NAME

Regexp::Pattern − Convention/framework for modules that contain collection of regexes

SPECIFICATION VERSION

0.2

VERSION

This document describes version 0.2.12 of Regexp::Pattern (from Perl distribution Regexp-Pattern), released on 2020−02−07.

SYNOPSIS

Subroutine interface:

```
use Regexp::Pattern; # exports re()

my $re = re('YouTube::video_id');
say "ID does not look like a YouTube video ID" unless $id =~ /\A$re\z/;

# a dynamic pattern (generated on-demand) with generator arguments
my $re2 = re('Example::re3', {variant=>"B"});
```

Hash interface (a la Regexp::Common but simpler with regular/non−magical hash that is only 1−level deep):

```
use Regexp::Pattern 'YouTube::video_id';
say "ID does not look like a YouTube video ID"
    unless $id =~ /\A$RE{video_id}\z/;

# more complex example

use Regexp::Pattern (
    're',                                 # we still want the re() function
    'Foo::bar' => (-as => 'qux'),         # the pattern will be in your $RE{qux}
    'YouTube::*',                         # wildcard import
    'Example::re3' => (variant => 'B'),   # supply generator arguments
    'JSON::*' => (-prefix => 'json_'),    # add prefix
    'License::*' => (
      # filtering options
      -has_tag    => 'family:cc',         # only select patterns that has this tag
      -lacks_tag  => 'type:unversioned', # only select patterns that does not have
      -has_tag_matching   => qr/^type:/, # only select patterns that has a tag matc
      -lacks_tag_matching => qr/^type:/, # only select patterns that does not have

      # other options
      -prefix  => 'pat_',       # add prefix
      -suffix  => '_license',   # add suffix
    ),
);
```

DESCRIPTION

Regexp::Pattern is a convention for organizing reusable regexp patterns in modules, as well as framework to provide convenience in using those patterns in your program.

Structure of an example Regexp::Pattern::* module

```
package Regexp::Pattern::Example;


our %RE = (
    # the minimum spec
```

```
    re1 => { pat => qr/\d{3}-\d{3}/ },

    # more complete spec
    re2 => {
        summary => 'This is regexp for blah', # plaintext
        description => <<'_',

A longer description in *Markdown* format.

_
        pat => qr/\d{3}-\d{3}(?:-\d{5})?/,
        tags => ['A','B'],
        examples => [
            # examples can be tested using 'test-regexp-pattern' script
            # (distributed in Test-Regexp-Pattern distribution). examples can
            # also be rendered in your POD using
            # Pod::Weaver::Plugin::Regexp::Pattern.
            {
                str => '123-456',
                matches => 1,
            },
            {
                summary => 'Another example that matches',
                str => '123-456-78901',
                matches => 1,
            },
            {
                summary => 'An example that does not match',
                str => '123456',
                matches => 0,
            },
            {
                summary => 'An example that does not get tested',
                str => '123456',
            },
            {
                summary => 'Another example that does not get tested nor rendered t
                str => '234567',
                matches => 0,
                test => 0,
                doc => 0,
            },
        ],
    },

    # dynamic (regexp generator)
    re3 => {
        summary => 'This is a regexp for blah blah',
        description => <<'_',

...

_
        gen => sub {
```

```
            my %args = @_;
            my $variant = $args{variant} || 'A';
            if ($variant eq 'A') {
                return qr/\d{3}-\d{3}/;
            } else { # B
                return qr/\d{3}-\d{2}-\d{5}/;
            }
        },
        gen_args => {
            variant => {
                summary => 'Choose variant',
                schema => ['str*', in=>['A','B']],
                default => 'A',
                req => 1,
            },
        },
        tags => ['B','C'],
        examples => [
            {
                summary => 'An example that matches',
                gen_args => {variant=>'A'},
                str => '123-456',
                matches => 1,
            },
            {
                summary => "An example that doesn't match",
                gen_args => {variant=>'B'},
                str => '123-456',
                matches => 0,
            },
        ],
    },

    re4 => {
        summary => 'This is a regexp that does capturing',
        # it is recommended that your pattern does not capture, unless
        # necessary. capturing pattern should tag with 'capturing' to let
        # users/tools know.
        tags => ['capturing'],
        pat => qr/(\d{3})-(\d{3})/,
        examples => [
            {str=>'123-456', matches=>[123, 456]},
            {str=>'foo-bar', matches=>[]},
        ],
    },

    re5 => {
        summary => 'This is another regexp that is anchored and does (named) captur
        # it is recommended that your pattern is not anchored for more
        # reusability, unless necessary. anchored pattern should tag with
        # 'anchored' to let users/tools know.
        tags => ['capturing', 'anchored'],
        pat => qr/^(?<cap1>\d{3})-(?<cap2>\d{3})/,
        examples => [
```

```
            {str=>'123-456', matches=>{cap1=>123, cap2=>456}},
            {str=>'something 123-456', matches=>{}},
        ],
    },
);
```

A Regexp::Pattern::* module must declare a package global hash variable named `%RE`. Hash keys are pattern names, hash values are pattern definitions in the form of defhashes (see DefHash).

Pattern name should be a simple identifier that matches this regexp: `/\A[A-Za-z_][A-Za-z_0-9]*\z/`. The definition for the qualified pattern name `Foo::Bar::baz` can then be located in `%Regexp::Pattern::Foo::Bar::RE` under the hash key `baz`.

Pattern definition hash should at the minimum be:

```
{ pat => qr/.../ }
```

You can add more stuffs from the defhash specification, e.g. summary, description, tags, and so on, for example (taken from Regexp::Pattern::CPAN):

```
{
    summary     => 'PAUSE author ID, or PAUSE ID for short',
    pat         => qr/[A-Z][A-Z0-9]{1,8}/,
    description => <<~HERE,
    I'm not sure whether PAUSE allows digit for the first letter. For safety
    I'm assuming no.
    HERE
    examples => [
        {str=>'PERLANCAR', matches=>1},
        {str=>'BAD ID', anchor=>1, matches=>0},
    ],
}
```

Examples. Your regexp specification can include an `examples` property (see above for example). The value of the `examples` property is an array, each of which should be a defhash. For each example, at the minimum you should specify `str` (string to be matched by the regexp), `gen_args` (hash, arguments to use when generating dynamic regexp pattern), and `matches` (a boolean value that specifies whether the regexp should match the string or not, or an array/hash that specifies the captures). You can of course specify other defhash properties (e.g. `summary`, `description`, etc). Other example properties might be introduced in the future.

If you use Dist::Zilla to build your distribution, you can use the plugin [Regexp::Pattern] to test the examples during building, and the Pod::Weaver plugin [–Regexp::Pattern] to render the examples in your POD.

Using a Regexp::Pattern::* module

Standalone

A Regexp::Pattern::* module can be used in a standalone way (i.e. no need to use via the Regexp::Pattern framework), as it simply contains data that can be grabbed using a normal means, e.g.:

```
use Regexp::Pattern::Example;

say "Input does not match blah"
    unless $input =~ /\A$Regexp::Pattern::Example::RE{re1}{pat}\z/;
```

Via Regexp::Pattern, sub interface

Regexp::Pattern (this module) also provides `re()` function to help retrieve the regexp pattern. See "re" for more details.

Via Regexp::Pattern, hash interface

Additionally, Regexp::Pattern (since v0.2.0) lets you import regexp patterns into your `%RE` package hash variable, a la Regexp::Common (but simpler because the hash is just a regular hash, only 1−level deep, and not magical).

To import, you specify qualified pattern names as the import arguments:

```
use Regexp::Pattern 'Q::pat1', 'Q::pat2', ...;
```

Each qualified pattern name can optionally be followed by a list of name-value pairs. A pair name can be an option name (which is dash followed by a word, e.g. `-as`, `-prefix`) or a generator argument name for dynamic pattern.

Wildcard import. Instead of a qualified pattern name, you can use 'Module::SubModule::*' wildcard syntax to import all patterns from a pattern module.

Importing into a different name. You can add the import option `-as` to import into a different name, for example:

```
use Regexp::Pattern 'YouTube::video_id' => (-as => 'yt_id');
```

Prefix and suffix. You can also add a prefix and/or suffix to the imported name:

```
use Regexp::Pattern 'Example::*' => (-prefix => 'example_');
use Regexp::Pattern 'Example::*' => (-suffix => '_sample');
```

Filtering. When wildcard-importing, you can select the patterns you want using a combination of these options: `-has_tag` (only select patterns that have a specified tag), `-lacks_tag` (only select patterns that do not have a specified tag), `-has_tag_matching` (only select patterns that has at least one tag matching specified regex pattern), `-lacks_tag_matching` (only select patterns that does not have any tags matching specified regex pattern).

Recommendations for writing the regex patterns

- Regexp pattern should be written as a `qr//` literal

  Using a string literal is less desirable. That is:

  ```
  pat => qr/foo[abc]+/,
  ```

  is preferred over:

  ```
  pat => 'foo[abc]+',
  ```

- Regexp pattern should not be anchored (unless really necessary)

  That is:

  ```
  pat => qr/foo/,
  ```

  is preferred over:

  ```
  pat => qr/^foo/, # or qr/foo$/, or qr/\Afoo\z/
  ```

  Adding anchors limits the reusability of the pattern. When composing pattern, user can add anchors herself if needed.

  When you define an anchored pattern, adding tag `anchored` is recommended:

  ```
  tags => ['anchored'],
  ```

- Regexp pattern should not contain capture groups (unless really necessary)

  Adding capture groups limits the reusability of the pattern because it can affect the groups of the composed pattern. When composing pattern, user can add captures herself if needed.

  When you define a capturing pattern, adding tag `capturing` is recommended:

  ```
  tags => ['capturing'],
  ```

FUNCTIONS

re

Exported by default. Get a regexp pattern by name from a `Regexp::Pattern::*` module.

Usage:

```
re($name[, \%args ]) => $re
```

`$name` is *MODULE_NAME::PATTERN_NAME* where *MODULE_NAME* is name of a `Regexp::Pattern::*` module without the `Regexp::Pattern::` prefix and *PATTERN_NAME* is a key to the `%RE` package global hash in the module. A dynamic pattern can accept arguments for its generator, and you can pass it as hashref in the second argument of `re()`.

Anchoring. You can also put `-anchor => 1` in `%args`. This will conveniently wraps the regex inside `qr/\A(?:...)\z/`.

Die when pattern by name `$name` cannot be found (either the module cannot be loaded or the pattern with that name is not found in the module).

FAQ

My pattern is not anchored, but what if I want to test the anchored version?

You can add `anchor=>1` or `gen_args=>{-anchor=>1}` in the example, for example:

```
{
    summary     => 'PAUSE author ID, or PAUSE ID for short',
    pat         => qr/[A-Z][A-Z0-9]{1,8}/,
    description => <<~HERE,
    I'm not sure whether PAUSE allows digit for the first letter. For safety
    I'm assuming no.
    HERE
    examples => [
        {str=>'PERLANCAR', matches=>1},
        {str=>'BAD ID', anchor=>1, matches=>0, summary=>"Contains whitespace"},
        {str=>'NAMETOOLONG', gen_args=>{-anchor=>1}, matches=>0, summary=>"Too long
    ],
}
```

HOMEPAGE

Please visit the project's homepage at <https://metacpan.org/release/Regexp-Pattern>.

SOURCE

Source repository is at <https://github.com/perlancar/perl-Regexp-Pattern>.

BUGS

Please report any bugs or feature requests on the bugtracker website <https://rt.cpan.org/Public/Dist/Display.html?Name=Regexp-Pattern>

When submitting a bug or request, please include a test-file or a patch to an existing test-file that illustrates the bug or desired feature.

SEE ALSO

Regexp::Common. Regexp::Pattern is an alternative to Regexp::Common. Regexp::Pattern offers simplicity and lower startup overhead. Instead of a magic hash, you retrieve available regexes from normal data structure or via the provided `re()` function. Regexp::Pattern also provides a hash interface, albeit the hash is not magic.

Regexp::Common::RegexpPattern, a bridge module to use patterns in `Regexp::Pattern::*` modules via Regexp::Common.

Regexp::Pattern::RegexpCommon, a bridge module to use patterns in `Regexp::Common::*` modules via Regexp::Pattern.

App::RegexpPatternUtils

If you use Dist::Zilla: Dist::Zilla::Plugin::Regexp::Pattern, Pod::Weaver::Plugin::Regexp::Pattern, Dist::Zilla::Plugin::AddModule::RegexpCommon::FromRegexpPattern, Dist::Zilla::Plugin::AddModule::RegexpPattern::FromRegexpCommon.

Test::Regexp::Pattern and test-regexp-pattern.

AUTHOR

perlancar <perlancar@cpan.org>

COPYRIGHT AND LICENSE