The Java™ Tutorials

Trail: Essential Classes
Lesson: Regular Expressions

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Methods of the Matcher Class

This section describes some additional useful methods of the `Matcher` class. For convenience, the methods listed below are grouped according to functionality.

Index Methods

Index methods provide useful index values that show precisely where the match was found in the input string:

- `public int start()`: Returns the start index of the previous match.
- `public int start(int group)`: Returns the start index of the subsequence captured by the given group during the previous match operation.
- `public int end()`: Returns the offset after the last character matched.
- `public int end(int group)`: Returns the offset after the last character of the subsequence captured by the given group during the previous match operation.

Study Methods

Study methods review the input string and return a boolean indicating whether or not the pattern is found.

- `public boolean lookingAt()`: Attempts to match the input sequence, starting at the beginning of the region, against the pattern.
- `public boolean find()`: Attempts to find the next subsequence of the input sequence that matches the pattern.
- `public boolean find(int start)`: Resets this matcher and then attempts to find the next subsequence of the input sequence that matches the pattern, starting at the specified index.
- `public boolean matches()`: Attempts to match the entire region against the pattern.

Replacement Methods

Replacement methods are useful methods for replacing text in an input string.

- `public Matcher appendReplacement(StringBuffer sb, String replacement)`: Implements a non-terminal append-and-replace step.
- `public StringBuffer appendTail(StringBuffer sb)`: Implements a terminal append-and-replace step.
- `public String replaceAll(String replacement)`: Replaces every subsequence of the input sequence that matches the pattern with the given replacement string.
- `public String replaceFirst(String replacement)`: Replaces the first subsequence of the input sequence that matches the pattern with the given replacement string.
- `public static String quoteReplacement(String s)`: Returns a literal replacement `String` for the specified `String`. This method produces a `String` that will work as a literal replacement `s` in the `appendReplacement` method of the `Matcher` class. The `String` produced will match the sequence of characters in `s` treated as a literal sequence. Slashes (`'\'`) and dollar signs (`'$'`) will be given no special meaning.

Using the `start` and `end` Methods

Here's an example, `MatcherDemo.java`, that counts the number of times the word "dog" appears in the input string.

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

public class MatcherDemo {

    private static final String REGEX =
        "\\bdog\\b";
    private static final String INPUT =
        "dog dog dog doggie dogg";
    public static void main(String[] args) {
       Pattern p = Pattern.compile(REGEX);
       //  get a matcher object
       Matcher m = p.matcher(INPUT);
       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());
      }
   }
}


OUTPUT:

Match number 1
start(): 0
end(): 3
Match number 2
start(): 4
end(): 7
Match number 3
start(): 8
end(): 11
```

You can see that this example uses word boundaries to ensure that the letters `"d" "o" "g"` are not merely a substring in a longer word. It also gives some useful information about where in the input string the match has occurred. The `start` method returns the start index of the subsequence captured by the given group during the previous match operation, and `end` returns the index of the last character matched, plus one.

Using the `matches` and `lookingAt` Methods

The `matches` and `lookingAt` methods both attempt to match an input sequence against a pattern. The difference, however, is that `matches` requires the entire input sequence to be matched, while `lookingAt` does not. Both methods always start at the beginning of the input string. Here's the full code, `MatchesLooking.java`:

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

public class MatchesLooking {

    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) {

        // Initialize
        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());
    }
}


Current REGEX is: foo
Current INPUT is: fooooooooooooooooo
```

```
lookingAt(): true
matches(): false
```

Using `replaceFirst(String)` and `replaceAll(String)`

The `replaceFirst` and `replaceAll` methods replace text that matches a given regular expression. As their names indicate, `replaceFirst` replaces the first occurrence, and `replaceAll` replaces all occurences. Here's the `ReplaceDemo.java` code:

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

public class ReplaceDemo {

    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);
    }
}


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

In this first version, all occurrences of `dog` are replaced with `cat`. But why stop here? Rather than replace a simple literal like `dog`, you can replace text that matches *any* regular expression. The API for this method states that "given the regular expression `a*b`, the input `aabfooaabfooabfoob`, and the replacement string `-`, an invocation of this method on a matcher for that expression would yield the string `-foo-foo-foo-`."

Here's the `ReplaceDemo2.java` code:

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

public class ReplaceDemo2 {

    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);
        INPUT = m.replaceAll(REPLACE);
        System.out.println(INPUT);
    }
}


OUTPUT: -foo-foo-foo-
```

To replace only the first occurrence of the pattern, simply call `replaceFirst` instead of `replaceAll`. It accepts the same parameter.

Using `appendReplacement(StringBuffer,String)` and `appendTail(StringBuffer)`

The `Matcher` class also provides `appendReplacement` and `appendTail` methods for text replacement. The following example, `RegexDemo.java`, uses these two methods to achieve the same effect as `replaceAll`.

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

public class RegexDemo {

    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);
        Matcher m = p.matcher(INPUT); // get a matcher object
        StringBuffer sb = new StringBuffer();
        while(m.find()){
            m.appendReplacement(sb,REPLACE);
        }
        m.appendTail(sb);
        System.out.println(sb.toString());
    }
}
```

```
OUTPUT: -foo-foo-foo-
```

Matcher Method Equivalents in `java.lang.String`

For convenience, the `String` class mimics a couple of `Matcher` methods as well:

- `public String replaceFirst(String regex, String replacement)`: Replaces the first substring of this string that matches the given regular expression with the given replacement. An invocation of this method of the form `str.replaceFirst(regex, repl)` yields exactly the same result as the expression `Pattern.compile(regex).matcher(str).replaceFirst(repl)`
- `public String replaceAll(String regex, String replacement)`: Replaces each substring of this string that matches the given regular expression with the given replacement. An invocation of this method of the form `str.replaceAll(regex, repl)` yields exactly the same result as the expression `Pattern.compile(regex).matcher(str).replaceAll(repl)`

Previous page: Methods of the Pattern Class
Next page: Methods of the PatternSyntaxException Class