Moose & Mouse QUICK REFERENCE

http://perl.ermitejo.com/moose_quick_ref.pdf | .odt
Last modified on 2010-03-01

Written by MORIYA Masaki a.k.a. gardejo

Inspired by Oliver Gorwits (http://tinyurl.com/moosequickref)

CLASSES

```
package My::Class;
use namespace::clean;
use Any::Moose;         # strict, warnings, confess(), blessed()
# use namespace::clean -except => [qw(meta)];
extends @superclasses;
with @roles;
with ( $some_role => {
    -alias    => { $some_method => $other_method },
    -excludes => { $other_method },
} );
has $attribute => %option;  # cf. ATTRIBUTE CONSTRUCTION OPTIONS
has [ @attributes ] => %option;
has "+$attribute_in_superclass" => %option;
sub method { my ($self, @args) = @_; ... }
__PACKAGE__->meta->make_immutable; # returns 1
__END__
```

ROLES

```
package My::Role;
use namespace::clean;
use Any::Moose '::Role';
requires @methods;      # as "interface" in Java
with @other_roles;
has $attribute => %option;
sub method { ... };     # as "mix-in" in Ruby
1;
__END__
```

METHOD MODIFIERS

```
before @methods => sub { my ($self, @args) = @_; ... };
after  @methods => sub { my ($self, @args) = @_; ... };
around @methods => sub {
    my ($next, $self, @args) = @_; ...
    my @return_values = $self->$next(@args); ...
};
override @methods => sub {
    my ($self, @args) = @_; ...
    super(); ...
};
augment @methods => sub {
    my ($self, @args) = @_; ...
    inner(); ...
};

before 2, before 1,
    around 2, around 1,
        wrapped method,
    around 1, around 2,
after 1, after 2

before qr{ ... } => sub { ... };    # only in a class
around [ @methods ] => sub { ... };
```

CONSTRUCTION AND DESTRUCTION

```
around BUILDARGS => sub {
    my ($next, $class, @args) = @_;
    if ( ... ) { ... }
    else { return $class->$next(@args); }
};
sub BUILD { my ($self, @args) = @_; ... }
sub DEMOLISH { my ($self, $arg) = @_; ... }
sub DEMOLISHALL { my ($self, $arg) = @_; ... }
```

ATTRIBUTE CONSTRUCTION OPTIONS

Attribute Extension

```
metaclass => $metaclass
# Moose::Meta::Attribute::Custom::$metaclass

traits => [ @traits ]
# Moose::Meta::$kind::Custom::Trait::$trait
# Moose::Meta::Attribute::Native::Trait::$trait
```

Accessor Methods Generation

```
is => 'rw'
is => 'ro'
is => 'bare'

accessor => $accessor_method    # is => 'rw'

reader   => $reader_method      # override is => 'ro'

writer   => $writer_method
```

Type Constraints and Coercions

```
isa => $type_constraint     # cf. TYPE CONSTRAINTS AND COERCIONS
isa => "$some_type | $other_type"

does => $role # for Interfaces

coerce     => $bool

weak_ref   => $bool         # conflicts with 'coerce'

auto_deref => $bool         # requires 'isa'
                            # use "Native Traits" instead!
```

Constructor Arguments

```
init_arg => $argument_key
init_arg => undef

required => $bool
```

Lazy Building, Predication and Cleaning

```
lazy => $bool

lazy_build => $bool         # lazy      => 1,
                            # predicate => "has_$attribute",
                            # clearer   => "clear_$attribute",
                            # builder   => "_build_$attribute",

predicate => $predicator_method

clearer   => $clearer_method

builder   => $builder_method
```

Default Values

```
default => $default_value
default => sub { my ($self) = @_; ... } # to assign a reference

builder => $builder_method # better than default
```

Triggers

```
trigger => sub { my ($self, $new_value) = @_; ... }
```

Delegation

```
handles => [ @delegated_methods ]
handles => { $delegating_method => $delegated_method }
handles => { $curried_method => [ $delegated_method, @args ] }
handles => qr{ $pattern_of_delegated_methods }
handles => $role_name
handles => sub { ... }
# cf. HELPER METHODS FROM NATIVE TRAITS
```

Documentation

```
documentation => $documentation_string
```

TYPE CONSTRAINTS AND COERCIONS

Type Hierarchy

```
Any
Maybe[TypeName] : Undef or TypeName
Item
    Bool
    Undef
    Defined
        Value
            Num
                Int
            Str
                RoleName  : loaded role (deprecated)
                ClassName : loaded class
        Ref
            ScalarRef
            ArrayRef or ArrayRef[TypeName]
            HashRef or HashRef[TypeName]
            CodeRef
            RegexpRef
            GlobRef
                FileHandle
            Object
                Role
```

Type Expressions

```
ArrayRef[Str]           # Parameterized Types

'DateTime'              # Automatically Created Types

SomeTime | OtherType    # Type Unions
```

To Define My Own Types

```
package My::App::Types;
use Any::Moose (
    'X::Types' => [ -declare => [ @my_types ] ],
    'X::Types::Moose' => [ @built_in_types ],
);

type MyType, where { ... }, message { ... };

subtype MyType, as ParentType, where { ... }, message { ... };

class_type MyType, { class => 'My::Class' };

duck_type MyType, [ @methods ];

enum MyType, @values;

# These sugar functions return an anonymous type object.
```

Type Coercions

```
coerce MyType, from OtherType, via { ... };
```

MooseX::* EXPANSION MODULES

```
use MooseX::StrictConstructor;      # "strict" %init_arg

use MooseX::Types;  # instead of Moose::Util::TypeConstraints

with qw(MooseX::Clone);
# $self->clone; Note: traits => [qw(Clone)] for refs.

with qw(MooseX::Getopt MooseX::SimpleConfig);

use MooseX::ClassAttribute;         # class_has => %option

with qw(MooseX::Object::Pluggable); # $self->load_plugin($p)

use MooseX::Role::Parameterized;
# parameter $parameter => %option; role { ... }

use MooseX::Orochi; # bind_constructor $registry => (...);

...
```

HELPER METHODS FROM NATIVE TRAITS

Bool

```
set()                   # $bool = 1
unset()                 # $bool = 0
toggle()                # from 1 to 0, from 0 to 1
not()                   # not $bool
```

Counter

```
set($new_count)         # $count = $new_count
inc()                   # $count ++
dec()                   # $count --
reset()                 # $count = $default_value
```

Number

```
set($new_num)           # $num = $new_num
add($adding_num)        # $num += $adding_num
sub($subtracting_num)   # $num -= $subtracting_num
mul($multiplying_num)   # $num *= $multiplying_num
div($deviding_num)      # $num /= $deviding_num
mod($deviding_num)      # $num %= $deviding_num
abs()                   # $num = abs $num
```

String

```
inc()                   # $str ++ (from 'a' to 'b')
append($other_str)      # $str .= $other_str
prepend($other_str)     # $str = $other_str . $str
replace($ptn, $rep)     # $str =~ s/$ptn/$rep/
                        # ('foo' or qr{foo}xmsi)
match($ptn)             # $str =~ m/$ptn/
chop()                  # chop $str
chomp()                 # chomp $str
clear()                 # $str = q{}
length()                # length $str
substr(@args)           # substr $str, $offset, $len, $rep
```

Hash

```
get(@keys)              # @$h{@keys}
set(%keys_and_vals)     # $h = { $key => $val, ... }
delete(@keys)           # delete @$h{@keys}
keys()                  # keys %$h
exists($key)            # exists $h->{$key}
defined($key)           # defined $h->{$key}
values()                # values %$h
kv()                    # k/v pairs as an arr. of arr. ref.
elements()              # k/v pairs as a flattened list
clear()                 # %$h = ()
count()                 # scalar keys %$h
accessor($key)          # get($key)
accessor($key, $val)    # set($key => $val)
```

Array

```
count()                 # scalar @$a
is_empty()              # if scalar @$a
elements()              # @$a
get($idx)               # $a->[$idx]
pop()                   # pop @$a
push(@vals)             # push @$a, @vals
shift()                 # shift @$a
unshift(@vals)          # unshift @$a, @vals
splice($offset, $len, @vals)  # splice @$a, $offset, $len, @vals
first(sub { ... })      # first { ... } @$a
grep(sub { ... })       # grep { ... } @$a
map(sub { ... })        # map { ... } @$a
reduce(sub { ... })     # reduce { ... } @$a
                        # ( $a, $b is $_[0], $_[1] )
sort(sub { ... })       # sort { ... } @$a (or sort @$a)
sort_in_place(sub { ... })  # @$a = sort { ... } @$a
shuffle()               # shuffle @$a
uniq()                  # uniq @$a
join($string)           # join $string, @$a
set($idx, $val)         # $a->[$idx] = $val
delete($idx)            # delete $a->[$idx]
insert($idx, $val)      # splice @$a, $idx, 0, $val
clear()                 # @$a = ()
accessor($idx)          # get($idx)
accessor($idx, $val)    # set($idx, $val)
natatime($n, $c)        # $i = natatime $n, @$a;
                        # while (@v = $i->()) { $c->(@v); }
```

Code

```
execute(@args)          # &$c(@args)
execute_method(@args)   # $invocant->$c(@args)
```