}, \text{"girlfriend"}, \text{"boycott"}, \text{"girlcott"}\}$ .
- (дизъюнкция, чередование)  $R|S$  обозначает объединение  $R$  и  $S$ . Например,  $\{\text{"ab"}, \text{"c"}\}|\{\text{"ab"}, \text{"d"}, \text{"ef"}\} = \{\text{"ab"}, \text{"c"}, \text{"d"}, \text{"ef"}\}$ . [4]
- (замыкание Клини, звезда Клини)  $R^*$  обозначает минимальное надмножество множества  $R$ , которое содержит  $\epsilon$  и замкнуто относительно конкатенации. Это есть множество всех строк, полученных конкатенацией нуля или более строк из  $R$ . Например,  $\{\text{"Go"}, \text{"Russia"}\}^* = \{\epsilon, \text{"Go"}, \text{"Russia"}, \text{"GoGo"}, \text{"GoRussia"}, \text{"RussiaGo"}, \text{"RussiaRussia"}, \text{"GoGoGo"}, \text{"GoGoRussia"}, \text{"GoRussiaGo"}, \dots\}$ .

[править] Синтаксис

[править] Представление символов

Основная статья: Представление символов в регулярных выражениях

[править] Обычные символы (литералы) и специальные символы (метасимволы)

Большинство символов в регулярном выражении представляют сами себя за исключением специальных символов `[ ] \ ^ $ . | ? * + ( ) { }`, которые могут быть предварены символом `\` (обратная косая черта) («экранированы», «защищены») для представления их самих в качестве символов текста. Можно экранировать целую последовательность символов, заключив её между `\Q` и `\E`.

Пример	Соответствие
<code>a\.</code>	<code>a</code> . или <code>a</code>
<code>a\\b</code>	<code>a\b</code>
<code>a[F]</code>	<code>a[F]</code>
<code>\Q+*/\E</code>	<code>+*/</code>

Аналогично могут быть представлены другие специальные символы (набор символов, требующих экранирования, может отличаться в зависимости от конкретной реализации). Часть символов, которые в той или иной реализации не требуют

экранирования (например, угловые скобки < >), могут быть экранированы из соображений удобочитаемости.

[править] Любой символ

Метасимвол . (точка) означает один любой символ, но в некоторых реализациях исключая символ новой строки.

[править] Символьные классы (наборы символов)

Набор символов в квадратных скобках [ ] именуется символьным классом и позволяет указать интерпретатору регулярных выражений, что на данном месте в строке может стоять один из перечисленных символов. В частности, [абв] задаёт возможность появления в тексте одного из трёх указанных символов, а [1234567890] задаёт соответствие одной из цифр. Возможно указание диапазонов символов: например, [А-Яа-я] соответствует всем буквам русского алфавита, за исключением букв «Ё» и «ё».[5] Если требуется указать символы, которые не входят в указанный набор, то используют символ ^ внутри квадратных скобок, например [^0-9] означает любой символ, кроме цифр.

Добавление в набор специальных символов путём экранирования — самый бесхитростный способ. Однако в современных регулярных выражениях унаследован также и традиционный подход — см. Традиционные регулярные выражения.

Некоторые символьные классы можно заменить специальными метасимволами:

Символ	Описание
\d	Соответствует цифре. Эквивалентно [0-9]
\D	Соответствует нецифровому символу. Эквивалентно [^0-9]
\s	Соответствует символу пробела. Эквивалентно [\f\n\r\t]
\S	Соответствует любому непробельному символу. Эквивалентно [^\f\n\r\t]
\w	Соответствует латинской букве, цифре или подчеркиванию. Эквивалентно [A-Za-z0-9_]
\W	Соответствует любому символу, кроме латинской буквы, цифры или подчеркивания. Эквивалентно [^A-Za-z0-9_]

[править] Позиция внутри строки

Следующие символы позволяют спозиционировать регулярное выражение относительно элементов текста: начала и конца строки, границ слова.

Представление	Позиция	Пример	Соответствие
^	Начало строки	^a	aaa aaa
\$	Конец строки	a\$	aaa aaa
\b	Граница слова	a\b	aaa aaa
		\ba	aaa aaa
\B	Не граница слова	\Ba\b	aaa aaa
\G	Предыдущий успешный поиск	\Ga	aaa aaa (поиск остановился на 4-й позиции — там, где не нашлось a)

[править] Квантификация (поиск последовательностей)

Квантификатор после символа, символьного класса или группы определяет, сколько раз предшествующее выражение может встречаться. Следует учитывать, что квантификатор может относиться более чем к одному символу в регулярном выражении, только если это символьный класс или группа.

Представление	Число повторений	Пример	Соответствие	
{n}	Ровно n раз	colou{3}r	colouuur	
{m,n}	От m до n включительно	colou{2,4}r	colouur, colouuur, colouuuur	
{m,}	Не менее m	colou{2,}r	colouur, colouuur, colouuuur и т. д.	
{,n}	Не более n	colou{,3}r	color, colour, colouur, colouuur	
Представление	Число повторений	Эквивалент	Пример	Соответствие
*	Ноль или более	{0,}	colou*r	color, colour, colouur и т. д.
+	Одно или более	{1,}	colou+r	colour, colouur и т. д. (но не color)

?                      Ноль или одно                      {0,1}                      colour?r                      color, colour

Часто используется последовательность `.*` для обозначения любого количества любых символов между двумя частями регулярного выражения.

Символьные классы в сочетании с квантификаторами позволяют устанавливать соответствия с реальными текстами. Например, столбцами цифр, телефонами, почтовыми адресами, элементами HTML-разметки и др.

Если символы `{ }` не образуют квантификатор, их специальное значение игнорируется.

[править] Жадная и ленивая квантификация

Пример использования жадных и ленивых выражений

Выражение `(<.*>)` соответствует строке, содержащей несколько тегов HTML-разметки, целиком.

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

Чтобы выделить отдельные теги, можно применить ленивую версию этого выражения:

`(<.*?>)` Ей 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>`.

В некоторых реализациях 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, если в тексте есть более одного HTML-тега, то этому выражению соответствует целиком строка, содержащая множество тегов.

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

Эту проблему можно решить двумя способами.

1. Учитывать символы, не соответствующие желаемому образцу `(<[^\>]*)` для вышеописанного случая).
2. Определить квантификатор как нежадный (ленивый, англ. *lazy*) — большинство реализаций позволяют это сделать, добавив после него знак вопроса.

Использование ленивых квантификаторов может повлечь за собой обратную проблему, когда выражению соответствует слишком короткая, в частности, пустая строка.

Жадный      Ленивый

*	*?
+	+?
{n,}	{n,}?

Также общей проблемой как жадных, так и ленивых выражений являются точки возврата для перебора вариантов выражения. Точки ставятся после каждой итерации квантификатора. Если интерпретатор не нашёл соответствия после квантификатора, то он начинает возвращаться по всем установленным точкам, пересчитывая оттуда выражение по-другому.

[править] Ревнивая квантификация (Сверхжадная)

При поиске выражения `(a+a+)+b` в строке `aaaaa` интерпретатор пойдёт приблизительно по следующему пути:

1. `aaaaa`
2. `aaaaa`
3. `aaaaa`
4. `aaaaa`
5. `aaaaa`
6. `aaaaa`
7. `aaaaa` — и только тут, проверив все точки возврата, сдастся.

При использовании ревнивого квантификатора будет выполнен только первый шаг алгоритма.

В отличие от обычной (жадной) квантификации, ревнивая квантификация не только старается найти максимально длинный вариант, но ещё и не позволяет алгоритму возвращаться к предыдущим шагам поиска для того, чтобы найти возможные соответствия для оставшейся части регулярного выражения.

Использование ревнивых квантификаторов увеличивает скорость поиска, особенно в тех случаях, когда строка не соответствует регулярному выражению. Кроме того, ревнивые квантификаторы могут быть использованы для исключения нежелательных совпадений.

Жадный	Ревнивый
*	*+
?	?+
+	++
{n,}	{n,}+

Пример

Соответствие

ab(xa)\*+a    abxaabxhaa; но не abxaabxhaa, так как буква a уже занята

[править] Группировка

[править] Обозначение группы

Круглые скобки используются для определения области действия и приоритета операций. Шаблон внутри группы обрабатывается как единое целое и может быть квантифицирован. Например, выражение (тр[ау]м-?)\* найдёт последовательность вида трам-трам-трумтрам-трум-трамтрум.

[править] Обратная связь

Одно из применений группировки — повторное использование ранее найденных групп символов (подстрок, блоков, отмеченных подвыражений). При обработке выражения подстроки, найденные по шаблону внутри группы, сохраняются в отдельной области памяти и получают номер начиная с единицы. Каждой подстроке соответствует пара скобок в регулярном выражении. Квантификация группы не влияет на сохранённый результат, то есть сохраняется лишь первое вхождение. Обычно поддерживается до 9 нумерованных подстрок с номерами от 1 до 9, но некоторые интерпретаторы позволяют работать с большим количеством. Впоследствии в пределах данного регулярного выражения можно использовать обозначения от \1 до \9 для проверки на совпадение с ранее найденной подстрокой.

Например, регулярное выражение (та|ту)-\1 найдёт строку та-та или ту-ту, но пропустит строку та-ту.

Также ранее найденные подстроки можно использовать при замене по регулярному выражению. В таком случае в замещающий текст вставляются те же обозначения, что и в пределах самого выражения.

[править] Группировка без обратной связи

Если группа используется только для группировки и её результат в дальнейшем не потребуется, то можно использовать группировку вида (?шаблон). Под результат такой группировки не выделяется отдельная область памяти и, соответственно, ей не назначается номер. Это положительно влияет на скорость выполнения выражения, но понижает удобочитаемость.

[править] Атомарная группировка

Атомарная группировка (вида (?>шаблон)), так же как группировка без обратной связи, не создаёт обратных связей. В отличие от неё, такая группировка запрещает возвращаться назад по строке, если часть шаблона уже найдена.

Пример	Соответствие	Создаваемые группы
a(bc b x)cc	abccaxcc abccaxcc	abccaxcc abccaxcc
a(?:bc b x)cc		
a(>bc b x)cc	abccaxcc но не abccaxcc: вариант x найден, остальные проигнорированы	нет
a(>x*)xa	не найдётся axxxa: все x заняты, и нет возврата внутрь группы	

Атомарная группировка выполняется ещё быстрее, чем группировка без обратной связи, и сохраняет процессорное время при выполнении остального выражения, так как запрещает проверку любых других вариантов внутри группы, когда один вариант уже найден. Это очень полезно при оптимизации групп со множеством различных вариантов.

[править] Модификаторы

Модификаторы действуют с момента вхождения и до конца регулярного выражения или противоположного модификатора. Некоторые интерпретаторы могут применить модификатор ко всему выражению, а не с момента его вхождения.

Синтаксис	Описание	
(?i)	Включает	нечувствительность выражения к регистру символов (англ. case insensitivity)
(?-i)	Выключает	
(?s)	Включает	режим соответствия точки символам переноса строки и возврата каретки
(?-s)	Выключает	
(?m)	Символы ^ и \$ вызывают соответствие только	после и до символов новой строки
(?-m)		с началом и концом строки
(?x)	Включает	режим без учёта пробелов между частями регулярного выражения и позволяет использовать # для комментариев
(?-x)	Выключает	

Группы-модификаторы можно объединять в одну группу: (?i-sm). Такая группа включает режим i и выключает режим s, m. Если использование модификаторов требуется только в пределах группы, то нужный шаблон указывается внутри группы после модификаторов и двоеточия. Например, (?-i)(?i:tv)set найдёт TVset, но не TVSET.

[править] КомментарииMay 12, 2018

Для добавления комментариев в регулярное выражение можно использовать группы-комментарии вида (?#комментарий). Такая группа интерпретатором полностью игнорируется и не проверяется на вхождение в текст. Например, выражение A(?#тут комментарий)Б соответствует строке АБ.

[править] Перечисление

Вертикальная черта разделяет допустимые варианты. Например, gray|grey соответствует gray или grey. Следует помнить, что перебор вариантов выполняется слева направо, как они указаны.

Если требуется указать перечень вариантов внутри более сложного регулярного выражения, то его нужно заключить в группу. Например, gray|grey или gr(a|e)y описывают строку gray или grey. В случае с односимвольными альтернативами предпочтителен вариант gr[ae]y, так как сравнение с символьным классом выполняется проще, чем обработка группы с проверкой на все её возможные модификаторы и генерацией обратной связи.

[править] Просмотр вперёд и назад

В большинстве реализаций регулярных выражений есть способ производить поиск фрагмента текста, «просматривая» (но не включая в найденное) окружающий текст, который расположен до или после искомого фрагмента текста. Например, таким способом легко найти имя тега HTML, не включая в результат поиска окружающие его угловые скобки или другие знаки, но и не упуская их «из внимания» при поиске нужного контекста. Просмотр с отрицанием используется реже и «следит» за тем, чтобы указанные соответствия, напротив, не встречались до или после искомого текстового фрагмента.

Представление	Вид просмотра	Пример	Соответствие
(?=шаблон)	Позитивный просмотр вперёд	Людиков(?=XVI)	ЛюдиковXV, ЛюдиковXVI, ЛюдиковXVIII, ЛюдиковLXVII, ЛюдиковXXL
(?!шаблон)	Негативный просмотр вперёд (с отрицанием)	Людиков(?!XVI)	ЛюдиковXV, ЛюдиковXVI, ЛюдиковXVIII, ЛюдиковLXVII, ЛюдиковXXL
(?<=шаблон)	Позитивный просмотр назад	(?<=Сергей )Иванов	Сергей Иванов, Игорь Иванов



(?!шаблон)      Негативный просмотр      (?<Сергей)Иванов      Сергей Иванов, Игорь Иванов  
назад (с отрицанием)

[править] Поиск по условию

Во многих реализациях регулярных выражений существует возможность выбирать, по какому пути пойдёт проверка в том или ином месте регулярного выражения на основании уже найденных значений.

Представление	Пояснение	Пример	Соответствие
(?(?=если)то  иначе)	Если операция просмотра успешна, то далее выполняется часть то, иначе выполняется часть иначе. В выражении может использоваться любая из четырёх операций просмотра. Следует учитывать, что операция просмотра нулевой ширины, поэтому части то в случае позитивного или иначе в случае негативного просмотра должны включать в себя описание шаблона из операции просмотра.	(?(?<=а)м п)	мам,пап
(?(n)то  иначе)	Если n-я группа вернула значение, то поиск по (а)?(?(1)м п) условию выполняется по шаблону то, иначе по шаблону иначе.		мам,пап

[править] Разновидности регулярных выражений

[править] Базовые регулярные выражения POSIX

(англ. basic regular expressions (BRE)). Традиционные регулярные выражения UNIX. Синтаксис базовых регулярных выражений 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 данную версию включены метасимволы:

- .
- [ ]
- [^ ]
- ^ (действует только в начале выражения)
- \$ (действует только в конце выражения)
- \*
- \{ \} — первоначальный вариант для { }
- \( \) — первоначальный вариант для ( )
- \n, где n — номер от 1 до 9

Особенности:

- Звёздочка должна следовать после выражения, соответствующего единичному символу. Пример: [xyz]\*.
- Выражение \ (блок)\* следует считать неправильным. В некоторых случаях оно соответствует нулю или более повторений строки блок. В других оно соответствует строке блок\*.
- Внутри символьного класса специальные значения символов, в основном, игнорируются. Особые случаи:
  - Чтобы добавить символ ^ в набор, его следует поместить туда не первым.
  - Чтобы добавить символ - в набор, его следует поместить туда первым или последним. Например:
    - шаблон DNS-имени, куда могут входить буквы, цифры, минус и точка-разделитель: [-0-9a-zA-Z.];
    - любой символ, кроме минуса и цифры: [^0-9].
  - Чтобы добавить символ [ или ] в набор, его следует поместить туда первым. Например:
    - [][ab] соответствует ], [, а или b.

[править] Расширенные регулярные выражения POSIX

(англ. extended regular expressions (ERE)). Синтаксис в основном аналогичен традиционному.

- Отменено использование обратной косой черты для метасимволов { } и ( ).
- Обратная косая черта перед метасимволом отменяет его специальное значение (см. Представление специальных символов).
- Отвергнута теоретически нерегулярная конструкция \n.
- Добавлены метасимволы +, ?, |.

См. также статью: Символьные классы POSIX.

[править] Регулярные выражения, совместимые с Perl

Основная статья: PCRE

Perl-совместимые регулярные выражения (англ. Perl-compatible regular expressions (PCRE)) имеют более богатый и в то же время предсказуемый[источник не указан 772 дня] синтаксис, чем даже POSIX ERE. По этой причине очень многие приложения используют именно Perl-совместимый синтаксис регулярных выражений.

[править] Нечёткие регулярные выражения

В некоторых случаях регулярные выражения удобно применить для анализа текстовых фрагментов на естественном языке, т.е. написанных людьми, и возможно содержащих опечатки, либо нестандартные варианты употребления слов. Например, если проводить опрос (допустим, на веб-сайте) "какой станцией метро вы пользуетесь", может оказаться что "Невский проспект" посетители могут указать как:

1. Невский
2. Несвк. просп.
3. Нев. порсект
4. наб. Конала Гребеедова ("Канал Грибоедова" это название второго выхода ст.м. Невский Проспект)

Здесь обычные регулярные выражения неприменимы, в первую очередь из-за того что входящие в образцы слова могут совпадать не очень точно (нечётко), но тем не менее было бы удобно описывать регулярными выражениями структурные зависимости между элементами образца, например, в нашем случае, указать что совпадение может быть с образцом "Невский проспект" ИЛИ "Канал Грибоедова", притом "проспект" может быть сокращено до "пр" или отсутствовать, а перед "Канал" может находиться сокращение "наб."

Эта задача сродни полнотекстовому поиску, отличаясь в том что здесь короткий фрагмент должен сравниваться с набором образцов, а при полнотекстовом поиске наоборот образец обычно один, в то время как фрагмент текста очень большой - или задаче Разрешения лексической многозначности, которая, однако, не позволяет задать структурирующие отношения между элементами образца.

Существует небольшое количество библиотек реализующих механизм регулярных выражений с возможностью нечёткого сравнения:

- TRE - бесплатная библиотека на C, использующая синтаксис регулярных выражений похожий на POSIX (стабильный проект);
- FRED - open-source библиотека на Java, использующая Lisp-образный синтаксис и лишённая многих возможностей обычных регулярных выражений, но сосредоточенная на различного рода автоматических заменах фрагментов текста (бета-версия).

[править] Реализации

- NFA (англ. nondeterministic finite-state automata — 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 (англ. deterministic finite-state automata — 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.


[править] Примечания

- ↑ [java.sun.com](http://java.sun.com)
- ↑ MSDN
- ↑ Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Синтаксический анализ. — Мир. — М.: 1978 Т. 2.
- ↑ Во многих книгах используются символы `[a-z]` или `[A-Z]` вместо `|`.
- ↑ Для использования последовательностей `\b` необходимо установить правильную кодовую страницу, в которой эти последовательности будут идти в порядке от и до указанных символов. Для русского языка это Windows-1251, ISO 8859-5 и Юникод, так как в DOS-855, DOS-866 и KOI8-R русские буквы не идут одной целой группой или не упорядочены по алфавиту. Отдельное внимание следует уделять буквам с диакритическими знаками, наподобие русских Ё/ё, которые, как правило, разбросаны вне основных диапазонов символов.

[править] Литература

- Фридл, Дж. Регулярные выражения. — СПб.: «Питер», 2001. — 352 с. — (Библиотека программиста). — ISBN 5-318-00056-8
- Смит, Билл. Методы и алгоритмы вычислений на строках (regex) = Computing Patterns in Strings. — М.: «Вильямс», 2006. — 496 с. — ISBN 0-201-39839-7
- Форта, Бен. Освой самостоятельно регулярные выражения. 10 минут на урок = Sams Teach Yourself Regular Expressions in 10 Minutes. — М.: «Вильямс», 2005. — 184 с. — ISBN 5-8459-0713-6
- Мельников С. В. Perl для профессиональных программистов. Регулярные выражения. — М.: «Бином», 2007. — 190 с. — (Основы информационных технологий). — ISBN 978-5-94774-797-3

[править] См. также

	Регулярные выражения в Викиучебнике?
	<div>lgijnlktbdfdd d;hnd;glbndb r;rgn;bprt</div>


Clearing download files

Directory created : C:\Users\Martian\Documents\tmssoftware\TMS Component Pack\cache

Tasks left :

Clearing download files

Directory created : C:\Users\Martian\Documents\tmssoftware\TMS Component Pack\cache

Tasks left :

Clearing download files

Directory created : C:\Users\Martian\Documents\tmssoftware\TMS Component Pack\cache

Tasks left :

- Шаблон поиска

[править] Ссылки

- Справочник и ресурсы по регулярным выражениям (англ.) — учебник и детальное описание синтаксиса с примерами, сравнение различных интерпретаторов и прочее
- MSDN — Знакомство с регулярными выражениями (рус.)
- Теория и методика построения регулярных выражений. Проблема самообразования (рус.)
- Конструктор регулярных выражений с подсветкой синтаксиса (рус.)
- Конструктор регулярных выражений с подсветкой синтаксиса (англ.)
- Perl регулярные выражения: статьи автора книги «Perl для профессиональных программистов. Регулярные выражения» (рус.)
- Видео урок по использованию регулярных выражений в AutoIt" (рус.)
- TRE - Бесплатная и переносимая библиотека для нечёткого сравнения с помощью регулярных выражений
- FREJ (Fuzzy Regular Expressions for Java) - Нечёткие регулярные выражения для Java на SourceForge.net

Источник — «[http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5\\_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F)»

Категории: Программирование | Формальные языки | Интерфейс командной строки | Сопоставление с образцом

Скрытые категории: Википедия:Нет источников с мая 2009 | Википедия:Статьи с утверждениями без источников более 14 дней | Статьи со ссылками на Викиучебник

Личные инструменты

- Представиться / зарегистрироваться

Пространства имён

- Статья
- Обсуждение

Варианты

Просмотры

- Чтение
- Текущая версия
- Правка
- История

Действия

Поиск

Навигация

- Заглавная страница
- Рубрикация
- Указатель А — Я
- Избранные статьи
- Случайная статья

- Текущие события

#### Участие

- Сообщить об ошибке
- Портал сообщества
- Форум
- Свежие правки
- Новые страницы
- Справка
- Пожертвования

#### Печать/экспорт



- Создать книгу
- Скачать как PDF
- Версия для печати

#### Инструменты

- Ссылки сюда
- Связанные правки
- Спецстраницы
- Постоянная ссылка
- Цитировать страницу

#### На других языках

- ڦيڤرغا
- Български
- Català
- Český
- Dansk
- Deutsch
- Ελληνικά
- English
- Esperanto
- Español
- Euskara
- عسرف
- Suomi
- Français
- Galego
- תירג
- □□□□□□
- Hrvatski
- Magyar
- Íslenska
- Italiano
- □□□
- □□□
- Македонски
- Nederlands
- Norsk (bokmål)
- Polski
- Português
- Română
- Simple English
- Slovenčina
- Српски / Srpski
- Svenska
- □□□□□
- ໂ໊໊໊
- Türkçe
- Українська

- [Tiếng Việt](#)
  - [□ □](#)
  - ☐ Последнее изменение этой страницы: 14:12, 29 июня 2011.
  - ☐ Текст доступен по лицензии [Creative Commons Attribution-ShareAlike](#); в отдельных случаях могут действовать дополнительные условия. Подробнее см. [Условия использования](#).
- [Wikipedia®](#) — зарегистрированная торговая марка Wikimedia Foundation, Inc., некоммерческой организации.
- ☐ [Свяжитесь с нами](#)
    - [Политика конфиденциальности](#)
    - [Описание Википедии](#)
    - [Отказ от ответственности](#)
- 
  - 
  -