Expressions regulars en Java

Introducció a les expressions regulars en Java

Les expressions regulars són una forma de descriure un conjunt de cadenes basades en característiques comunes compartides per cada cadena en el conjunt. S'utilitzen per buscar, editar, o manipular text i dades.

La sintaxi de les expressions regulars en l'API **java.util.regex** és molt similar a la del llenguatge Perl.

El paquet **java.util.regex** es compon fonamentalment de tres classes: **Pattern, Matcher** i **PatternSyntaxException.**

- Un objecte de **Pattern** és una representació compilada d'una expressió regular. La classe Pattern no proporciona constructors públics. Per crear un patró, cal invocar algun dels seus mètodes **public static compile()**, els quals retornen un objecte **Pattern.** Aquests mètodes accepten una expressió regular com a primer argument.
- Un objecte **Matcher** és el motor que interpreta el patró i realitza comprovacions de concordança contra una cadena d'entrada. Igual que la classe Pattern, Matcher no defineix cap constructor públic. Per obtenir un objecte Matcher cal invocar el mètode **matcher()** en un objecte Pattern.
- Un objecte **PatternSyntaxException** és una excepció sense control que indica un error de sintaxi en un patró d'expressió regular.

Per tal de provar les expressions regulars utilitzarem el següent programa:

```
/**
* RegexTest.java
* Test per provar patrons d'expressions regulars
* @author: Jose Moreno
*/
import java.util.Scanner;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class RegexTest {

   public static void main(String[] args){
          Scanner scan = new Scanner(System.in);
          String regexp;  //expressió regular del patró
          String cadena;  //cadena a verificar
          boolean sortir = false;  //flag per sortir del programa
       do {
         regexp = null;
         //entrar l'expressió regular
         System.out.print("Entra una expressio regular: ");
                regexp = scan.next();
                //crear el patró amb l'expressió sol·licitada
         Pattern patro = Pattern.compile(regexp);
         //entrar la cadena a verificar
         System.out.print("Entra una cadena: ");
                cadena = scan.next();
                //crear el verificador
         Matcher matcher = patro.matcher(cadena);
                //començar la verificació
```

```
        boolean trobat = false;
        while (matcher.find()) {
                        System.out.format(
                                "Trobat el text " + "\"%s\" entre els indexs " +
            "%d i %d.\n",
            matcher.group(), matcher.start(), matcher.end()
          );
          trobat = true;
        }
        if(!trobat){
          System.out.println("No s'han trobat coincidències");
        }
      } while (!sortir);
   }
}
```

Les expressions regulars estan formades pels següents tipus de components:

- Literals de cadena
- Classes de caràcters
- Caràcters predefinits
- Quantificadors

Literals de cadena

És la forma més simple d'expressió regular. Es considera que cada caràcter d'un String de longitud n caràcters ocupa una cel·la, entre la 0 i la n (tot i que els caràcters realment ocupen de la 0 a la n-1).

Per verificar la concordança, es comparen la cadena i el patró cel·la a cel·la. Un cop trobada una concordança, es consumeixen les cel·les que concorden i es continua buscant concordances amb el que queda del String.

Es poden fer servir **metacaràcters**:

```
<([{\^-=$!|]})?*+.>
```

Aquests metacaràcters tenen un significat especial llevat que vagin precedits del símbol de ***backslash*** **'\'**.

També es poden tancar els caràcters a buscar entre **\Q** i **\E**.

Classes de caràcters

Són conjunts de caràcters que poden donar concordança positiva per a un únic caràcter d'un String d'entrada donat.

Expressió	Descripció
[abc]	Classe senzilla: a, b, or c
[^abc]	Negació: Qualsevol caràcter llevat de a, b, or c
[a-zA-Z]	Rang: a fins z, o A fins Z, inclosos
[a-d[m-p]]	Unió: a fins d, o m fins p: [a-dm-p]
[a-z&&[def]]	Intersecció: caràcters inclosos en ambdós grups
[a-z&&[^bc]]	Substracció: a fins z, llevat de b and c
[a-z&&[^m-p]]	Substracció: a fins z, i no m fins p

Classes de caràcters predefinides

Expressió	Descripció

.	Qualsevol caràcter
\d	Un dígit [0-9]
\D	Un caràcter no dígit: [^0-9]
\s	Un espai en blanc: [\t\n\x0B\f\r]
\S	Un caràcter no espai en blanc: [^\s]
\w	Un caràcter de paraula: [a-zA-Z_0-9]
\W	Un caràcter no de paraula [^\w]

Per definir en Java una expressió regular predefinida cal doblar el '\' per indicar-ho al compilador.

```
private final String REGEX = "\\d";
```

Quantificadors

S'utilitzen per especificar el nombre d'aparicions del patró per buscar concordances.

Es poden combinar per donar tres comportaments diferents:

- **Greedy** (cobdiciós): forcen el comprovador a consumir tota la cadena abans d'intentar la primera comprovació.Si la comprovació amb tota la cadena falla, torna el primer caràcter i prova novamene la verificació, repetint el procés fins que troba una coincidència o no queden més carácters.
- **Reluctant** (reticent): comença per l'inici de la cadena i va consumint caràcter a caràcter fins a trobar una coincidència. L'últim intent el fa amb tota la cadena.
- **Possessive** (possessiu): Sempre consumeix tota la cadena amb un únic intent.

Greedy	Reluctant	Possessive	Significat
X?	X??	X?+	X, 0-1
X*	X*?	X*+	X, 0-*
X+	X+?	X++	X, 1-*
X{n}	X{n}?	X{n}+	X, exactament n vegades
X{n,}	X{n,}?	X{n,}+	X, almenys n vegades
X{n,m}	X{n,m}?	X{n,m}+	X, entre n i m vegades, ambdues incloses

Delimitadors de contorn

Serveixen per fer verificacions sobre la ubicació.

Expressió	Descripció
^	Inici de línia
$	Final de línia
\b	Contorn de paraula
\B	No contorn de paraula
\A	Inici de l'entrada
\G	Final de la coincidència anterior
\Z	Final de l'entrada, excepte pel terminador final, si és el cas
\z	Final de l'entrada

Mètodes de la classe Pattern

Pattern amb flags

La classe Pattern admet un mètode compile() que permet especificar modificadors (*flags*) al seu comportament.

```
final int flags = Pattern.CASE_INSENSITIVE | Pattern.UNICODE_CASE;
Pattern pattern = Pattern.compile("aa", flags);
```

El paràmetre de flags és una màscara de bits que pot incloure els camps *public static* de la classe:

- **Pattern.CANON_EQ**. Habilita l'equivalència canònica. Per defecte està desactivat.
- **Pattern.CASE_INSENSITIVE**. Habilita la distinció entre majúscules i minúscules assumint que s'està utilitzant el joc de caràcters US-ASCII.
- **Pattern.COMMENTS**. Permet espais en blanc i comentaris en el patró
- **Pattern.DOTALL**. Per defecte desactivat. Si s'activa, habilita que el punt (.) accepti, a més de qualsevol caràcter, el terminador de línia.
- **Pattern.LITERAL**. Habilita que el patró sigui considerat com a un literal de caràcters, eliminant el significat especials dels seus metacaràcters i seqüències d'escapament. Els flags CASE_INSENSITIVE i UNICODE_CASE matenen, però, el seu efecte.
- **Pattern.MULTILINE**. Habilita el mode multilinía. Per defecte està desactivat.
- **Pattern.UNICODE_CASE**. Habilita mode UNICODE per treballar en combinació amb el flag CASE_INSENSITIVE.
- **Pattern.UNIX_LINES**. Habilita els canvis de línia a l'estil Unix ('\n').

És possible habilitar alguns *flags* amb **expressions incloses als patrons**:

Flag	Expressió
Pattern.CANON_EQ	Cap
Pattern.CASE_INSENSITIVE	(?i)
Pattern.COMMENTS	(?x)
Pattern.MULTILINE	(?m)
Pattern.DOTALL	(?s)
Pattern.LITERAL	Cap
Pattern.UNICODE_CASE	(?u)
Pattern.UNIX_LINES	(?d)

Mètode Pattern.matches(String)

```
String regexp = "\\d";
String s = "1";
boolean b = Pattern.matches(s,regexp) //b=true
```

Mètode Pattern.split(String)

Aquest mètode d'instància de la classe Pattern retorna un array de String amb un element per cada fragment en què queda dividida la cadena original per les coincidències amb el patró.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class SplitDemo {

   private static final String REGEX = ":";
   private static final String INPUT = "one:two:three:four:five";

   public static void main(String[] args) {
      Pattern p = Pattern.compile(REGEX);
      String[] items = p.split(INPUT);
      for(String s : items) {
         System.out.println(s);
      }
   }
}
```

La sortida de l'exemple anterior seria:

```
one
two
three
four
five
```

public static String quote(String s)

Retorna un String que es pot utilitzar per crear un patró que concordi amb la cadena passada com a paràmetre. Si aquesta conté metacaràcters o caràcters d'escapament, perden el seu significat.

```
String cadena = “abc123ef45”;
String patro = Pattern.quote(cadena);
```

Mètodes de la classe String per verificar patrons

La classe String també conté mètodes per treballar amb expressions regulars i aplicar patrons.

public boolean matches(String regexp)

Verifica si la cadena encaixa amb l'expressió regular passada com a paràmetre.

La invocació str.matches(regex) és equivalent a Pattern.matches(regex, str).

public String [] split(String regexp, int limit=0)

Divideix la cadena utilitzant com a divisors els caràcters que encaixen amb l'expressió regular, amb el límit de blocs indicat. Si s'invoca sense el segon paràmetre, es considera 0.

La invocació str.split(regex, n) equival a Pattern.compile(regex).split(str, n).

public String replace(CharSequence target,CharSequence replacement)

Reemplaça cada subcadena que encaixi amb el literal *target* amb el literal *replacement*, procedint des del començament de la cadena fins al final.

Mètodes de la classe Matcher

Mètodes d'índex

Informen de la posició on s'ha trobat la coincidència.

- **public int start()**: posició inicial de la coincidència anterior.
- **public int start(int group)**: posició inicial de la subseqüència capturada pel grup donat durant l'anterior operació de comprovació.
- **public int end()**: desplaçament des del darrer caràcter de la darrera verificació.
- **public int end(int group)**: desplaçament des del darrer caràcter de la subseqüència capturada pel grup donat durant l'anterior operació de comprovació.

Mètodes d'anàlisi

Informen si la cadena d'ajusta al patró.

- **public boolean lookingAt()**: Intenta verificar contra el patró la seqüència començant per començament de la regió. Retorna true si i només si un prefix de la seqüència entrada encaixa amb el patró
- **public boolean find()**: Intenta trobar la següent subseqüència que encaixa amb el patró.
- **public boolean find(int start)**: Reinicia el *matcher* i intenta trobar la següent subseqüència que encaixa amb el patró, començant a l'índex especificat.
- **public boolean matches()**: Intenta verificar tota la regió contra el patró.

Mètodes de reemplaçament

Són útils per reemplaçar text en una cadena.

- **public Matcher appendReplacement(StringBuffer sb, String replacement)**

```
Pattern p = Pattern.compile("cat");
Matcher m = p.matcher("one cat two cats in the yard");
StringBuffer sb = new StringBuffer();
while (m.find()) {
   m.appendReplacement(sb, "dog");
}
m.appendTail(sb);
System.out.println(sb.toString()); //one dog two dogs in the yard
```

- **public StringBuffer appendTail(StringBuffer sb)**: S'utilitza després de appendReplacement() per afegir el que queda de la seqüència d'entrada.
- **public String replaceAll(String replacement)**: Reemplaça totes les ocurrències de la cadena entrada que encaixa amb el patró per la cadena donada replacement.
- **public String replaceFirst(String replacement)**: Reemplaça la primera ocurrència de la seqüència d'entrada que encaixa amb el patró per la cadena donada.
- **public static String quoteReplacement(String s)**: Retorna un literal que es pot utilitzar com a literal de reemplaçament per al mètode appendReplacement().

Mètodes de la classe String per fer reemplaçaments

- **public String replaceFirst(String regex, String replacement)**: Reemplaça el primer substring que s'ajusta a l'expressió regular amb la cadena replacement especificada. La invocació str.replaceFirst(regex, repl) equival a Pattern.compile(regex).matcher(str).replaceFirst(repl).
- **public String replaceAll(String regex, String replacement)**: Reemplaça cada substring que s'ajusta a l'expressió regular amb la cadena replacement especificada. La invocació str.replaceAll(regex, repl) equival a Pattern.compile(regex).matcher(str).replaceAll(repl).

Mètodes de la classe PatternSyntaxException

PatternSyntaxException és una excepció no verificada que indica un error en un patró d'expressió regular. Proporciona els següents mètodes per informar de l'error:

- **public String getDescription()**: Retorna la descripció de l'error.
- **public int getIndex()**: Retorna la posició on s'ha produït l'error.
- **public String getPattern()**: Retorna el patró d'expressió regular que ha produït l'error.
- **public String getMessage()**: Retorna una cadena multilínia que conté la descripció de l'error de sintaxi i l'índex, el patró d'expressió regular erroni i una indicació visual de l'índex d'error en el patró.