Curs de Perl: Moose

(Orientació a Objectes)

Alex Muntada*

<alexm@alexm.org>

2012.03.10

**use Perl;*

Índex

Capítol 1

Sistema d'objectes de Perl 5

La base del sistema d'objectes són els mòduls (o paquets).

1.1 Mòduls

Com en qualsevol altre llenguatge permeten:

- reutilització
- separació
- encapsulament
- ...

1.1.1 Espai de noms

```
use Foo;
say $INC{'Foo.pm'};
# /path/to/lib/Foo.pm

use Foo::Bar;
say $INC{'Foo/Bar.pm'};
# /path/to/lib/Foo/Bar.pm
```

1.1.2 On es busquen

```
our @INC;
use lib '/path/to/lib';

$ export PERL5LIB=/path/to/lib
$ perl -V
```

1.1.3 Quin aspecte tenen

Abans es feien d'aquesta manera:

```
package HelloWorld;

# our $VERSION = '1.00';

use vars qw( $VERSION );
$VERSION = '1.00';

# ...

1;
```

A partir de Perl 5.14 ja es poden fer d'aquesta altra:

```
use 5.014;
use Modern::Perl '2012';

package HelloWorld 1.00 {
    # ...
}

1;
```

1.1.4 Com s'utilitzen

```
use 5.014;
use Modern::Perl '2012';

package HelloWorld {
    sub hello {
        return "Hello, world!";
    }
}

# package main {
    say HelloWorld::hello();
# }
```

Llistat 1.1: *HelloWorld-hello.pl*

1.1.5 Com es defineixen

Aquesta és la definició d'un mòdul més complex:

```
use 5.014;
use Modern::Perl '2012';

package HelloWorld 1.00 {
    use Exporter qw( import );

    our @EXPORT    = qw( hello );
    our @EXPORT_OK = qw( japh );
```

```
    sub hello {
        return "Hello, world!";
    }

    sub japh {
        return "Just another Perl hacker,";
    }
}

1;
```

Llistat 1.2: *lib/1.00/HelloWorld.pm*

1.1.6 Crida a les funcions del mòdul

```
use 5.014;
use Modern::Perl '2012';

use lib 'lib/1.00';
use HelloWorld 1.00;

say hello();
say HelloWorld::japh();
```

Llistat 1.3: *hello1.pl*

1.1.7 Importació de símbols

```
use 5.014;
use Modern::Perl '2012';

use lib 'lib/1.00';
use HelloWorld 1.00 qw( japh );

# say hello(); => FAIL
say japh();
```

Llistat 1.4: *hello2.pl*

1.1.8 require vs. use

`require`...

- s'avalua en temps d'execució,
- un sol cop,
- inclou el resultat al context actual.

`use` fa el mateix però…

- s'avalua en temps de compilació,

- només admet *barewords* com a nom del mòdul,
- sense `eval` no es poden utilitzar variables pel nom,
- permet afegir una restricció de versió mínima del mòdul.

1.1.9 use vs. no

```
# use Module;
BEGIN { require Module; Module->import(); }

# use Module ();
BEGIN { require Module; }

# use Module LIST;
BEGIN { require Module; Module->import( LIST ); }

# no Module LIST;
BEGIN { require Module; Module->unimport( LIST ); }
```

1.2 Classes, mètodes i atributs

El mateix exemple d'abans…

```
use 5.014;
use Modern::Perl '2012';

package HelloWorld 1.00 {
    use Exporter qw( import );

    our @EXPORT    = qw( hello );
    our @EXPORT_OK = qw( japh );

    sub hello {
        return "Hello, world!";
    }

    sub japh {
        return "Just another Perl hacker,";
    }
}

1;
```

Llistat 1.5: *lib/1.00/HelloWorld.pm*

1.2.1 Constructors, atributs, getters i setters

Els objectes són en realitat referències:

```
use 5.014;
use Modern::Perl '2012';

package HelloWorld 1.01 {
```

```
    use Carp qw( croak );

    sub new {
        my ( $class, %args ) = @_;

        my $obj = {
            hello => "Hello, world!",
            japh  => "Just another Perl hacker,",
            %args,
        };

        return bless $obj, $class;
    }

    sub hello {
        my $self = shift;

        return @_ ? $self->set_hello(@_) : $self->get_hello();
    }

    sub japh {
        my $self = shift;

        return @_ ? $self->set_japh(@_) : $self->get_japh();
    }

    sub AUTOLOAD {
        my $self = shift;
        my @args = @_;

        our $AUTOLOAD;
        my ($method) = $AUTOLOAD =~ /::(\w+)$/;

        if    ( $method =~ /^get_(\w+)/ && exists $self->{$1} ) {
            return $self->{$1};
        }
        elsif ( $method =~ /^set_(\w+)/ && exists $self->{$1} ) {
            my $new_value = $args[0];
            my $old_value = $self->{$1};
            $self->{$1}   = $new_value;

            return $new_value;
        }
        else {
            croak "Bad method name <$method>!";
        }
    }

    sub DESTROY { }
}

1;
```

Llistat 1.6: *lib/1.01/HelloWorld.pm*

1.2.2 Instàncies

```
use 5.014;
use Modern::Perl '2012';

use lib 'lib/1.01';
use HelloWorld 1.01;

my $h = HelloWorld->new();
say $h->hello();
say $h->japh();

my $j = HelloWorld->new( hello => 'JAPH!' );
say $j->hello();
```

Llistat 1.7: *hello3.pl*

1.2.3 Accessors

```
use 5.014;
use Modern::Perl '2012';

use lib 'lib/1.01';
use HelloWorld 1.01;

my $j = HelloWorld->new();

# $j->hello(); => OK

my $old_hello = $j->get_hello();
$j->set_hello( $j->get_japh() );

say $old_hello;
say $j->get_hello();
# say $j->get_foobar(); => FAIL
```

Llistat 1.8: *hello4.pl*

1.3 Herència

```
use 5.014;
use Modern::Perl '2012';

package Baz {
    use parent qw( Foo Bar );
}

package Baz {
    BEGIN {
        require Foo;
        require Bar;
        push our @ISA, qw(Foo Bar);
    }
```

```
}
```

Capítol 2

Moose

2.1 Introducció

- Construït damunt del sistema d'objectes de Perl 5.
- Pren característiques d'altres llenguatges com Smalltalk, Common Lisp i Perl 6.
- La forma més moderna de programar amb objectes en Perl modern.
- Té sabors alternatius: `Mouse`, `Moo`, `Mo`.

2.1.1 Classes, mètodes i atributs

```
use 5.014;
use Modern::Perl '2012';

package Pet {
    use Moose;

    has 'name' => (
        is  => 'ro',
        isa => 'Str',
    );
}

package Cat {
    use Moose;

    extends 'Pet';

    has 'diet' => (
        is  => 'rw',
        isa => 'Str',
        );

    has 'birth_year' => (
        is       => 'ro',
        isa      => 'Int',
            default  => sub { (localtime)[5] + 1900 },
    );
```

```
    sub meow {
        say "Meow!";
    }

    sub age {
        my $self = shift;
        my $year = (localtime)[5] + 1900;

        return $year - $self->birth_year;
    }
}

package main {
    my $fat = Cat->new( name => 'Fatty', age => 8, diet => 'Sea Treats' );
    say $fat->name, ' eats ', $fat->diet;

    $fat->diet('Low Sodium Kitty Lo Mein');
    say $fat->name, ' eats ', $fat->diet;
}
```

Llistat 2.1: *cat.pl*

2.1.2 Hello, world!

```
use 5.014;
use Modern::Perl '2012';

package HelloWorld 2.00 {
    use Moose;

    has 'hello' => (
        is      => 'rw',
        isa     => 'Str',
        default => 'Hello, world!',
    );

    has 'japh' => (
        is      => 'ro',
        isa     => 'Str',
        default => 'Just another Perl hacker,',
    );
}

1;
```

Llistat 2.2: *lib/2.00/HelloWorld.pm*

```
use 5.014;
use Modern::Perl '2012';

use lib 'lib/2.00';
use HelloWorld 2.00;

my $h = HelloWorld->new();

say $h->hello();
```

```
$h->hello('hello');
say $h->hello();

say $h->japh();
# $h->japh('japh'); => FAIL

my $j = HelloWorld->new( hello => 'JAPH!' );
say $j->hello();
```

Llistat 2.3: *hello5.pl*

2.1.3 Hello, world! *accessible*

```
use 5.014;
use Modern::Perl '2012';

package HelloWorld 2.01 {
    use Moose;

    has 'hello' => (
        is      => 'rw',
        isa     => 'Str',
        default => 'Hello, world!',
        reader  => 'get_hello',
        writer  => 'set_hello',
    );

    has 'japh' => (
        is      => 'ro',
        isa     => 'Str',
        default => 'Just another Perl hacker,',
        reader  => 'get_japh',
        writer  => 'set_japh',
    );
}

1;
```

Llistat 2.4: *lib/2.01/HelloWorld.pm*

```
use 5.014;
use Modern::Perl '2012';

use lib 'lib/2.01';
use HelloWorld 2.01;

my $j = HelloWorld->new();

# $j->hello(); => FAIL

my $old_hello = $j->get_hello();
$j->set_hello( $j->get_japh() );

say $old_hello;
say $j->get_hello();
# say $j->get_foobar(); => FAIL
```

Llistat 2.5: *hello6.pl*

2.2 Modificadors de mètodes

2.2.1 before, after, around

```
package Cat {
    # ...

    before 'age' => sub {
        my $self = shift;

        die "cannot ask for " . $self->name . " age"
            unless $self->doesnt_care;
    };

    after 'age' => sub {
        warn "too late to care\n";
    };
}

package HelloWorld {
    # ...

    around 'hello' => sub {
        my $orig = shift;
        my $self = shift;

        my $text = join q{ :: }, @_;

        return $self->$orig($text);
    };
}
```

2.2.2 super, override

```
package CheaterKitty {
    # ...

    extends 'Cat';

    override 'age' => sub {
        my $self = shift;

        my $age = super();
        $age   /= 2
            if $age > 1;

        return $age;
    };
}
```

2.2.3 inner, augment

```
use 5.014;
use Modern::Perl '2012';

package Document {
    use Moose;

    sub as_xml {
        my $self = shift;

        my $xml = "<document>";
        $xml   .= inner();
        $xml   .= "</document>";

        return $xml;
    }
}

package Report {
    use Moose;

    extends 'Document';

    augment 'as_xml' => sub {
        my $self = shift;

        return "<report>This is a report</report>";
    };
}

package main {
    my $report = Report->new();
    say $report->as_xml();
}
```

Llistat 2.6: *report.pl*

2.3 Subtipus i coercions

```
use 5.014;
use Modern::Perl '2012';

package Foo {
    use Moose;
    use Moose::Util::TypeConstraints;
    use Moose::Autobox;

    subtype 'ArrayRefOfPositiveInts',
        as 'ArrayRef[Int]',
        message { "The array you provided has a negative number" },
        where {
            my @array    = @$_;
            my $positive = 1;
```

```
            for my $next (@array) {
                $positive &&= ( $next >= 0 ) or last;
            }

            return $positive;
        };

    coerce 'ArrayRefOfPositiveInts',
        from 'Int',
        via { [ $_ ] };

    has 'sizes'    => (
        is     => 'ro',
        isa    => 'ArrayRefOfPositiveInts',
        coerce => 1,
    );
}

package main {
    Foo->new( sizes => 42 );

    # Foo->new( sizes => [ 42, -42 ] ); => FAIL
}
```

Llistat 2.7: *foo.pl*

2.4 Rols i trets

2.4.1 Per què?

- Objectes: què és el sistema.
  - Com són els objectes.
  - El model de dades és estable.
  - Habitualment és necessari documentar-lo.
- Rols: què fa el sistema.
  - La lògica de negoci és canviable en el temps.
  - Facilita la creació de tests.
  - Eviten documentar els algorismes, el codi és la documentació.
  - Els trets a Moose són rols composats en temps d'execució.
- Vistes: interacció amb els usuaris.
  - Diferents dispositius tenen entrades i sortides diferents.
  - Aspectes diversos: presentació, blog, llibre, etc.
  - Formats de sortida: HTML, text, ODT, PDF, etc.

2.4.2 Composició de rols

```
use 5.014;
use Modern::Perl '2012';

package HelloWorld 2.02 {
    use Moose::Role;

    requires qw( hello );

    has 'japh' => (
        is      => 'ro',
        isa     => 'Str',
        default => 'Just another Perl hacker,',
    );

    sub japh_hello {
        my $self = shift;

        return $self->japh . ' ' . $self->hello . '!';
    }
}

1;
```

Llistat 2.8: *lib/2.02/HelloWorld.pm*

```
use 5.014;
use Modern::Perl '2012';

package Pet 2.02 {
    use Moose;

    has 'name' => (
        is  => 'ro',
        isa => 'Str',
    );
}

1;
```

Llistat 2.9: *lib/2.02/Pet.pm*

```
use 5.014;
use Modern::Perl '2012';

package Cat 2.02 {
    use Moose;

    extends 'Pet';

    has 'diet' => (
        is  => 'rw',
        isa => 'Str',
    );

    has 'birth_year' => (
```

```
        is       => 'ro',
        isa      => 'Int',
        default  => sub { (localtime)[5] + 1900 },
    );

    has 'hello' => (
        is      => 'rw',
        isa     => 'Str',
        lazy    => 1,
        default => sub { "hello " . shift->name },
    );

    with 'HelloWorld';

    sub meow {
        say "Meow!";
    }

    sub age {
        my $self = shift;
        my $year = (localtime)[5] + 1900;

        return $year - $self->birth_year;
    }
}

1;
```

Llistat 2.10: *lib/2.02/Cat.pm*

```
use 5.014;
use Modern::Perl '2012';

use lib 'lib/2.02';
use Cat 2.02;

my $fat = Cat->new( name => 'Kitty', age => 8, diet => 'Sea Treats' );
say $fat->japh_hello()
    if $fat->DOES('HelloWorld');
```

Llistat 2.11: *hello7.pl*

2.5 Delegació

Importa mètodes d'un atribut sense crear una relació d'herència o composició de rols:

```
package Website {
    use Moose;

    has 'uri'   => (
        is      => 'ro',
        isa     => 'URI',
        handles => [qw( host path )],
    );
}
```

2.6 Constructors i destructors

Compte! No definiu cap mètode `new` ni `DESTROY`. Disposeu d'aquests mecanismes:

- `BUILDARGS` abans de crear l'objecte.
- `BUILD` després de crear l'objecte.
- `DEMOLISH` durant la destrucció de l'objecte.

2.7 Protocol de meta-objectes i immutabilitat

Tota aquest màgia és possible gràcies al MOP:

```
my $meta = User->meta();

for my $attribute ( $meta->get_all_attributes ) {
    print $attribute->name(), "\n";

    if ( $attribute->has_type_constraint ) {
        print "  type: ", $attribute->type_constraint->name, "\n";
    }
}

for my $method ( $meta->get_all_methods ) {
    print $method->name, "\n";
}
```

Per optimitzar l'execució i per evitar que una classe canviï:

```
__PACKAGE__->meta->make_immutable;
```

2.8 Extensions de Moose

Algunes de les extensions recomanades a `Moose::Manual::MooseX`:

- `Moose::Autobox`
- `MooseX::StrictConstructor`
- `MooseX::Params::Validate`
- `MooseX::Getopt`
- `MooseX::Singleton`

Altres extensions interessants:

- `MooseX::Declare`
- `MooseX::ClassAttribute`
- `MooseX::Daemonize`
- `MooseX::Role::Parameterized`

- `MooseX::SemiAffordanceAccessor`
- `MooseX::NonMoose`

Capítol 3

Pràctica

A partir de la classe `HelloWorld`, dissenyeu un sistema de presentació de diapositives que pugui generar sortida en diferents formats.

- Heu d'utilitzar `Moose`.
- Separeu el què és el sistema del què fa amb `Moose::Role`.
- Utilitzeu els patrons Model-Vista-Controlador (MVC) o Data-Context-Interaction (DCI).
- El sistema ha de generar com a mínim una versió en HTML.
- Opcionalment pot generar versions en ODF, PDF, etc.

3.1 TMTOWTDI

Debat sobre les propostes:

- Punts forts i punts febles.
- Alternatives.