Matching Positions with Anchors and Boundaries

Victor Grazi
ORACLE JAVA CHAMPION, SPEAKER AND GEEK

@vgrazi

Introduction

Matching occurrences at (or not at):

- **Word boundaries**
- **Line boundaries**

Matching occurrences when following or preceding a pattern (or not):

- Look-behind
- Look-ahead
- Negative look-behind
- Negative look-ahead

Boundaries

\btom

tom-tom

Boundaries

tom\b

tom-tom

Boundaries

\btom\b

tom

Boundaries

\btom\b

tom-tom

Boundaries

\btom-tom\b

tom-tom

Look-around

Rich (?=Man)

consumes

anchors

Rich Man

Anchor/Boundary	Meaning
^	Beginning of a line
$	End of a line
\b	The position of, or after, a word character
\B	The position of, or after, a non-word character
\A	The beginning of the input string
\G	The end of the previous match
\Z	The end of the input except for possibly a final terminator
\z	The end of the input

Syntax	Meaning	Explanation
(?<=regex)	Look-behind	Asserts that the regex pattern immediately precedes the current capture position
(?=regex)	Look-ahead	Asserts that the regex pattern immediately follows the current capture position
(?<!regex)	Negative look-behind	Asserts that the regex pattern does not immediately precede the current capture position
(?!regex)	Negative look-ahead	Asserts that the regex pattern does not immediately follow the current capture position

Syntax	Meaning	Explanation
(?<=regex)	Look-behind	Asserts that the regex pattern immediately precedes the current capture position
(?<!regex)	Negative look-behind	Asserts that the regex pattern does not immediately precede the current capture position
(?=regex)	Look-ahead	Asserts that the regex pattern immediately follows the current capture position
(?!regex)	Negative look-ahead	Asserts that the regex pattern does not immediately follow the current capture position

Syntax	Meaning	Explanation
(?<=regex)	Look-behind	Asserts that the regex pattern immediately precedes the current capture position
(?<!regex)	Negative look-behind	Asserts that the regex pattern does not immediately precede the current capture position
(?=regex)	Look-ahead	Asserts that the regex pattern immediately follows the current capture position
(?!regex)	Negative look-ahead	Asserts that the regex pattern does not immediately follow the current capture position

Syntax	Meaning	Explanation
(?<=regex)	Look-behind	Asserts that the regex pattern immediately precedes the current capture position
(?<!regex)	Negative look-behind	Asserts that the regex pattern does not immediately precede the current capture position
(?=regex)	Look-ahead	Asserts that the regex pattern immediately follows the current capture position
(?!regex)	Negative look-ahead	Asserts that the regex pattern does not immediately follow the current capture position
(?>regex)	Atomic non-capture group	Does not capture or backtrack

Replace, using Look-arounds

Replace “cane” with “bar”...
but only if it follows “candy ”

Sugar cane, candy cane

consumes

anchors

(?<=candy)cane

“Sugar cane, candy cane”
.replaceAll(“(?<=candy)cane”,“bar”)

→ Sugar cane, candy bar

Multiple look-arounds in sequence

- Legal! (Since they just assert a position)
- Must not contradict...
- or no match is found...
- (No exception thrown)

Anchors & Boundaries

Both describe a position in an expression, but don't capture any characters.

"Anchors" refer to positional patterns, only needing to check a single character such as end of line or end of input.

"Boundaries" refer to word boundaries, checking the character before and after.

Summary

Matching occurrences at (or not at):

- **Word boundaries**
- **Anchors**

Matching occurrences when following or preceding a pattern (or not):

- Look-behind
- Look-ahead
- Negative look-behind
- Negative look-ahead

"The Greatest Regex Trick Ever!"