Regular Expression Subroutines

Perl 5.10, PCRE 4.0, and Ruby 1.9 support regular expression subroutine calls. These are very similar to regular expression recursion. Instead of matching the entire regular expression again, a subroutine call only matches the regular expression inside a capturing group. You can make a subroutine call to any capturing group from anywhere in the regex. If you place a call inside the group that it calls, you'll have a recursive capturing group.

As with regex recursion, there is a wide variety of syntax that you can use for exactly the same thing. Perl uses `(?1)` to call a numbered group, `(?+1)` to call the next group, `(?-1)` to call the preceding group, and `(?&name)` to call a named group. You can use all of these to reference the same group. `(?+1)(?'name'[abc])(?1)(?-1)(?&name)` matches a string that is five letters long and consists only of the first three letters of the alphabet. This regex is exactly the same as `[abc](?'name'[abc])[abc][abc][abc]`.

PCRE was the first regex engine to support subroutine calls. `(?P<name>[abc])(?1)(?P>name)` matches three letters like `(?P<name>[abc])[abc][abc]` does. `(?1)` is a call to a numbered group and `(?P>name)` is a call to a named group. The latter is called the "Python syntax" in the PCRE man page. While this syntax mimics the syntax Python uses for named capturing groups, it is a PCRE invention. Python does not support subroutine calls or recursion. PCRE 7.2 added `(?+1)` and `(?-1)` for relative calls. PCRE 7.7 adds all the syntax used by Perl 5.10 and Ruby 2.0. Recent versions of PHP, Delphi, and R also support all this syntax, as their regex functions are based on PCRE.

The syntax used by Ruby 1.9 and later looks more like that of backreferences. `\g<1>` and `\g'1'` call a numbered group, `\g<name>` and `\g'name'` call a named group, while `\g<-1>` and `\g'-1'` call the preceding group. Ruby 2.0 adds `\g<+1>` and `\g'+1'` to call the next group. `\g<+1>(?<name>[abc])\g<1>\g<-1>\g<name>` and `\g'+1'(?'name'[abc])\g'1'\g'-1'\g'name'` match the same 5-letter string in Ruby 2.0 as the Perl example does in Perl. The syntax with angle brackets and with quotes can be used interchangeably.

JGsoft V2 supports all three sets of syntax. As we'll see later, there are differences in how Perl, PCRE, and Ruby deal with capturing, backreferences, and backtracking during subroutine calls. While they copied each other's syntax, they did not copy each other's behavior. JGsoft V2, however, copied their syntax and their behavior. So JGsoft V2 has three different ways of doing regex recursion, which you choose by using a different syntax. But these differences do not come into play in the basic examples on this page.

Boost 1.42 copied the syntax from Perl but its implementation is marred by bugs, which are still not all fixed in version 1.62. Most significantly, quantifiers other than `*` or `{0,}` cause subroutine calls to misbehave. This is partially fixed in Boost 1.60 which correctly handles `?` and `{0,1}` too.

Boost does not support the Ruby syntax for subroutine calls. In Boost `\g<1>` is a backreference—not a subroutine call—to capturing group 1. So `([ab])\g<1>` can match `aa` and `bb` but not `ab` or `ba`. In Ruby the same regex would match all four strings. No other flavor discussed in this tutorial uses this syntax for backreferences.

Matching Balanced Constructs

Recursion into a capturing group is a more flexible way of matching balanced constructs than recursion of the whole regex. We can wrap the regex in a capturing group, recurse into the capturing group instead of the whole regex, and add anchors outside the capturing group. `\A(b(?:m|(?1))*e)\z` is the generic regex for checking that a string consists entirely of a correctly balanced construct. Again, `b` is what begins the construct, `m` is what can occur in the middle of the construct, and `e` is what can occur at the end of the construct. For correct results, no two of `b`, `m`, and `e` should be able to match the same text. You can use an atomic group instead of the non-capturing group for improved performance: `\A(b(?>m|(?1))*e)\z`.

Similarly, `\Ao*(b(?:m|(?1))*eo*)+\z` and the optimized `\Ao*+(b(?>m|(?1))*+eo*+)++\z` match a string that consists of nothing but a sequence of one or more correctly balanced constructs, with possibly other text in between. Here, `o` is what can occur outside the balanced constructs. It will often be the same as `m`. `o` should not be able to match the same text as `b` or `e`.

`\A(\((?>[^()]|(?1))*\))\z` matches a string that consists of nothing but a correctly balanced pair of parentheses, possibly with text between them. `\A[^()]*+(\((?>[^()]|(?1))*+\)[^()]*+)++\z`.

Matching The Same Construct More Than Once

A regex that needs to match the same kind of construct (but not the exact same text) more than once in different parts of the regex can be shorter and more concise when using subroutine calls. Suppose you need a regex to match patient records like these:

```
Name: John Doe
Born: 17-Jan-1964
Admitted: 30-Jul-2013
Released: 3-Aug-2013
```

Further suppose that you need to match the date format rather accurately so the regex can filter out valid records, leaving invalid records for human inspection. In most regex flavors you could easily do this with this regex, using free-spacing syntax:

```
^Name:\ (.*)\r?\n
Born:\ (?:3[01]|[12][0-9]|[1-9])
       -(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
       -(?:19|20)[0-9][0-9]\r?\n
Admitted:\ (?:3[01]|[12][0-9]|[1-9])
           -(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
           -(?:19|20)[0-9][0-9]\r?\n
Released:\ (?:3[01]|[12][0-9]|[1-9])
           -(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
           -(?:19|20)[0-9][0-9]$
```

With subroutine calls you can make this regex much shorter, easier to read, and easier to maintain:

```
^Name:\ (.*)\r?\n
Born:\ (?'date'(?:3[01]|[12][0-9]|[1-9])
               -(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
               -(?:19|20)[0-9][0-9])\r?\n
Admitted:\ \g'date'\r?\n
Released:\ \g'date'$
```

Separate Subroutine Definitions

In Perl, PCRE, and JGsoft V2, you can take this one step further using the special DEFINE group: `(?(DEFINE)(?'subroutine'regex))`. While this looks like a conditional that references the non-existent group DEFINE containing a single named group “subroutine”, the DEFINE group is a special syntax. The fixed text `(?(DEFINE)` opens the group. A parenthesis closes the group. This special group tells the regex engine to ignore its contents, other than to parse it for named and numbered capturing groups. You can put as many capturing groups inside the DEFINE group as you like. The DEFINE group itself never matches anything, and never fails to match. It is completely ignored. The regex `foo(?(DEFINE)(?'subroutine'skipped))bar` matches `foobar`. The DEFINE group is completely superfluous in this regex, as there are no calls to any of the groups inside it.

With a DEFINE group, our regex becomes:

```
(?(DEFINE)(?'date'(?:3[01]|[12][0-9]|[1-9])
                  -(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
                  -(?:19|20)[0-9][0-9]))
^Name:\ (.*)\r?\n
Born:\ (?P>date)\r?\n
Admitted:\ (?P>date)\r?\n
Released:\ (?P>date)$
```

Quantifiers On Subroutine Calls

Quantifiers on subroutine calls work just like a quantifier on recursion. The call is repeated as many times in sequence as needed to satisfy the quantifier. `([abc])(?1){3}` matches `abcb` and any other combination of four-letter combination of the first three letters of the alphabet. First the group matches once, and then the call matches three times. This regex is equivalent to `([abc])[abc]{3}`.

Quantifiers on the group are ignored by the subroutine call. `([abc]){3}(?1)` also matches `abcb`. First, the group matches three times, because it has a quantifier. Then the subroutine call matches once, because it has no quantifier. `([abc]){3}(?1){3}` matches six letters, such as `abbcab`, because now both the group and the call are repeated 3 times. These two regexes are equivalent to `([abc]){3}[abc]` and `([abc]){3}[abc]{3}`.

While Ruby does not support subroutine definition groups, it does support subroutine calls to groups that are repeated zero times. `(a){0}\g<1>{3}` matches `aaa`. The group itself is skipped because it is repeated zero times. Then the subroutine call matches three times, according to its quantifier. This also works in PCRE 7.7 and later. It doesn't work (reliably) in older versions of PCRE or in any version of Perl because of bugs.

The Ruby version of the patient record example can be further cleaned up as:

```
(?'date'(?:3[01]|[12][0-9]|[1-9])
        -(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
        -(?:19|20)[0-9][0-9]){0}
^Name:\ (.*)\r?\n
Born:\ \g'date'\r?\n
Admitted:\ \g'date'\r?\n
Released:\ \g'date'$
```

Make a Donation

Did this website just save you a trip to the bookstore? Please make a donation to support this site, and you'll get a **lifetime of advertisement-free access** to this site!