Onigmo (Oniguruma-mod) Regular Expressions正则表达式

Version 6.1.0

2016/12/25

syntax: ONIG_SYNTAX_RUBY (default)

1. Syntax elements 语法元素

- \ escape (enable or disable meta character) 转义 (允许或禁止元字符)
- | alternation (可选，或操作)
- (...) group (分组)
- [...] character class (字符集合)

2.Characters 元字符

- \t horizontal tab (水平制表符) (0x09)
- \v vertical tab (垂直制表符) (0x0B)
- \n newline (line feed) (换行符) (0x0A)
- \r carriage return (回车符) (0x0D)
- \b backspace (退格) (0x08)
- \f form feed (换页符) (0x0C)
- \a bell (响铃) (0x07)
- \e escape (退出) (0x1B)
- \nnn octal char (十进制字符) (encoded byte value)
- \xHH hexadecimal char (十六进制字符) (encoded byte value)
- \x{7HHHHHHH} wide hexadecimal char(宽十六进制字符) (character code point value)
- \uHHHH wide hexadecimal char (宽十六进制字符) (character code point value)
- \cx control char (控制字符) (character code point value)
- \C-x control char (控制字符) (character code point value)
- \M-x meta (x|0x80) (元) (character code point value)
- \M-\C-x meta control char (元控制字符) (character code point value)

(* \b as backspace is effective in character class only，只有在字符集的情况下，\b才能使用)

ONIG_SYNTAX_PERL: \o{nnn} (octal char十进制字符) can be also used也可以使用.

3.Character types 字符类型

- **.** any character (except newline) 任意字符(除了换行符)
- **\w** word character 单词字符

```
Not Unicode 非Unicode的情形:  alphanumeric and "_". 此时，\w匹配字母、数字和下划线
• Unicode 在Unicode的情形: General_Category - (Letter|Mark|Number|Connector_Punctuation) 此时，\w匹配内容(包含字母、数字、标点、连接符号)，不包含中文等其他字符

• It depends on ONIG_OPTION_ASCII_RANGE option that non-ASCII char includes or not. 是否包含"非ASCII码"，取决于选项ONIG_OPTION_ASCII_RANGE.
```

- **\W** non-word char 非单词字符
- **\s** whitespace char 空白字符

```
Not Unicode非Unicode的情形:  \t, \n, \v, \f, \r, \x20 包含水平制表符、换行符、垂直制表符、换页符、回车符
• Unicode在Unicode的情形: 0009, 000A, 000B, 000C, 000D, 0085(NEL),
•              General_Category -- Line_Separator
•                               -- Paragraph_Separator
•                               -- Space_Separator
•           It depends on ONIG_OPTION_ASCII_RANGE option that non-ASCII char   includes or not.是否包含"非ASCII码"，取决于选项 ONIG_OPTION_ASCII_RANGE.
```

- **\S** non-whitespace char 非空白字符
- **\d** decimal digit char 十进制的数字

```
Unicode: General_Category -- Decimal_Number

• It depends on ONIG_OPTION_ASCII_RANGE option that non-ASCII char includes or not.
```

- **\D** non-decimal-digit char 非十进制的数字
- **\h** hexadecimal-digit char [0-9a-fA-F] 十六进制的数字
- **\H** non-hexadecimal-digit char 非十六进制的数字

Character Property 字符属性，类似于POSIX字符集：\p{digit} 等同于[[:digit:]]

```
* \p{property-name} 使用\p{属性名}，可用属性名见下方，\p和\P为相反。
    * \p{^property-name}     (negative) 小写的\p{^pattern}等于大写的\P{pattern}
    * \P{property-name}      (negative)
```

property-name 可用属性名:

```
 + works on all encodings
   Alnum, Alpha, Blank, Cntrl, Digit, Graph, Lower, Print, Punct, Space, Upper, XDigit, Word, ASCII
 + works on EUC_JP, Shift_JIS, CP932, Hiragana, Katakana, Han, Latin, Greek, Cyrillic
 + works on UTF-8, UTF-16, UTF-32
   see UnicodeProps.txt
```

\p{Lower} 小写字母字符：[a-z]
\p{Upper} 大写字母字符：[A-Z]
\p{ASCII} 所有 ASCII：[\x00-\x7F]
\p{Alpha} 字母字符：[\p{Lower}\p{Upper}]
\p{Digit} 十进制数字：[0-9]
\p{Alnum} 字母数字字符：[\p{Alpha}\p{Digit}]
\p{Punct} 标点符号：!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
\p{Graph} 可见字符：[\p{Alnum}\p{Punct}]
\p{Print} 可打印字符：[\p{Graph}\x20]
\p{Blank} 空格或制表符：[\t]
\p{Cntrl} 控制字符：[\x00-\x1F\x7F]
\p{XDigit} 十六进制数字：[0-9a-fA-F]
\p{Space} 空白字符：[\t\n\x0B\f\r]
\p{InGreek} 所有希腊字符，比如α，β，γ，Σ.....
\p{Lu} 大写字母（开启区分大小写）
\p{Sc} 货币符号(比如$，€，¥...)
\P{InGreek} 所有字符，除去希腊字符(注意P是大写的)
Unicode Property：字符属于标点、空格、字母等等。每个Unicode字符只能属于唯一Unicode Property。.NET、Java、PHP和Ruby等语言支持。具体分类为：
字符\p{L}
\p{Ll}或\p{Lowercase_Letter}：小写字符（必须有大写的形式）。
\p{Lu}或\p{Uppercase_Letter}：大写字符（必须有小写的形式）。
\p{Lt}或\p{Titlecase_Letter}：全词首字母大写的字符。
\p{L&}或\p{Cased_Letter}：存在大小写形式的字符（Ll，Lu，Lt的组合）。
\p{Lm}或\p{Modifier_Letter}：音标修饰字符。
\p{Lo}或\p{Other_Letter}：不具有大小写的字符或字形。
附加符号\p{M}
\p{Mn}或\p{Non_Spacing_Mark}：与其他字符结合，不额外占用空间的字符，例如日耳曼语元音变音。
\p{Mc}或\p{Spacing_Combining_Mark}：与其他字符结合，额外占用空间的字符，例如马拉雅拉姆文#元音字母及附标。
\p{Me}或\p{Enclosing_Mark}：包含其他字符的字符，例如圆圈、方块。
分隔符p{Z}
\p{Zs}或\p{Space_Separator}：a whitespace character that is invisible, but does take up space。
\p{Zl}或\p{Line_Separator}：line separator character U+2028。
\p{Zp}或\p{Paragraph_Separator}：paragraph separator character U+2029。
符号\p{S}
\p{Sm}或\p{Math_Symbol}：数学符号。
\p{Sc}或\p{Currency_Symbol}：通货符号。
\p{Sk}或\p{Modifier_Symbol}：组合为其他字符的符号。
\p{So}或\p{Other_Symbol}：其他符号。
数值字符\p{N}
\p{Nd}或\p{Decimal_Digit_Number}：所有文本中的数字0至9字符，不含形意符号。
\p{Nl}或\p{Letter_Number}：看起来像字母的符号，包含罗马数字。
\p{No}或\p{Other_Number}：上角标或下角标数字，或者其他不属于0至9的数字。不含形意符号。
标点符号\p{P}
\p{Pd}或\p{Dash_Punctuation}：连字号或连接号。
\p{Ps}或\p{Open_Punctuation}：开括号。
\p{Pe}或\p{Close_Punctuation}：闭括号。
\p{Pi}或\p{Initial_Punctuation}：开引号。
\p{Pf}或\p{Final_Punctuation}：闭引号。
\p{Pc}或\p{Connector_Punctuation}：连接词的标点符号，如下划线。
\p{Po}或\p{Other_Punctuation}：其他标点符号。
其它符号\p{C}（包括不可见控制字符与未用码位）
\p{Cc}或\p{Control}：ASCII或Latin-1控制字符0x00-0x1F与0x7F-0x9F。
\p{Cf}或\p{Format}：不可见的格式化指示字符。
\p{Co}或\p{Private_Use}：私用码位。
\p{Cs}或\p{Surrogate}：UTF-16编码的代理对的一半。
\p{Cn}或\p{Unassigned}：未被使用的码位。

备注： \p{Punct} works slightly different on Unicode encodings and the other encodings. It matches the nine characters "$+<=>^`|~" on non-Unicode encodings (which is the same as [[:punct:]]), but not on Unicode encodings.

\p{Punct} 在Unicode编码和其它编码下，结果有所不同。在其它编码时，等同于[[:punct:]]，它匹配九个字符 "$+<=>^`|~" ； 在Unicode下则不然。
在Unicode编码下，\p{XPosixPunct} 匹配这九个字符(在EmEditor环境下测试，两者匹配结果相同，不明白为何)。

- **\R** Linebreak换行符

```
Unicode:    (?>\x0D\x0A|[\x0A-\x0D\x{85}\x{2028}\x{2029}])
• Not Unicode:  (?>\x0D\x0A|[\x0A-\x0D])
```

\X Extended Grapheme cluster 扩展图形集

```
Unicode:  See: Unicode Standard Annex #29 UNICODE TEXT SEGMENTATION
http://unicode.org/reports/tr29/
• Not Unicode:  (?>\x0D\x0A|(?m:.))
```

4.Quantifier 数量词

- **greedy 贪婪，匹配器被强制要求第一次尝试匹配时读入整个输入串，如果第一次尝试匹配失败，则从后往前逐个字符地回退并尝试再次匹配，直到匹配成功或没有字符可回退。**
  - **?** 1 or 0 times 出现0或1次
  - ***** 0 or more times 出现0或多次
  - **+** 1 or more times 出现1或多次
  - **{n,m}** at least n but no more than m times 出现最少n次，最多m次
  - **{n,}** at least n times 出现最少n次
  - **{,n}** at least 0 but no more than n times ({0,n}) 出现最少0次，最多n次，等同于{0，n}
  - **{n}** n times 出现n次
- **reluctant 懒惰，从输入串的首(字符)位置开始，在一次尝试匹配查找中只勉强地读一个字符，直到尝试完整个字符串。**
  - **??** 1 or 0 times 出现0或1次
  - ***?** 0 or more times 出现0或多次
  - **+?** 1 or more times 出现1或多次
  - **{n,m}?** at least n but not more than m times
  - **{n,}?** at least n times
  - **{,n}?** at least 0 but not more than n times (== {0,n}?)
- **possessive *(greedy and does not backtrack once match)*独占(等同贪婪，但是匹配到后不回溯)**
  - **?+** 1 or 0 times
  - ***+** 0 or more times
  - **++** 1 or more times

({n,m}+, {n,}+, {n}+ are possessive op. in ONIG_SYNTAX_JAVA and ONIG_SYNTAX_PERL only) ex. /a+/ === /(?>a)/

5.Anchors 边界/锚点

- **^** beginning of the line 行首
- **$** end of the line 行尾
- **\b** word boundary 单词边界
- **\B** non-word boundary 非单词边界
- **\A** beginning of string 字符串首
- **\Z** end of string, or before newline at the end 字符串尾，或换行符之前的末位
- **\z** end of string字符串尾
- **\G** where the current search attempt begins当前搜索试图开始的地方

6.Character class 字符集合

- **^...** negative class (lowest precedence) 取反，比如[^a-z]
- **x-y** range from x to y 范围,比如a-z
- **[...]** set (character class in character class) 分组，比如[aoeiu]
- **..&&..** intersection (low precedence, only higher than ^) 交集， [a-z&&[aeiou]]，示例 [a-w&&[^c-g]z] ==> ([a-w] AND ([^c-g] OR z)) ==> [abh-w]

  备注：If you want to use '[', '-', or ']' as a normal character in character class, you should escape them with ''.
  倘若要使用 '[', '-', 或 ']' ，需要使用''来转义，在EmEditor中用\转义。

  POSIX 字符集： 表达方式bracket括号 ([:xxxxx:], negate 取反[:^xxxxx:])**

- Not Unicode Case 非Unicode编码:

  alnum alphabet or digit char 字母和数字
  alpha alphabet 字母
  ascii code value: [0 - 127] ASCII码
  blank \t, \x20 空格
  cntrl 控制符
  digit 0-9 数字
  graph \x21-\x7E and all of multibyte encoded characters 图形符号
  lower 小写字母
  print \x20-\x7E and all of multibyte encoded characters 打印字符
  punct 拼音
  space \t, \n, \v, \f, \r, \x20 空白
  upper 大写
  xdigit 0-9, a-f, A-F 十六进制数字
  word alphanumeric, "_" and multibyte characters 单词

- Unicode Case Unicode编码:

  alnum Letter | Mark | Decimal_Number
  alpha Letter | Mark
  ascii 0000 - 007F
  blank Space_Separator | 0009
  cntrl Control | Format | Unassigned | Private_Use | Surrogate
  digit Decimal_Number
  graph [[:^space:]] && ^Control && ^Unassigned && ^Surrogate
  lower Lowercase_Letter

print [[:graph:]] | Space_Separator
punct Connector_Punctuation | Dash_Punctuation | Close_Punctuation | Final_Punctuation | Initial_Punctuation | Other_Punctuation | Open_Punctuation | 0024 | 002B | 003C | 003D | 003E | 005E | 0060 | 007C | 007E
space Space_Separator | Line_Separator | Paragraph_Separator | 0009 | 000A | 000B | 000C | 000D | 0085
upper Uppercase_Letter
xdigit 0030 - 0039 | 0041 - 0046 | 0061 - 0066 (0-9, a-f, A-F)
word Letter | Mark | Decimal_Number | Connector_Punctuation

It depends on ONIG_OPTION_ASCII_RANGE option and ONIG_OPTION_POSIX_BRACKET_ALL_RANGE option that POSIX brackets match non-ASCII char or not.

7.Extended groups 扩展分组

- **(?#...)** comment 注释
- **(?imxdau-imx)** option on/off 选项开关
  i: ignore case 忽略大小写
  m: multi-line (dot (.) also matches newline) 多行模式，点号匹配换行符
  x: extended form 扩展格式

  character set option (character range option) 字符设置选项(字符范围选项)
  d: Default (compatible with Ruby 1.9.3) 默认(兼容Ruby 1.9.3)
  \w, \d and \s doesn't match non-ASCII characters. 表达式\w,\d和\s不匹配非ASCII字符。
  \b, \B and POSIX brackets use the each encoding's rules.表达式\b,\B和POSIX字符集使用各自的编码规则。

  a: ASCII
  ONIG_OPTION_ASCII_RANGE option is turned on.
  \w, \d, \s and POSIX brackets doesn't match non-ASCII characters.
  \b and \B use the ASCII rules.
  u: Unicode
  ONIG_OPTION_ASCII_RANGE option is turned off.
  \w (\W), \d (\D), \s (\S), \b (\B) and POSIX brackets use the each encoding's rules.

- **(?imxdau-imx:subexp)** option on/off for subexp 表达式的选项开/关
- **(?:subexp)** non-capturing group 非捕获组
- **(subexp)** capturing group 捕获组
- **(?=subexp)** look-ahead肯定前视
- **(?!subexp)** negative look-ahead否定前视
- **(?<=subexp)** look-behind 肯定回顾
- **(?<!subexp)** negative look-behind 否定回顾

备注：Subexp of look-behind must be fixed-width. But top-level alternatives can be of various lengths. ex. (?<=a|bc) is OK. (?<=aaa(?:b|cd)) is not allowed.
In negative look-behind, capturing group isn't allowed, but non-capturing group (?:) is allowed.

前视的表达式必须固定宽度。但如果使用了或操作，顶层的长度可以变化，比如(?<=a|bc)是可以的，但 (?<=aaa(?:b|cd)) 是不可以的。

- **\K** keep 保持
  Another expression of look-behind. Keep the stuff left of the \K, don't include it in the result.
  回顾的另一种写法。(?<=几度)夕阳红 与几度\K 夕阳红是等效的

- **(?>subexp)** atomic group 原子组 no backtracks in subexp. 表达式中不回溯。默认的机制会进行回溯，降低了运行效率，而“原子组”则不进行回溯。回溯就是将字符串反复拆解匹配；而原子组则是最小单元，不拆解直接匹配。

- **(?subexp), (?'name'subexp)** define named group 命名组
  (Each character of the name must be a word character.) 命名的每个字符必须是单词。
  Not only a name but a number is assigned like a capturing group. 不仅是名称，编号也可以像捕获组一样分配。
  Assigning the same name to two or more subexps is allowed. 允许分配相同的名称给两个或多个表达式。

  示例：正则表达式：`(?<year>\d{4})-(?<date>\d{2}-(?<day>\d\d))`

- **(?(cond)yes-subexp), (?(cond)yes-subexp|no-subexp)** conditional expression 条件表达式 (有点复杂，还没搞清楚)
  Matches yes-subexp if (cond) yields a true value, matches no-subexp otherwise. 条件返回真值时，执行Yes表达式；返回假值时，执行No表达式。
  Following (cond) can be used: 可以使用以下的条件：

(n) (n >= 1) Checks if the numbered capturing group has matched something. 检查编号的捕获组是否已经匹配到内容。

(), ('name') Checks if a group with the given name has matched something. 检查一个命名组是否已经匹配到内容。

BUG: If the name is defined more than once, the left-most group is checked, but it should be the same as \k. 如果命名定义多次，最左边的组会被应用。但它应该与\k一样。

- **(?~subexp)** absence operator (experimental) 缺席运算符(实验)
  Matches any string which doesn't contain any string which matches subexp.表达式中的字符串不包含。 例如，(?~中)华 这个表达式，匹配中华牙膏中的华字；也匹配清华大学中的清华；匹配光华路中的光华。
  More precisely, (?~subexp) matches the complement set of a set which *.subexp.* matches. This is regular in the meaning of formal language theory.
  Similar to (?:(?!subexp).)*, but easy to write.

E.g.:示例： (?~abc) matches匹配: "", "ab", "aab", "ccdd", etc. It doesn't match不匹配: "abc", "aabc", "ccabcdd", etc.

\/*(?~*\/)*\/ matches C style comments: "/*/", "/ foobar */", etc.

\A\/*(?~*\/)*\/\z doesn't match "/**/ /". *This is different from \A\/*.* ?*\/\z which uses a reluctant quantifier (.*?).

Unlike (?:(?!abc).)*c, (?~abc)c matches "abc", because (?~abc) matches "ab". (?~) never matches.

Theoretical backgrounds are discussed in Tanaka Akira's paper and slide (both Japanese):

```
                  * Absent Operator for Regular Expression
                    https://staff.aist.go.jp/tanaka-akira/pub/prosym49-akr-
paper.pdf
                                          * 正規表現における非包含オペレータの提案
                    https://staff.aist.go.jp/tanaka-akira/pub/prosym49-akr-
presen.pdf
```

8 Backreferences 返回引用

When we say "backreference a group," it actually means, "re-match the same text matched by the subexp in that group."

\n \k \k'n' (n >= 1) backreference the nth group in the regexp
\k<-n> \k'-n' (n >= 1) backreference the nth group counting backwards from the referring position
\k \k'name' backreference a group with the specified name

When backreferencing with a name that is assigned to more than one groups, the last group with the name is checked first, if not matched then the previous one with the name, and so on, until there is a match.

- Backreference by number is forbidden if any named group is defined and ONIG_OPTION_CAPTURE_GROUP is not set.
- ONIG_SYNTAX_PERL: \g{n}, \g{-n} and \g{name} can also be used.
  If a name is defined more than once in Perl syntax, only the left-most group is checked.

backreference with recursion level

(n >= 1, level >= 0)

\k<n+level> \k'n+level'
\k \k'n-level'
\k<-n+level> \k'-n+level'
\k<-n-level> \k'-n-level'

\k<name+level> \k'name+level'
\k \k'name-level'

Refer a group on the recursion level relative to the referring position.

ex 1.

/\A(?|.|(?:(?.)\g\k))\z/.match("reee")
/\A(?|.|(?:(?.)\g\k<b+0>))\z/.match("reer")

\k<b+0> refers to the (?.) on the same recursion level with it.

ex 2.

```
r = Regexp.compile(<<'REGEXP'.strip, Regexp::EXTENDED)
(? \g \g* \g ){0}
(? < \g \s* > ){0}
(? [a-zA-Z_:]+ ){0}
(? [^<&]+ (\g | [^<&]+)* ){0}
(? </ \k<name+1> >){0}
\g
REGEXP

p r.match("fbbbf").captures
```

9 Subexp calls ("Tanaka Akira special") 表达式调用

When we say "call a group," it actually means, "re-execute the subexp in that group." “调用分组”的意思是“在分组内重新执行表达式”。

- \g<0> \g'0' call the whole pattern recursively
- \g \g'n' (n >= 1) call the nth group
- \g<-n> \g'-n' (n >= 1) call the nth group counting backwards from the calling position
- \g<+n> \g'+n' (n >= 1) call the nth group counting forwards from the calling position
- \g \g'name' call the group with the specified name
  - Left-most recursive calls are not allowed.

    ex. (?a|\gb) => error
    (?a|b\gc) => OK
  - Calls with a name that is assigned to more than one groups are not allowed in ONIG_SYNTAX_RUBY.
  - Call by number is forbidden if any named group is defined and ONIG_OPTION_CAPTURE_GROUP is not set.
  - The option status of the called group is always effective. ex. /(?-i:\g)(?i:(?a)){0}/.match("A")
  - ONIG_SYNTAX_PERL:
    Use (?&name), (?n), (?-n), (?+n), (?R) or (?0) instead of \g<>.
    Calls with a name that is assigned to more than one groups are allowed, and the left-most subexp is used.

10 Captured group 捕获组

Behavior of an unnamed group (...) changes with the following conditions.
(But named group is not changed.)

case 1. /.../ (named group is not used, no option)

(...) is treated as a capturing group.

case 2. /.../g (named group is not used, 'g' option)

(...) is treated as a non-capturing group (?:...).

case 3. /..(?..)../ (named group is used, no option)

(...) is treated as a non-capturing group.
numbered-backref/call is not allowed.

case 4. /..(?..)../G (named group is used, 'G' option)

(...) is treated as a capturing group.
numbered-backref/call is allowed.

where
g: ONIG_OPTION_DONT_CAPTURE_GROUP
G: ONIG_OPTION_CAPTURE_GROUP

('g' and 'G' options are argued in ruby-dev ML)

A-1. Syntax-dependent options 语法相关的选项

- ONIG_SYNTAX_RUBY
  (?m): dot (.) also matches newline
- ONIG_SYNTAX_PERL, ONIG_SYNTAX_JAVA and ONIG_SYNTAX_PYTHON
  (?s): dot (.) also matches newline
  (?m): ^ matches after newline, $ matches before newline
- ONIG_SYNTAX_PERL
  (?d), (?l): same as (?u)

A-2. Original extensions 原始扩展

- hexadecimal digit char type \h, \H
- named group (?...), (?'name'...)
- named backref \k
- subexp call \g, \g

A-3. Missing features compared with perl 5.18.0 与Perl相比缺少的特征

- \N{name}, \N{U+xxxx}, \N
- \l,\u,\L,\U, \C

\v, \V, \h, \H

- (?{code})
- (??{code})
- (?|...)
- (?[])
- (*VERB:ARG)
- \Q...\E
  This is effective on ONIG_SYNTAX_PERL and ONIG_SYNTAX_JAVA.

A-4. Differences with Japanized GNU regex(version 0.12) of Ruby 1.8

- add character property (\p{property}, \P{property})
- add hexadecimal digit char type (\h, \H)
- add look-behind
  (?<=fixed-width-pattern), (?<!fixed-width-pattern)
- add possessive quantifier. ?+, *+, ++
- add operations in character class. [], &&
  ('[' must be escaped as an usual char in character class.)
- add named group and subexp call.
- octal or hexadecimal number sequence can be treated as
  a multibyte code char in character class if multibyte encoding
  is specified.
  (ex. [\xa1\xa2], [\xa1\xa7-\xa4\xa1])
- allow the range of single byte char and multibyte char in character
  class.
  ex. /[a-<>]/ in EUC-JP encoding.
- effect range of isolated option is to next ')'.
  ex. (?:(?i)a|b) is interpreted as (?:(?i:a|b)), not (?:(?i:a)|b).
- isolated option is not transparent to previous pattern.
  ex. a(?i)* is a syntax error pattern.
- allowed unpaired left brace as a normal character.
  ex. /{/, /({)/, /a{2,3/ etc...
- negative POSIX bracket [:^xxxx:] is supported.
- POSIX bracket [:ascii:] is added.
- repeat of look-ahead is not allowed.
  ex. /(?=a)*/, /(?!b){5}/
- Ignore case option is effective to escape sequence.
  ex. /\x61/i =~ "A"

In the range quantifier, the number of the minimum is optional.
/a{,n}/ == /a{0,n}/
The omission of both minimum and maximum values is not allowed.
/a{,}/

- /{n}?/ is not a reluctant quantifier.
  /a{n}?/ == /(?:a{n})?/
- invalid back reference is checked and raises error.
  /\1/, /(a)\2/
- Zero-width match in an infinite loop stops the repeat,
  then changes of the capture group status are checked as stop condition.
  /(?:()|())\1\2/ =~ ""
  /(?:\1a|())/ =~ "a"

A-5. Features disabled in default syntax 默认语法中禁止的特征

- capture history

  (?@...) and (?@...)

  ex. /(?@a)*/.match("aaa") ==> [<0-1>, <1-2>, <2-3>]

  see sample/listcap.c file.

A-6. Problems 问题

- Invalid encoding byte sequence is not checked.

  ex. UTF-8

  - Invalid first byte is treated as a character.
    /./u =~ "\xa3"
  - Incomplete byte sequence is not checked.
    /\w+/ =~ "a\xf3\x8ec"

// END