

DSLs в Perl

Как?

Руслан Закиров <ruz@bestpractical.com>
Best Practical Solutions, 2008

Прототипы

Прототипы

(\$)

(&)

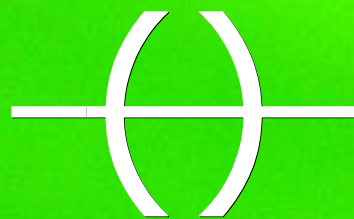
(@)

Прототипы (завтрак)

```
set($$) {...}
```

```
set key => $val;
```

Прототипы (запястья)



Прототипы

(&)

блок

функция

Прототипы

```
grep {} @_  
sub grep(&@) {}
```

Прототипы

```
sub run(&) {};  
run { my $... };
```

Методы

Методы

$\$0 = \text{new } X \text{ } k \Rightarrow \$v;$

Методы

```
perl -e 'title is „foo“;'
```

Методы

Невозможно найти
метод "title" в пакете
"is"

Методы

МОЖЕТ ВМЫ ЗАБЫЛИ
ЗАГРУЗИТЬ "IS"?

Методы

этап исполнения

Методы

```
perl -e 'sub foo { title is  
    „foo“ }'
```

нет ошибки

Методы

```
my $a = {};  
local *is::AUTOLOAD = sub {  
    shift; # is  
    $a{$AUTOLOAD} = join ' ', @_  
};  
$call->(); # что-то
```

Объединим

Объединим (img_simple.pl)

```
sub img(&) {  
  my $code = shift;  
  my %attr;  
  local *is::AUTOLOAD = sub {...};  
  $code->();  
  my $attrs = join ' ',  
    map $_.'='."$attr{$_},  
    keys %attr;  
  print „<img $attrs />“;  
}
```

Слишком
просто?

Усложним (img_strict.pl)

```
my %attr_checks = (  
  img => {  
    _mandatory => [qw(src alt)],  
    src => {  
      canonicalizer => sub {...},  
      provides => sub {...},  
    },  
    # ...  
  },  
);
```

Усложним (img_strict.pl)

```
canonicalizer => sub {  
    return "/static/images/${_}[0]"  
    unless $_[0] =~ /^\\/;/  
    return $_[0];  
}
```

Усложним (img_strict.pl)

```
provides => sub {  
    return unless my ($w, $h) =  
        ($_[0] =~ /-(\d+)x(\d+)\./);  
    return (width => $w, height => $h);  
}
```

Результат (img_strict.pl)

```
img {  
  alt is 'feed',  
  src is 'f-14x14.png'  
};
```

Результат (img_strict.pl)

```

```

Отладка

Отладка (carp.pl)

нет атрибута
'boo' у тега 'img'
at carp.pl **line 10**

Отладка (carp.pl)

```
local $Carp::CarpLevel = 1;  
Carp::croak(...);
```

Отладка (carp.pl)

нет атрибута

'boo' у тега 'img'

at carp.pl **line 16**

МЕСТО ВЫЗОВА

Отладка (carp.pl)

```
main::___ANON___()  
called at carp.pl line 13
```

Отладка (carp.pl)

```
sub img(&) {  
    local *___ANON___  
        = "img_impl";
```

```
    ...
```

```
}
```

Отладка (carp.pl)

bla-bla at carp.pl line 17

main::**img_impl**() called at...

main::**img**('CODE(...)') called at...

Грабли №1

Грабли №1

```
page {...};  
sub page(&) {...};
```

Грабли №1

Невозможно вызвать
метод "page" без
пакета или объекта

Грабли №2

Грабли №2

```
sub page(&) {  
    local *title::page = sub {}  
    shift->();  
};  
page { page title „qwe“ };
```

Грабли №2

Методы
проигрывают
протипам

Избавляемся от
методов

Без методов (wo_methods.pl)

```
our %ATTR;  
sub attrs(&) {  
    %ATTR = shift->()  
}  
sub img(&) {  
    local %ATTR;  
    $code->();  
    ...  
}
```

Без методов (wo_methods.pl)

```
img { attrs {  
  alt => 'boo', src => 'href'  
} };
```

Вложенные структуры

Вложения (div_simple.pl)

```
<div> <div>
```

```
    ЧТО-ТО
```

```
</div> </div>
```

Вложения (div_simple.pl)

```
sub div(&) {  
    my $code = shift;  
    my $inside = $code->();  
    return "<div>$inside</div>";  
}  
print div { div {'some'} };
```

Вложения (div_simple.pl)

```
<div> <div>
```

```
    ЧТО-ТО
```

```
</div> </div>
```

Если что-то
сложнее?

Усложняем

```
div {  
  div {1}; div {2}  
};
```

Усложняем

```
<div>
```

```
<div> 2 </div>
```

```
</div>
```

Усложняем

ФИГНЯ ВЫШЛА

Простое решение

Простое решение (div_sol1.pl)

```
sub div(&) {  
    my $code = shift;  
    my @inside = $code->();  
    return "<div>@inside</div>";  
}
```

Простое решение (div_sol1.pl)

```
print div { div {1}, div {2} },  
         div { div {3}, div {4} };
```

Код не вставишъ

:(

явный print
отстой

запятые в топку

Да придут с
вами КОНТЕКСТЫ

Контексты (context1.pl)

```
unless ( defined wantarray ) {  
    # void  
}  
elsif ( wantarray ) {  
    # array  
}  
else {  
    # scalar  
}
```

Контексты

```
sub div(&) {  
    my $res = join " , shift->();  
    unless ( defined wantarray ) {  
        print „<div>$res</div>“;  
    } else {  
        return „<div>$res</div>“;  
    }  
}
```

Контексты (context1.pl)

```
div { div {1}; div {2} };
```

```
<div>1</div>
```

```
<div><div>2</div></div>
```

Буферизация

Бу-эфиры (buffers1.pl)

```
sub buffered {  
    my $buf = "";  
    local *STDOUT;  
    open STDOUT, '>', \ $buf;  
    return $buf . join("", shift->());  
}
```

Бу-эфиры (buffers1.pl)

```
sub div(&) {  
    my $res = buffered(shift);  
    return „<div>$res</div>”  
    if defined wantarray;  
    print „<div>$res</div>”;  
}
```

Бу-эфиры (buffers1.pl)

```
div {  
  div{'menu'};  
  my $some = 'some';  
  div{$some},  
  div{'tail'}  
};
```

ЯХУ! :)

ПЕРИС ХИЛТОН

БЛА-БЛА. ТУТ Я

УСТАЛ

РИСОВАТЬ

СЛАЙДЫ

Грабли №3

Грабли №3

```
div {  
  'some';  
  my $some = 'some';  
  div{$some}  
};
```

Грабли №3

```
print 'some';  
outs('some');  
x {'some'}  
{ my $x=...; 'some', div{$x} }
```

Установка функций

Установка функций

```
my @tags = qw(a b);  
foreach my $t ( @tags ) {  
    no strict 'refs';  
    *{'main::' . $t} = sub (&) {  
        ...  
    };  
}
```

Установка функций

`sub import;`

`Exporter;`

`Symbol;`

И прочие

Прото-цепочки

Это не химия
И не биология

ЭТО
(&;\$)

Прото-цепочки

```
sub a(&,$) {  
    print 'a ', context wantarray, "\n"  
}  
sub b(&,$) {  
    print 'b ', context wantarray, "\n"  
}
```

Прото-цепочки

a {} b {} ;

b scalar

a void

Промежуточное представление

Пром-представления

```
*{'main::'.$t} = sub (&,$) {  
  my ($code, $next) = @_;  
  unless ( defined wantarray ) {  
    return _tag($t, $code, $next);  
  } else {  
    return bless sub {  
      return _tag($t, $code, $next)  
    }, 'MyTag';  
  }  
};
```

Пром-представления

```
package MyTag;  
use overload "" => sub {  
    return buffered($_[0])  
};  
1;
```

Пром-представления

a {'head'}

b {'middle'}

c {'tail'};

im_represent.pl

Пром-представления

```
my %h = ( k1 => 'v2', k2 => 'v2');  
a {  
  my ($k, $v) = ("", "");  
  my $closure = b { $k } c { $v };  
  while (($k, $v) = each %h ) {  
    $closure->();  
  }  
  return "";  
};
```

Грабли №4

Габбли №4

```
div {print „hehe“};
```

```
<div>hehe1</div>
```

Грабли №4

```
sub buffered {  
  ...  
  my $tail = join(" ", shift->());  
  return $buf . $tail  
};
```

Грабли №4 (компромис)

```
my @tail = shift->();  
return join "", @tail  
    unless length $buf;  
my $tail = pop @tail;  
$tail = "" unless blessed($tail);  
return $buf . join("", @tail) . $tail;
```

[tail_problem.pl](#)

Экранирование

Экранирование

```
sub _escape {  
    return unless defined $_[0];  
    my $v = shift;  
    $v = „$v“;  
    $v =~ s/.../.../g;  
    return $v;  
}
```

Экранирование

```
sub escape(@_) {  
  return map  
    blessed($_)  
    && $_->isa('MyTag')  
    ? $_ : _escape($_),  
  @_;  
}
```

Экранирование

```
sub buffered {  
    ...  
    return join "", $buf, escape(@tail);  
}  
sub _tag {  
    my ($tag, $code, $next) = @_;  
    ...  
    print join "", $res, escape $next;  
}
```

Все работает
escaping.pl

На закуску к экранам

```
sub outs(@) { print _escape(join ", @_") }
```

```
sub raw(@) { print join ", @_" }
```

```
# грабли №3
```

```
div { outs('some'); div {...} };
```

MyTD ГОТОВ.

Осталось

ТОЛЬКО...

Недостающие части

объединить все примеры

запаковать все

```
sub template($$);  
# template 'index' => run {};  
sub show($@);  
# show 'index', arg => $arg, ...;
```

Другие промежуточные представления

Что хотите

массивы

хеши

объекты

Тут должны были
быть грабли №5

Доп инструменты

`overload::constant q => sub { }`

`caller` – возвращает прототип,
контекст и многое другое,
например аргументы

`Want` – новый уровень работы с
контекстами, можно расширить до
работы с `tail`-вызовами,
аргументами

BCE