Regex

L'expressió regular o regex, és extremadament poderosa en la cerca i manipulació de cadenes de text. Una línia d'expressions regulars pot reemplaçar fàcilment diverses dotzenes de línies de codis de programació.

Regex és compatible amb tots els llenguatges de seqüències de comandos (Perl, Python, PHP i JavaScript); així com llenguatges de programació de propòsit general com Java; i fins i tot processadors de text com Word per a buscar textos.

Sintaxis

Per a crear una expressió regular pots usar:

- un caràcter qualsevol, es cerca la coincidència exacta.
- un caràcter especial ., +, *, ?, ^, $, (,), [,], {, }, | tenen un significat propi.
- una seqüència d'escapament
    - \car per a representar un caràcter que té un significat especial en les expressions regulars.
    - \\ per a representar el caràcter "\"(barra invertida).
    - una seqüència d'escapament predefinida

Seqüència d'escapament

Està formada per la barra invertida seguida d'un o més caràcters.

- \n nova línia
- \t tabulació
- \r retorn de carro
- \nnn un número octal de fins a 3 dígits
- \xhh un codi hexadecimal de dos dígits
- \uhhhh un Unicode de 4 dígits
- \uhhhhhhhh un Unicode de 8 dígits.

Metacaracters

Coincideix amb un única caràcter.

- . qualsevol caràcter
- \d qualsevol caràcter del 0 al 9 (un dígit), equival a [0-9]
- \D qualsevol caràcter diferent del 0 al 9 (un no dígit), equival a [^0-9]
- \s qualsevol caràcter separador, equival a [\t\n\x0B\f\r]

- \S qualsevol caràcter no separador, equival a [^\s]
- \w qualsevol caràcter alfabètic, equival a [a-zA-Z_0-9]
- \W qualsevol caràcter no alfabètic, equival a [^\w]
- \b qualsevol caràcter límit de paraula, equival a [a-zA-Z0-9_]
- \B qualsevol caràcter no límit de paraula, equival a [^\b]

Conjunt de caràcters

Els [ ] creen un conjunt de caràcters, [abc] representa un caràcter "a", "b" o "c".

El - crea un rang de valors, [a-f] representa un caràcter des de "a" fins a la "f".

El caràcter ^ representa la negació, [^0-9] representa un caràcter que no és des de "0" fins al "9".

Quantificadors

- + representa un o més, [0-9]+ representa un o més dígit del "0" al "9".
- * zero o més, [0-9]* representa zero o més dígit del "0" al "9", accepta una cadena buida.
- ? zero o un (opcional), [+-]? representa un "+", "-".
- {m} una quantitat m exacta, [0-9]{3} representa 3 dígits del "0" al "9".
- {m,n} una quantitat de m fins a n (tots dos inclusivament), [0-9]{3,6} representa de 3 a 6 dígits del "0" al "9".
- {m,} una quantitat de m més, [0-9]{3,} representa 3 o més dígits del "0" al "9"..

Posició

- ^ inici de línia, ^[a-z] representa comença amb una lletra minúscula.
- $ final de línia, [0-9]$ representa termina amb un dígit.

Operador O

| és l'operador o que permet crear alternatives, [un|dos|tres] representa el text "un", "dos" o "tres".

Java

Java Regex és una API per a definir patrons per a buscar o manipular cadenes, està en el paquet java.util.regex i proporciona la interfície MatchResult i les classe Matcher, Pattern i PatternSyntaxException.

La classe Matcher proporciona els mètodes:

- boolean matches() provar si l'expressió regular coincideix amb el patró.
- boolean find() troba la següent expressió que coincideix amb el patró.

- boolean find(int start) troba la següent expressió que coincideix amb el patró del número inicial dau.
- String group() retorna la subseqüència coincident.
- int start() retorna l'índex inicial de la subseqüència coincident.
- int end() retorna l'índex final de la subseqüència coincident.
- int groupCount() retorna el número total de la subseqüència coincident.

La classe Pattern proporciona els mètodes:

- static Pattern compile(String regex) compila l'expressió regular donada i retorna la instància del patró.
- Matcher matcher(CharSequence input) crea un comparador que coincideix amb l'entrada donada amb el patró.
- static boolean matches(String regex, CharSequence input) Funciona com la combinació de mètodes de compilació i comparació. Compila l'expressió regular i fa coincidir l'entrada donada amb el patró.
- String[] split(CharSequence input) divideix la cadena d'entrada donada entorn de les coincidències del patró donat.
- String pattern() retorna el patró d'expressió regular.

La classe PatternSyntaxException se llança si el patró de l'expressió regular està mal escrit.

Exemples

En els exemples s'usen els mètodes de les classes anteriors i els mètodes de la classe String usen una expressió regular, matches, replaceAll, replaceFirst, split.

s

El regex "s" busca la cadena "s".

```
Pattern p = Pattern.compile("s");
Matcher m = p.matcher("as");
boolean res = m.matches(); // false
res = Pattern.compile("s").matcher("as").matches();      // false
res = Pattern.matches("s", "as");       // false
String cad = "as", regex = "s";
res = cad.matches(regex);   // false
```

as

El regex "as" busca la cadena "as".

```
String cad = "els dos ases es van anar a pasturar", regex = "as";
Pattern pattern = Pattern.compile(regex, Pattern.CASE_INSENSITIVE); // no distingeix majúscules i minúscules
Matcher matcher = pattern.matcher(cad); // crea el comparador
while (matcher.find()) {      // mentre troba coincidències
```

```
    System.out.println(matcher.group()
     + " comença en: " + matcher.start()
     + " acaba en: " + matcher.end() + " ");
}
 // visualitza
as comença en: 8 acaba en: 10
as comença en: 28 acaba en: 30
```

El mètode group torna la coincidència, start l'índex d'inici de la coincidència i end l'índex final de la coincidència.

dos

El regex "dos" busca la cadena "dos".

```
res = Pattern.matches("dos", "dos ases");        // false
```

.s

El regex ".s" busca qualsevol caràcter i una "s". El punt substitueix qualsevol caràcter.

```
String cad = "as", regex = ".s";
System.out.println(cad.matches(regex));        // true
```

..s.+

El regex "..s.+" busca dos caràcters qualsevol, una "s" seguida de 1 o més caràcter qualsevol. El "+" representa una repetició de 1 o més del "."

```
String cad = "dos ases", regex = "..s.+";
System.out.println(cad.matches(regex));        // true
cad = "tres asses";
System.out.println(cad.matches(regex));        // false
```

.s\b

El regex ".s\b" busca un caràcter qualsevol i una "s" al final d'una paraula.

➜ L'ús en Java necessita doblar el "\" ja que és un caràcter especial.

```
String cad = "dos ases", regex = ".s\\b";.          // cal doblar el "\"
cad = cad.replaceAll(regex, "us");      // reemplaça totes les aparicions de l'expressió per "us"
```

La cadena resultant és "dus assus", els caràcters subratllats són els substituïts, el "as" no es substitueix perquè no està al final d'una paraula.

\bases\b

El regex "\bases\b" busca la paraula "ases" en el text.

```
String cad = "els dos ases es van anar a pasturar", regex = "\\bases\\b";.      // cal doblar el "\"
Pattern pattern = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
```

```
Matcher matcher = pattern.matcher(cad);
while (matcher.find()) {
   System.out.println(matcher.group()
         + " comença en: " + matcher.start()
         + " acaba en: " + matcher.end() + " ");
}
// visualitza
as comença en: 8 acaba en: 12
```

[0-9]

El regex "[0-9]" busca un dígit, també, es pot escriure "\d".

```
String cad = "d'on ei123xen987to0ts aq92389u87e8sts nombres", regex = "\\d";.      // cal doblar el "\"
System.out.println(Arrays.toString(cad.split(regex)));
// visualitza
[d'on ei, , , xen, , , to, ts aq, , , , , u, , e, sts nombres]
```

[0-9]+

El regex "[0-9]+" busca 1 o més dígit, també, es pot escriure "\d+", un nombre.

```
String cad = "d'on ei123xen987to0ts aq92389u87e8sts nombres", regex = "[0-9]+";
System.out.println(Arrays.toString(cad.split(regex)));
// visualitza
[d'on ei, xen, to, ts aq, u, e, sts nombres]
```

[+][0-9]*

El regex "[+][0-9]*" busca un " " o "+" seguit de 0 o més dígits del 0 al 9, un nombre positiu.

```
String cad = "+23 98 -23 0 423 +892 3 -98 23982438 -2 -1", regex = "[ +][0-9]*";
System.out.println(cad.replaceAll(regex, "x"));
// visualitza
xxx-23xxxxxx-98xx-2x-1
```

[a-zA-Z_][0-9a-zA-Z_]*

El regex "[a-zA-Z_][0-9a-zA-Z_]*" busca una lletra de la "a" a la "z" o de la "A" a la "Z" seguit "_" de 0 o més dígits del 0 al 9 lletres de la "a" a la "z" o de la "A" a la "Z", un identificador.

```
String id = "valorPrincipal", regex = "[a-zA-Z_][0-9a-zA-Z_]*";
if (id.matches(regex)) {
   System.out.println(id + " és un identificador correcte");
}
```

^[\w]+\.(gif|png|jpg|jpeg)$

El regex "^[\w]+\.(gif|png|jpg|jpeg)$" busca una o més lletres o el " " a l'inici seguit de "." i que acaba amb "gif" o "png" o "jpg" o "jpeg", el nom d'un fitxer d'imatge.

```
String nomImg = "as de Picas.jpg", regex = "^[\\w ]+\\.(gif|png|jpg|jpeg)$"; .  // cal doblar el "\"
System.out.println(nomImg + (nomImg.matches(regex) ? "" : " no") + " és un nom correcte");
```

^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$

El regex "^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$" valida emails.

El ^ indica al principi.

([.-]?\w+)* és 0 o més [.-]?\w+ és un caràcter opcional "." o "-" seguit d'1 o més lletres, valida el nom de l'usuari.

@ és el "@".

\w+([.-]?\w+)* és la mateixa expressió que l'anterior, s'usa per a validar el nom de l'organització.

(\.\w{2,3})+ és un "." seguit de dos o tres lletres que pot ocórrer una o més vegades, s'usa per a validar el tipus de l'organització.

El $ indica al final.

```
String email = "Armando-Grescas@org.es";
String regex = "^\\w+([.-]?\\w+)*@\\w+([.-]?\\w+)*(\\.\\w{2,3})+$";.          // cal doblar el "\"
System.out.println(email + (email.matches(regex) ? "" : " no") + " és un email correcte");
```