Regulární výrazy

Regulární výrazy jsou určeny pro práci s textovými řetězci, jsou součástí J2SDK až od verze 1.4, v předchozích verzích je potřeba použít některou z externích knihoven, např. knihovnu ORO ze http://jakarta.apache.org/, která je volně k použití.

Základní myšlenkou je porovnání textového řetězce s předpřipraveným vzorem.Nejčastěji se zjišťuje, zda řetězec odpovídá vzoru, lze také rozdělit řetězec na části (rozdělovačem je zadaný vzor) nebo části řetězce odpovídající vzoru nahradit jiným řetězcem.

Vzory

Vzory jsou základní prvky pro vytváření regulárních výrazů – vzorem se popisuje, jak má přípustný řetězec vypadat. Následuje přehled základních pravidel pro vytváření vzorů, další možnosti jsou uvedeny v dokumentaci.

Jednotlivé konkrétní znaky:

x	konkrétní znak
\\	zpětné lomítko (jeden znak)
\0*n*	znak zadaný pomocí oktalové soustavy (0-377)
\x*hh*	znak zadaný pomocí hexadecimální soustavy
\u*hhhh*	znak z Unicode zadaný pomocí hexadecimální soustavy
\t	tabulátor
\n	znak LF (linefeed), též se nazývá newline, obvykle označuje konec řádky,
\e	znak escape

Množiny znaků, tj. jeden ze znaků v zadané množině

[*abc***]**	jednoduchá množina (tj. jeden ze znaků a b c)
[^*abc***]**	negace, tj. žádný ze znaků a b c,
[*a-zA-Z***]**	rozsah znaků (znaky a až z a znaky A-Z)

Zástupné znaky, tj. znaky, která zastupují nahrazují jeden z množiny znaků,

.	(tečka) libovolný znak s výjimkou konce řádku
\d	číslice, tj. [0-9]
\D	nečíslice, tj. [^0-9]
\s	„netisknutelné“ znaky, tj. mezera, tabulátor, konec řádku a konec stránky
\S	opak \s
\w	písmena, číslice a podtržítko, tj. [a-zA-Z0-9_]
\W	opak \w

Označení hranic

^	začátek řetězce
$	konec řetězce
\b	hranice slova

Znaky pro vyjádření počtu opakování předchozího znaku (množiny znaků)

?	předchozí znak bude 1x nebo 0x,
*****	0 a více opakování předchozího znaku,
+	předchozí znak je minimálně jedenkrát,
{*n***}**	přesně *n* opakování předchozího znaku,
{*n***,}**	minimálně *n* opakování předchozího znaku,
{*m***,***n***}**	minimálně *m* a maximálně *n* opakování předchozího znaku,

Další operátory

XY	znaky X a Y musí být vedle sebe,
X|Y	buď znak X nebo znak Y,
(X**)**	označení skupiny,
n	obsah *n*-té skupiny,

Příklady vzorů[1]

Popisy odpovídají použití metody find().

ahoj	řetězec, který obsahuje podřetězec ahoj,
\bahoj\b	řetězec, který obsahuje slovo ahoj (jsou zde uvedeny hranice slova),
`^\s*$`	"prázdný" řetězec, tj. řetězec který je prázdný, či obsahuje pouze mezery nebo tabulátory,
`\s+$`	mezery a tabulátory na konci řetězce,
`^[a-wyz].*`	řetězec začínající písmeny a až z s výjimkou písmene x,
`\s(try)|(catch)\s`	řetězec obsahuje slovo try či catch (nebo oboje),

Třídy a metody

Regulární výrazy jsou v Javě soustředěny do balíčku java.util.regex.

Třída Pattern má tyto základní funkce:

- kontrola a „překlad" vzoru do interní formy pomocí metody **public static Pattern compile(String regex)** - při překladu může vzniknout výjimka **PatternSyntaxException** (potomek RunTimeException, tj. nemusí se povinně odchytávat),
- porovnání vzoru s konkrétním řetězcem a vytvoření instance třídy Matcher pomocí metody **public Matcher matcher()**,
- rozdělení řetězce dle vzoru do pole řetězců metodou **public String [] split()**,

Třída Matcher má tyto možnosti:

- metody pro prohledání dle vzoru, které vracejí hodnotu true či false:
  - **public boolean matches()** - porovnává celý řetězec se vzorem
  - **public boolean find()** – metoda hledá od začátku řetězce výskyt části, která odpovídá vzoru, při následném vyvolání metoda hledá další výskyt odpovídající vzoru (tj. lze postupně vyhledat všechny výskyty odpovídající vzoru), část odpovídající vzoru nalezenou při posledním vyvolání metody find lze získat pomocí metody **public String group()**,

[1] Pokud jsou vzory zadány jako řetězce ve zdrojovém kódu, je potřeba si uvědomit, že zpětné lomítko je speciální znak pro řetězcovou konstantu a proto je potřeba pro vložení zpětného lomítko vzoru uvést dvě zpětná lomítka.

- metody pro nahrazování části řetězce odpovídající vzoru jiným řetězce (substituce):
  - **public String replaceAll(String replacement)** - nahradí všechny výskyty odpovídající vzoru zadaným řetězcem,
  - **public String replaceFirst(String replacement)** - nahradí první výskyt odpovídající vzoru zadaným řetězcem,
- metody pro práci se skupinami zadanými ve vzoru (zadávají se pomocí kulatých závorek), tyto metody lze použít až po vyvolání některé z metod pro prohledávání:
  - **public String group(int group)** - vrací řetězce odpovídající příslušné skupině ve vzoru, skupiny se číslují od 1,
  - **public int groupCount()** - vrací počet skupin,

Typický postup použití regulárních výrazů vypadá následovně:

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
if ( m.matches() ) {
  ...
}
```

V JDK 1.4 došlo též k rozšíření třídy String o metody pracující s regulárními výrazy

- **public boolean matches(String regex) -** stejné jako Pattern.matches(regex, str)
- **public String replaceFirst(String regex, String replacement) -** stejné jako Pattern.compile(regex).matcher(str).replaceFirst(repl)
- **public String replaceAll(String regex, String replacement) -** stejné jako Pattern.compile(regex).matcher(str).replaceAll(repl)
- **public String[] split(String regex, int limit) -** stejné jako Pattern.compile(regex).split(str,n)
- **public String[] split(String regex) -** stejné jako Pattern.compile(regex).split(str)

Příklady aplikací

Grep

Tento program vypisuje ze zadaných souborů řádky, které obsahují zadaný vzor[2]. Program obsahuje metodu main, v rámci které se prochází příkazová řádka – přeloží se vzor a poté se postupně pro jednotlivé soubory vyvolává metoda grep. V rámci této metody se čtou jednotlivé věty a zjišťuje se výskyt zadaného vzoru v rámci každé řádky pomocí metody find.

```
import java.io.*;
import java.util.regex.*;

public class Grep {

    static Pattern pat;
```

[2] Ve Windows nelze některé přípustné vzory zadat na příkazové řádce díky omezením operačního systému.

```
    static void grep (String soubor) {
        try {
            BufferedReader vstup = new BufferedReader(
                                new FileReader (soubor));
            String radek;
            while ((radek = vstup.readLine()) != null ){
                if (pat.matcher(radek).find()){
                    System.out.println(soubor+": "+radek);
                }
            }
            vstup.close();
        }
        catch (IOException e) {
            System.out.println ("Chyba na vstupu souboru "+soubor);
        }
    }

    public static void main (String [] args) {
        if (args.length < 2 ) {
            System.out.println("grep VZOR SOUBOR ...");
            System.exit(1);
        }
        try {
            pat = Pattern.compile(args[0]);
        }
        catch (PatternSyntaxException pe) {
            System.out.println(pe.getMessage());
            System.exit(1);
        }
        for (int i=1; i< args.length; i++) {
            grep(args[i]);
        }
    }
}
```

Zrušení mezer

Následují ukázky kódu, které mají za úkol zrušit mezery z řetězce (vstupního řádku). Jsou použity metody třídy String. V případě použití v cyklu je efektivnější předkompilovat vzory a použít odpovídající metody tříd Pattern a Matcher.

zrušení mezer na začátku řádku:

```
radek = radek.replaceFirst("^\\s+","");
```

zrušení mezer na konci řádku:

```
radek = radek.replaceFirst("\\s+$","");
```

zrušení všech mezer v řádku:

```
radek = radek.replaceAll("\\s+","");
```

zrušení mezer na konci řádku pomocí tříd Pattern a Matcher:

```
static Pattern pat = Pattern.compile("\\s+");
radek = pat.matcher(radek).replaceAll("");
```

Vybrání podřetězce

Nechť zpracovávaný soubor má následující strukturu[3]:

ČísloSítě,EthKarta=BootSoubor
např.:
0x92660003,0020AF3096C2=3c509umc.sys
0x92660003,0020AF3096CF=3c509umc.sys
0x92660003,0020AF30AD33=3c509umc.sys

Následující část kódu vytiskne z tohoto vstupního souboru čísla eth. karet. Ve vzoru jsou uvedeny kulaté závorky, které označí skupinu v každé porovnávané řádce (před vybráním skupiny musí být volána metoda matches() nebo metoda find()). Obsah skupiny se získá pomocí metody group s parametrem označujícím pořadí skupiny v rámci vzoru (pozor, skupiny ve vzoru se počítají od 1).

```
Pattern pat = Pattern.compile(".*,(.*)=.*");
try {
    BufferedReader vstup = new BufferedReader(
                           new FileReader (soubor));
    String radek;
    while ((radek = vstup.readLine()) != null ) {
        Matcher matcher = pat.matcher(radek);
        if (matcher.matches()) {
            System.out.println(matcher.group(1));
        }
    }
    vstup.close();
}
catch (IOException e) {
    System.out.println ("Chyba na vstupu souboru "+soubor);
}
```

Split

Tento program používá stejný vstupní soubor jako předchozí příklad. Cílem programu je spočítat počet výskytů jednotlivých bootovacích souborů. Každý vstupní řádek se rozdělí pomocí metody split na jednotlivé části. Jako vzor se pro rozdělení použije řetězec "[,=]". Po rozdělení se zkontroluje, zda vznikly tři části, poté se zruší případné mezery. Pro počítání výskytů se používá pomocná třída Counter, jednotlivé záznamy jsou uloženy v mapě.

```
import java.io.*;
import java.util.regex.*;
import java.util.*;

class Counter {
    int pocet = 1;
    public String toString () {
        return Integer.toString(pocet);
    }
}
```

[3] Jedná se o konfigurační soubor bootconf.sys, který slouží pro bootovaní bezdiskových stanic v síti Novell Netware

```
public class Split {
    static Pattern pat = Pattern.compile("[,=]");

    public static void main (String [] args) {
        Map seznam = new HashMap();
        if (args.length == 0 ) {
            System.out.println("Nutno zadat jmeno souboru");
            System.exit(1);
        }
        for (int i=0; i< args.length; i++) {
            String soubor = args[i];
            try {
                BufferedReader vstup = new BufferedReader(
                                    new FileReader (soubor));
                String radek;
                while ((radek = vstup.readLine()) != null ) {
                    String [] vysl = pat.split(radek);
                    if (vysl.length != 3) continue;
                    String bootFile = vysl[2].replaceAll(" +","");
                    if (seznam.containsKey(bootFile)) {
                        ((Counter)seznam.get(bootFile)).pocet++;
                    }
                    else {
                        seznam.put(bootFile,new Counter());
                    }
                }
                vstup.close();
            }
            catch (IOException e) {
                System.out.println ("Chyba na vstupu
                                            souboru "+soubor);
            }
        };
        System.out.println(seznam);
    }
}
```