ГЛАВА 4

Альтернативы, группы и обратные ссылки

Вы уже видели, как работают группы. Создание группы путем заключения текста в круглые скобки упрощает выполнение ряда операций, перечисленных ниже:

- чередование, т.е. выбор одного из нескольких возможных шаблонов;
- создание подшаблонов;
- захват (запоминание) групп для последующих обращений к ним с помощью обратных ссылок;
- применение операций к групповому шаблону, например квантификатору;
- использование групп без функции захвата;
- атомарные группы (дополнительная возможность).

В примерах этой главы наряду с полным текстом поэмы "The Rime of the Ancient Mariner" (файл *rime.txt*) используется ряд дополнительных текстов. Нашим основным инструментов будет настольная версия приложения RegExr, написанная с использованием технологии Adobe AIR (об установке приложения см. в главе 2), однако будут привлекаться и другие средства, такие как редактор *sed*.

Чередование

Термин *чередование* (alteration) означает возможность выбора альтернативных вариантов (альтернатив) шаблона при поиске совпадений. Предположим, требуется определить, сколько раз артикль *the* встречается в тексте поэмы "The Rime of the Ancient Mariner". Проблема заключается в том, что в поэме артикль может встречаться в различных формах: *THE*, *The* и *the*. Альтернативы позволяют справиться с этой проблемой.

Откройте настольное приложение RegExr, дважды щелкнув на его значке, и скопируйте в него текст поэмы из файла *rime.txt*, находящегося в архиве примеров.

Введите в верхнем текстовом поле такой шаблон:

```
(the|The|THE)
```

и вы увидите, как в расположенном под ним поле с текстом поэмы выделятся все вхождения артикля *the* (рис. 4.1). Для просмотра скрытой части текста воспользуйтесь полосой прокрутки.

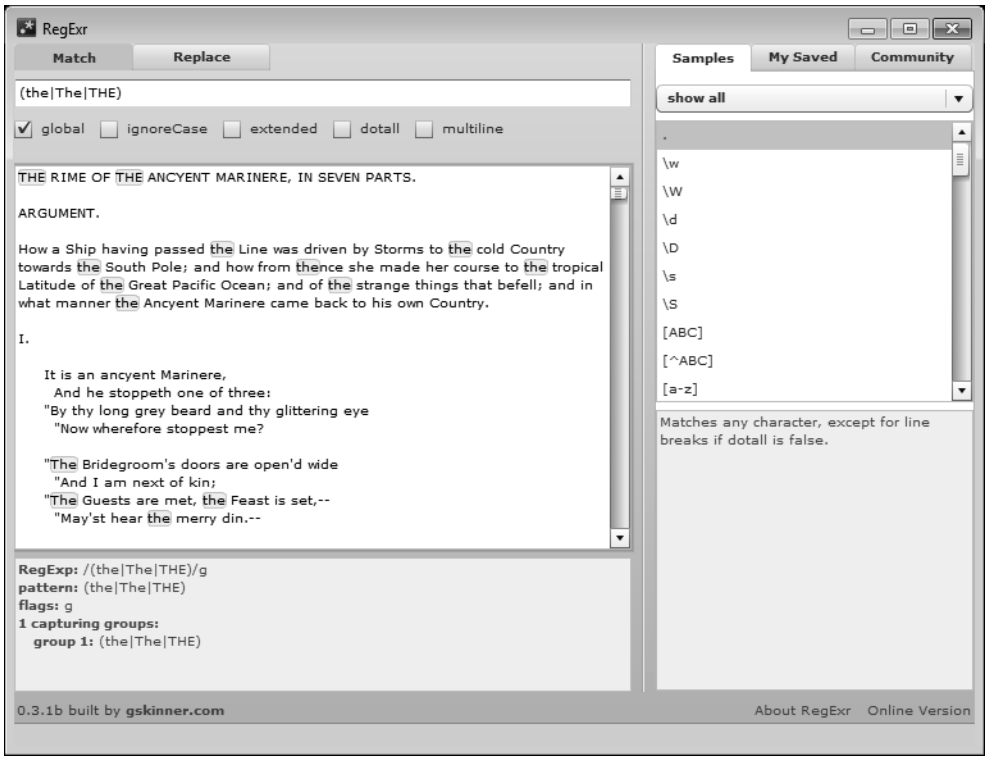

Рис. 4.1. Использование чередования шаблонов

Применив опции, эту группу можно записать в более компактном виде. Опции позволяют конкретизировать способ поиска совпадений с шаблоном в тексте. Например, следующая опция:

```
(?i)
```

делает шаблон нечувствительным к регистру, так что вместо прежнего шаблона, включающего набор альтернатив, можно использовать шаблон следующего вида:

```
(?i)the
```

Введите его в RegExr и сравните полученный результат с предыдущим. Вместо того чтобы изменять шаблон, можно установить флажок ignoreCase (игнорировать регистр). Результат будет тем же. Перечень доступных опций (модификаторов) приведен в табл. 4.1.

Таблица 4.1. Опции, используемые в регулярных выражениях

Опция	Описание	Поддержка
`(?d)`	Строки Unix	Java
`(?i)`	Игнорировать регистр символов	PCRE, Perl, Java
`(?J)`	Разрешить одинаковые имена подшаблонов	PCRE*
`(?m)`	Обрабатывать текст как многострочный	PCRE, Perl, Java

Окончание табл. 4.1

Опция	Описание	Поддержка
`(?s)`	Обрабатывать текст как одну строку	PCRE, Perl, Java
`(?u)`	Обрабатывать шаблоны как строки Unicode	Java
`(?U)`	Делает модификаторы “нежадными” по умолчанию	PCRE
`(?x)`	Игнорировать пробельные символы и комментарии	PCRE, Perl, Java
`(?-...)`	Сброс или отключение опций	PCRE

[*] См. раздел “Named Subpatterns” на странице `http://www.pcre.org/pcre.txt`.

Далее мы рассмотрим применение альтернатив в *grep*. Кстати, опции, приведенные в табл. 4.1, в *grep* не работают, поэтому мы будем использовать исходный шаблон, содержащий перечисление альтернатив. Для подсчета количества строк, в которых встречается артикль *the*, причем независимо от регистра символов и того, сколько именно раз шаблон встречается в строке, используйте такую команду:

```
grep -Ec "(the|The|THE)" rime.txt
```

что должно привести к следующему результату:

```
327
```

Однако это еще не вся история, поэтому не расслабляйтесь.
Ниже приведен подробный анализ того, как работает данная команда.

- Опция `-E` означает, что вы хотите использовать расширенные регулярные выражения (ERE), а не базовые (BRE). Это позволяет избавиться от необходимости экранировать скобки и вертикальную черту (`\(THE\|The\|the\)`), что надо было бы сделать в случае использования BRE.
- Опция `-c` указывает на необходимость вывода количества строк, в которых обнаружены совпадения (а не собственно количества совпадений).
- Скобки объединяют варианты выбора, или альтернативы, заданные в виде *the*, *The* и *THE*, в одну группу.
- Символ вертикальной черты разделяет альтернативы, обработка которых осуществляется слева направо.

Чтобы получить фактическое количество вхождений артикля в тексте поэмы, необходимо использовать следующую команду:

```
grep -Eo "(the|The|THE)" rime.txt | wc -l
```

возвращающую каждое совпадение в виде отдельной строки, что приводит к следующему результату:

```
412
```

Проанализируем эту команду.

- Опция `-o` указывает на то, что отображать необходимо лишь ту часть строки, которая совпадает с шаблоном, хотя это и не очевидно по той причине, что канал (`|`) перенаправляет вывод команде `wc`.
- В данном контексте вывод команды `grep` перенаправляется в поток ввода команды `wc`. Команда `wc` — это команда подсчета слов, опция `-l` которой задает подсчет количества входных строк.

Откуда взялась столь большая разница в значениях: 327 и 412? Это произошло потому, что опция `-c` задает лишь подсчет строк, в которых встречаются совпадения с шаблоном, но ведь в одной строке может встретиться несколько совпадений. Если в команде `wc -l` дополнительно использовать опцию `-o`, то каждое вхождение искомого слова в любой из его форм будет появляться на отдельной строке и учитываться при подсчете, что и приводит к получению большего значения.

Выполним аналогичный поиск совпадений с помощью Perl, используя следующую команду:

```
perl -ne 'print if /(the|The|THE)/' rime.txt
```

Эту команду можно оптимизировать за счет применения опции `(?i)`, делающей ненужным использование списка альтернатив:

```
perl -ne 'print if /(?i)the/' rime.txt
```

Но и последнюю команду можно дополнительно улучшить, добавив модификатор `i` вслед за последним разделителем шаблона:

```
perl -ne 'print if /the/i' rime.txt
```

Результат останется тем же. Однако чем проще, тем лучше. Список дополнительных *модификаторов* (называемых также *флагами*) приведен в табл. 4.2. Одновременно у вас появляется возможность сравнить (разумеется, с учетом различий в синтаксисе) эти модификаторы с опциями, приведенными в табл. 4.1.

Таблица 4.2. Модификаторы (флаги) Perl*

Модификатор	Описание
`a`	Поиск соответствий для сокращений \d, \s, \w и классов POSIX только в диапазоне символов ASCII
`c`	Не сбрасывать текущую позицию поиска при неудачном исходе сопоставления
`d`	Использовать собственные правила платформы, заданные по умолчанию
`g`	Глобальное сопоставление, т.е. поиск всех вхождений шаблона
`i`	Игнорировать регистр при сопоставлении
`l`	Использовать правила текущей локали
`m`	Обрабатывать исходный текст как многострочный
`p`	Сохранять строку, которая совпала
`s`	Обрабатывать исходный текст как единую строку

Окончание табл. 4.2

Модификатор	Описание
u	Использовать правила Unicode при сопоставлении
x	Игнорировать пробельные символы и комментарии

[*] См. http://perldoc.perl.org/perlre.html#Modifiers

Подшаблоны

Когда говорят о *подшаблонах* в регулярных выражениях, то под этим термином чаще всего подразумевают группу или группы, входящие в другую группу. Подшаблон — это шаблон в шаблоне. Часто, хотя и не всегда, совпадение с подшаблоном проверяется лишь в том случае, если найдено совпадение для предшествующего ему шаблона. Подшаблоны можно конструировать множеством способов, но нас интересуют в первую очередь те из них, которые определяются с помощью круглых скобок.

В некотором смысле вы уже познакомились с подшаблонами, когда работали со следующим шаблоном:

```
(the|The|THE)
```

Здесь мы имеем дело с тремя подшаблонами. Первый из них — *the*, второй — *The*, третий — *THE*, но в данном случае поиск совпадений для второго подшаблона осуществляется независимо от поиска совпадений для первого подшаблона. (Поиск совпадений начинается с крайнего слева шаблона.)

А вот пример, в котором работа одного подшаблона (подшаблонов) зависит от результатов работы предыдущего:

```
(t|T)h(eir|e)
```

Этому выражению будут соответствовать подстроки, начинающиеся с одной из букв `t` или `T`, за которой следует буква `h`, за которой, в свою очередь, следует либо последовательность букв `eir`, либо буква `e`. Таким образом, данный шаблон совпадет с любым из следующих слов:

- *the*
- *The*
- *their*
- *Their*

В данном случае второй подшаблон — `(e|eir)` — зависит от первого — `(t|T)`.

Использовать круглые скобки в подшаблонах необязательно. Вот пример определения подшаблонов с помощью классов:

```
\b[tT]h[ceinry]*\b
```

Кроме слов *the* и *The* данному шаблону будут соответствовать также слова *thee*, *thy*, *thence* и много других. Указание двух границ слов (`\b`) означает, что шаблону будут соответствовать только целые слова, а не подстроки, входящие в состав других слов.

Проанализируем, как работает этот шаблон.

- Метасимволу `\b` соответствует начало слова.
- Выражение `[tT]` — это символьный класс, которому соответствует либо буква `t` нижнего регистра, либо буква `T` верхнего регистра.
- Далее шаблон находит (или пытается найти) букву `h` нижнего регистра.
- Второй (и последний) подшаблон также записан в виде символьного класса `[ceinry]` с последующим квантификатором, которому соответствует нуль или несколько символов.
- Наконец, шаблон оканчивается еще одной границей слова `\b`.

Хочу обратить ваше внимание на одну интересную особенность, характеризующую нынешнее состояние дел в области регулярных выражений. Будучи, как правило, строгой, терминология регулярных выражений в некоторых случаях довольно размыта. Пытаясь дать определения *подшаблона* и некоторых других терминов, используемых в данной книге, я просмотрел множество источников, стремясь привести их к общему знаменателю. Подозреваю, кое-кто может утверждать, что символьный класс нельзя причислять к подшаблонам. Но, поскольку символьные классы могут функционировать как шаблоны, я считаю, что для применения к ним термина "подшаблон" есть все основания.

Захватывающие группы и обратные ссылки

Если весь шаблон или некоторая часть его содержимого заключается в круглые скобки, образуя группу, то содержимое этой группы захватывается и временно сохраняется в памяти. Впоследствии на сохраненное содержимое можно ссылаться с помощью обратных ссылок вида

```
\1
```

или

```
$1
```

где переменные `\1` или `$1` ссылаются на первую захваченную группу, `\2` или `$2` — на вторую и т.д. Редактор *sed* воспринимает лишь ссылки вида `\1`, тогда как Perl воспринимает ссылки обоих типов.

Первоначально в редакторе *sed* разрешалось использовать только ссылки в диапазоне от `\1` до `\9`, но в настоящее время это ограничение, по всей видимости, снято.

С подобным вы уже встречались в предыдущих главах, однако я все же приведу соответствующий пример. Его суть заключается в перестановке слов в одной из строк поэмы, за что я заранее приношу свои извинения ее автору Сэмюэлу Тейлору Кольриджу. Щелкнув в окне приложения RegExr на вкладке Replace, введите в верхнем текстовом поле следующий шаблон:

```
(It is) (an ancyent Marinere)
```

Прокрутите обрабатываемый текст (верхняя текстовая область) вниз, пока не увидите подсвеченную строку, и введите во втором текстовом поле следующий текст:

```
$2 $1
```

В результате в этой строке слова окажутся переставленными (рис. 4.2):

```
an ancyent Marinere It is,
```

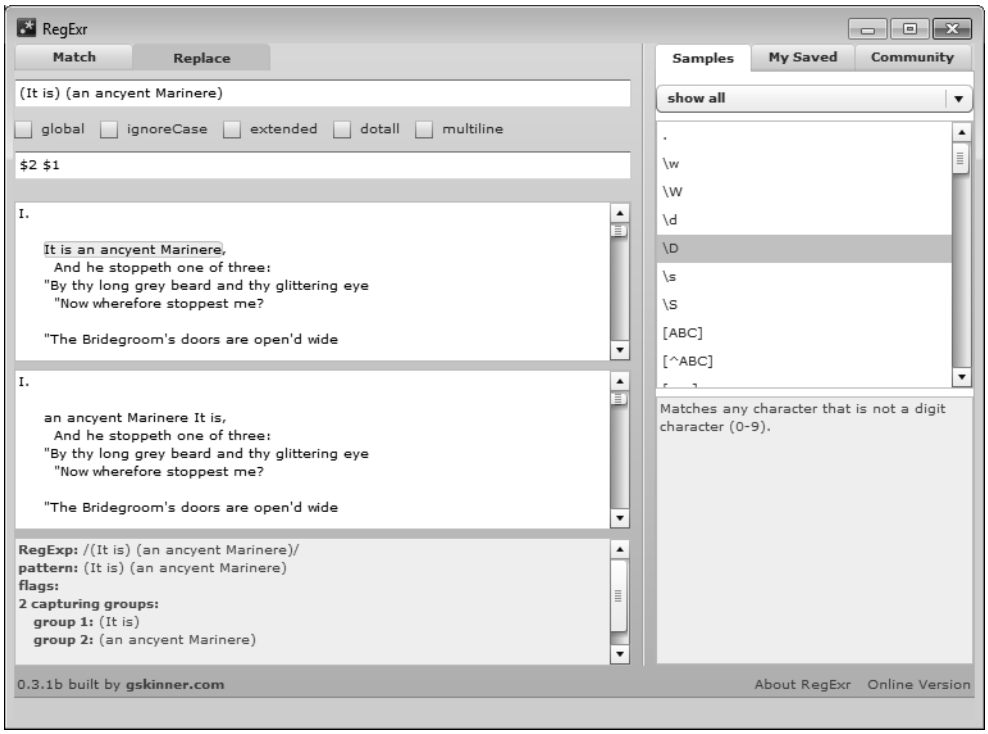

Рис. 4.2. Использование обратных ссылок `$1` и `$2`

Для получения того же результата с помощью редактора *sed* необходимо выполнить следующую команду:

```
sed -En 's/(It is) (an ancyent Marinere)/\2 \1/p' rime.txt
```

Получаемый при этом вывод

```
an ancyent Marinere It is,
```

полностью согласуется с результатом, полученным с помощью приложения RegExr. Ниже приведен подробный анализ работы данной команды.

- Опция `-E` означает, что вы хотите использовать расширенные регулярные выражения (ERE), что, в частности, позволяет избавиться от экранирования скобок.
- Опция `-n` подавляет заданный по умолчанию вывод каждой строки.
- Команда подстановки выполняет поиск совпадения с текстом “It is an ancyent Marinere”, захватывая его в две группы.

- Та же команда подстановки направляет в вывод измененную строку, полученную путем перестановки местами частей исходной строки, доступных по обратным ссылкам `\1` и `\2`.
- Символ `p` в конце команды подстановки означает вывод строки на печать.

Аналогичная команда Perl выглядит следующим образом:

```
perl -ne 'print if s/(It is) (an ancyent Marinere)/\2 \1/'↳
 rime.txt
```

Заметьте, что в этой команде используется синтаксис обратных ссылок в стиле `\1`. Разумеется, в Perl с равным правом можно использовать синтаксис вида `$1`:

```
perl -ne 'print if s/(It is) (an ancyent Marinere)/$2 $1/'↳
 rime.txt
```

Мне нравится та простота, с какой Perl позволяет вывести выделенную строку на печать. Скажу несколько слов о полученном выводе:

```
an ancyent Marinere It is,
```

Преобразование нарушило общепринятые правила использования прописных и строчных букв, но Perl позволяет исправить это с помощью директив `\u` и `\l`:

```
perl -ne 'print if s/(It is) (an ancyent Marinere)/\u$2↳
 \l$1/' rime.txt
```

Теперь результат выглядит гораздо лучше:

```
An ancyent Marinere it is,
```

Объясню, как мы этого добились:

- символ `\u` ничему не соответствует и преобразует следующий символ в верхний регистр;
- символ `\l` ничему не соответствует и преобразует следующий символ в нижний регистр;
- директива `\U` (здесь не используется) преобразует все символы следующей строки в верхний регистр;
- директива `\L` (здесь не используется) преобразует все символы следующей строки в нижний регистр.

Эти директивы действуют до тех пор, пока не будут отменены (например, `\l` отменяет действие `\U`). Поэкспериментируйте с ними самостоятельно, чтобы проверить, как они работают.

Именованные группы

Именованные группы — это захватывающие группы, которым присвоены имена. Доступ к сохраненному содержимому таких групп может осуществляться с использованием имен, а не целых чисел. Их использование демонстрируется ниже на примере Perl.

```
perl -ne 'print if s/(?<one>It is) (?<two>an ancyent Marinere)/↳
\u$+{two}\l$+{one}/' rime.txt
```

В этой команде именованные группы создаются и используются следующим образом.

- Присвоение группам имен `one` и `two` осуществляется путем дописывания последовательностей `?<one>` и `?<two>` в начале содержимого соответствующей группы в круглых скобках.
- Последовательность `$+{one}` ссылается на группу `one`, а последовательность `$+{two}` — на группу `two`.

Допускается повторное использование именованных групп в том шаблоне, в котором они были поименованы. Сейчас поясню, что имеется в виду. Предположим, выполняется поиск строки, содержащей подстроку в виде шести следующих подряд нулей:

```
000000
```

Это тривиальный пример, но для наших целей его будет вполне достаточно. Присвоим имя группе, состоящей из трех нулей, с помощью следующего шаблона (имя `z` выбрано произвольно):

```
(?<z>0{3})
```

Далее эту группу можно использовать примерно так:

```
(?<z>0{3})\k<z>
```

так:

```
(?<z>0{3})\k'z'
```

или так:

```
(?<z>0{3})\g{z}
```

Апробируйте эти выражения в RegExr, и вы убедитесь в том, что все они отлично работают. Ряд других синтаксических конструкций, предназначенных для работы с именованными группами, приведен в табл. 4.3.

Таблица 4.3. Синтаксис именованных групп

Синтаксис	Описание
`(?<`*имя*`>...)`	Именованная группа
`(?`*имя*`...)`	Другая именованная группа
`(?P<`*имя*`>...)`	Именованная группа в Python
`\k<`*имя*`>`	Ссылка на группу по ее имени в Perl
`\k'`*имя*`'`	Ссылка на группу по ее имени в Perl
`\g{`*имя*`}`	Ссылка на группу по ее имени в Perl
`\k{`*имя*`}`	Ссылка на группу по ее имени в .NET
`(?P=`*имя*`)`	Ссылка на группу по ее имени в Python

Незахватывающие группы

Существуют также *незахватывающие* группы, содержимое которых не сохраняется в памяти. Иногда это оказывается преимуществом, особенно в тех случаях, когда вы не собираетесь ссылаться на группу. Отсутствие необходимости сохранения данных в памяти может обеспечивать более высокую производительность, хотя при выполнении простых примеров, приведенных в данной книге, об этом вряд ли стоит задумываться.

Помните, что собой представляла первая из групп, которые обсуждались в этой главе? Вот как она выглядела:

```
(the|The|THE)
```

Никакой необходимости в использовании обратных ссылок здесь нет, поэтому можно создать незахватывающую группу такого вида:

```
(?:the|The|THE)
```

По аналогии с примером, приведенным в самом начале главы, в это выражение можно было бы добавить опцию, делающую шаблон нечувствительным к регистру (хотя эта опция фактически устраняет необходимость в использовании группы):

```
(?i)(?:the)
```

Эту же группу можно записать в таком виде:

```
(?:(?i)the)
```

а еще лучше в таком:

```
(?i:the)
```

Букву `i`, устанавливающую данную опцию, можно записывать между вопросительным знаком и двоеточием, что и было сделано в последнем случае.

Атомарные группы

Атомарные группы — это разновидность незахватывающих групп. В случае использования движка регулярных выражений, работающего в режиме поиска с возвратом, атомарная группа отключает этот режим, но не для всего регулярного выражения, а лишь для той его части, которая заключена в данной группе. Соответствующий синтаксис выглядит примерно так:

```
(?>the)
```

В каких ситуациях имеет смысл использовать атомарные группы? К числу операций, способных существенно замедлять работу механизма регулярных выражений, относится поиск с возвратом. Это связано с тем, что такой поиск требует перебора всех возможностей, на что уходит много времени и расходуются значительные вычислительные ресурсы. Иногда операции такого рода становятся основными пожирателями времени. Существует даже термин — *катастрофический поиск с возвратом.*

Можете либо вообще отказаться от поиска с возвратом, используя движки наподобие re2 (`http://code.google.com/p/re2/`), либо отключать этот режим для некоторых частей регулярных выражений, создавая атомарные группировки.

Основная цель, которую я преследую в книге, — это ознакомление читателя с регулярными выражениями. Вопросов производительности я почти не касаюсь, но именно с ними, по моему мнению, наиболее тесно связано использование атомарных группировок.

В главе 5 речь пойдет о символьных классах.

О чем вы узнали в главе 4

- Каким образом альтернативы обеспечивают возможность выбора между двумя и более шаблонами.
- Какие существуют модификаторы опций и как они используются в шаблоне.
- Какие виды подшаблонов существуют.
- Использование захватывающих групп и обратных ссылок.
- Использование именованных групп и ссылок на них.
- Использование незахватывающих групп.
- Что такое атомарная группа.

На заметку

- Adobe Air — кроссплатформенная среда выполнения, позволяющая использовать HTML, JavaScript, Flash и ActionScript для создания веб-приложений, способных выполняться в качестве автономных клиентских приложений без использования браузера. Более подробную информацию по этому вопросу можно получить по адресу `http://www.adobe.com/products/air.html`.
- Python (`http://www.python.org`) — высокоуровневый язык программирования, имеющий собственную реализацию механизма регулярных выражений (`http://docs/python.org/library/re.html`).
- .NET (`http://www.microsoft.com/net`) — программная платформа для компьютеров, работающих под управлением Windows, в которой также реализован механизм регулярных выражений (`http://msdn.microsoft.com/en-us/library/hs600312.aspx`).
- Более полное описание атомарных групп приведено на сайтах `http://www.regularexpressions.info/atomic.html` и `http://stackoverflow.com/questions/6488944/atomic-group-and-non-capturing-group`.