

새로운
Perl OOP로
개과천선하기

Yet Another Sunday Perl Seminar

aero

Perl FUD

- Perl은 구닥다리 언어다.
- Perl은 OOP를 잘 지원하지 못한다.
- Perl하면 고리타분하고 P*t*on, R**y하면 신세대다.

-모 블로그에서 인용

“오늘 학교에서 펄책 들여보고 있으니 옆의 형이 사장되가는 언어를 왜 공부하냐고 하더군요 π,π ”

Perl OOP의 시작

Programming is Hard, Let's Go Scripting...

<http://www.perl.com/pub/a/2007/12/06/soto-11.html>

I don't really know much about Python. I only stole its object system for Perl 5. I have since repented.

- Larry Wall

- Perl OOP는 Perl 5부터 도입되었다.
- Perl 5 != (보통의 Perl에 대한 인식 = Perl 4 + CGI)

<추천링크>

<http://www.slideshare.net/Tim.Bunce/perl-myths-200802-with-notes>

<http://greenokapi.net/talks/ReadablePerl.pdf>

Perl OOP 기초

- Perl 클래스는 Package를 기반으로 구성된다.

Java

```
class Point
{
    private int X;
    private int Y;
    public Point(int i, int j)
    {
        X=i;
        Y=j;
    }
    . . .
}
```

//객체 생성

```
Point p = new Point(10,20);
```

Perl

```
package Point;

sub new {
    my ($class, $i, $j) = @_;
    my $self = { X=>$i, Y=>$j };
    bless $self, $class;
    return $self;
}
. . .
```

#객체 생성

```
my $p = Point->new(10,20);
```

Perl OOP 기초

- Perl 객체는 레퍼런스를 클래스 이름으로 태깅(Tagging)한 것이다.
- 따라서 Perl 객체는 대부분 해시기반이지만 레퍼런스에 담기는 스칼라,배열,해시,서브루틴 등 모두 객체가 될 수 있다.

```
package Point;
```

```
sub new {
    my ($class, $i, $j) = @_ ;
    my $self = { X=>$i, Y=>$j }; #익명해시(해시 레퍼런스)를 $self(이것도 관례적이름)에 넣고
    bless $self, $class; #그것을 bless 함수를 사용하여 $self를 $class로 태깅
    return $self;
}
```

```
package main;
use Data::Dumper;
my $p = Point->new(10,20);
print Dumper($p);
```

<결과>

```
$VAR1 = bless( {
    'X' => 10,
    'Y' => 20,
}, 'Point' );
```

Perl OOP 기초

- Perl 클래스 메소드는 패키지내의 서브루틴으로 구현된다.
- Perl 에서는 생성자(new)도 클래스 메소드다.

```
package Point;
sub new {
    my ($class, $i, $j) = @_;
    my $self = { X=>$i, Y=>$j };
    bless $self, $class;
    return $self;
}
sub get_point {
    my ($self) = @_;
    return ($self->{X}, $self->{Y});
}
sub add_x {
    my ($self, $n) = @_;
    $self->{X}+=$n;
}

```

실제 사용시에는 대부분 package는 별도의 파일로 모듈형식으로 만든다.

```
package main;
my $p = Point->new(10,20);
my ($x,$y) = $p->get_point();
print "$x $y\n"; # 10 20
$p->add_x(2);
my ($x2,$y2) = $p->get_point();
print "$x2 $y2\n"; # 12 20

```

Perl OOP 기초

- 메소드 호출 방법

1. `my $p = Point->new(10,20); # 직접호출`
2. `my $p = new Point (10,20); # 간접호출`
3. `my $p = Point::new('Point', 10, 20); # 위 호출들은 이 같은 의미`

1. `$p->add_x(2); # 직접호출`
2. `add_x $p (2); # 간접호출`
3. `Point::add_x($p,2); #위 호출들은 이 같은 의미`

- 1,2는 상속시 현 객체에 메소드가 없으면 상속을 거슬러 올라가며 찾아서 호출한다.
- 2의 간접호출 방식은 Java,C++과 비슷하게 보인다고 저렇게 사용하는 사람도 있는데 경우에 따라 예상치 않은 side effect가 있을 수 있으므로 사용하지 않는 것이 정석임. (직접호출 방식만 사용하자.)

Perl OOP 상속

```
package Point;

sub new {
    my ($class, $i, $j) = @_;
    my $self = { X=>$i, Y=>$j };
    bless $self, $class;
    return $self;
}

sub get_point {
    my ($self) = @_;
    return ($self->{X}, $self->{Y});
}

sub add_x {
    my ($self, $n) = @_;
    $self->{X}+=$n;
}
}
```

```
package Point3D;
use base 'Point';
#our @ISA=qw/Point/; #와 같은 뜻

sub new {
    my ($class, $i, $j, $k) = @_;
    my $self = $class->SUPER::new($i,$j);
    $self->{Z} = $k;
    bless $self, $class;
    return $self;
}

sub get_point {
    my ($self) = @_;
    return ( $self->SUPER::get_point(), $self->{Z} )
}
}
```

```
package main;
my $p = Point3D->new(10,20,30);
my ($x,$y,$z) = $p->get_point();
print "$x $y $z\n";
$p->add_x(5); #Point3D에 없으므로 Point쪽을 찾아 호출
my ($x2,$y2,$z2) = $p->get_point();
print "$x2 $y2 $z2\n";
```

Perl OOP 상속

- Perl에서 상속은 @ISA배열로 단지 없는 메소드를 어느 부모 클래스(패키지)에서 탐색할 것인가를 지정하는 것이다.
- 다이아몬드 상속구조에서 메소드 탐색시 MRO방식을 사용할 수 있다.(Class::C3, mro, MRO::Compat)
<http://translate.google.com/translate?hl=ko&u=http://gihyo.jp/dev/serial/01/modern-perl/0002>
- 다중 상속을 지원한다.

Perl OOP의 특징

- 기본으로 최소한의 기능만 지원하며 Java,C++같이 키워드를 통해서 특정한 방식(private,public등의 캡슐화, virtual,abstract등의 추상화 등)의 객체구현을 강요하지 않는다.
- 위의 특징들은 Closure 등 문법적 기교와 모듈 등으로 사용가능하다.
- 하나의 OOP방식이 아닌 다양한 방식으로 아주 유연하게 확장시킬 수 있다.(장점이자 단점)

Perl OOP 모듈의 역사

- Perl 클래스는 상속시 멤버변수,메소드의 오버라이드를 기본적으로 막지 않으므로 예상치 못하게 부모 클래스의 멤버변수를 덮어쓰거나 존재하지 않았던 멤버변수의 추가 등의 작업이 자유롭다.
- 이것을 방지하기 위해(private 속성) 멤버변수에 패키지이름을 앞에 붙이는 방법 Psuedo hash, Restricted hash와 함께 타입렉시컬을 써서 컴파일 타임에 검사하는 방법, Inside-Out 객체를 사용하는 방법 등 다양한 방법과 기능의 모듈이 사용되었다.
- 하지만 실수를 막기 위해 그러한 테크닉과 모듈을 사용하는 비용이 문서,주석 등을 명확히 하고 작성자가 알아서 조심하는 비용보다 더 크지 요즘은 잘 쓰지 않게 되었다.

Perl OOP 모듈의 역사

- Perl 객체를 다룰 때는 모든 변수를 캡슐화 하여 직접 접근은 하지 않으며 accessor 등의 메소드를 통해서 하는 것이 일반적 방법이다.
- 이런 멤버변수, accessor 등을 일일이 수동으로 만들려면 반복적인 귀찮고 번거로운 작업이 많다.
- 따라서 이런 작업을 편하게 하기 위해 Damian Conway씨의 책 Object Oriented Perl에도 소개되어 있는 Class::Struct, Class::MethodMaker, Class::Std 등 여러 모듈이 나왔으나 이것들도 요즘에는 거의 쓰이지 않는다.

요즘의 대세

Class::Accessor::Fast

Moose / Mouse

Class::Accessor::Fast

Class::Accessor 모듈의 일부기능을 포기하면서 속도를 높인 모듈

```
package Point;
use base qw/Class::Accessor::Fast/;
__PACKAGE__->mk_accessors(qw/x y/);

package main;
my $p = Point->new( { x=>10, y=>20 } );
print $p->x, " ", $p->y, "\n";
$p->x(15);
print $p->x, " ", $p->y, "\n";
```

부모,자식클래스간 공유데이터를 원하면 Class::Data::Inheritable,
Class::Accessor::Grouped

Moose

- Meta Object Protocol(런타임시에 클래스 계층구조와 메소드 정보를 살펴보고 조작할 수 있도록 하는 기능,API)를 기반으로 하는 어느 것보다 진보된 현대적 객체시스템
- Perl 6의 객체시스템과 유사
- 덩치가 커서 초기 로딩속도가 좀 느리며 메모리를 많이 먹는경향이 있다.
- Moose의 단점을 보완하기 위해 Moose의 잘 쓰이지 않은 기능을 빼서 경량화하고 속도를 높인 Mouse도 있음
- Any::Moose 모듈을 사용하면 알아서 Moose/Mouse를 로딩해줌

Moose 예제

```
package Point;

use Moose;

has 'x' => (is => 'rw', isa => 'Int');
has 'y' => (is => 'rw', isa => 'Int');

sub clear {
    my $self = shift;
    $self->x(0);
    $self->y(0);
}

package Point3D;
use Moose;

extends 'Point';

has 'z' => (is => 'rw', isa => 'Int');

after 'clear' => sub {
    my $self = shift;
    $self->z(0);
};
```

```
package main;

my $p = Point3D->new( x=>10,y=>20,z=>30 );
print $p->x, " ", $p->y, " ", $p->z, "\n« ;
$p->x(15);
print $p->x, " ", $p->y, " ", $p->z, "\n« ;
$p->clear;
print $p->x, " ", $p->y, " ", $p->z, "\n« ;
```

<결과>

```
10 20 30
15 20 30
0 0 0
```

뭘 써야 하나요?

- Perl 기본 객체시스템은 이해해야 합니다.
(That's why you should learn perl if you want to learn OO! You can learn how to make an object system, not just how to use it. – Dan Kogai 출처: <http://www.dan.co.jp/~dankogai/yapcasia2008/psl.html>)
- 간단한 객체는 기본객체 시스템 사용
- 중간규모의 작업이며 속도와 리소스를 생각한다면 Class::Accessor::Fast, Mouse
- 대규모작업이라면 Moose, Mouse
- 제일 잘나가는 Perl Web Framework인 Catalyst 도 Class::Accessor::Fast를 거쳐 Moose로 옮겨가고 있음

읽어볼 만한 참고자료

- <http://www.slideshare.net/dtreder/moose-527243>
- <http://www.bofh.org.uk/2009/02/22/moose-for-ruby-programmers>
- <http://www.kevinboone.com/javaperl.html>
- <http://www.iinteractive.com/moose/>