Regular Expression trong Java

Java cung cấp **java.util.regex** package cho pattern so khớp với các Regular Expression. Các Regular Expression trong Java là tương tự với Ngôn ngữ lập trình Perl và rất dễ dàng để học.

Một Regular Expression là một dãy liên tục của các ký tự đặc biệt mà giúp bạn so khớp hoặc tìm kiếm chuỗi hoặc tập hợp các chuỗi khác, sử dụng một cú pháp riêng biệt trong một pattern. Chúng có thể được sử dụng để tìm, chỉnh sửa và thao tác text và dữ liệu.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: **Ví dụ về Regular Expression trong Java**.

Gói java.util.regex chủ yếu chứa 3 lớp sau:

- **Lớp Pattern:** Một đối tượng Pattern là một phép biểu diễn được biên dịch của một Regular Expression. Lớp Pattern không cung cấp một public constructor nào. Để tạo một pattern, bạn đầu tiên phải gọi một trong các phương thức biên dịch static chung của nó, mà sau đó sẽ trả về một đối tượng Pattern. Những phương thức này chấp nhận một Regular Expression như là tham số đầu tiên.

- **Lớp Matcher:** Một đối tượng Matcher là phương tiện mà thông dịch pattern và thực hiện so khớp các hoạt động với một chuỗi đầu vào. Như lớp Pattern, Matcher không định nghĩa một public constructor nào. Bạn nhận được một đối tượng Matcher bằng việc gọi phương thức matcher trên một đối tượng Pattern.

- **PatternSyntaxException:** Một đối tượng PatternSyntaxException là một exception (ngoại lệ) chưa được kiểm tra mà chỉ dẫn một lỗi cú pháp trong mẫu Regular Expression.

Capture các Group trong Java

Capturing Groups là một cách coi nhiều ký tự như là một đơn vị đơn. Chúng được tạo bằng việc xác định vị trí của các ký tự để được nhóm vào trong một tập hợp các dấu ngoặc đơn. Ví dụ, Regular Expression (dog) tạo một group đơn chứa các chữ cái là “d”, “o” và “g”.

Capturing Groups được đánh số bởi việc tính toán số dấu ngoặc đơn mở từ trái qua phải. Ví dụ, trong Expression ((A)(B(C))) có 4 nhóm:

- ((A)(B(C)))

- (A)
- (B(C))
- (C)

Để tìm bao nhiêu group có mặt trong Expression đó, bạn gọi phương thức groupCount trên một đối tượng Matcher. Phương thức groupCount trả về một int minh họa số Capturing Groups có mặt trong mẫu của đối tượng Matcher.

Cũng có một group đặc biệt, là group 0, mà luôn luôn biểu diễn toàn bộ expression. Group này không được bao gồm trong kết quả của phương thức groupCount.

Ví dụ:

Ví dụ sau minh họa cách để tìm một chuỗi ký số từ chuỗi chữ-số đã cho:

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

public class RegexMatches
{
    public static void main( String args[] ){

      // String to be scanned to find the pattern.
      String line = "This order was placed for QT3000! OK?";
      String pattern = "(.*)(\\d+)(.*)";

      // Create a Pattern object
      Pattern r = Pattern.compile(pattern);

      // Now create matcher object.
      Matcher m = r.matcher(line);
      if (m.find( )) {
         System.out.println("Found value: " + m.group(0) );
         System.out.println("Found value: " + m.group(1) );
         System.out.println("Found value: " + m.group(2) );
```

```
      } else {
         System.out.println("NO MATCH");
      }
   }
}
```

Nó sẽ cho kết quả sau:

```
Found value: This order was placed for QT3000! OK?
Found value: This order was placed for QT300
Found value: 0
```

Cú pháp cho Regular Expression trong Java

Bảng dưới đây liệt kê tất cả các cú pháp siêu ký tự cho Regular Expression có sẵn trong Java.

Subexpression	So khớp
^	So khớp với phần bắt đầu của dòng (line)
$	So khớp với phần cuối của dòng
.	So khớp với bất kỳ ký tự đơn nào ngoại trừ newline. Sử dụng tùy chọn m cũng cho phép nó so khớp với newline
[...]	So khớp với bất kỳ ký tự đơn nào trong dấu ngoặc vuông
[^...]	So khớp với bất kỳ ký tự đơn nào không trong dấu ngoặc vuông
\A	Phần bắt đầu của cả chuỗi
\z	Phần cuối của cả chuỗi

\Z	Phần cuối của cả chuỗi
re*	So khớp với 0 hoặc nhiều sự xuất hiện của expression đặt trước
re+	So khớp với 1 hoặc nhiều của cái gì đó ở trước
re?	So khớp với 0 hoặc 1 sự xuất hiện của expression đặt trước
re{ n}	So khớp một cách chính xác với n lần xuất hiện của Expression đặt trước
re{ n,}	So khớp với n lần xuất hiện hoặc nhiều hơn của Expression đặt trước
re{ n, m}	So khớp với ít nhất n và nhiều nhất m lần xuất hiện của Expression đặt trước
a| b	So khớp với hoặc a hoặc b
(re)	Nhóm các Regular Expression và ghi nhớ text đã so khớp
(?: re)	Nhóm các Regular Expression mà không ghi nhớ text đã so khớp
(?> re)	So khớp với patter độc lập mà không truy tích ngược (backtrack)
\w	So khớp với các ký tự từ
\W	So khớp với các ký tự không phải từ
\s	So khớp với khoảng trống trắng. Tương đương với [\t\n\r\f].
\S	So khớp với các ký tự không là khoảng trống trắng

\d	So khớp với các ký số. Tương đương với [0-9].
\D	So khớp với ký tự không là ký số
\A	So khớp với phần bắt đầu của chuỗi
\Z	So khớp với phần kết thúc của chuỗi. Nếu một newline tồn tại, nó so khớp với ngay trước newline.
\z	So khớp với phần kết thúc của chuỗi
\G	So khớp với điểm, nơi mà so khớp cuối cùng kết thúc
\n	Tham chiếu ngược để capture group số "n"
\b	So khớp với các giới hạn từ bên ngoài dấu ngoặc vuông. So khớp với phím lùi (0x08) khi ở trong dấu ngoặc vuông
\B	So khớp các giới hạn không phải từ
\n, \t, etc.	So khớp với các newline, carriage return, tab, …
\Q	Thoát (trích dẫn) tất cả ký tự tới \E
\E	Kết thúc trích dẫn bắt đầu từ \Q

Các phương thức của lớp Matcher trong Java

Dưới đây là danh sách các phương thức hữu ích:

Các phương thức về Index trong Java

Các phương thức về index cung cấp các giá trị chỉ mục hữu ích giúp tìm kiếm sự so khớp chính xác trong chuỗi đầu vào.

STT	Phương thức và Miêu tả
1	**public int start()** Trả về chỉ mục bắt đầu của so khớp trước
2	**public int start(int group)** Trả về chỉ mục bắt đầu của dãy phụ được nắm bắt bởi group đã cho trong hoạt động so khớp trước.
3	**public int end()** Trả về offset sau ký tự cuối cùng được so khớp
4	**public int end(int group)** Trả về offset sau ký tự cuối cùng của dãy phụ được nắm bắt bởi group đã cho trong hoạt động so khớp trước

Các phương thức Study trong Java

Các phương thức Study duyệt lại chuỗi input và trả về một Boolean chỉ rằng có hay không pattern được tìm thấy.

STT	Phương thức và Miêu tả
1	**public boolean lookingAt()** So khớp với dãy input, bắt đầu từ khu vực đó, với pattern
2	**public boolean find()**

	Tìm dãy tiếp theo của dãy input mà so khớp với pattern
3	**public boolean find(int start** Đặt lại Matcher này và sau đó cố gắng tìm dãy tiếp theo của dãy input mà so khớp với pattern, bắt đầu từ chỉ mục đã xác định
4	**public boolean matches()** So khớp toàn bộ khu vực với pattern

Các phương thức thay thế vị trí trong Java

Các phương thức này rất hữu ích để thay thế text trong một chuỗi input:

STT	Phương thức và Miêu tả
1	**public Matcher appendReplacement(StringBuffer sb, String replacement)** Triển khai một bước phụ thêm-và-thay thế có giới hạn
2	**public StringBuffer appendTail(StringBuffer sb)** Triển khai một bước phụ thêm-và-thay thế không giới hạn
3	**public String replaceAll(String replacement)** Thay thế mỗi dãy phụ của dãy input mà so khớp pattern với chuỗi thay thế đã cho
4	**public String replaceFirst(String replacement)** Thay thế dãy phụ đầu tiên của dãy input mà so khớp pattern với chuỗi thay thế đã cho
5	**public static String quoteReplacement(String s)** Trả về một String thay thế cho String đã xác định. Phương thức này tạo một String mà sẽ làm việc như là một sự thay thế literal trong phương thức appendReplacement của

	lớp Matcher.

Các phương thức *start* và *end* trong Java

Ví dụ đơn giản sau tính toán số lần mà từ "cats" xuất hiện trong chuỗi input:

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

public class RegexMatches
{
    private static final String REGEX = "\\bcat\\b";
    private static final String INPUT =
                                    "cat cat cat cattie cat";

    public static void main( String args[] ){
       Pattern p = Pattern.compile(REGEX);
       Matcher m = p.matcher(INPUT); // get a matcher object
       int count = 0;

       while(m.find()) {
         count++;
         System.out.println("Match number "+count);
         System.out.println("start(): "+m.start());
         System.out.println("end(): "+m.end());
      }
   }
}
```

Nó sẽ cho kết quả:

```
Match number 1
start(): 0
end(): 3
Match number 2
```

```
start(): 4
end(): 7
Match number 3
start(): 8
end(): 11
Match number 4
start(): 19
end(): 22
```

Bạn có thể thấy rằng ví dụ này sử dụng các giới hạn từ để bảo đảm rằng chữ cái "c" "a" "t" không đơn thuần là một chuỗi phụ trong một từ dài hơn. Nó cũng cung cấp một số thông tin hữu ích về nơi nào trong chuỗi input xuất hiện sự so khớp.

Phương thức *start* trả về chỉ mục bắt đầu của dãy phụ được nắm bắt bởi group đã cho trong hoạt động so khớp trước, và phương thức *end* trả về chỉ mục của ký tự cuối cùng được so khớp, cộng với 1.

Các phương thức *matches* và *lookingAt* trong Java

Cả hai phương thức **matches và lookingAt** so khớp một dãy input với một pattern. Tuy nhiên, sự khác nhau ở đây là phương thức matches yêu cầu toàn bộ dãy input để được so khớp, trong khi phương thức lookingAt thì không.

Cả hai phương thức luôn luôn bắt đầu tại điểm bắt đầu của chuỗi input. Dưới đây là ví dụ giải thích tính năng này:

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

public class RegexMatches
{
    private static final String REGEX = "foo";
    private static final String INPUT = "fooooooooooooooooo";
    private static Pattern pattern;
    private static Matcher matcher;
```

```
    public static void main( String args[] ){
       pattern = Pattern.compile(REGEX);
       matcher = pattern.matcher(INPUT);

       System.out.println("Current REGEX is: "+REGEX);
       System.out.println("Current INPUT is: "+INPUT);

       System.out.println("lookingAt(): "+matcher.lookingAt());
       System.out.println("matches(): "+matcher.matches());
   }
}
```

Nó sẽ cho kết quả sau:

```
Current REGEX is: foo
Current INPUT is: fooooooooooooooo
lookingAt(): true
matches(): false
```

Các phương thức *replaceFirst* và *replaceAll* trong Java

Các phương thức **replaceFirst và replaceAll** trong Java thay thế text mà so khớp với một Regular Expression đã cho. Như tên của chúng đã cho biết, phương thức replaceFirst thay thế sự xuất hiện so khớp đầu tiên, trong khi phương thức replaceAll thay thế tất cả so khớp.

Ví dụ sau đây giải thích tính năng này:

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

public class RegexMatches
{
    private static String REGEX = "dog";
    private static String INPUT = "The dog says meow. " +
                                  "All dogs say meow.";
    private static String REPLACE = "cat";
```

```
    public static void main(String[] args) {
       Pattern p = Pattern.compile(REGEX);
       // get a matcher object
       Matcher m = p.matcher(INPUT);
       INPUT = m.replaceAll(REPLACE);
       System.out.println(INPUT);
   }
}
```

Nó sẽ cho kết quả sau:

```
The cat says meow. All cats say meow.
```

Các phương thức *appendReplacement* và *appendTail*trong Java

Lớp Matcher cũng cũng cung cấp hai phương thức **appendReplacement và appendTail** để thay thế văn bản.

Ví dụ sau giải thích tính năng này:

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

public class RegexMatches
{
   private static String REGEX = "a*b";
   private static String INPUT = "aabfooaabfooabfoob";
   private static String REPLACE = "-";
   public static void main(String[] args) {
      Pattern p = Pattern.compile(REGEX);
      // get a matcher object
      Matcher m = p.matcher(INPUT);
      StringBuffer sb = new StringBuffer();
      while(m.find()){
```

```
         m.appendReplacement(sb,REPLACE);
      }
      m.appendTail(sb);
      System.out.println(sb.toString());
   }
}
```

Nó sẽ cho kết quả sau:

```
-foo-foo-foo-
```

Các phương thức lớp PatternSyntaxException trong Java

Một lớp PatternSyntaxException là một exception chưa được kiểm tra mà chỉ dẫn lỗi cú pháp (syntax error) trong một mẫu Regular Expression. Lớp PatternSyntaxException cung cấp các phương thức sau để giúp bạn xác định cái gì gây ra lỗi.

STT	Phương thức và Miêu tả
1	**public String getDescription()** Thu nhận sự miêu tả về lỗi
2	**public int getIndex()** Thu nhận chỉ mục của lỗi
3	**public String getPattern()** Thu nhận mẫu Regular Expression sai sót
4	**public String getMessage()** Trả về một chuỗi nhiều dòng chứa sự mô tả về syntax error và chỉ mục của nó, mẫu Regular Expression sai sót, và chỉ dẫn có thể nhìn thấy của chỉ mục lỗi trong pattern đó.

Để hiểu sâu hơn các khái niệm được trình bày trong chương này, mời bạn tham khảo loạt bài: **Ví dụ về Regular Expression trong Java**.