NAME

MooseX::Params::Validate − an extension of Params::Validate using Moose's types

VERSION

version 0.16

SYNOPSIS

```
package Foo;
use Moose;
use MooseX::Params::Validate;

sub foo {
    my ( $self, %params ) = validated_hash(
        \@_,
        bar => { isa => 'Str', default => 'Moose' },
    );
    return "Hooray for $params{bar}!";
}

sub bar {
    my $self = shift;
    my ( $foo, $baz, $gorch ) = validated_list(
        \@_,
        foo   => { isa => 'Foo' },
        baz   => { isa => 'ArrayRef | HashRef', optional => 1 },
        gorch => { isa => 'ArrayRef[Int]', optional => 1 }
    );
    [ $foo, $baz, $gorch ];
}
```

DESCRIPTION

This module fills a gap in Moose by adding method parameter validation to Moose. This is just one of many developing options, it should not be considered the ''official'' one by any means though.

You might also want to explore `MooseX::Method::Signatures` and `MooseX::Declare`.

CAVEATS

It is not possible to introspect the method parameter specs; they are created as needed when the method is called and cached for subsequent calls.

EXPORTS

validated_hash(\@_, %parameter_spec)

This behaves similarly to the standard Params::Validate `validate` function and returns the captured values in a HASH. The one exception is where if it spots an instance in the `@_`, then it will handle it appropriately (unlike Params::Validate which forces you to shift you `$self` first).

The values in `@_` can either be a set of name-value pairs or a single hash reference.

The `%parameter_spec` accepts the following options:

isa The `isa` option can be either; class name, Moose type constraint name or an anon Moose type constraint.

does

The `does` option can be either; role name or an anon Moose type constraint.

default

This is the default value to be used if the value is not supplied.

optional

As with Params::Validate, all options are considered required unless otherwise specified. This option is passed directly to Params::Validate.

coerce

If this is true and the parameter has a type constraint which has coercions, then the coercion will be called for this parameter. If the type does have coercions, then this parameter is

ignored.

This function is also available under its old name, `validate`.

validated_list(\@_, %parameter_spec)

The `%parameter_spec` accepts the same options as above, but returns the parameters as positional values instead of a HASH. This is best explained by example:

```
sub foo {
    my ( $self, $foo, $bar ) = validated_list(
        \@_,
        foo => { isa => 'Foo' },
        bar => { isa => 'Bar' },
    );
    $foo->baz($bar);
}
```

We capture the order in which you defined the parameters and then return them as a list in the same order. If a param is marked optional and not included, then it will be set to `undef`.

The values in `@_` can either be a set of name-value pairs or a single hash reference.

Like `validated_hash`, if it spots an object instance as the first parameter of `@_`, it will handle it appropriately, returning it as the first argument.

This function is also available under its old name, `validatep`.

pos_validated_list(\@_, $spec, $spec, ...)

This function validates a list of positional parameters. Each `$spec` should validate one of the parameters in the list:

```
sub foo {
    my $self = shift;
    my ( $foo, $bar ) = pos_validated_list(
        \@_,
        { isa => 'Foo' },
        { isa => 'Bar' },
    );

    ...
}
```

Unlike the other functions, this function *cannot* find `$self` in the argument list. Make sure to shift it off yourself before doing validation.

The values in `@_` must be a list of values. You cannot pass the values as an array reference, because this cannot be distinguished from passing one value which is itself an array reference.

If a parameter is marked as optional and is not present, it will simply not be returned.

If you want to pass in any of the cache control parameters described below, simply pass them after the list of parameter validation specs:

```
sub foo {
    my $self = shift;
    my ( $foo, $bar ) = pos_validated_list(
        \@_,
        { isa => 'Foo' },
        { isa => 'Bar' },
        MX_PARAMS_VALIDATE_NO_CACHE => 1,
    );

    ...
}
```

ALLOWING EXTRA PARAMETERS

By default, any parameters not mentioned in the parameter spec cause this module to throw an error. However, you can have have this module simply ignore them by setting `MX_PARAMS_VALIDATE_ALLOW_EXTRA` to a true value when calling a validation subroutine.

When calling `validated_hash` or `pos_validated_list` the extra parameters are simply returned in the hash or list as appropriate. However, when you call `validated_list` the extra parameters will not be returned at all. You can get them by looking at the original value of `@_`.

EXPORTS

By default, this module exports the `validated_hash`, `validated_list`, and `pos_validated_list`.

If you would prefer to import the now deprecated functions `validate` and `validatep` instead, you can use the `:deprecated` tag to import them.

IMPORTANT NOTE ON CACHING

When a validation subroutine is called the first time, the parameter spec is prepared and cached to avoid unnecessary regeneration. It uses the fully qualified name of the subroutine (package + subname) as the cache key. In 99.999% of the use cases for this module, that will be the right thing to do.

However, I have (ab)used this module occasionally to handle dynamic sets of parameters. In this special use case you can do a couple things to better control the caching behavior.

- Passing in the `MX_PARAMS_VALIDATE_NO_CACHE` flag in the parameter spec this will prevent the parameter spec from being cached.

```
sub foo {
    my ( $self, %params ) = validated_hash(
        \@_,
        foo                          => { isa => 'Foo' },
        MX_PARAMS_VALIDATE_NO_CACHE => 1,
    );

}
```

- Passing in `MX_PARAMS_VALIDATE_CACHE_KEY` with a value to be used as the cache key will bypass the normal cache key generation.

```
sub foo {
    my ( $self, %params ) = validated_hash(
        \@_,
        foo                           => { isa => 'Foo' },
        MX_PARAMS_VALIDATE_CACHE_KEY => 'foo-42',
    );

}
```

MAINTAINER

Dave Rolsky <autarch AT urth DOT org>

BUGS

Please submit bugs to the CPAN RT system at http://rt.cpan.org/NoAuth/ReportBug.html?Queue=moosex-params-validate or via email at bug-moosex-params-validate AT rt DOT cpan DOT org.

AUTHOR

Stevan Little <stevan DOT little AT iinteractive DOT com>

COPYRIGHT AND LICENSE