NGÔN NGỮ LẬP TRÌNH JAVA

Nội dung

Cơ bản về ngôn ngữ lập trình Java

- Lập trình hướng đối tượng
- Biến, từ khoá, kiểu dữ liệu
- Biểu thức, các cấu trúc điều khiển
- Dữ liệu kiểu mảng

Các khía cạnh nâng cao của lập trình hướng đối tượng

- Thiết kế lớp
- Thiết kế lớp nâng cao
- Xử lý ngoại lệ
- Generics
- Java Collection Framework
- Multithread& Concurrency
- Database Programming
- Network Programming
- Send mail
- Testing
- Regular Expression

REGULAR EXPRESSION

Nội dung

1. Giới thiệu về RegEx
2. Mô tả pattern tìm kiếm
    1. Xâu bình thường & Ký tự đặc biệt
    2. Các lớp ký tự
    3. Các bổ ngữ
    4. Nhóm ký tự
    5. So khớp biên
3. Tìm kiếm và thay thế

1. Giới thiệu về Regular Expression

- Biểu thức chính quy là một cách thức để mô tả một tập các xâu dựa trên các đặc tính chung của xâu.
- Có thể được sử dụng để tìm kiếm, thao tác, và soạn thảo các đoạn văn bản, dữ liệu
- Cần phải hiểu được cú pháp cơ bản của Reg Ex => có thể làm việc với bất cứ biểu thức chính quy trên bất kỳ ngôn ngữ nào.
- Reg Ex trong java tương tự Reg Ex trong ngôn ngữ Pearl

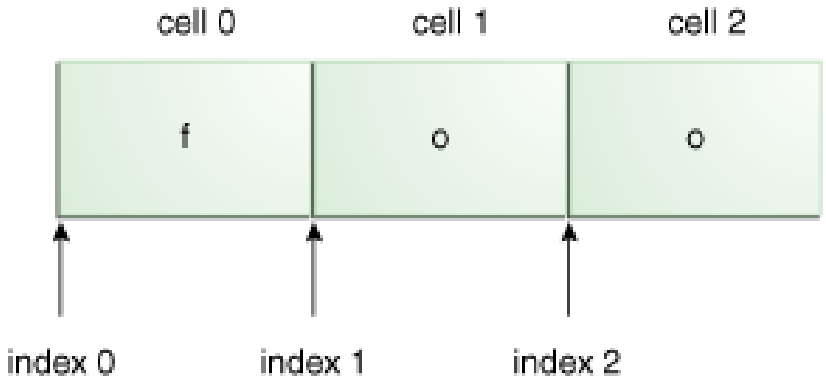
1. Giới thiệu về Regular Expression

- Trong Java : Reg Ex được hỗ trợ bởi gói java.util.regex
- Với 3 lớp: Pattern, Matcher, và PatternSyntaxExpression
- Pattern: 1 đối tượng lớp Pattern là một biểu diễn của 1 biểu thức chính quy
  - Pattern class ko cung cấp các public constructors
  - Để tạo 1 pattern, phải gọi một trong các phương thức “public static compile”
- Matcher: 1 đt lớp Matcher là một máy phiên dịch các pattern và sau đó thực hiện so khớp với một xâu đầu vào.
  - Matcher cũng ko cung cấp các public constructors
  - Phải gọi “matcher” method trên 1 đ/t pattern
- PatternSyntaxException : chứa các unchecked exception, chỉ định các lỗi cú pháp trong 1 reg ex pattern

RegexTestHarness

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

public class RegexTestHarness {

    public static void main(String[] args){
        Console console = System.console();
        if (console == null) {
            System.err.println("No console.");
            System.exit(1);
        }
        while (true) {

            Pattern pattern =
            Pattern.compile(console.readLine("%nEnter your regex: "));

            Matcher matcher =
            pattern.matcher(console.readLine("Enter input string to search: "));

            boolean found = false;
            while (matcher.find()) {
                console.format("I found the text" +
                    " \"%s\" starting at " +
                    "index %d and ending at index %d.%n",
                    matcher.group(),
                    matcher.start(),
                    matcher.end());
                found = true;
            }
            if(!found){
                console.format("No match found.%n");
            }
        }
    }
}
```

2. Mô tả xâu
2.1 Xâu bình thường & Ký tự đặc biệt




```
Enter your regex: foo
Enter input string to search: foo
I found the text foo starting at index 0 and ending at index 3.
```

```
Enter your regex: foo
Enter input string to search: foofoofoo
I found the text foo starting at index 0 and ending at index 3.
I found the text foo starting at index 3 and ending at index 6.
I found the text foo starting at index 6 and ending at index 9.
```

2. Mô tả xâu

2.1 Xâu bình thường & Ký tự đặc biệt

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

- Các ký tự đặc biệt hỗ trợ việc biểu diễn và so khớp các xâu trong pattern
- Có 2 cách để coi 1 ký tự đặc biệt là ký tự thông thường
  - Đặt ký tự backslash \ trước ký tự đặc biệt
  - Đặt trong cặp \Q và \E

2.2 Các lớp ký tự

Construct	Description
`[abc]`	a, b, or c (simple class)
`[^abc]`	Any character except a, b, or c (negation)
`[a-zA-Z]`	a through z, or A through Z, inclusive (range)
`[a-d[m-p]]`	a through d, or m through p: [a-dm-p] (union)
`[a-z&&[def]]`	d, e, or f (intersection)
`[a-z&&[^bc]]`	a through z, except for b and c: [ad-z] (subtraction)
`[a-z&&[^m-p]]`	a through z, and not m through p: [a-lq-z] (subtraction)

Dạng thức đơn giản nhất là đặt một tập các ký tự giữa cặp ngoặc [ ], thể hiện các tuỳ chọn so khớp

Simple class [bcr]at

```
Enter your regex: [bcr]at
Enter input string to search: bat
I found the text "bat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: cat
I found the text "cat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: rat
I found the text "rat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: hat
No match found.
```

Negation [^bcr]at

```
Enter your regex: [^bcr]at
Enter input string to search: bat
No match found.

Enter your regex: [^bcr]at
Enter input string to search: cat
No match found.

Enter your regex: [^bcr]at
Enter input string to search: rat
No match found.

Enter your regex: [^bcr]at
Enter input string to search: hat
I found the text "hat" starting at index 0 and ending at index 3.
```

Ranges [a-c]

```
Enter your regex: [a-c]
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: [a-c]
Enter input string to search: b
I found the text "b" starting at index 0 and ending at index 1.

Enter your regex: [a-c]
Enter input string to search: c
I found the text "c" starting at index 0 and ending at index 1.

Enter your regex: [a-c]
Enter input string to search: d
No match found.

Enter your regex: foo[1-5]
Enter input string to search: foo1
I found the text "foo1" starting at index 0 and ending at index 4.

Enter your regex: foo[1-5]
Enter input string to search: foo5
I found the text "foo5" starting at index 0 and ending at index 4.
```

Unions [0-4[6-8]]

```
Enter your regex: [0-4[6-8]]
Enter input string to search: 0
I found the text "0" starting at index 0 and ending at index 1.

Enter your regex: [0-4[6-8]]
Enter input string to search: 5
No match found.

Enter your regex: [0-4[6-8]]
Enter input string to search: 6
I found the text "6" starting at index 0 and ending at index 1.

Enter your regex: [0-4[6-8]]
Enter input string to search: 8
I found the text "8" starting at index 0 and ending at index 1.

Enter your regex: [0-4[6-8]]
Enter input string to search: 9
No match found.
```

Intersections

```
Enter your regex: [0-9&&[345]]
Enter input string to search: 3
I found the text "3" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[345]]
Enter input string to search: 4
I found the text "4" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[345]]
Enter input string to search: 5
I found the text "5" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[345]]
Enter input string to search: 2
No match found.

Enter your regex: [0-9&&[345]]
Enter input string to search: 6
No match found.
```

Intersection of two ranges

```
Enter your regex: [2-8&&[4-6]]
Enter input string to search: 3
No match found.

Enter your regex: [2-8&&[4-6]]
Enter input string to search: 4
I found the text "4" starting at index 0 and ending at index 1.

Enter your regex: [2-8&&[4-6]]
Enter input string to search: 5
I found the text "5" starting at index 0 and ending at index 1.

Enter your regex: [2-8&&[4-6]]
Enter input string to search: 6
I found the text "6" starting at index 0 and ending at index 1.

Enter your regex: [2-8&&[4-6]]
Enter input string to search: 7
No match found.
```

Subtraction

```
Enter your regex: [0-9&&[^345]]
Enter input string to search: 2
I found the text "2" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 3
No match found.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 4
No match found.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 5
No match found.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 6
I found the text "6" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 9
I found the text "9" starting at index 0 and ending at index 1.
```

Predefined Character Classes

Construct	Description
.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Predefined Character Classes (ex)

```
Enter your regex: .
Enter input string to search: @
I found the text "@" starting at index 0 and ending at index 1.

Enter your regex: .
Enter input string to search: 1
I found the text "1" starting at index 0 and ending at index 1.

Enter your regex: .
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \d
Enter input string to search: 1
I found the text "1" starting at index 0 and ending at index 1.

Enter your regex: \d
Enter input string to search: a
No match found.

Enter your regex: \D
Enter input string to search: 1
No match found.
```

Predefined Character Classes (ex)

```
Enter your regex: \D
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \s
Enter input string to search:
I found the text " " starting at index 0 and ending at index 1.

Enter your regex: \s
Enter input string to search: a
No match found.

Enter your regex: \S
Enter input string to search:
No match found.

Enter your regex: \S
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \w
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
```

2.3 Quantifiers ?*+{n,m}

RegEx	Ý nghĩa
X?	X, once or not at all
X*	X, zero or more times
X+	X, one or more times
X{n}	X, exactly *n* times
X{n,}	X, at least *n* times
X{n,m}	X, at least *n* but not more than *m* times

```
Enter your regex: a?
Enter input string to search:
I found the text "" starting at index 0 and ending at index 0.

Enter your regex: a*
Enter input string to search:
I found the text "" starting at index 0 and ending at index 0.

Enter your regex: a+
Enter input string to search:
No match found.
```

Zero-length matches

```
Enter your regex: a?
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
I found the text "" starting at index 1 and ending at index 1.

Enter your regex: a*
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
I found the text "" starting at index 1 and ending at index 1.

Enter your regex: a+
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
```

Zero-length matches (cont)

```
Enter your regex: a?
Enter input string to search: aaaaa
I found the text "a" starting at index 0 and ending at index 1.
I found the text "a" starting at index 1 and ending at index 2.
I found the text "a" starting at index 2 and ending at index 3.
I found the text "a" starting at index 3 and ending at index 4.
I found the text "a" starting at index 4 and ending at index 5.
I found the text "" starting at index 5 and ending at index 5.

Enter your regex: a*
Enter input string to search: aaaaa
I found the text "aaaaa" starting at index 0 and ending at index 5.
I found the text "" starting at index 5 and ending at index 5.

Enter your regex: a+
Enter input string to search: aaaaa
I found the text "aaaaa" starting at index 0 and ending at index 5.
```

2.4. Nhóm ký tự với ( )

- Nhóm ký tự được sử dụng để nhóm nhiều ký tự lại như là một đơn vị duy nhất, đặt trong cặp ( )

```
Enter your regex: (dog){3}
Enter input string to search: dogdogdogdogdogdog
I found the text "dogdogdog" starting at index 0 and ending at index 9.
I found the text "dogdogdog" starting at index 9 and ending at index 18.

Enter your regex: dog{3}
Enter input string to search: dogdogdogdogdogdog
No match found.


Enter your regex: [abc]{3}
Enter input string to search: abccabaaaccbbbc
I found the text "abc" starting at index 0 and ending at index 3.
I found the text "cab" starting at index 3 and ending at index 6.
I found the text "aaa" starting at index 6 and ending at index 9.
I found the text "ccb" starting at index 9 and ending at index 12.
I found the text "bbc" starting at index 12 and ending at index 15.

Enter your regex: abc{3}
Enter input string to search: abccabaaaccbbbc
No match found.
```

2.5 So khớp biên

Boundary Construct	Description
^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

2.5 So khớp biên (cont)

- ^ matches the beginning of a line, and $ matches the end.

```
Enter your regex: ^dog$
Enter input string to search: dog
I found the text "dog" starting at index 0 and ending at index 3.

Enter your regex: ^dog$
Enter input string to search:       dog
No match found.

Enter your regex: \s*dog$
Enter input string to search:              dog
I found the text "            dog" starting at index 0 and ending at index 15.

Enter your regex: ^dog\w*
Enter input string to search: dogblahblah
I found the text "dogblahblah" starting at index 0 and ending at index 11.
```

2.5 So khớp biên (cont)

- To check if a pattern begins and ends on a word boundary (as opposed to a substring within a longer string), just use \b on either side; for example, \bdog\b

```
Enter your regex: \bdog\b
Enter input string to search: The dog plays in the yard.
I found the text "dog" starting at index 4 and ending at index 7.

Enter your regex: \bdog\b
Enter input string to search: The doggie plays in the yard.
No match found.
```

2.5 So khớp biên (cont)

- To match the expression on a non-word boundary, use \B instead:

```
Enter your regex: \bdog\B
Enter input string to search: The dog plays in the yard.
No match found.

Enter your regex: \bdog\B
Enter input string to search: The doggie plays in the yard.
I found the text "dog" starting at index 4 and ending at index 7.
```

2.5 So khớp biên (cont)

- To require the match to occur only at the end of the previous match, use \G:

```
Enter your regex: dog
Enter input string to search: dog dog
I found the text "dog" starting at index 0 and ending at index 3.
I found the text "dog" starting at index 4 and ending at index 7.

Enter your regex: \Gdog
Enter input string to search: dog dog
I found the text "dog" starting at index 0 and ending at index 3.
```

3. Tìm kiếm và thay thế

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

public class RegexeFindReplace {
   public static void main(String[] args) {
      String input = "This is an apple. These are 33 (Thirty-three) apples";
      String regexe = "apple";         // pattern to be matched
      String replacement = "orange";  // replacement pattern

      // Step 1: Allocate a Pattern object to compile a regexe
      Pattern pattern = Pattern.compile(regexe, Pattern.CASE_INSENSITIVE);

      // Step 2: Allocate a Matcher object from the pattern, and provide the input
      Matcher matcher = pattern.matcher(input);

      // Step 3: Perform the matching and process the matching result
      String output = matcher.replaceAll(replacement);     // all matches
      //String output = matcher.replaceFirst(replacement); // first match only
      System.out.println(output);
   }
}
```

Renames files of a give directory

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

public class RegexeRenameFiles {
   public static void main(String[] args) {
      String regexe = ".class$";   // ending with ".class"
      String replacement = ".out"; // replace with ".out"

      // Allocate a Pattern object to compile a regexe
      Pattern pattern = Pattern.compile(regexe, Pattern.CASE_INSENSITIVE);
      Matcher matcher;

      File dir = new File("d:\\temp");  // directory to be processed
      int count = 0;
      File[] files = dir.listFiles();   // list all files and dirs
      for (File file : files) {
         if (file.isFile()) {   // file only, not directory
            String inFilename = file.getName();    // get filename, exclude path
            matcher = pattern.matcher(inFilename); // allocate Matches with input
            if (matcher.find()) {
               ++count;
               String outFilename = matcher.replaceAll(replacement);
               System.out.print(inFilename + " -> " + outFilename);

               if (file.renameTo(new File(dir + "\\" + outFilename))) {  // execute rename
                  System.out.println(" SUCCESS");
               } else {
                  System.out.println(" FAIL");
               }
            }
         }
      }
      System.out.println(count + " files processed");
   }
}
```