Expresiones Regulares

david@edu.xunta.es

Contenidos

Introducción....3
Usos de las expresiones regulares....3
Programación....3
Herramientas de líneas de comandos....3
Búsquedas en procesadores ofimáticos....4
Búsqueda en el la consola de salida del Netbeans....4
Reemplazando texto en el Notepad++....5
Directivas del servidor web Apache....5
Caracteres utilizados....6
Cuantificadores....7
Modificadores de la expresión regular....7
Principio y final de cadena....8
Paréntesis y grupos....8
Ejemplos:....9

Introducción

Una expresión regular es una cadena formada por un conjunto de caracteres que nos permite crear un patrón para realizar búsquedas (y sustituciones) en un texto. Dicho patrón encajará (o no) en una o varias partes del texto, resultando muy sencillo su manejo posterior.

Las expresiones regulares bien utilizadas nos evitan la programación de líneas y líneas de codigo ejecutando los típicos métodos de los objetos tipo *String* como pueden ser extraer subcadenas, localizar posiciones, devolver longitudes, etc.

Usos de las expresiones regulares

Las utilidades de las expresiones regulares van desde validaciones a datos simples (código postal, NIF, IBAN, matrículas, números, fechas, etc.) hasta un compilador de un lenguaje de programación (los errores sintácticos en un programa básicamente se descubren porque el código fuente no 'encaja' con su expresión regular).

Aunque el uso de expresiones regulares es muy común, existen multitud de entornos en donde son utilísimas y a veces no somos conscientes de su posible uso:

Programación

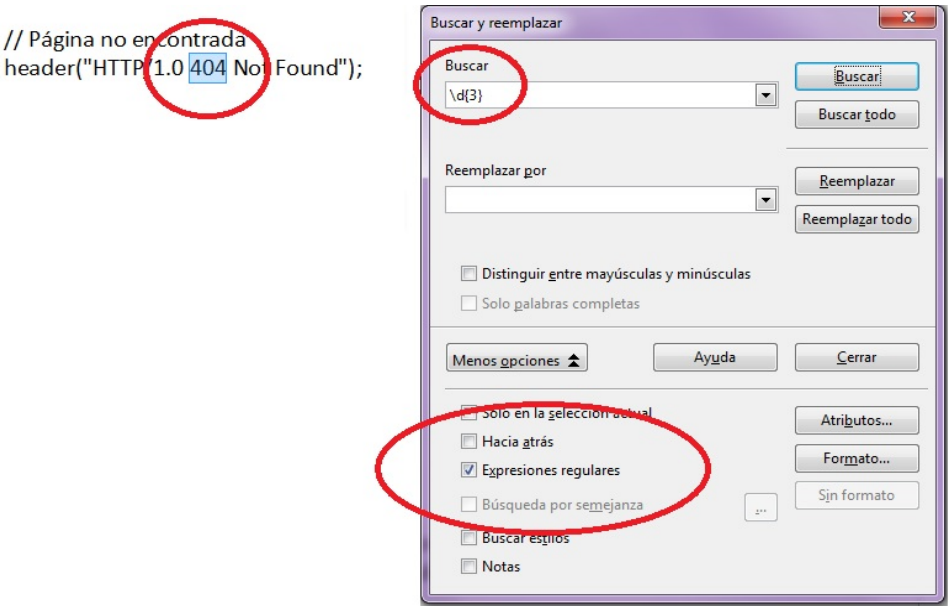
Cualquier aplicación que maneje datos de tipo cadena es susceptible de utilizar toda la potencia de estas expresiones. Desde validaciones de los datos de usuario introducidos en un formulario web hasta procesamiento masivo de textos.

Herramientas de líneas de comandos

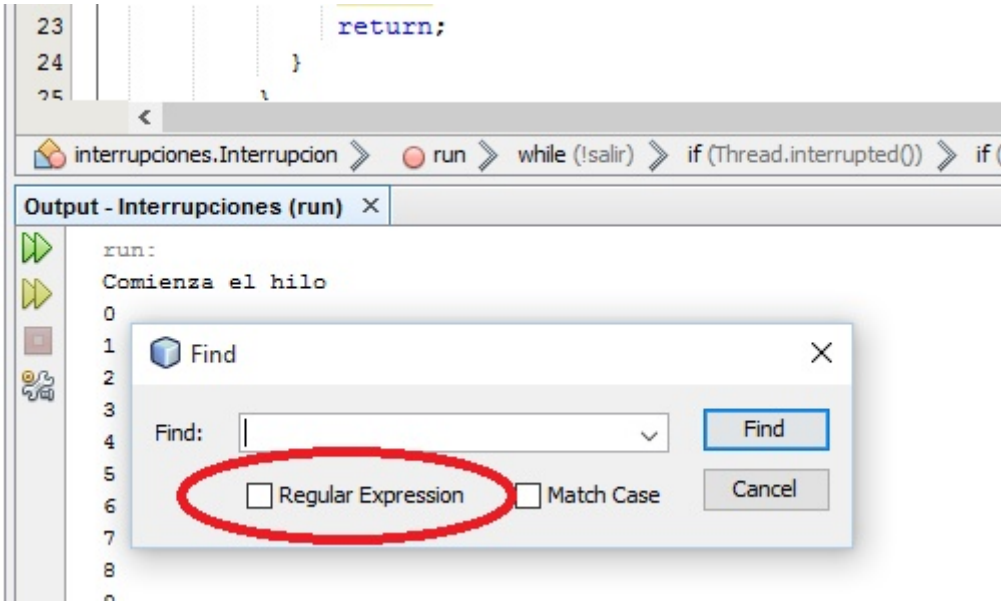
Multitud de aplicaciones de la línea de comandos del sistema operativo permiten el uso de expresiones regulares. Desde un simple *ls* hasta comandos tipo *grep*, *find*, *sed*, etc.

La herramienta *sqlite3* permite pasarle a los comandos *.dump .schema .tables e .indices* una expresión regular para trabajar con las tablas cuyo nombre encaje con dicho parámetro.

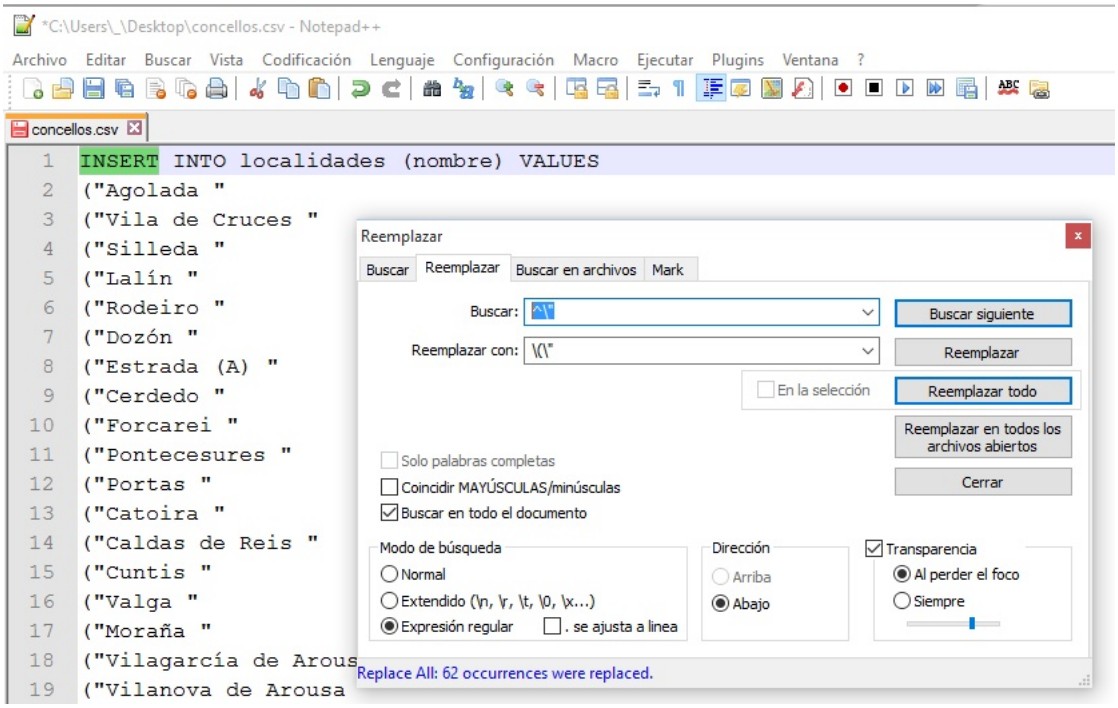
Búsquedas en procesadores ofimáticos



Búsqueda en el la consola de salida del Netbeans



Reemplazando texto en el Notepad++



Directivas del servidor web Apache

En multitud de directivas Apache es posible utilizar expresiones regulares.

Un ejemplo son las directivas *DirectoryMatch, FilesMatch y LocationMatch*, que son similares a *Directory, Files y Location*, para definir propiedades a una carpeta, archivo o URL, pero indicando una expresión regular para especificar los nombres afectados.

Otro ejemplo muy interesante lo encontramos si activamos el módulo *mod_rewrite* en el *httpd.conf. É*ste se ejecutará entre el *request* a un servidor y la ejecución de cualquier script y así podremos reescribir la URL pedida por el usuario. Para ello incluiremos una directiva en el archivo *.htaccess* como la siguiente:

```
RewriteEngine  On
RewriteRule    ^([a-zA-Z]+)/([a-zA-Z]+)/([a-zA-Z\d]+)\.htm$
               miScript.php?categoria=$1&producto=$2&talla=$3 [L,NC]
RewriteRule    ^articulos/(.+).php codigo/ver_articulo.php?nombre=$1
```

En este ejemplo conseguimos simular páginas estáticas que serán redirigidas a un *script* de generación dinámica de una web en función de los parámetros indicados en la URL (carpeta – subcarpeta y nombre de la página).

Caracteres utilizados

En una expresión regular se pueden poner los caracteres que se quiera, pero existen una serie de caracteres especiales que nos permiten construir un patrón flexible que encaje con las cadenas deseadas.

\d	Un dígito (del 0 al 9)
\w	Un carácter alfanumérico (letras, números o _)
\s	Un carácter de espacio en blanco (espacio, tabulación, retorno de carro, …)
\D	No es un dígito
\W	Cualquier carácter no alfanumérico
\S	No es un carácter de espacio en blanco …
.	Cualquier carácter excepto el salto de línea
|	Encaja con alguna de las expresiones que tiene a los lados
\	Para escapar caracteres especiales como el punto \. o la propia \\

Cuando incluímos unos corchetes en una expresión regular, éstos encajarán siempre con UN CARÁCTER, cualquiera de los que se encuentren en su interior, pero solo con uno. Dentro de los corchetes se pueden incluir rangos de caracteres utilizando un guión, teniendo en cuenta que los números pueden incluirse pero se toman siempre como caracteres, no existen rangos numéricos.

Como primer carácter dentro de los corchetes se puede utilizar el operador ^, que indicará que se encajará con cualquier carácter que NO esté presente.

[caracteres]	Encaja con cualquier carácter incluído en el interior. Se permiten rangos.
[^caracteres]	Encaja con cualquier carácter NO incluído en el interior. Se permiten rangos.

p[ae]z	encajará únicamente con 'paz' y con 'pez', (pero no con 'paez')
[A-Za-z]\d	encajará con una letra cualquiera (mayúscula o minúscula) seguida por un número
[A-Za-z\d]	encajará con una letra cualquiera (mayúscula o minúscula) o un número
1[0-5]	encajará con una cadena que contenga un número del 10 al 15 ambos inclusive
[0-9]	es lo mismo que \d
[^0-9]	es lo mismo que \D
[^aeiou]	encajará con cualquier carácter que no sea una vocal (caracteres especiales, etc.)

Cuantificadores

Los cuantificadores afectan a la subexpresión o carácter anterior a su localización y nos permiten modelizar las posibles repeticiones de caracteres.

{n}	se repetirá exactamente n veces
{n,}	se repetirá al menos n veces (n o más veces)
{n,m}	se repetirá entre n y m veces (ambos incluídos)
?	es equivalente a {0,1}, nos permite indicar opcionalidad
*	es equivalente a {0,}
+	es equivalente a {1,}
*?	Cuantificador *reluctant* (reacio). Se queda con el "encaje" más pequeño.

Por lo general en la búsqueda del cumplimiento del patrón con cuantificadores se intenta siempre encajar lo máximo posible *(greedy match* – codicioso*).* Existen mencanismos en las distintas librerías para revertir este comportamiento (*reluctant match* – reacio).

Por ejemplo "\(.*\) " encajará una sola vez con " … (bla) … (bla) ..." en vez de dos veces. Utilizando \(.*?\) sí encajará dos veces como cabría esperar.

Modificadores de la expresión regular

Por lo general (dependiendo del lenguaje de programación o entorno de trabajo) podemos asociar distintos modificadores a la expresión regular.

Por ejemplo, un modificador tipo '*ignore case*' para no tener que especificar continuamente A-Za-z dentro de los corchetes, con uno de ellos sería suficiente. Otros modificadores útiles son los que indican (en el caso de textos multilínea) si hay que tener en cuenta cada línea independientemente o bien todo el texto en su conjunto.

Principio y final de cadena

Es conveniente resaltar que una expresión regular encaja en un texto cuando se encuentra al menos una ocurrencia del patrón en dicho texto. Esto es útil para realizar búsquedas (porque se podrá recorrer todas las ocurrencias encontradas), pero es un problema si queremos validar un dato concreto obtenido, por ejemplo, de un formulario web. Es decir, el patrón \d{5} de un código postal encajaría en el texto "cp36001 xxx" introducido por un usuario cuando obviamente no sería válido. Para solucionarlo tenemos dos caracteres más:

^	Principio de cadena
$	Fin de cadena

La expresión ^\d{5}$ sí encajaría con un texto que contenga únicamente cinco dígitos.

Es recomendable estudiar detenidamente el comportamiento de las clases y los métodos utilizados a la hora de programar utilizando expresiones regulares. Existen lenguajes (librerías más exactamente) que pueden asumir el ^ … $ en las expresiones pasadas como parámetro. Por ejemplo, utilizando el paquete *java.util.regex*, el método *matches* de las clases *String*, *Pattern o Matcher*, devuelve *true* si encaja exactamente (asume el ^...$), en cambio los métodos *find* o *lookingAt* no lo hacen.

El comportamiento de los caracteres ^ y $ pueden variar en función del uso del modificador de multilínea, pues puede significar principio/final de línea o principio/final del texto completo.

Paréntesis y grupos

Podemos agrupar partes de la expresión regular creando subexpresiones a las que se le pueden aplicar cuantificadores.

A veces los paréntesis son necesarios para agrupar de manera correcta las subexpresiones, pero otras los incluiremos a propósito puesto que cada paréntesis crea automáticamente un grupo que, una vez encajada la expresión en un texto concreto, podremos obtener el texto de ese grupo de una manera directa.

Por ejemplo, podemos crear una expresión regular para un correo electrónico en donde creamos dos grupos para extraer automáticamente el usuario y el servidor, o otros dos grupos para extraer fácilmente los números y las letras del NIF: ^(\d{8})([TRWAGMYFPDXBNJZSQVHLCKE])$

Podremos utilizar los valores de los grupos obtenidos directamente en la propia expresión regular, usando la barra y a continuación el nº del grupo, o bien a través del lenguaje de programación utilizado que siempre nos proporciona una manera directa de acceder.

```
<([a-zA-Z]\w*?)>.*?</\1>              Encajamos con un elemento XML independientemente del nombre
```

Utilizar grupos también nos evita, una vez validado un texto, tener que acudir a funciones de cadena para extraer las partes que nos interesan.

Ejemplos:

NOTA: muchos son mejorables: código postal (se podría ajustar al menos los dos primeros dígitos a los códigos de provincia existentes), nif (no contempla variantes donde el 1º carácter es una letra) o correo electrónico (está muy genérico, habría que saber exactamente los caracteres válidos y no válidos, longitud del dominio de alto nivel …)

- Código Postal ^\d{5}$
- NIF ^\d{8}[TRWAGMYFPDXBNJZSQVHLCKE]$
- Matrículas (nuevas) ^\d{4}[BCDFGHJKLMNPRSTVWXYZ]{3}$
- N.º entero positivo ^\d+$
- N.º entero con signo ^[+\-]?\d+$
- N.º real con hasta 4 decimales ^\d*[\.,]?\d{0,4}$ dejando todo opcional encajaría “” !
- N.º del día de la semana ^[1-7]$
- N.º del mes ^([12]?\d)|(3[01])$
- Hora ^(1?\d)|(2[0-3])$
- Minutos (o segundos) ^[1-5]?\d$
- Archivo con extensión de imagen .\.(gif|png|jpe?g)$
- IP ^([01]?\d?\d|2[0-4]\d|25[0-5])(\.[01]?\d?\d|2[0-4]\d|25[0-5]){3}$
- Correo electrónico ^.+@.+\.([\da-zA-Z]){2,}$

Para probar estos ejemplos y todos los que se le ocurran, lo más sencillo es recurrir a páginas de internet en donde se pueden realizar un test online de una cadena cualquiera contra una expresión regular.