5

Слова, строки и специальные символы

В этой главе содержатся рецепты, связанные с поиском и манипулированием текстом в различных контекстах. Одни рецепты демонстрируют, как реализовать расширенные функции поисковых механизмов, такие как поиск одного из нескольких слов или поиск слов, находящихся рядом друг с другом. Другие помогут вам отыскивать целые строки, содержащие определенные слова, удалять повторяющиеся слова или экранировать метасимволы регулярных выражений.

Основная цель этой главы состоит в том, чтобы продемонстрировать различные приемы и конструкции регулярных выражений в действии. Чтение этой главы сродни тренировке в применении большого числа особенностей регулярных выражений, и должно помочь вам наработать навыки применения регулярных выражений в решении стоящих перед вами задач. Во многих случаях мы ограничиваемся поиском простого текста, но предлагаемые нами шаблоны решений могут быть адаптированы вами под решение конкретных проблем.

5.1. Поиск определенного слова

Задача

Перед вами стоит простая задача – отыскать все вхождения слова «cat», без учета регистра символов. Выражение должно обнаруживать только те вхождения, которые являются отдельными словами. Оно не должно обнаруживать вхождения, являющиеся частью других слов, таких как `hellcat`, `application` или `Catwoman`.

Решение

Решить эту проблему помогут метасимволы границы слова:

```
\bcat\b
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

Как можно использовать это регулярное выражение для поиска всех совпадений, демонстрируется в рецепте 3.7. Как заменить найденные совпадения другим текстом, демонстрируется в рецепте 3.14.

Обсуждение

Метасимволы границы слова, находящиеся с обоих концов регулярного выражения, гарантируют, что совпадение с текстом `cat` будет обнаруживаться, только если он появляется только как самостоятельное слово. Точнее, границы слова требуют, чтобы фрагмент `cat` отделялся от остального текста началом или концом строки, пробельными символами, знаками пунктуации или другими символами, не являющимися символами слова.

Механизмы регулярных выражений интерпретируют буквы, цифры и символ подчеркивания, как символы слова. Более подробно границы слова обсуждаются в рецепте 2.6.

В диалектах JavaScript, PCRE и Ruby могут возникнуть проблемы с интернациональным текстом, так как в этих диалектах при определении границ слова в учет принимаются только символы кодировки ASCII. Другими словами, границы слова обнаруживаются только в позициях между совпадениями с выражениями `‹^|[^A-Za-z0-9_]›` и `‹[A-Za-z0-9_]›` или между совпадениями с выражениями `‹[A-Za-z0-9_]›` и `‹[^A-Za-z0-9_]|$›`. То же относится и к Python, если при создании регулярного выражения не был установлен флаг `UNICODE` или `U`. Эта особенность препятствует использованию метасимвола `‹\b›` для поиска «только целого слова» в тексте, содержащем символы из алфавитов, отличных от алфавита Latin. Например, в JavaScript, PCRE и Ruby регулярное выражение `‹\büber\b›` будет обнаруживать совпадение внутри слова `darüber`, и не будет обнаруживать внутри текста `dar über`. В большинстве случаев такой результат представляет собой полную противоположность желаемому. Проблема обусловлена тем, что символ `ü` интерпретируется не как символ слова, поэтому метасимвол границы слова обнаруживает совпадение между символами `rü`. В свою очередь, между пробелом и символом `ü` граница слова не обнаруживается, потому что они образуют непрерывную последовательность символов, не являющихся символами слова.

Решить эту проблему можно, применив вместо метасимвола границы слова опережающую и ретроспективную проверки (*проверки соседних символов*). Подобно метасимволу границы слова проверка соседних символов обеспечивает совпадение нулевой длины с некоторой позицией. В диалектах PCRE (когда эта библиотека скомпилирована с поддержкой UTF-8) и Ruby 1.9 имеется возможность имитировать поддержку границы слова в кодировке Юникод, например `‹(?‹=\P{L}|^)cat(?=\P{L}|$)›`. В этом регулярном выражении использованы инверти-

рованные метасимволы свойства Letter Юникода (‹\P{L}›), которые рассматривались в рецепте 2.7. Проверка соседних символов рассматривается в рецепте 2.16. Если потребуется, чтобы проверка соседних символов интерпретировала цифры и символ подчеркивания как символы слова (подобно метасимволу ‹\b›), достаточно будет заменить две конструкции ‹\P{L}› символьным классом ‹[^\p{L}\p{N}_]›.

В JavaScript и Ruby 1.8 не поддерживаются ни ретроспективная проверка, ни свойства Юникода. Обойти отсутствие поддержки ретроспективной проверки в этих диалектах можно сопоставлением с символом, не являющимся символом слова, перед каждым совпадением, а затем либо удалять его из каждого совпадения, либо возвращать его обратно в строку при выполнении операции замещения (примеры использования фрагментов совпадений в замещающем тексте приводятся в рецепте 3.15). Кроме того, отсутствие поддержки свойств Юникода (вместе с тем фактом, что в обоих языках программирования метасимволы ‹\w› и ‹\W› поддерживают только символы ASCII) означает, что придется обходиться менее универсальным решением. Кодовые пункты из категории Letter рассеяны по всему набору символов Юникода, поэтому потребовались бы тысячи символов, чтобы имитировать конструкцию ‹\p{L}› с помощью экранированных последовательностей Юникода и символьных классов. Неплохим компромиссом мог бы стать символьный класс ‹[A-Za-z\xAA\xB5\xBA\xC0-\xD6\xD8-\xF6\xF8-\xFF]›, которому соответствуют все буквы Юникода в восьмибитовом адресном пространстве, то есть первые 256 кодовых пунктов Юникода со значениями от 0x00 до 0xFF (список соответствующих символов приводится на рис. 5.1). Этот символьный класс будет совпадать со множеством (или, в инвертированной форме, исключать множество) наиболее часто используемых символов, находящихся за пределами адресного пространства семибитного набора ASCII.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
⋮																
4		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z					
6		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z					
⋮																
A											ª					
B						µ					º					
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý	þ	ÿ

Рис. 5.1. *Алфавитные символы Юникода в восьмибитном адресном пространстве*

Ниже демонстрируется, как можно заменить все вхождения слова «cat» словом «dog» в JavaScript. Этот пример корректно опознает все наиболее типичные акцентированные символы, поэтому слово ëcat останется без изменений. Для этого потребовалось сконструировать свой собственный символьный класс и отказаться от использования встроенных метасимволов ‹\b› и ‹\w›:

```
// 8-битные алфавитные символы
var L = 'A-Za-z\xAA\xB5\xBA\xC0-\xD6\xD8-\xF6\xF8-\xFF';
var pattern = '([^{L}]|^)cat([^{L}]|$)'.replace(/{L}/g, L);
var regex = new RegExp(pattern, 'gi');

// заменить cat на dog и вернуть в строку
// любые совпавшие дополнительные символы
subject = subject.replace(regex, '$1dog$2');
```

Примечательно, что для вставки специальных символов в строковый литерал в JavaScript используется форма записи \x*HH* (где *HH* – это двухзначное шестнадцатеричное число). Следовательно, переменная L, которая передается в регулярное выражение, будет содержать литеральные версии символов. Если необходимо, чтобы в регулярное выражение передавались сами метапоследовательности \x*HH*, их необходимо экранировать в строковом литерале с помощью символа обратного слэша (например, "\\x*HH*") . Однако в данном случае это не имеет большого значения и не окажет влияния на совпадение с регулярным выражением.

См. также

Рецепты 5.2, 5.3 и 5.4.

5.2. Поиск любого слова из множества

Задача

Необходимо отыскать любое слово из списка, не выполняя несколько операций поиска в испытуемом тексте.

Решение

С использованием конструкции выбора

Самое простое решение заключается в создании конструкции выбора из требуемых слов:

```
\b(?:one|two|three)\b
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

Более сложные примеры поиска похожих слов рассматриваются в рецепте 5.3.

Пример решения на языке JavaScript

```
var subject = 'One times two plus one equals three.';

var regex = /\b(?:one|two|three)\b/gi;

subject.match(regex);
// вернет массив с четырьмя совпадениями: ['One','two','one','three']

// Эта функция делает то же самое,
// но принимает искомые слова в виде массива.
// Любые метасимволы регулярных выражений,
// присутствующие внутри искомых слов,
// перед поиском экранируются символами обратного слэша.

function match_words (subject, words) {
    var regex_metachars = /[(){}[\]*+?.\\^$|,\-]/g;

    for (var i = 0; i < words.length; i++) {
        words[i] = words[i].replace(regex_metachars, '\\$&');
    }

    var regex = new RegExp('\\b(?:' + words.join('|') + ')\\b', 'gi');

    return subject.match(regex) || [];
}

match_words(subject, ['one','two','three']);
// вернет массив с четырьмя совпадениями: ['One','two','one','three']
```

Обсуждение

С использованием конструкции выбора

Это регулярное выражение состоит из трех частей: границы слова с обоих концов, несохраняющая группа и список слов (отделяются друг от друга оператором выбора ‹|›). Границы слова гарантируют, что регулярное выражение не совпадет с частью более длинного слова. Несохраняющая группировка ограничивает область действия операторов выбора – в противном случае для достижения того же эффекта регулярное выражение пришлось бы записать как ‹\bone\b|\btwo\b|\bthree\b›. Каждое из слов просто соответствует самому себе.

Так как механизм регулярных выражений пытается найти совпадение с каждым словом из списка, просматривая его слева направо, мож-

но заметить некоторое увеличение производительности, если в начало списка поместить слова, которые могут встречаться в испытуемом тексте с большей долей вероятности. Так как слова с обеих сторон окружены границами слова, они могут появляться в любом порядке. Однако, если бы в выражении отсутствовали границы слова, тогда важно было бы первыми указать наиболее длинные слова, в противном случае выражение `‹awe|awesome›` никогда не смогло бы найти совпадение со словом «awesome», так как оно всегда обнаруживало бы совпадение с «awe» в начале слова.

Обратите внимание, что это регулярное выражение предназначено лишь для того, чтобы в общих чертах продемонстрировать возможность совпадения с одним словом из списка.

Поскольку в данном примере оба слова `‹two›` и `‹three›` начинаются с одного и того же символа, есть возможность помочь механизму регулярных выражений, переписав регулярное выражение как `‹\b(?:one|t(?:wo|hree))\b›`. В рецепте 5.3 приводятся дополнительные примеры, демонстрирующие, как эффективнее выполнять поиск одного слова из списка похожих слов.

Пример решения на языке JavaScript

Пример на языке JavaScript сопоставляет тот же список слов двумя разными способами. В первом случае просто создается регулярное выражение и с помощью метода `match()`, имеющегося у строк в языке JavaScript, выполняется поиск в испытуемом тексте. При вызове методу `match()` передается регулярное выражение, использующее флаг `/g` («global» – глобальный). Он возвращает массив всех совпадений, обнаруженных в строке, или значение `null`, если не было найдено ни одного совпадения.

Во втором случае используется функция (`match_words()`), принимающая испытуемую строку, внутри которой требуется выполнить поиск, и массив искомых слов. Эта функция сначала экранирует любые метасимволы регулярных выражений, которые могут присутствовать в искомых словах, и затем компонует из списка слов новое регулярное выражение, используемое для поиска в строке.

Функция возвращает массив обнаруженных совпадений или пустой массив, если сгенерированным регулярным выражением не было найдено ни одного совпадения. Благодаря применению флага нечувствительности к регистру символов (`/i`) искомые слова могут состоять из символов верхнего и нижнего регистра в любой комбинации.

См. также

Рецепты 5.1, 5.3 и 5.4.

5.3. Поиск похожих слов

Задача

В данном случае необходимо решить несколько задач:

- Отыскать все вхождения обоих слов `color` и `colour` в строке.
- Отыскать любое из трех слов, оканчивающееся на `bat`, `cat` или `rat`.
- Отыскать любое слово, оканчивающееся на `phobia`.
- Отыскать наиболее распространенные варианты записи имени «Steven»: `Steve`, `Steven` и `Stephen`.
- Отыскать совпадение с любой формой термина «regular expression».

Решение

Ниже приводятся регулярные выражения, решающие поставленные задачи в порядке их следования. Во всех решениях используется режим нечувствительности к регистру символов.

Color или colour

```
\bcolou?r\b
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

Bat, cat или rat

```
\b[bcr]at\b
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

Слова, заканчивающиеся на «phobia»

```
\b\w*phobia\b
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

Steve, Steven или Stephen

```
\bSte(?:ven?|phen)\b
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

Варианты написания термина «regular expression»

```
\breg(?:ular•expressions?|ex(?:ps?|e[sn])?)\b
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

Обсуждение

Использование границ слова для обеспечения совпадения с целыми словами

Для обеспечения совпадения с целыми словами во всех пяти регулярных выражениях используются метасимволы границы слова (‹\b›) . Во всех шаблонах используются разные способы сопоставления с вариантами слов, которым они соответствуют.

Рассмотрим каждое из решений подробнее.

Color или Colour

Это регулярное выражение будет совпадать со словом color или colour, но не будет совпадать со словом colorblind. В нем используется квантификатор ‹?›, чтобы сделать необязательным совпадение с предшествующим ему символом «u». Квантификаторы, такие как ‹?›, действуют не как шаблонные символы, знакомые многим. Они непосредственно связаны с предшествующими им элементами, которые могут представлять собой либо одиночный элемент (как литерал символа «u» в данном случае), либо группу элементов, заключенных в круглые скобки. Квантификатор ‹?› повторяет предшествующий ему элемент ноль или один раз. Механизм регулярных выражений сначала пытается сопоставить элемент, с которым связан квантификатор, и если эта попытка оказывается неудачной, механизм продолжает движение вперед, без сопоставления элемента. Любой квантификатор, допускающий нулевое число повторений, фактически делает необязательным совпадение с предшествующим ему элементом, и это как раз то, что требуется в данном случае.

Bat, cat или rat

В этом регулярном выражении для совпадения с символом «b», «c» или «r» используется символьный класс, за которым следуют литералы символов «at». То же самое можно было бы сделать с помощью выражений ‹\b(?:b|c|r)at\b›, ‹\b(?:bat|cat|rat)\b› или ‹\bbat\b|\bcat\b|\brat\b›. Однако всякий раз, когда выбор между возможными совпадениями заключается в выборе единственного символа из списка, лучше использовать символьные классы. Мало того, что символьные классы обеспечивают более компактную и удобочитаемую форму записи (благодаря отсутствию операторов выбора и возможности использовать диапазоны, такие как A–Z), большинство механизмов регулярных выражений также обеспечивают превосходную оптимизацию обработки символьных классов. Операторы выбора требуют от механизма регулярных выраже-

ний использовать дорогостоящий, в смысле вычислительных ресурсов, алгоритм возвратов, тогда как для сопоставления с символьными классами задействуется более простой метод поиска.

Тем не менее, несколько слов предостережения. Символьные классы относятся к элементам регулярных выражений, которые наиболее часто применяются неверно. Возможно, проблема в недостаточно подробной документации или, возможно, в недостаточно внимательном ознакомлении с нею. Как бы то ни было, не повторяйте ошибок, характерных для начинающих. Символьные классы способны обеспечить совпадение лишь с одним из символов, перечисленных в них, – и только это.

Ниже приводятся два наиболее типичных примера неправильного использования символьных классов:

Помещение слов в символьные классы

Вне всяких сомнений, выражение `<[cat]{3}>` будет обнаруживать совпадение со словом `cat`, но оно также будет совпадать со словами `act`, `ttt` и любыми другими комбинациями из перечисленных символов. То же относится и к инвертированным классам, таким как `<[^cat]>`, которому соответствует любой символ, за исключением `c`, `a` и `t`.

Попытка использовать оператор выбора в символьном классе

Символьный класс по определению обеспечивает выбор между символами, перечисленными в нем. Конструкции `<[a|b|c]>` соответствует один символ из множества «abc|». Что, вполне возможно, совсем не то, что вы подразумевали. Но даже если это не так, данный символьный класс содержит лишние символы вертикальной черты.

Все подробности, которые могут пригодиться для правильного и эффективного использования символьных классов, приводятся в рецепте 2.3.

Слова, заканчивающиеся на «phobia»

Как и в предыдущем регулярном выражении, здесь также используется квантификатор, позволяющий обнаруживать совпадения с разными вариантами слов в строке. Например, регулярному выражению соответствуют такие слова, как `arachnophobia` и `hexakosioihexekontahexaphobia`, а так как квантификатор `‹*›` допускает нулевое число повторений, ему также будет соответствовать само слово `phobia`. Если необходимо, чтобы перед окончанием «phobia» присутствовал хотя бы один символ, следует заменить квантификатор `‹*›` на `‹+›`.

Steve, Steven или Stephen

Данное регулярное выражение включает в себя пару особенностей, использованных в предыдущих примерах. Несохраняющая группировка, записываемая как `‹(?:…)›`, ограничивает область действия оператора выбора `‹|›`. Квантификатор `‹?›`, применяемый к первой альтернативе

внутри группы, обеспечивает необязательность предшествующего ему символа ‹n›. Это увеличивает эффективность (и краткость) выражения по сравнению с эквивалентным выражением ‹\bSte(?:ve|ven|phen)\b›. Те же соображения эффективности заставляют поместить строковый литерал ‹Ste› в начало регулярного выражения, а не повторять его трижды, как в выражениях ‹\b(?:Steve|Steven|Stephen)\b› или ‹\bSteve\b|\bSteven\b|\bStephen\b›. Некоторые механизмы возвратов недостаточно интеллектуальны, чтобы заметить, что любой текст, соответствующий этим двум последним выражениям, должен начинаться с последовательности символов Ste. По мере продвижения механизма по испытуемой строке в поисках совпадения он сначала попытается отыскать границу слова и затем убедиться, что следующий символ – это символ S. Если совпадение не найдено, механизм вынужден будет опробовать все возможные альтернативы, присутствующие в регулярном выражении, прежде чем он сможет переместиться к следующей позиции в строке и повторить попытку. Если человек легко может заметить, что это будет напрасной тратой сил (так как все альтернативы в регулярном выражении начинаются с последовательности «Ste»), то механизм даже не подозревает об этом. Если же выражение записано как ‹\bSte(?:ven?|phen)\b›, механизм тут же сообразит, что он не сможет отыскать соответствие в строке, которая не начинается с указанных символов.

Более подробно о работе механизма возвратов рассказывается в рецепте 2.13.

Варианты написания термина «regular expression»

Последний пример в этом рецепте соединяет в себе конструкцию выбора, символьные классы и квантификаторы, чтобы отыскать совпадение со всеми типичными вариантами записи термина «regular expression». Поскольку на первый взгляд это регулярное выражение может показаться немного сложным, разобьем его на составляющие и рассмотрим их по отдельности.

Регулярное выражение, которое приводится ниже, записано в режиме свободного форматирования, который не поддерживается диалектом JavaScript. Поскольку в режиме свободного форматирования пробелы игнорируются, литералы пробелов были экранированы символами обратного слэша:

```
\b              # Проверка совпадения с границей слова.
reg             # Соответствует “reg”.
(?:             # Группировка, но несохраняющая...
  ular\         #   Соответствует “ular “.
  expressions?  #   Соответствует “expression” или “expressions”.
 |              #  или...
  ex            #   Соответствует “ex”.
  (?:           #   Группировка, но несохраняющая...
```

```
    ps?         #     Соответствует “p” или “ps”.
   |            #    или...
    e           #     Соответствует “e”.
    [sn]        #     Соответствует одному из символов из множества “sn”.
  )             #   Конец несохраняющей группы.
    ?           #     Повторить предшествующую группу ноль или один раз.
)               # Конец несохраняющей группы.
\b              # Проверка совпадения с границей слова.
```

Параметры: режим свободного форматирования, нечувствительность к регистру символов
Диалекты: .NET, Java, PCRE, Perl, Python, Ruby

Данный шаблон совпадет с любой из следующих семи строк:

- regular expressions
- regular expression
- regexps
- regexp
- regexes
- regexen
- regex

См. также

Рецепты 5.1, 5.2 и 5.4.

5.4. Поиск любых слов, за исключением некоторых

Задача

Необходимо с помощью регулярного выражения отыскать совпадения с любыми полными словами, за исключением слова `cat`. Слово `Catwoman` и другие подобные слова, которые просто содержат в себе символы «cat», должны совпадать, а слово `cat` – нет.

Решение

Исключить совпадения с нежелательными словами можно с помощью негативной опережающей проверки, которая является ключевым элементом следующего регулярного выражения:

```
\b(?!cat\b)\w+
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

Обсуждение

Несмотря на то, что с помощью инвертированного символьного класса (записывается как ‹[^···]›) легко можно исключить возможность совпадения с определенными символами, тем не менее, нельзя просто записать ‹[^cat]›, и надеяться, что будут исключены совпадения со словом cat. ‹[^cat]› – это допустимое регулярное выражение, но ему соответствует любой символ, за исключением символов c, a и t. Но, хотя выражение ‹\b[^cat]+\b› могло бы помочь исключить совпадения со словом cat, оно точно так же исключило бы совпадения со словом cup, потому что в этом слове содержится запрещенный символ c. Регулярное выражение ‹\b[^c][^a][^t]\w*› ничуть не лучше, потому что оно будет отвергать любые слова, в которых первым следует символ c, вторым – a или третьим – t. Кроме того, это регулярное выражение не накладывает ограничений на первые три символа слова – оно лишь совпадает со словами, содержащими, по крайней мере, три символа, поскольку ни один из инвертированных символьных классов не является необязательным.

Учитывая все вышесказанное, рассмотрим, как регулярное выражение, представленное в начале этого рецепта, позволяет решить поставленную задачу:

```
\b    # Проверка соответствия границе слова.
(?!   # Проверка, что регулярное выражение, следующее ниже, не найдет
      # совпадения, начиная с этой позиции...
  cat #   Соответствие "cat".
  \b  #   Проверка соответствия границе слова.
)     # Конец негативной опережающей проверки.
\w+   # Соответствует одному или более символу слова.
```

Параметры: режим свободного форматирования, нечувствительность к регистру символов

Диалекты: .NET, Java, PCRE, Perl, Python, Ruby

Ключевым элементом этого шаблона является негативная опережающая проверка, которая имеет синтаксис ‹(?!···)›. Негативная опережающая проверка отклоняет возможность совпадения с последовательностью символов cat, которая следует за границей слова, и не запрещает регулярному выражению совпадать с этими же символами, если они не следуют именно в таком порядке, или когда они составляют часть более длинного или более короткого слова. В конце регулярного выражения отсутствует проверка совпадения с границей слова, потому что ее присутствие не повлияет на возможность совпадения с регулярным выражением. Квантификатор ‹+› в подвыражении ‹\w+› повторит метасимвол символа слова максимально возможное число раз, то есть он всегда будет обнаруживать совпадение до ближайшей границы слова.

Если применить это регулярное выражение к испытуемому тексту categorically match any word except cat, оно обнаружит пять совпадений: categorically, match, any, word и except.

Варианты

Поиск слов, не содержащих других слов

Если вместо того, чтобы пытаться отыскать совпадения с любыми словами, кроме слова `cat`, необходимо отыскать любые слова, которые *не содержат* в себе слово `cat`, требуется немного иной подход:

```
\b(?:(?!cat)\w)+\b
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

В предыдущем разделе этого рецепта метасимвол границы слова в начале регулярного выражения обеспечивал удобную привязку, которая позволила просто выполнить негативную опережающую проверку в начале слова.

Решение, используемое здесь, не столь эффективно; тем не менее, эта конструкция часто используется, чтобы обеспечить возможность совпадения с любым словом, за исключением определенного слова или шаблона. Это достигается за счет повторения группы, содержащей негативную опережающую проверку и единственный метасимвол, обозначающий символ слова. Перед сопоставлением с каждым символом механизм регулярных выражений проверяет отсутствие совпадения со словом `cat`, начиная с текущей позиции.

В отличие от предыдущего регулярного выражения, в данном выражении требуется присутствие метасимвола границы слова. В противном случае оно могло бы совпасть с первой частью слова перед вхождением в него слова `cat`.

См. также

Рецепт 2.16, где приводится более полное обсуждение проверок соседних символов (позитивных и негативных, опережающих и ретроспективных проверок).

Рецепты 5.1, 5.5, 5.6 и 5.11.

5.5. Поиск любого слова, за которым не следует указанное слово

Задача

Необходимо отыскать совпадение с любым словом, непосредственно за которым не следует слово `cat`, игнорируя присутствующие между ними любые пробельные символы, знаки пунктуации и другие символы, не являющиеся символами слова.

Решение

Секретным ингредиентом этого регулярного выражения является негативная опережающая проверка:

```
\b\w+\b(?!\W+cat\b)
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

В рецептах 3.7 и 3.14 показано, как это регулярное выражение можно реализовать в программном коде.

Обсуждение

Как и во многих других рецептах этой главы, совпадение с целым словом обеспечивается совместной работой метасимволов границы слова (‹\b›) и символа слова (‹\w›). Более подробное описание этих особенностей можно найти в рецепте 2.6.

Вторая часть этого выражения заключена в конструкцию ‹(?!…)›, которая представляет собой негативную опережающую проверку. Опережающая проверка предписывает механизму регулярных выражений временно пройти вперед по строке, чтобы проверить возможность совпадения текста, находящегося непосредственно за текущей позицией, с шаблоном, находящимся внутри опережающей проверки. При этом любые символы, совпавшие с выражением внутри опережающей проверки, не поглощаются – проверка просто убеждается, что совпадение возможно. Поскольку здесь используется негативная опережающая проверка, результат проверки инвертируется.

Другими словами, если шаблон внутри опережающей проверки может обнаружить совпадение непосредственно за текущей позицией, попытка сопоставления считается неудачной и механизм регулярных выражений продвигается на одну позицию вперед, чтобы предпринять новую попытку сопоставления со всем регулярным выражением, начиная со следующего символа в испытуемой строке. Более подробное обсуждение опережающей (и ретроспективной) проверки можно найти в рецепте 2.16.

Что касается шаблона внутри опережающей проверки: конструкции ‹\W+› соответствует один или более символов, не являющихся символами слова, которые находятся перед словом ‹cat›, а завершающий метасимвол границы слова гарантирует, что совпадение будет обнаружено только со словами, за которыми не следует последовательность символов cat, в виде самостоятельного слова, но допускается, что следующее слово начинается с символов cat.

Следует заметить, что это регулярное выражение будет совпадать даже со словом cat, при условии, что за ним не следует второе слово cat. Если потребуется избежать совпадения со словом cat, следует объединить

это регулярное выражение с регулярным выражением из рецепта 5.4, что в результате даст выражение `‹\b(?!cat\b)\w+\b(?!\W+cat\b)›`.

Варианты

Если необходимо обеспечить совпадение только со словами, за которыми следует слово `cat` (не включая в совпадение само слово `cat` и предшествующие ему символы, не являющиеся символами слова), замените негативную опережающую проверку на позитивную и радуйтесь жизни:

```
\b\w+\b(?=\W+cat\b)
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, JavaScript, PCRE, Perl, Python, Ruby

См. также

Рецепт 2.16, где приводится более полное обсуждение проверок соседних символов (позитивных и негативных, опережающих и ретроспективных проверок).

Рецепты 5.4 и 5.6.

5.6. Поиск любого слова, которому не предшествует определенное слово

Задача

Необходимо отыскать любое слово, которое не следует непосредственно за словом `cat`, игнорируя присутствующие между ними любые символы, не являющиеся символами слова.

Решение

Взгляд назад

Ретроспективная проверка позволяет проверить наличие некоторого текста непосредственно перед текущей позицией. Она предписывает механизму регулярных выражений временно вернуться назад по строке, чтобы проверить возможность совпадения текста, оканчивающегося в позиции, где находится ретроспективная проверка. Более подробное обсуждение ретроспективной проверки можно найти в рецепте 2.16.

Следующие три регулярных выражения используют негативную ретроспективную проверку, которая записывается как `‹(?‹!···)›`. К сожалению, диалекты регулярных выражений, рассматриваемые в этой книге, отличаются допустимыми шаблонами, которые можно помещать внутрь ретроспективной проверки. В результате получившиеся

решения работают немного по-разному. Дополнительные подробности можно найти в разделе «Пояснения» на стр. 11, в этом рецепте.

Слова, не предшествующие слову «cat»

```
(?<!\bcat\W+)\b\w+
```

Параметры: нечувствительность к регистру символов
Диалект: .NET

```
(?<!\bcat\W{1,9})\b\w+
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, PCRE

```
(?<!\bcat)(?:\W+|^)(\w+)
```

Параметры: нечувствительность к регистру символов
Диалекты: .NET, Java, PCRE, Perl, Python, Ruby 1.9

Имитация ретроспективной проверки

Диалекты JavaScript и Ruby 1.8 вообще не поддерживают ретроспективную проверку, хотя и предоставляют возможность опережающей проверки. Однако, благодаря тому, что в данной задаче ретроспективная проверка находится в самом начале регулярного выражения, имеется отличная возможность имитировать ретроспективную проверку, разбив регулярное выражение на две части, как показано в следующем примере на JavaScript:

```
var subject = 'My cat is furry.',
    main_regex = /\b\w+/g,
    lookbehind = /\bcat\W+$/i,
    lookbehind_type = false, // негативная ретроспективная проверка
    matches = [],
    match,
    left_context;

while (match = main_regex.exec(subject)) {
    left_context = subject.substring(0, match.index);

    if (lookbehind_type == lookbehind.test(left_context)) {
        matches.push(match[0]);
    } else {
        main_regex.lastIndex = match.index + 1;
    }
}

// совпадения: ['My','cat','furry']
```