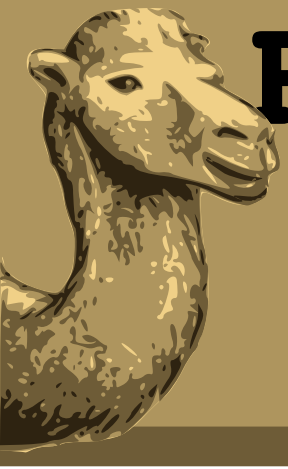
Perl Mongers im Ruhrgebiet



Ruhr.pm

Introspektion und dynamische Akzessoren

Autor: Veit Wahlich

EMail: veit AT ruhr.pm.org

Datum: 13. Dezember 2010

http://ruhr.pm.org/

Ruhr.pm

Introspektion / Reflexion

- Selbstkenntnis des Programms ueber seinen Aufbau oder den eigenen Programmcode
  - veschiedene Eingriffspunkte niederer Ebenen
    - Typeglobs, Source Filters, ...
  - Bereitstellung von Eingriffspunkten hoeherer Ebenen durch z.B. OO-Frameworks
    - i.d.R. beschraenkt auf das Framework
- Moeglichkeiten zum Eingriff in den eigenen Code oder das eigene Verhalten

Ruhr.pm

introspect.pl

```
use Digest::MD5;
my $base = 'Digest::MD5::';

my %stash = %{$base};
my @stash_keys = sort keys %stash;

printf "%s: Global deklarierte, definierte"
  . " Skalare: %s\n",  $pkg, join ', ',
  grep {defined ${$base . $_}} @stash_keys;

printf "%s: Global deklarierte, nicht leere"
  . " Arrays: %s\n", $pkg, join ', ',
  grep {@{$base . $_}} @stash_keys;

printf "%s: Global deklarierte, nicht leere"
  . " Hashs: %s\n",  $pkg, join ', ',
  grep {%{$base . $_}} @stash_keys;

printf "%s: Globale Subs: %s\n",
  $pkg , join ', ',
  grep {defined &{$base . $_}} @stash_keys;
```

Einfache Introspektion des Moduls **`Digest::MD5`** durch Zugriff auf den Stash **`%{'Digest::MD5::'}`** des Packages, ohne dabei tief in Typeglobs einsteigen zu muessen.

Auflistung aller global deklarierten und definierten bzw. gefuellten Skalare, Arrays und Hashs sowie Subs in globalem Kontext.

Sie auch erweitertes Beispiel introspect/introspect.pl

Ruhr.pm

Akzessoren

- Methoden fuer den Zugriff auf die Attribute eines Objekts
  - sog. „Setter“ und „Getter“
- erlauben einem Objekt auf einfache Weise, die eigenen Attribute zu ueberwachen
  - Pruefung von Werten, bevor sie in ein Attribut geschrieben werden
  - Ausfuehren einer Aktion, wenn ein Attribut gelesen wird
  - ...

Ruhr.pm

statische_akzessoren.pl

```
use Kfz;

my $k = Kfz->new
          ->setMarke('VEB Sachsenring')
          ->setModell('Trabant');

print "Mein Auto ist ein ", $k->getModell,
  " aus dem Hause " , $k->getMarke,
  " und es faehrt noch.\n" ;
```

Erzeugung eines Objekts der Beispiel-Datenklasse **`Kfz`** und Nutzung von Akzessoren, um die Attribute „Marke“ und „Modell“ zu manipulieren (Setter) sowie wieder auszulesen (Getter).

Ruhr.pm

Kfz.pm

```
package Kfz;

sub new{
  my ($class) = @_;
  my $self = {
        marke  => undef,
        modell => undef
    };
  bless $self, $class;
  return $self;
}

sub setMarke{
  my ($self, $marke) = @_;
  $self->{marke} = $marke;
  return $self;
}

sub getMarke{
  my ($self) = @_;
  return $self->{marke};
}

# ... und weitere Akzessoren
1;
```

Die Beispiel-Datenklasse **`Kfz`** mit Konstruktor, der die Attribute **`marke`** und **`modell`** festlegt.

Fuer den manipulierenden Zugriff auf das Attribut **`marke`** wird die einfache Setter-Methode **`setMarke`** angelegt, um das Attribut zu beziehen, die Getter-Methode **`getMarke`**.

Sie auch erweitertes Beispiel statische_akzessoren/Kfz.pm

Ruhr.pm

Dynamische Akzessoren

- Akzessoren, die bei Bedarf automatisch erzeugt werden
  - auf Basis der existierenden Attribute, z.B.
    - definiert ueber spezielle Datenstrukturen
    - durch Introspektion dem Code entnommen
    - ...
- ermoeglicht durch **`sub AUTOLOAD{}`** und **`$AUTOLOAD`** oder aufsetzende Module
  - Aufruf von AUTOLOAD(), wenn die gesuchte Funktion/Methode nicht im Code existiert

Ruhr.pm

Kfz.pm, dynamische Variante

```
package Kfz;

use Carp;

our $AUTOLOAD;

sub new{
  my ($class) = @_;
  my $self = {
        marke  => undef,
        modell => undef
    };
  bless $self, $class;
  return $self;
}

sub DESTROY{}

# ...
```

Die Beispiel-Datenklasse **Kfz** wird mit dynamischen Akzessoren ausgestattet. Der Konstruktor bleibt gleich, soll nun aber beliebig viele Attribute aufnehmen koennen, ohne dass neuer Code geschrieben werden muss.

Die Variable **$AUTOLOAD** wird spaeter den Namen der gesuchten Funktion enthalten und wird dazu als Package-Global deklariert.

DESTROY() wird angelegt, damit sie nicht durch **AUTOLOAD()** laeuft.

Ruhr.pm

Kfz.pm, dynamische Variante

```
# ...

sub AUTOLOAD{
  my ($self, $value) = @_;

  my ($do, $attr)
    = ($AUTOLOAD =~ /\:(set|get)(\w+)$/);

  defined $attr
    && exists $self->{$attr = lc $attr}
    || croak "Sub $AUTOLOAD does not exist";

  if ($do eq 'get'){
    return $self->{$attr};
  }
  else {
    $self->{$attr} = $value;
    return $self;
  }

}

1;
```

AUTOLOAD() wird fuer jede nicht existierende Funktion/Methode aufgerufen.

Der Name der gesuchten Sub steht (inkl. Paket-Name) in **$AUTOLOAD**, wir fischen uns die Art des Aufrufs („set“ oder „get“) sowie den Attribut-Namen heraus.

Wir pruefen, ob ein Attribut mit diesem Namen existiert und liefern entsprechend der Art des Aufrufs dessen Wert aus bzw. manipulieren ihn.

Ruhr.pm

Kombiniert in komplexem Beispiel

- Programmlogik fuer Introspektion und dynamische Akzessoren wird vollstaendig ererbt
- einfache Deklaration von Attributen als skalare Variablen in globalem Kontext einer Klasse
  - definieren „Typ“ und Default-Werte fuer Attribute
- Akzessoren fuer alle Attribute
  - Pruefung des Eingabewertes bei Manipulation ueber Setter mittels Parse-Methode je „Typ“
- leicht erweiterbar um Array- und Hash-Attribute

Ruhr.pm

adam_und_eva.pl

```
use strict;
use warnings;

use Person;

use Data::Dumper;

my $adam = new Person();
my $eva = new Person();

print Dumper($adam, $eva);

$adam->setVorname('Adam')
     ->setGeschlecht('m')
     ->setPartner($eva);

$eva->setVorname('Eva')
    ->setGeschlecht('f')
    ->setPartner($adam);

print Dumper($adam, $eva);
```

Beispiel fuer die Nutzung der einfachen Datenklasse **`Person`**, welche wiederum Introspektions-Funktionalitaet und dynamische Akzessoren von der Klasse **`Introspector`** erbt.

Ruhr.pm

Beispiel: Person.pm

```
package Person;
use base qw(Introspector);

our $any_Vorname = '';
our $any_Nachname = '';
our $posint_Alter = 0;
our $sex_Geschlecht = 'undef';
our $person_Partner = 'undef';

sub parse_posint($$){
  die "May contain only characters [0-9].\n" if $_[1] !~ /^\d+$/;
  return int $_[1];
}

sub parse_sex($$){
  die "Must be set to 'f', 'm' or 'i'.\n" if $_[1] !~ /^[fmi]$/;
  return $_[1];
}

sub parse_person($$){
  !defined $_[1] || ref($_[1]) eq 'Person'
    || die "Must be object of class Person or undef.\n";
  return $_[1];
}
1;
```

Ruhr.pm

Beispiel: Introspector.pm

```perl
package Introspector;

use strict;
use warnings;

use Carp;

use vars qw($AUTOLOAD);

# Konstruktor "new", akzeptiert hier keine Parameter.
sub new($){
  my ($class) = @_;

  # Wir speichern die Attribute direkt im Hash des Objekts, ihre Typen
  # jedoch im unter _types referenzierten Hash.
  my $self = {_types => {}};

  # Strikte Refernzen muessen fuer den Zugriff auf den Stash ueber $class
  # deaktiviert werden.
  no strict qw(refs);
  # Aus dem Stash von $class suchen wir alle skalar deklarierten und
  # definierten Variablen...
  # FIXME: Wie koennen wir hier *alle* skalar deklarierten Variablen finden,
  # statt nur der definierten?
  my @scalar_attrs = grep{defined ${"${class}::$_"}} keys %{"${class}::"};
```

Ruhr.pm

Beispiel: Introspector.pm

```
    # ... und iterieren ueber alle gefundenen.
    foreach my $attr (@scalar_attrs){

      # Wir beachten nur solche Variablennamen, die dem gewuenschten Muster
      # "{typ}_{name}" entsprechen.
      if (my($type, $name) = ($attr =~ /^([a-z][a-z0-9]*)_([a-zA-Z]\w*)$/)){

        # Attribute werden unabhaengig von ihrer Gross-/Kleinschreibung
        # betrachtet.
        $name = lc $name;
        # Der Wert der globalen Variable ist der Default-Wert und wird initial
        # unter dem Namen des Attributs hinterlegt.
        # Der Platzhalter "undef" dient dazu, Attribute initial nicht
        # definiert zu speichern.
        $self->{$name}
          = ${"${class}::$attr"} eq 'undef' ? undef : ${"${class}::$attr"};
        # Den zugehoerigen Typen speichern wir unter dem Namen des Attributs
        # im Hash unter _types.
        $self->{_types}->{$name} = $type;
      }
    }

    bless $self, $class;
    return $self;
} # Ende sub new
```

Ruhr.pm

Beispiel: Introspector.pm

```perl
sub AUTOLOAD{
  my ($self, @args) = @_;

  # Der Name der urspruenglich gesuchten Funktion steht in der globalen
  # Variable $AUTOLOAD.
  # Wir verwerfen den normalerweise vorhandenen Paketnamen.
  (my $method = $AUTOLOAD) =~ s/^.*::/;

  # Zurueck gehen, falls die Funktion DESTROY gesucht wurde.
  return if($method eq 'DESTROY');

  # Beziehe die Klasse des Objekts. Abbrechen, falls kein Klassenname
  # gesetzt ist (d.h. die Funktion nicht als Methode eines Objekts
  # aufgerufen).
  my $class = ref $self
    || croak "Function $method does not exist";

  # Spalte den Methodennamen auf in Typ (Setter-/Getter-Methode) und
  # Attributnamen auf.
  my ($action, $attr) = ($method =~ /^(set|get)_?([a-z0-9]\w*)$/i);
  # Breche ab, falls der Methodenname nicht dem Muster entspricht oder das
  # Attribut nicht existiert.
  defined $attr && exists $self->{$attr = lc $attr}
    || croak "Method $method does not exist in $class";
```

Ruhr.pm

Beispiel: Introspector.pm

```
# Falls wir ein Setter-Aufruf sind.
if ($action eq 'set'){
  my $retval;
  no strict qw(refs);

  # Uebergebe die Funktionsparameter an die Parse-Funktion (parse_{typ}())
  # des Attribut-Typs. Fange Exceptions ab fuer den Fall, dass die Parse-
  # Funktion nicht existiert oder sie die Verarbeitung abbricht
  # (z.B. ungueltige Eingabedaten).
  eval {
    my $parser = 'parse_' . $self->{_types}->{$attr};
    $retval = $self->$parser(@args);
    1;
   } || do {
    # Abbrechen, falls das Attribut nicht gesetzt werden konnte
    # -- alternativ koennte hier auch undef zurueck gegeben werden.
     croak "Error setting attribute '$attr': $@";
  };

  # Da die Parse-Funktion erfolgreich war, setze ihren Rueckgabewert als
  # neuen Wert des Attributs.
  $self->{$attr} = $retval;

  # Gebe das Objekt zurueck.
  return $self;
} # Ende if
```

Ruhr.pm

Beispiel: Introspector.pm

```
  # Sonst ist sie ein Getter ...
  else {
    # ... und gibt einfach den aktuellen Wert des Attributs zurueck.
    return $self->{$attr};
  }
} # Ende sub AUTOLOAD

# Parse-Methode fuer int-Attribute; der Wert wird explizit in int gecastet.
sub parse_int($$){
  return int $_[1];
}

# Parse-Methode fuer word-Attribute; abbrechen, falls der Wert Zeichen
# enthaelt, die nicht aus der Klasse [0-9a-zA-Z] sind.
sub parse_word($$){
  die "Only 'word' characters [0-9a-zA-Z_] allowed.\n"
    unless (defined $_[1] && $_[1] !~ /\W/s);
  return $_[1];
}

# Parse-Methode fuer any-Attribute; akzeptiere unveraendert alle Werte.
sub parse_any($$){
  return $_[1];
}

1; # Ende Introspector
```

Ruhr.pm

Vielen Dank
fuer Eure Aufmerksamkeit