6 Построение эффективных регулярных выражений

Управляемый регулярным выражением механизм НКА встречается в Perl, пакетах Java, языках .NET, Python и PHP (список далеко не полон, за дополнительной информацией обращайтесь к таблице на с. 194). Природа этого механизма такова, что незначительные изменения в регулярном выражении могут кардинально изменить результат и время поиска. Проблемы, которых в механизме ДКА попросту нет, в НКА выходят на первый план. Возможности точной регулировки механизма НКА позволяют *творить* выражения, хотя для непосвященных это порой вызывает немало проблем. Настоящая глава поможет вам овладеть этим искусством.

Наша цель — правильность и эффективность. Это означает, что выражение должно находить все нужное, не находить ничего лишнего, и притом быстро. Правильность рассматривалась в главах 4 и 5, а в этой главе будут рассмотрены вопросы эффективности механизма НКА и то, как обратить их в свою пользу (там, где это уместно, будет приведена и информация о ДКА, но эта глава в первую очередь посвящена механизмам НКА). Главное, что для этого нужно, — доскональное понимание возврата и умение избегать его там, где это возможно. Мы рассмотрим некоторые практические приемы написания эффективных выражений, которые не только ускоряют их работу, но и при хорошем понимании механики их обработки, помогут вам создавать более сложные выражения.

Глава начинается с подробного примера, который демонстрирует, насколько важными могут быть эти проблемы. Затем, чтобы подготовиться к восприятию более сложных приемов, описанных далее, мы снова рассмотрим базовую процедуру возврата, описанную в предыдущей главе, с упором на эффективность и глобальные последствия возврата. Далее рассматриваются некоторые стандартные приемы внутренней оптимизации, способные довольно заметно влиять на эффективность, и особенности построения выражений для тех реализаций, в которых эти приемы используются. Наконец, все сказанное объединяется в нескольких «убойных» приемах, которые помогут вам конструировать чрезвычайно эффективные регулярные выражения НКА.

Проверки и возвраты

Приведенные примеры демонстрируют общие ситуации, возникающие при использовании регулярных выражений. Анализируя эффективность конкретного примера, я часто привожу число отдельных проверок, используемых механизмом регулярных выражений при поиске совпадения. Например, при поиске ⌜`marty`⌟ в строке `smarty` происходит шесть отдельных проверок: сначала ⌜`m`⌟ сравнивается с `s` (неудача), затем ⌜`m`⌟ сравнивается с `m`, ⌜`a`⌟ — с `a` и т. д. (все эти проверки проходят успешно). Я также часто сообщаю количество возвратов (в данном примере ноль, хотя неявный возврат для повторного применения регулярного выражения со второго символа можно посчитать за один).

Я привожу эти конкретные величины не потому, что здесь так важна точность, а скорее для того, чтобы избежать использования туманных слов «много», «мало», «лучше», «терпимо» и т. д. Не подумайте, что использование регулярных выражений в НКА сводится к подсчету проверок или возвратов; я просто хочу, чтобы вы представляли себе порядок этих величин.

Еще одно важное замечание: вы должны понимать, что эти «точные» числа, вероятно, в разных программах будут разными. Я привожу лишь базовые показатели для тех примеров, которые, как я надеюсь, вам еще пригодятся. Однако приходится учитывать и другой важный фактор — оптимизацию, выполняемую конкретной программой. Достаточно «умная» реализация может полностью устранить поиск конкретного регулярного выражения, если она заранее решит, что оно в любом случае не совпадет с имеющейся строкой (например, из-за отсутствия в строке некоторого символа, который обязательно должен присутствовать в возможном совпадении). Мы рассмотрим некоторые приемы оптимизации в этой главе, но общие принципы важнее частных случаев.

Традиционный НКА и POSIX НКА

Анализируя эффективность выражения, следует учитывать тип механизма используемой программы (традиционный НКА или POSIX НКА). Как будет показано в следующем разделе, некоторые проблемы относятся лишь к одному из этих типов. Иногда изменение, не влияющее на один механизм, сильно отражается на работе другого. Еще раз подчеркну, что понимание базовых принципов поможет вам правильно оценивать все ситуации по мере их возникновения.

Убедительный пример

Начнем с примера, который наглядно продемонстрирует, какими важными могут быть проблемы возврата и эффективности. На с. 257 мы построили выражение

⌜"(\\.|[^\\"])*"⌟ для поиска строк, заключенных в кавычки. В строке могут присутствовать внутренние кавычки, экранированные символом \. Это регулярное выражение работает, но в механизме НКА конструкция выбора, применяемая к каждому символу, работает крайне неэффективно. Для каждого «обычного» символа в строке (не кавычки и не экранированного символа) механизм должен проверить ⌜\\.⌟ обнаружить неудачу и вернуться, чтобы в результате найти совпадение для ⌜[^\\"]⌟. Если выражение используется в ситуации, когда важна эффективность, конечно, хотелось бы немного ускорить обработку этого выражения.

Простое изменение — начинаем с более вероятного случая

Поскольку в средней строке, заключенной в кавычки, обычных символов больше, чем экранированных, напрашивается простое изменение — сменить порядок альтернатив и поставить ⌜[^\\"]⌟ на первое место, а ⌜\\.⌟ — на второе. Если ⌜[^\\"]⌟ стоит на первом месте, то возврат происходит лишь при обнаружении экранированного символа в строке (и, конечно, при несовпадении *, поскольку конструкция выбора не совпадает лишь в том случае, если не совпадают все альтернативы). Рисунок 6.1 наглядно демонстрирует отличия между этими двумя выражениями. Уменьшение количества стрелок в нижней половине означает, что для первой альтернативы совпадения находятся чаще. Это приводит к уменьшению количества возвратов.

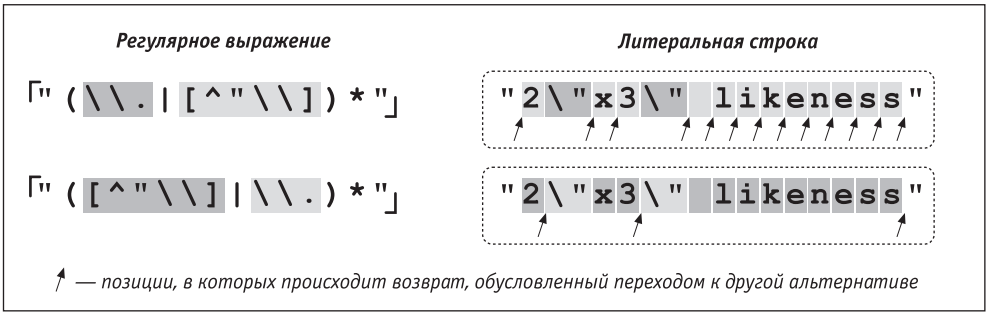

Рис. 6.1. Изменение порядка альтернатив (для традиционного НКА)

Оценивая последствия такого изменения для эффективности, необходимо задать себе несколько ключевых вопросов:

- Какой механизм выиграет от этих изменений — традиционный НКА, POSIX НКА или оба?

- Когда изменение приносит наибольшую пользу — когда текст совпадает, когда текст не совпадает или в любом случае?

❖ Подумайте над этими вопросами, затем проверьте свои ответы на с. 288. Прежде чем переходить к следующему разделу, убедитесь в том, что вы хорошо понимаете смысл ответов и их обоснование.

Эффективность и правильность

Самый важный вопрос, который необходимо себе задать при любых попытках повышения эффективности: а не повлияет ли изменение на правильность совпадения? Изменение порядка альтернатив допустимо лишь в том случае, если порядок не влияет на успешность совпадения. Рассмотрим выражение `"(\\.|[^"])*"`, более раннюю (☞ 255) и ошибочную версию выражения из предыдущего раздела. В этом выражении инвертированный символьный класс не содержит символ \\ и поэтому может совпасть в случаях, в которых он совпадать не должен. Если регулярное выражение применяется только к «правильным» данным, с которыми оно *должно* совпадать, проблема останется незамеченной. Полагая, что выражение работает нормально, а перестановка альтернатив повысит эффективность поиска, вы попадете в беду. Перестановка, после которой `[^"]` оказывается на первом месте, приводит к неверному совпадению во всех случаях, когда целевой текст содержит экранированную кавычку:

```
"You need a 2\"3\" photo."
```

Итак, прежде чем беспокоиться об эффективности, обязательно убедитесь в том, что выражение работает правильно.

Следующий шаг — локализация максимального поиска

Из рис. 6.1 ясно видно, что в обоих случаях квантификатор * должен последовательно перебрать все нормальные символы, при этом он снова и снова входит в конструкцию выбора (и круглые скобки) и выходит из нее. Все эти действия сопряжены с лишней работой, от которой хотелось бы по возможности избавиться.

Однажды, работая над аналогичным выражением, я вдруг понял, что выражение можно оптимизировать, если учесть, что когда выражение `[^"\\]` относится к «нормальным» (не кавычки и не экранированные символы) случаям, то при его замене на `[^\\"]+` одна итерация `(…)*` прочитает все последовательно стоящие обычные символы. При отсутствии экранированных символов будет прочитана вся строка. Это позволяет найти совпадение практически без возвратов и сокращает

многократное повторение * до абсолютного минимума. Я был очень доволен своим открытием.

Этот пример будет подробно рассмотрен ниже, но даже беглый взгляд на статистику наглядно демонстрирует преимущества нового выражения. На рис. 6.2 показано, как происходит поиск в традиционном НКА. Модификация исходного выражения `"(\\.|[^\\"])*"` (верхняя пара на рис. 6.2) уменьшает количество возвратов, связанных с конструкцией выбора, а также число итераций квантификатора *. Нижняя пара на рис. 6.2 показывает, что объединение этой модификации с изменением порядка альтернатив приводит к еще большему повышению эффективности.

Рис. 6.2. Последствия добавления плюса (традиционный механизм НКА)

Добавление квантификатора + уменьшает количество возвратов, обусловленных конструкцией выбора, что, в свою очередь, приводит к уменьшению количества итераций *. Квантификатор * относится к подвыражению в круглых скобках, а каждая итерация сопряжена с немалыми затратами при входе в круглые скобки и выходе из них, поскольку механизм должен хранить информацию о том, какой текст совпадает с подвыражением в скобках (эта тема подробно рассматривается ниже).

Таблица 6.1 аналогична таблице, приведенной во врезке, но в ней рассматривается меньшее количество примеров и имеются дополнительные столбцы для количества итераций *. При модификации выражения количество проверок и возвратов увеличивается незначительно, но количество итераций уменьшается очень заметно. Налицо заметное повышение эффективности.

ПОСЛЕДСТВИЯ ПРОСТЫХ ИЗМЕНЕНИЙ

❖ *Ответ на вопрос со с. 286.*

Для какого типа механизма? Изменение практически никак не повлияет на работу механизма POSIX НКА. Поскольку этот механизм в любом случае должен опробовать все комбинации элементов регулярного выражения, порядок проверки альтернатив не важен. Однако в традиционном НКА порядок альтернатив, ускоряющий поиск совпадения, является преимуществом, поскольку механизм может остановиться сразу же после того, как будет найдено первое совпадение.

Для какого результата? Изменение приводит к ускорению поиска лишь при наличии совпадения. НКА может сделать вывод о неудаче только после того, как будут проверены все возможные комбинации (повторяю, POSIX НКА проверяет их в любом случае). Следовательно, если попытка окажется неудачной, значит, были опробованы все комбинации, поэтому порядок не важен.

В следующей таблице перечислено количество проверок и возвратов для некоторых случаев (чем меньше число, тем лучше).

	Традиционный НКА				**POSIX НКА**	
	`「"(\\.|[^"])*"」`		`「"([^"]|\\.)*"」`		**Оба выражения**	
Текст примера	**Пров.**	**Возвр.**	**Пров.**	**Возвр.**	**Пров.**	**Возвр.**
`"2\"x3\" likeness"`	32	14	22	4	48	30
`"makudonarudo"`	28	14	16	2	40	26
`"very...`*99 символов*`...long"`	218	109	111	2	325	216
`"No \"match\" here`	124	86	124	86	124	86

Как видите, в POSIX НКА оба выражения дают одинаковые результаты, а в традиционном НКА для нового выражения быстродействие возрастает (уменьшается количество возвратов). В ситуации без совпадения (последний пример в таблице) оба механизма проверяют все возможные комбинации, поэтому и результаты оказываются одинаковыми.

Таблица 6.1. Эффективность совпадений для традиционного НКА

	`「"([^\\"]|\\.)*"」`			`「"([^\\"]+|\\.)*"」`		
Текст примера	**Пров.**	**Возвр.**	**Итер**	**Пров.**	**Возвр.**	**Итер.**
`"makudonarudo"`	16	2	13	17	3	2
`"2\"x3\" likeness"`	22	4	15	25	7	6
`"very...`*99 символов*`...long"`	111	2	108	112	3	2