Java: Regular Expression

Masudul Haque

What is Regular Expression?

- A Regular expression is a pattern describing a certain amount of text.

- A regular expression, often called a pattern, is an expression that describes a set of strings.
  - Wikipedia

Why Regex?

- Matching/Finding
- Doing something with matched text
- Validation of data
- Case insensitive matching
- Parsing data (ex: html)
- Converting data into diff. form etc.

java.util.regex package

- **Pattern**: To create a pattern, you must first invoke one of its public static compile methods, which will then return a Pattern object. These methods accept a regular expression as the first argument.
- **Matcher:** A Matcher object is the engine that interprets the pattern and performs match operations against an input string.
- **PatternSyntaxException**: A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern.

Metacharacters

\	Quote the next meta-character.
^	Match at the beginning
.	Match any character except new line
$	Match at the end, before new line
|	Alternation
()	Grouping
[]	Character class
{}	Match m to n times
+	One or more times
*	Zero or more times
?	Zero or one times

Non printable chars

\t	tab	(HT, TAB)
\n	newline	(LF, NL)
\r	return	(CR)
\f	form feed	(FF)
\a	alarm (bell)	(BEL)
\e	escape (think troff)	(ESC)
\033	octal char	(example: ESC)
\x1B	hex char	(example: ESC)
\x{263a}	long hex char	(example: Unicode SMILEY)
\cK	control char	(example: VT)
\N{name}	named Unicode character	

Character class

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)

Pre-defined character classes

Construct	Descriptions
.	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]

Posix Character class

Construct	Description
\p{Lower}	A lower-case alphabetic character: [a-z]
\p{Upper}	An upper-case alphabetic character:[A-Z]
\p{ASCII}	All ASCII:[\x00-\x7F]
\p{Alpha}	An alphabetic character:[\p{Lower}\p{Upper}]
\p{Digit}	A decimal digit: [0-9]
\p{Alnum}	An alphanumeric character:[\p{Alpha}\p{Digit}]
\p{Punct}	Punctuation: One of !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
\p{Graph}	A visible character: [\p{Alnum}\p{Punct}]
\p{Print}	A printable character: [\p{Graph}\x20]
\p{Blank}	A space or a tab: [\t]
\p{Cntrl}	A control character: [\x00-\x1F\x7F]
\p{XDigit}	A hexadecimal digit: [0-9a-fA-F]
\p{Space}	A whitespace character: [\t\n\x0B\f\r]

java.lang.Character class

Construct	Description
\p{javaLowerCase}	Equivalent to java.lang.Character.isLowerCase()
\p{javaUpperCase}	Equivalent to java.lang.Character.isUpperCase()
\p{javaWhitespace}	Equivalent to java.lang.Character.isWhitespace()
\p{javaMirrored}	Equivalent to java.lang.Character.isMirrored()

Other Classes

Construct	Description
\p{IsLatin}	A Latin script character (script)
\p{InGreek}	A character in the Greek block (block)
\p{Lu}	An uppercase letter (category)
\p{IsAlphabetic}	An alphabetic character (binary property)
\p{Sc}	A currency symbol
\P{InGreek}	Any character except one in the Greek block (negation)
[\p{L}&&[^\p{Lu}]]	Any letter except an uppercase letter (subtraction)

Quantifier

- **Greedy** quantifiers are considered "greedy" because they force the matcher to read in, or *eat*, the entire input string prior to attempting the first match.

- **Reluctant** quantifiers, however, take the opposite approach: They start at the beginning of the input string, then reluctantly eat one character at a time looking for a match. The last thing they try is the entire input string.

- **Possessive** quantifiers always eat the entire input string, trying once (and only once) for a match. Unlike the greedy quantifiers, possessive quantifiers never back off, even if doing so would allow the overall match to succeed.

Quantifier

Greedy	Reluctant	Possessive	Meaning
X?	X??	X?+	X, once or not at all
X*	X*?	X*+	X, zero or more times
X+	X+?	X++	X, one or more times
X{n}	X{n}?	X{n}+	X, exactly n times
X{n,}	X{n,}?	X{n,}+	X, at least n times
X{n,m}	X{n,m}?	X{n,m}+	X, at least n but not more than m times

Boundary Matches

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

Capturing Groups

Capturing groups are a way to treat multiple characters as a single unit.

- int groupCount()
- int start()
- int end()
- String group(int)

Backreferences

Pattern class constant

Constant	Equivalent Embedded Flag Expression
Pattern.CANON_EQ	None
Pattern.CASE_INSENSITIVE	(?i)
Pattern.COMMENTS	(?x)
Pattern.MULTILINE	(?m)
Pattern.DOTALL	(?s)
Pattern.LITERAL	None
Pattern.UNICODE_CASE	(?u)
Pattern.UNIX_LINES	(?d)

Matcher Class

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.

Matcher Class

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.

Matcher Class

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.