Filtres de codi
(source filters)

Xavier Noria
Barcelona.pm
25 de novembre de 2004

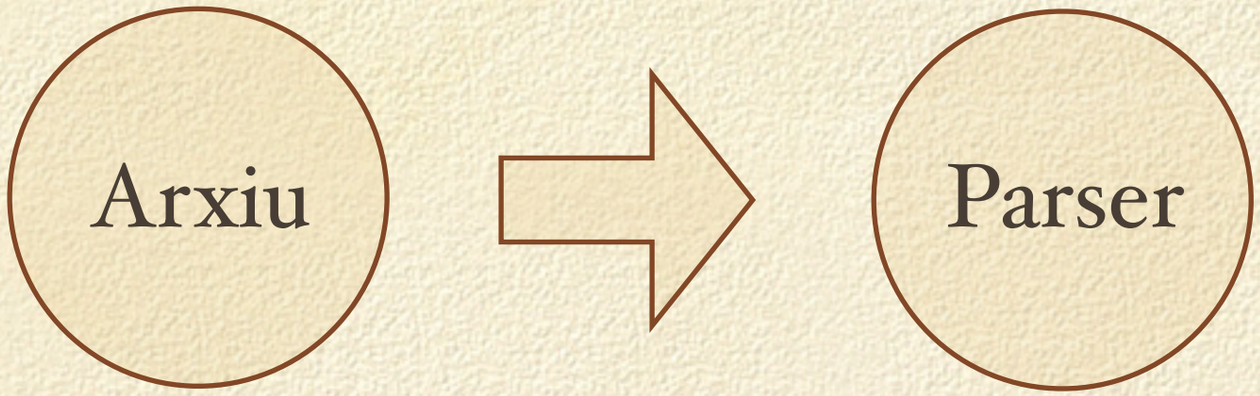
Què són?

Què són els filtres de codi?

- Els filtres de codi són uns mòduls especials que intercepten un flux de codi abans que li arribi al parser

- Reben el codi, normalment el transformen, i incorporen el resultat al flux de codi de nou per a que sigui processat pel parser

- Històricament s'ideen com una solució per a poder distribuir codi xifrat


Arxiu
Parser


Arxiu
Filtre
Parser

Exemple

```
use Filter::cpp;

#define FRED 1
$a = 2 + FRED;
print "a = $a\n";
#ifdef FRED
print "Hello FRED\n";
#else
print "Where is FRED\n";
#endif
```

```
a = 3
Hello FRED
```

Com s'escriuen?

Com s'escriuen?

- Hi ha tres maneres d'escriure filtres de codi:
  - C
  - Filtres externs
  - Perl

Filtres en C

- El filtres en C fan anar directament els *hooks* que Perl ofereix a l'efecte
- En general no cal complicar-se tant, llevat que el propòsit ho mereixi
- Per exemple, poden saltar-se el parser

Exemple

```
print "Hello World!\n";
```

```
$ perlcc -B -o foo foo.pl
```

```
#! perl
use ByteLoader 0.05;
CBLPdarwin-2level^@0.05^@^@^@^@^D^@^@^@^D^LY
a^@^@^@^Ad^Me@^Lh ^^@^@^@^Ad^E^L^B\234     ^D\204^@^D^N^@^@^@^N...
^@^V^^@^@^@^Bd^Bt^@^@^@^D^L^A\230^^@^@^@^Cd^B^LW8Pmain^@^T
^@^@^K^N^@^@^@^Emain^@^E^@^@^@^HC^@^@^@^H^R^@^@^@PNmain::
_<foo.pl^@^T^@^@`^M^R^@^@^@^CR^@\
...
```

```
$ perl foo
Hello World!
```

Filtres externs

- Hom pot delegar la tasca a un filtre extern
- En tal cas disposem de dos mòduls específics
  - Filter::exec
  - Filter::sh

Exemple

```
use Acme::Pythonic;
use Filter::sh qw(cat > eraseme && echo 'print "done\n"')

my $s = 0
for my $x in 1..100:
    $s += $x
print "$s\n"
```

```
% perl foo.pl
done
% cat eraseme

my $s = 0;
for my $x (1..100) {
    $s += $x
}
print "$s\n";
```

Filter::Util::Call

- Filter::Util::Call, de Paul Marquess, fa accessible la API C des de Perl

- Les principals funcions són:

  - `filter_add()`

  - `filter_read()`

  - `filter_del()`

Method filter

```
package Rot13;

use Filter::Util::Call;

sub import {
    my ($class, @args) = @_;
    filter_add({});
}

sub filter {
    my $self = shift;
    my $status = filter_read();
    tr/a-zA-Z/n-za-mN-ZA-M/ if $status > 0;
    return $status;
}

1;
```

Closure filter

```
package SuperSubstitutor;

use Filter::Util::Call;

sub import {
      my ($class, $start, $stop, $from, $to) = @_;
      my $inside = 0;

      filter_add(sub {
          my $status = filter_read();
          if ($status > 0) {
              $inside = 1 if $inside == 0 and /$start/;
              if ($inside) {
                  s/$from/$to/;
                  filter_del() if /$stop/;
              }
          }
          return $status;
      });
}

1;
```

Inconvenients

- Filter::Util::Call és potent, però sovint hom necessitaria quelcom més simple:
- `no ModuleName;`
- `__END__`
- `__DATA__`
- Cadenes, regexps, POD, ...

Filter::Simple

- Filter::Simple, de Damian Conway, és un wrapper de Filter::Util::Call que els adreça

```
package Bang;

use Filter::Util::Call;

sub import {
    filter_add({});
}

sub filter {
    my $self = shift;
    my $status = filter_read();
    s/foo/bar/g if $status > 0;
    return $status;
}

1;
```

```
package Bang;

use Filter::Simple sub {
    s/foo/bar/g;
};

1;
```

No em toquis els strings!

- Filter::Simple permet especificar sobre què vols treballar, alguns exemples:
  - `code`: no quotelikes, POD, `__DATA__`
  - `executable`: no POD ni `__DATA__`
  - `string`: només cadenes
  - `regex`: només expressions regulars

Exemple

```
package RegexCI;
use Filter::Simple;
FILTER_ONLY regex => sub { $_ = "(?i:$_)" };
```

```
use RegexCI;
my $arg = shift;
print "a 0!\n" if $arg =~ /^3$/;
```

```
% perl -c -MFilter::ExtractSource foo.pl
foo.pl syntax OK
use RegexCI;
my $arg = shift;
print "a 0!\n" if $arg =~ /(?i:^3$)/;
```

Documentació

- No hi ha massa documentació
- La pàgina *perlfilter* introdueix el tema
  - Sembla que enlloc està documentat com escriure un filtre en C
  - Per a filtres en Perl només cal llegir la documentació de Filter::Util::Call o Filter::Simple

Depuració

- Hi ha diverses maneres de veure el que un filtre de codi escrit en Perl genera
  - Filter::tee, Filter::ExtractSource
  - A mà:

```
sub import {
    my ($class, %cfg) = @_;
    $DEBUG = $cfg{debug};
}

FILTER_ONLY code => sub {
    # modify $_
    if ($DEBUG) {
        s/$Filter::Simple::placeholder/BLANKED_OUT/g;
        print;
        $_ = '1;';
    }
};
```

Gotcha

- Els filtres de codi no funcionen en temps d'execució
  - `require`
  - `eval EXPR`

Quina poca feina!

Acme::Pythonic

```
sub exp_mod:
    my ($i, $j, $n) = @_
    my $r = 1
    while $j:
        if $j % 2:
            $r = $i*$r % $n
        $j >>= 1
        $i = $i**2 % $n
    return $r
```

Lingua::Romana::Perligata

```
use Lingua::Romana::Perligata;

adnota Illud Cribrum Eratothenis

maximum tum val inquementum tum biguttam tum stadium egresso scribe.
vestibulo perlegementum da meo maximo .
maximum tum novumversum egresso scribe.
da II tum maximum conscribementa meis listis.
dum damentum nexto listis decapitamentum fac sic
        lista sic hoc tum nextum recidementum cis vannementa da listis.
        next tum biguttam tum stadium tum nextum tum novumversum
                scribe egresso.
    cis
```

Gràcies!