Expresiones regulares

Introducción

Numerosos problemas de tratamientos de ristras requieren la localización en un texto de subristras que cumplen un determinado patrón: fechas, números de DNI, números de la seguridad social, e-mails, direcciones web, etc. La búsqueda exacta que proporcionan operaciones como el método *indexOf* de la clase *String* de Java resulta insuficiente para resolver este tipo de problemas.

Una expresión regular es una ristra de caracteres que representa un patrón de búsqueda. El patrón más simple es el de búsqueda exacta, en el que cada carácter se representa a sí mismo (por ejemplo, la expresión *"niño"* serviría para buscar la subristra *"niño"*), pero la potencia de las expresiones regulares radica en que determinados caracteres no se representan a sí mismos, sino que actúan como metacaracteres que indican la aparición de otros caracteres o combinaciones de caracteres. Por ejemplo, la expresión *"niñ(o|a)"* podría servir para buscar las subristras *"niño"* o *"niña"* debido a la presencia de los metacaracteres paréntesis, que sirven para agrupar, en combinación con el metacarácter '|' que indica alternativa.

El que algunos caracteres actúen como metacaracteres implica que, en caso de que haya que buscar subristras que los contengan, sea necesario representarlos usando secuencias de escape, anteponiendo una barra inclinada '\' al carácter en cuestión. Por ejemplo, la expresión *"\(75\)"* busca la subristra *"(75)"*, mientras que *"(75)"* busca la subristra *"75"*. La barra inclinada también se usa, como veremos, para dotar de significado a caracteres que, en otras circunstancias, serían interpretados de forma normal (por ejemplo, la expresión *"d"* busca la letra *'d'*, mientras que *"\d"* busca un dígito).

Por otra parte, la barra diagonal también se usa en Java y otros lenguajes para formar secuencias de escape para representar símbolos especiales dentro de una ristra. Por ejemplo, si queremos incluir comillas en una ristra en Java tendríamos que

anteponerles una barra, dado que las comillas funcionan como delimitadores, tal como se ve en el siguiente ejemplo:

```
String dlg = "Pedro dijo: \"hola\"";
```

El que la barra diagonal funcione como carácter de escape en Java implica que cuando una expresión regular que incluya secuencias de escape se quiera representar como un literal ristra habrá que doblar las barras:

```
(expreg) "\d" → (Java String) "\\d"
```

En este documento mostraremos las expresiones regulares tal cual, sin doblar las barras; deberá tenerse en cuenta a la hora de trasladar dichas expresiones a un programa en Java.

Metacaracteres

Los metacaracteres son caracteres o conjuntos de caracteres que tienen un significado especial y representan la aparición de otros caracteres o secuencias de caracteres en la ristra de búsqueda.

Conjuntos y rangos

Determinados metacaracteres sirven para indicar la aparición de un carácter correspondiente a un conjunto o rango de caracteres (Tabla 1).

Mc	Significado	Patrón ejemplo	Ejemplos de ristras que cumplen el patrón
.	Cualquier carácter	.iña	"ciña", "niña", "piña", "riña", "tiña", …
[]	Cualquier carácter del conjunto encerrado entre corchetes	niñ[oa]	"niño", "niña"
-	(dentro de corchetes) representa cualquier carácter de un rango de caracteres	[0-9]	"0", "1", "2", "3", "4", "5", "6", "7", "9"
^	(detrás de abre-corchete) representa la negación de la expresión contenida en los corchetes	[^0-9]	Cualquier carácter que no sea un dígito

Tabla 1

Las expresiones contenidas entre corchetes se pueden combinar:

```
[0-9a-z] representa "cualquier dígito o letra minúscula"
```

Posición en líneas

Cuando se trabaja con ristras que representan un texto con varias líneas, existen metacaracteres para representar si la subristra buscada debe encontrarse al comienzo o final de línea (Tabla 2).

Mc	Significado	Patrón ejemplo	Ejemplos de subristras que cumplen el patrón
^	(al principio de la expresión) indica un comienzo de línea	^ser	"ser o no ser"
$	(al final de la expresión) indica un final de línea	ser$	"ser o no ser"

Tabla 2

Posición en palabras

Existen metacaracteres para indicar si la subristra buscada debe encontrarse, o no, en el extremo de una palabra (Tabla 3), entendiendo como palabra cualquier combinación formada por letras (minúsculas o mayúsculas, dígitos o el carácter '_').

Mc	Significado	Patrón ejemplo	Ejemplos de subristras que cumplen el patrón
\b	Indica extremo de palabra (dependiendo de si está al principio o al final de la expresión indicará comienzo, final, o palabra completa)	\bser	"servir"
		ser\b	"coser"
		\bser\b	"ser"
\B	Indica no extremo de palabra (dependiendo de si está al principio o al final de la expresión indicará no comienzo, no final, o enmedio)	\Bcos	"descoser", "cascos"
		cos\B	"descoser", "coser"
		\Bcos\B	"descoser"

Tabla 3

Abreviaturas de clase

Las abreviaturas de clase (Tabla 4) son patrones predefinidos, expresados como secuencias de escape, que expresan de forma más sencilla patrones de uso frecuente.

Ab	Significado	Equivalencia
\d	Cualquier dígito	[0-9]
\D	Cualquier carácter que no sea un dígito	[^0-9]
\s	Un carácter espaciador	[\t\n\x0b\r\f]
\S	Un carácter no espaciador	[^\s]
\w	Un carácter "de palabra"	[a-zA-Z_0-9]
\W	Un carácter que no sea "de palabra"	[^\w]

Tabla 4

Cuantificadores

Los cuantificadores (Tabla 5) indican la repetición del elemento que los precede.

Mc	Significado	Patrón ejemplo	Ejemplos de subristras que cumplen el patrón
{m}	Indica que el elemento precedente se repite m veces	\d{3}	"124", "523", "897", "847", …
{m,}	Indica que el elemento precedente se repite al menos m veces	\d{2,}	"16", "23", "897", "8423457", …
{m,n}	Indica que el elemento precedente se repite entre m y n veces	\d{1,3}	"1", "23", "897", "847", …
?	Indica que el elemento precedente se repite 0 o 1 veces (equivale a {0,1})	cast?a	"casa", "casta"
*****	Indica que el elemento precedente se repite 0 o más veces (equivale a {0,})	cal*e	"cae", "cale", "calle", "callle", …
+	Indica que el elemento precedente se repite al menos 1 vez (equivale a {1,})	calla+	"calla", "callaa", "callaaa", "callaaaa", "callaaaaa", …

Tabla 5

Agrupamiento y alternativa

Los paréntesis se usan para agrupar porciones de una expresión regular, formando subexpresiones. Ello permite tratar las subexpresiones como un todo, pudiendo, por ejemplo, asignarles cuantificadores:

```
\d\d-\d\d-\d\d\d\d se puede convertir en: (\d{2}-){2}\d{4}
```

Además, permite hacer referencia, en una operación de manipulación de ristras, al patrón encontrado por cada expresión; esto es posible porque cada subexpresión correspondiente se numera de izquierda a derecha, dándole un identificador que es dicho número precedido por el carácter '$'. Supongamos que tenemos una ristra, T, que representa un texto que tiene algunos errores, como, por ejemplo, que se han introducido espacios entre algunas palabras y un signo de puntuación que le sigue, como en el ejemplo:

```
T = "Los números son : uno , dos    , tres ."
```

Para localizar cuándo se da esa situación, podemos usar una expresión como:

```
\w\s+[,;.:]
```

Que es equivalente a:

```
(\w)(\s+)([,;.:])
```

La diferencia, es que en el segundo caso, al estar las partes de la expresión agrupadas con paréntesis, son referenciables:

(\w)	$1
(\s+)	$2
([,;.:])	$3

De esta manera, podríamos usar una operación de manipulación de ristras para hacer algo como:

```
Sustituir en la ristra T todas las apariciones de
(\w)(\s+)([,;.:]) por $1$3
```

Dada una subristra de T que coincida con el patrón de búsqueda, lo que se haría es sustituirla por la parte de esa subristra que coincide con el grupo $1 del patrón, concatenada con la que coincide con el grupo $3, eliminando de esta manera el grupo $2, que corresponde a los separadores no deseados antes de un signo de puntuación.