

Java Regex Primer

William Denniss

April 5, 2004

Java Regex Primer

Since version 1.4, Java has had support for Regular Expressions in the core API. Java Regex follows the same basic principles used in other languages, just with different access methods, and some subtle differences with the patterns. This primer is aimed towards developers already familiar with regex in other languages wanting a brief outline of its support in Java. It may also be beneficial to developers learning regex if used in conjunction with detailed documentation explaining the construction of regex patterns.

Reading the javadoc for `java.util.regex.Pattern` is a must to see how the Java regex patterns are different from other languages such as Perl.

Most of the functions discussed herein are from the `java.util.regex.Matcher` class with a few from `java.util.regex.Pattern`. Reading this text in conjunction with the javadoc of those classes is advised.

Matching

The most straight forward regex operation is matching. That is testing a string to see if it matches a regex pattern as a whole. Example:

```
String regex = "[0-9]{4}-[0-9]{4}";
String content1 = "1234-3495";
String content2 = "1234-3495a";
```

```
Matcher m;
```

```
m = Pattern.compile(regex).matcher(content1);
System.out.println(m.matches()); //true
```

```
m = Pattern.compile(regex).matcher(content2);
System.out.println(m.matches()); //false
```

Java also provides two easier ways of performing the same match - the static method `Pattern.matches(String regex, CharSequence input)` and the `matches(String regex)` method of `String`.

Searching

Regex's other main use is for searching. This is much broader than simply validating an entire string against a pattern. Its functions include fast and easy pattern replacement, counting substring instances, and carving up delimited strings to name a few.

Find, Start & End

As well as matching the entire string, one can search a string to see if it contains a regex pattern. Example:

```
String regex = "[0-9]{4}-[0-9]{4}";
String content = "This is an example: 1234-3495";

Matcher m = Pattern.compile(regex).matcher(content);

System.out.println(m.find()); //true
```

When `find` is called, if a pattern is found, the start and end characters of the pattern can be retrieved using the `start()` and `end()` methods. This can be used to extract parts of the string based on the pattern.

Subsequent calls of `find` move the “pointer” to the next match of the pattern (if any) and again, the `start()` and `end()` can be used. When there are no more patterns to be found, `find` returns “false”. Example:

```
String regex = "[0-9]{4}-[0-9]{4}";
String content = "This is an example: 1234-3495 ok? And again: 1234-3495.";

Matcher m = Pattern.compile(regex).matcher(content);

m.find(); // finds first pattern

System.out.println("First Find");

System.out.println("1 " + content.substring(0, m.start()));
System.out.println("2 " + content.substring(m.start(), m.end()));
System.out.println("3 " + content.substring(m.end()));

m.find(); // finds second pattern

System.out.println("Second Find");

System.out.println("4 " + content.substring(0, m.start()));
System.out.println("5 " + content.substring(m.start(), m.end()));
System.out.println("6 " + content.substring(m.end()));

yields
```

First Find

```
1 This is an example:
2 1234-3495
3 ok? And again: 1234-3495.
```

Second Find

```
4 This is an example: 1234-3495 ok? And again:
5 1234-3495
6 .
```

Split

The `Pattern` class offers the convenience method `split` so one can carve up a string without using `find`, `start`, and `end`. Example:

```
String regex = "[0-9]{4}-[0-9]{4}";
String content = "This is an example: 1234-3495 ok? And again: 1234-3495.";

String [] carved = Pattern.compile(regex).split(content);

for (int i = 0; i < carved.length; i++) {
    System.out.println(carved[i]);
}
```

yields

```
This is an example:
ok? And again:
.
```

Capturing Groups (Backreferences)

When `find` is called, the values in groups (bracketed sequences) are stored as backreferences, just as they are for other languages. They are accessed through `Matcher`'s `group(int)` method. The first group (0) is always the entire pattern. `groupCount()` can be used to get the total number of backreferences (NOTE: This doesn't include the first backreference, so in actual fact there are `groupCount()+1` groups that can be accessed by the `group(int)` method. Example:

```
String regex = "([0-9]{4})-([0-9]{4})";
String content = "This is an example: 1234-3495 ok?";

Matcher m = Pattern.compile(regex).matcher(content);

m.find();

for (int i = 0; i < m.groupCount() + 1; i++) {
    System.out.println(i + " " + m.group(i));
}
```

yields

```
0 1234-3495
1 1234
2 3495
```

Replacing

The `replaceAll` and `replaceFirst` methods can be used to do a bulk find and replace on the string. This is a much faster method than using `find`, `start` and `end` to achieve the same end.

One handy trick is that you can still use the callbacks by prepending the number with a `$` (eg `$1`) and just using it in the text. Example:

```
String regex = "[0-9]{4}-[0-9]{4}";
String content = "This is an example: 1234-3495 ok?";

Matcher m = Pattern.compile(regex).matcher(content);

m.find();

m.replaceAll("[\\$0]");
```

yields

```
This is an example: [1234-3495] ok?
```