Mooooooooooooooooose

sunnavy@bestpractical.com

November 08, 2008

moose

What Moose is not

- NOT a cow

What Moose is not

- NOT a cow
- NOT contain any melamine

What Moose is not

- NOT a cow
- NOT contain any melamine
- NOT Perl 6

What Moose is not

- NOT a cow
- NOT contain any melamine
- NOT Perl 6
- NOT just a toy

What Moose is not

- NOT a cow
- NOT contain any melamine
- NOT Perl 6
- NOT just a toy
- NOT a new object system

Google knows something about Moose

Interested in that animal?

Let's talk about that animal later

What Moose here really is

- an extension of the existing object system of Perl 5

What Moose here really is

- an extension of the existing object system of Perl 5
- very much inspired by Perl 6

What Moose here really is

- an extension of the existing object system of Perl 5
- very much inspired by Perl 6
- built on top of Class::MOP(Welcome to Meta world!)

What Moose here really is

- an extension of the existing object system of Perl 5
- very much inspired by Perl 6
- built on top of Class::MOP(Welcome to Meta world!)
- postmodern(just like Perl ;)

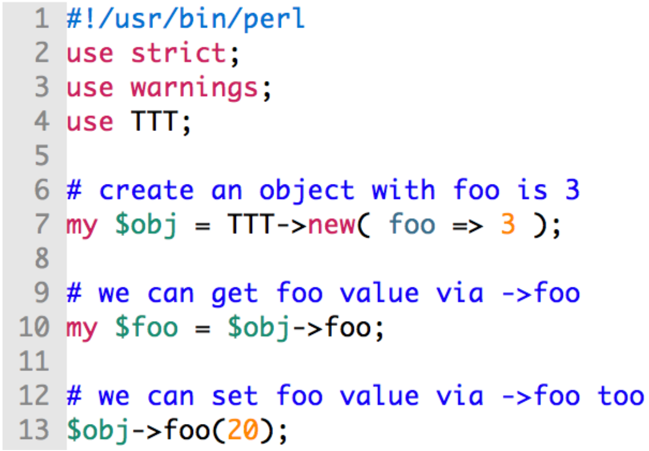
What Moose here really is

- an extension of the existing object system of Perl 5
- very much inspired by Perl 6
- built on top of Class::MOP(Welcome to Meta world!)
- postmodern(just like Perl ;)
- ready to use

Official Definition

Moose is a postmodern object system for Perl 5 that takes the tedium out of writing object-oriented Perl. It borrows all the best features from Perl 6, CLOS (LISP), Smalltalk, Java, BETA, OCaml, Ruby and more, while still keeping true to its Perl 5 roots.

The Request



```
#!/usr/bin/perl
use strict;
use warnings;
use TTT;

# create an object with foo is 3
my $obj = TTT->new( foo => 3 );

# we can get foo value via ->foo
my $foo = $obj->foo;

# we can set foo value via ->foo too
$obj->foo(20);
```

What we normally OOP

```perl
package TTT;
use strict;
use warnings;

sub new {
    my $class = shift;
    my %args  = @_;
    my $ref   = {};
    bless $ref, $class;
    $ref->foo($args{foo}) if exists $args{foo};
    return $ref;
}

sub foo {
    my $self = shift;
    $self->{foo} = shift if @_;
    return $self->{foo};
}
1;
```

If we have Moose . . .

```
package TTT;
use Moose;

has 'foo' => ( is => 'rw' );

1;
```

You may point out. . .

You skipped use strict; and use warnings; on purpose to reduce lines!

So, I'm telling you. . .

Well, I skipped those two lines because Moose does that for me.

Thanks, Moose!

use 'has' to install attributes

$$\text{has \$name}|\text{@\$}names => \%options$$

%options

- is ('ro|rw')

%options

- is ('ro|rw')
- isa (type constraints)

%options

- is ('ro|rw')
- isa (type constraints)
- does ($role)

%options

- is ('ro|rw')
- isa (type constraints)
- does ($role)
- required (1|0)

%options

- is ('ro|rw')
- isa (type constraints)
- does ($role)
- required (1|0)
- default

%options

- is ('ro|rw')
- isa (type constraints)
- does ($role)
- required (1|0)
- default
- predicate (method name to check for initialization)

%options

- is ('ro|rw')
- isa (type constraints)
- does ($role)
- required (1|0)
- default
- predicate (method name to check for initialization)
- clearer (method name to uninitialize)

is

- ro

is

- ro
- rw

is

```perl
#!/usr/bin/perl
package TTT;
use Moose;
has 'foo';

package main;
my $ttt = TTT->new( foo => "I'm foo!" );

# works, but we'd better not do this at $work
$ttt->{foo};

# wrong, there's no foo accessor
$ttt->foo;
```

isa - type

```
Any
Item
    Bool
    Maybe[`a]
    Undef
    Defined
        Value
            Num
              Int
            Str
              ClassName
        Ref
            ScalarRef
            ArrayRef[`a]
            HashRef[`a]
            CodeRef
            RegexpRef
            GlobRef
              FileHandle
            Object
                Role
```

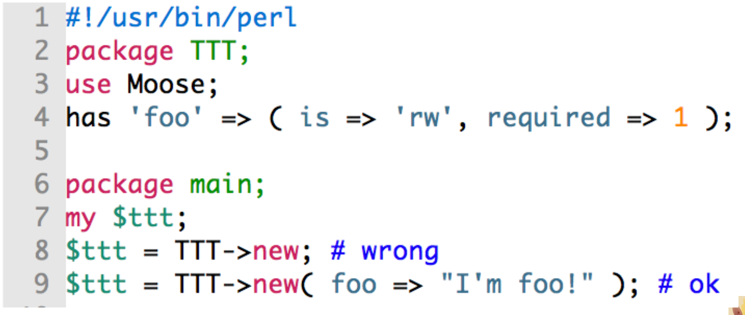
isa - type

```
#!/usr/bin/perl
package TTT;
use Moose;

has 'shipwright' => ( is => 'rw', isa => 'Shipwright' );

package main;
use Shipwright;

my $ttt = TTT->new;

my $sw = Shipwright->new( repository => 'fs:/tmp/fs' );

# invalid type, so this will die
$ttt->shipwright( "I'm not Shipwright" );

# this's ok
$ttt->shipwright( $sw );
```

does - Role

Let's talk about Roles later

required

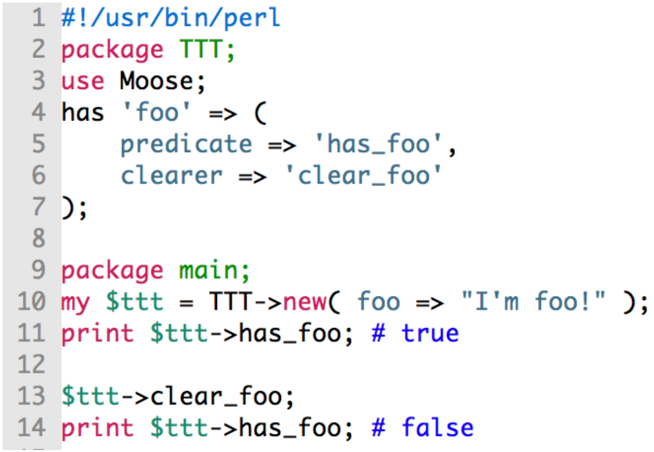


```perl
#!/usr/bin/perl
package TTT;
use Moose;
has 'foo' => ( is => 'rw', required => 1 );

package main;
my $ttt;
$ttt = TTT->new; # wrong
$ttt = TTT->new( foo => "I'm foo!" ); # ok
```

default

```perl
# for simple var, this's ok.
has 'foo' => ( is => 'rw', default => 'Hello' );

# for ref, this's wrong usually.
has 'foo' => ( is => 'rw', default => [1,2] );

# for ref, we need to wrap the ref value to a sub.
has 'foo' => ( is => 'rw', default => sub { [1,2] });
```

predicate and clearer



```
#!/usr/bin/perl
package TTT;
use Moose;
has 'foo' => (
      predicate => 'has_foo',
      clearer => 'clear_foo'
);

package main;
my $ttt = TTT->new( foo => "I'm foo!" );
print $ttt->has_foo; # true

$ttt->clear_foo;
print $ttt->has_foo; # false
```

Code Reuse

From S12: Classes are primarily in charge of object management, and only secondarily in charge of software reuse. In Perl 6, roles take over the job of managing software reuse.

What's a Role?

From S12: Depending on how you care to look at it, a role is like a partial class, or an interface with default implementation, or a set of generic methods and their associated data, or a class closed at compile time.

use 'with' to apply Roles

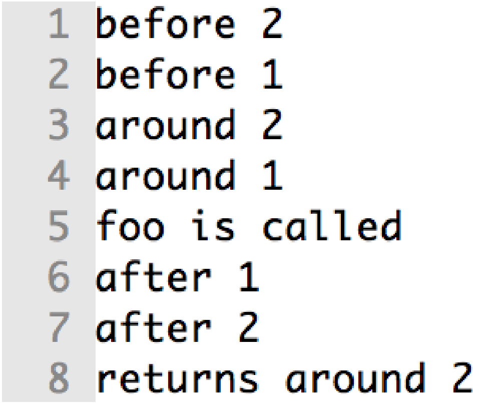
```perl
#!/usr/bin/perl
package TTT::Role;
use Moose::Role;

has 'message' => (
    is      => 'rw',
    isa     => 'Str',
    default => 'Hello, I am TTT'
);

package My::TTT;
use Moose;

with 'TTT::Role';

package main;

# wrong! Role can't be instanced
my $ttt = TTT::Role->new;

my $my_ttt = My::TTT->new();
$my_ttt->message; # 'Hello, I am TTT'
```

- before

- before
- after

- before
- after
- around

```perl
#!/usr/bin/perl
package TTT;
use Moose;
use Perl6::Say;
sub foo { say 'foo is called'; return 'foo'; };
before 'foo' => sub { say 'before 1' };
before 'foo' => sub { say 'before 2' };
after 'foo' => sub { say 'after 1' };
after 'foo' => sub { say 'after 2' };
around 'foo' => sub {
    say 'around 1';
    $_[0]->();
    return 'returns around 1'
};
around 'foo' => sub {
    say 'around 2';
    $_[0]->();
    return 'returns around 2'
};

package main;
my $ttt = TTT->new();
print $ttt->foo;
```



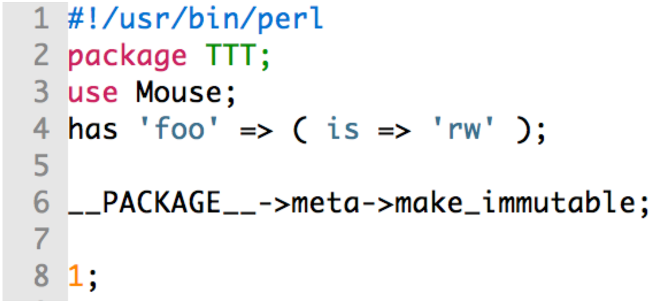
```
before 2
before 1
around 2
around 1
foo is called
after 1
after 2
returns around 2
```

- override/super

- override/super
- augment/inner

- override/super
- augment/inner
- ...

an easy way to speed up Moose



```perl
#!/usr/bin/perl
package TTT;
use Mouse;
has 'foo' => ( is => 'rw' );

__PACKAGE__->meta->make_immutable;

1;
```

Mouse is a lightweighted Moose. It provides a subset of Moose's functionality.

the compile time penalty

Though significant progress has been made over the years, the compile time penalty is a non-starter for some applications.

It's easy to do

s/Mouse/Moose/g;

Less is better sometimes

- less code

Less is better sometimes

- less code
- less tests

Less is better sometimes

- less code
- less tests
- less bugs

So. . .

Let's Moose from now on

Until. . .

Perl 6 is finished

More Info

- http://www.iinteractive.com/moose

More Info

- http://www.iinteractive.com/moose
- #moose on irc.perl.org

More Info

- http://www.iinteractive.com/moose
- #moose on irc.perl.org
- subscribe to moose@perl.org

Thanks!

Any Questions?