Renée Bäcker

Was ist neu in Perl 5.20?

Wie in den vergangenen Jahren gibt es auch in diesem Frühjahr eine neue Version von Perl 5. Mittlerweile gibt es schon die Version 20 und auch diese bringt einige Neuerungen, die in diesem Artikel beschrieben werden sollen. Auch wenn es das Perl-Magazin in Zukunft nicht mehr geben wird, wird es diese Art von Artikel jährlich geben: zu finden unter http://perl-academy.de

Im Gegensatz zu den letzten Versionen gibt es in Perl 5.20 eine große entscheidende Veränderung: Es gibt Subroutinen-Signaturen. In älteren Ausgaben von $foo wurden schon Module vorgestellt die Methoden-Signaturen einführen: Artikel "Methoden-Signaturen" in Ausgabe 12 und als Teil des Moose-Tutorials in Ausgabe 16.

Um die weiteren nützlichen Änderungen in Perl 5.20 dennoch ausreichend würdigen zu können, gibt es die Signaturen als eigenständigen Artikel direkt im Anschluss an diesen hier.

Widmen wir uns aber den Neuerungen in der neuen Perl-Version.

Postfix-Dereferenzierung

Das Dereferenzieren von Datenstrukturen ist nicht gerade eine Wohltat für die Augen. Manchmal erkennt man vor lauter Sigils und geschweiften Klammern gar nicht mehr, worum es eigentlich geht. Abhilfe - auch wenn das ebenfalls gewöhnungsbedürftig ist - verspricht die Postfix-Dereferenzierung.

Auch hier gehen die Perl 5 Porters den Weg, viele neue Features erst einmal als *experimentell* zu kennzeichnen, und das gilt auch für dieses Feature. Und es muss über das *feature* Pragma aktiviert werden:

```
use feature 'postderef';
no warnings 'experimental::postderef';
```

Danach kann zusätzlich zu dem altbewährten Dereferenzieren auch die Postfix-Dereferenzierung verwendet werden:

```
my @colors    = qw(blue yellow red);
my $color_ref = \@colors;

# bisheriges Derefenzieren
my @copy = @{$color_ref};

# Postfix-Dereferenzierung
my @copy = $color_ref->@*;
```

Statt dem vorangestellten Sigil und den -- in manchen Situationen überflüssigen - geschweiften Klammern wird der bekannte Pfeil -> verwendet und das passende Sigil gefolgt vom Asterisk. Dieses Schema wird auch bei den weiteren Dereferenzierungen eingesetzt:

```
$sref->$*;  # same as ${ $sref }
$aref->@*;  # same as @{ $aref }
$href->%*;  # same as %{ $href }
$cref->&*;  # same as &{ $cref }
$gref->**;  # same as *{ $gref }
```

Das funktioniert natürlich auch für komplexere Datenstrukturen:

```
my $data = {
    servers => {
        server1 => {
            ip => [
                '127.0.0.1',
                '192.168.2.161',
            ],
        },
    },
};

my @ips = $data->{servers}->{server1}
              ->{ip}->@*;
```

Damit erkennt man schneller als bei der altbewährten Methode, dass am Ende ein Array herauskommt. Der Vorteil bei der neuen Methode ist auch, dass der komplette Ausdruck von links nach rechts gelesen werden kann, ohne dass man das Sigil am Anfang im Hinterkopf behalten muss.

Möchte man auf einen Slot innerhalb einer *Globreferenz* zugreifen, wird der abschließende Asterisk durch den Zugriff auf den Slot ersetzt:

```
# same as *{ $gref }{ $slot }
$gref->*{ $slot };
```

Die Postfix-Dereferenzierung funktioniert auch bei Slices:

```
$aref->@[ ... ];  # same as @$aref[ ... ]
$href->@{ ... };  # same as @$href{ ... }
$aref->%[ ... ];  # same as %$aref[ ... ]
$href->%{ ... };  # same as %$href{ ... }
```

Hier ist deutlich schneller erkennbar, dass ein Slice einer Referenz erwünscht ist und nicht die Dereferenzierung eines Array- bzw. Hashelements.

Einige dieser Ausdrücke können auch interpoliert werden. Dazu muss aber das verwandte Feature *postderef_qq* aktiviert werden:

```
use feature qw/postderef postderef_qq/;

my $name_ref = \'$foo';
print "Sie lesen $name_ref->$*\n";
```

Folgende Ausdrücke können interpoliert werden:

```
$sref->$*
$aref->@*
$aref->@[ ... ]
$href->@{ ... }
```

Um an den letzten Index in einer Arrayreferenz zu kommen, braucht man nicht mehr

```
my $max_index = $#$aref;
```

sondern

```
my $max_index = $aref->$#*;
```

Performanz-Verbesserungen

Ein paar Perl 5 Porters haben sich auch der Verbesserung der Performanz verschrieben. In den nachfolgenden Absätzen ist zu sehen, dass es viele kleine Verbesserungen gibt, die für sich genommen nicht immer die großen Schritte machen, aber in der Summe kommt doch einiges zusammen.

Hash-Lookups bei Slices

Es ist schon länger so, dass zur Compile-Zeit der interne Hashwert für Hash-Lookups mit konstanten Schlüsseln (z.B. `$hash{schluessel}`) vorberechnet wurde. Damit wird der Hash-Lookup stark beschleunigt. Diese Optimierung wurde jetzt auch für Hash-Slices (z.B. `$hash{qw/schluessel1 schluessel2/}`) umgesetzt.

and- und or-Oparatoren im void-Kontext

Kombinierte `and`- und `or`-Operatoren im *void*-Kontext, wie z.B. bei `unless( $a && $b )` oder `if( $a || $b )`, kürzen jetzt ab, so dass bei `unless( $a && $b )` das `$b` erst gar nicht überprüft wenn `$a` schon unwahr ist.

Patternmatching mit Nicht-Unicode

Eine Performanzregression wurde in Perl 5.11.2 für Nicht-Unicde Patternmatching, bei denen die Groß-/Kleinschreibung keine Rolle spielt, eingebaut. Dabei wurde eine Optimierung für Zeichen aus dem ASCII-Bereich deaktiviert. Diese Optimierung ist in Perl 5.20 wieder aktiviert.

Copy-On-Write

Perl hat einen neuen Copy-On-Write-Mechanismus, der das Kopieren des internen Stringpuffers vermeidet wenn ein Skalar einem anderen zugewiesen wird. Das macht das Kopieren von großen Strings viel schneller. Wenn einer der beiden Skalare nach der Zuweisung verändert wird, wird das wirkliche Kopieren angestoßen. Durch diesen neuen Mechanismus ist es unnötig Strings aus Effizienzgründen als Referenz an z.B. Subroutinen zu übergeben.

Dieses Feature gab es schon in Perl 5.18.0, war aber standardmäßig deaktiviert. Das wurde jetzt geändert, so dass man Perl nicht mehr mit der *Configure*-Option

```
-Accflags=PERL_NEW_COPY_ON_WRITE
```

kompilieren muss.

Soll das Copy-On-Write beim Kompilieren deaktiviert werden, muss man

```
-Accflags=PERL_NO_COW
```

verwenden.

Hier ein kleines Beispiel, an dem man das Copy-On-Write (*COW*) erkennen kann:

```
$ perl -MDevel::Peek -e
    '$a="abc"; $b = $a; Dump $a; Dump $b'
SV = PV(0x260cd80) at 0x2620ad8
  REFCNT = 1
  FLAGS = (POK,IsCOW,pPOK)
  PV = 0x2619bc0 "abc"\0
  CUR = 3
  LEN = 16
  COW_REFCNT = 2
SV = PV(0x260ce30) at 0x2620b20
  REFCNT = 1
  FLAGS = (POK,IsCOW,pPOK)
  PV = 0x2619bc0 "abc"\0
  CUR = 3
  LEN = 16
  COW_REFCNT = 2
```

Man sieht, dass beide Skalare den gleichen *PV*-Puffer verwenden und der COW-Referenzzähler auf 1 steht. Bei den *FLAGS* ist auch vermerkt, dass für den Skalar "Copy-On-Write" aktiv ist. Ändert man z.B. `$b` durch ein

```
$b =~ s/a/u/g;
```

dann sehen die Dumps so aus:

```
SV = PV(0x260cd80) at 0x2620ad8
  REFCNT = 1
  FLAGS = (POK,IsCOW,pPOK)
  PV = 0x2619bc0 "abc"\0
  CUR = 3
  LEN = 16
  COW_REFCNT = 2
SV = PV(0x940cf90) at 0x941da60
  REFCNT = 1
  FLAGS = (POK,pPOK)
  PV = 0x9418ea0 "ubc"\0
  CUR = 3
  LEN = 16
```

Bei dem geänderten Skalar ist das Flag *IsCOW* verschwunden und der String wurde in einen anderen Puffer kopiert.

Allgemeine Verbesserungen und Änderungen

Es gibt etliche allgemeine Verbesserungen und Änderungen, die in den folgenden Abschnitten gezeigt werden.

readdir() und die Fehlervariable $!

Die Funktion `readdir()` setzt die Spezialvariable `$!` nur im Fehlerfall. Diese Variable wird beim abschließenden `undef` nicht mehr auf `EBADF` gesetzt, wenn der Systemaufruf nicht diese Variable setzt.

Zeichen matchen in Regulären Ausdrücken

Eine Regression seit Perl 5.18.0 Reguläre Ausdrücke betreffend wurde gefixt. In dem Fehler wurden Zeichen aus dem Bereich `\x80 - \xFF` nicht gematcht wenn in einer Zeichenklasse neben `[:^ascii:]` noch andere Zeichen waren.

Neues Attribut für Subroutinen

Subroutinen können jetzt mit dem `prototype`-Attribut versehen werden. Wenn die Subroutine definiert oder deklariert wird, kann der Prototyp innerhalb eines `prototype`-Attributs anstelle der runden Klammern nach dem Namen geschrieben werden. Aus

```
sub foo ($$) {
}
```

wird dann

```
sub :prototype($$) {
}
```

Warum das neue Attribut notwendig wurde, wird im Artikel über die Subroutinen-Signaturen erläutert.

Zufallszahlengenerator für rand

Bisher verwendete Perl einen plattformsspezifischen Zufallszahlengenerator, das war entweder `rand()`, `random()` oder `drand48()`. Das bedeutete, dass die Qualität der Zufallszahl in Perl von Plattform zu Plattform variiert. Das geht von den 15 Bits von `rand()` auf Windows bis zu 48 Bit von `drand48()` auf POSIX-Plattformen wie Linux.

Mit Perl 5.20 verwendet Perl auf allen Plattformen seine eigene Implementierung von `drand48()`, was Perls `rand()`-Funktion kryptographisch aber nicht sicher macht.

Bessere 64-bit Unterstützung

Auf 64-Bit-Plattformen benutzen die internen Array-Funktionen jetzt 64-Bit-Offsets, wodurch es möglich ist, dass Perl Arrays mehr als 2**31 Elemente haben können -- solange genug Speicher zur Verfügung steht.

Die Engine für die Regulären Ausdrücke unterstützt jetzt Strings die länger sind als 2**31 Zeichen.

Einige `PerlIO_*`-Funktionen haben jetzt Parameter und Returnwerte von der Größe `SSize_t` anstatt *int*.

Neue slice Syntax

Für Hashes und Arrays gibt es eine neue Form von Slices. Mit `%hash{...}` bzw. `%array[...]` liefert eine Liste von Schlüssel-Wert-Paaren bzw. bei Arrays Index-Wert-Paaren.

```
use Data::Dumper;

my @array = qw(perl magazin foo);
my %map   = %array[0,2];

print Dumper \%map;
__END__
$VAR1 = {
          '2' => 'foo',
          '0' => 'perl',
        };


use Data::Dumper;

my %hash = qw(
    sprache  => 'perl',
    magazin  => 'foo',
    webseite => 'cpan.org',
    modul    => 'Moose',
);
my %map  = %hash{qw/sprache magazin/};

print Dumper \%map;
__END__
$VAR1 = {
          'magazin' => 'foo',
          'sprache' => 'perl',
        };
```

Neue Warnung bei möglichen Vorrangsproblemen

Beim Programmieren fällt es nicht unbedingt sofort auf wenn man in die Fallen der Vorrangsregeln von Operatoren tappt. Für solche Fälle ist es praktisch und wichtig, wenn der Compiler eine Warnung ausgibt. Eine solche Warnung wurde jetzt eingeführt für Fälle in denen ein Operator wie `return` und einem Operator mit niedriger Vorrangsstufe wie `or` gemischt werden.

Schreibt man

```
sub foo {
    return $a or $b;
}
```

bekommt man die Warnung

```
Possible precedence issue with \
    control flow operator at ....pl
```

angezeigt. Perl parst den `return` Befehl als

```
sub foo {
    (return $a) or $b;
}
```

und dadurch wird das `or $b` nie ausgeführt. Die Zeile ist also effektiv ein reines `return $a;`. Durch die Warnung wird man darauf aufmerksam gemacht, dass man entweder runde Klammern benutzen sollte (`return ($a or $b)`) oder Operatoren mit hoher Priorität (`return $a || $b`).

Namen für Dateihandles

Ein lexikalischer Dateihandle (wie er in `open my $fh ...` vorkommt) hat üblicherweise einen Namen basierend auf dem aktuellen Package und dem Namen der Variablen, also z.B. `main::$fh`. Im Falle von Rekursion hat der Dateihandle den *$fh*-Teil verloren. Dieser Fehler wurde beseitigt.

Auslesen von freigegebenem Speicher

In den seltenen Fällen dass ein Perl-Programm mit einem *HEREDOC*-Teil aufhörte und die abschließende Zeile keinen Zeilenumbruch enthielt konnte es passieren, dass der freigegebene Speicher während des Parsens ausgelesen wurde. Dieses Verhalten wurde abgestellt.

Kommentar oder kein Kommentar

Seit Perl 5.001 wurde in Regulären Ausdrücken wie `/[#$a]/x` oder `/[#]$a/x` das `#`-Zeichen als Kommentar betrachtet. Dabei sollte in Zeichenklassen das Zeichen seine besondere Bedeutung verlieren. Durch diesen Fehler wurde die Variable nie interpoliert. Dieser Fehler ist jetzt endlich behoben.

Nützliche Infos und Dokumentation

Das Debugging mit `gdb` ist für unerfahrene Programmierer nicht so einfach. In der *perlhacktips*-Dokumentation wurden einige zusätzliche Beispiele für die Verwendung von `gdb` hinzugefügt.

Kernmodule

Die Perl5-Porters tun sich nicht leicht damit, langjährige Mitglieder der Kernmodule wieder aus dem Kern zu entfernen und neue Mitglieder aufzunehmen. Manchmal ist es aber dennoch notwendig und auch mit Perl 5.20 gibt es da etwas Bewegung. Einige Module wurden entfernt, zu den Wichtigsten gehören `CPANPLUS`, `Log::Message` und `Archive::Extract`.

Zu den Modulen, die in (nicht ganz so naher Zukunft) aus dem Kern entfernt werden, gehört jetzt auch `CGI.pm`. Wer Webanwendungen schreibt und dabei das Modul verwendet, sollte es auf jeden Fall in die Liste der Abhängigkeiten eintragen. Bis so eine Ankündigung dass ein Modul entfernt wird in die Tat umgesetzt wird, vergehen üblicherweise viele Jahre, also kein Grund zur Hektik.

Neu hinzugekommen ist neben ein paar "kleineren" Modulen auch `IO::Socket::IP`.

Mit `IO::Socket::IP` bekommt der Perl-Kern ein Modul mit dem unabhängig von Protokollen mit IPv4 und IPv6 gearbeitet werden kann. Das Modul ist als Ersatz für das weitverbreitete `IO::Socket::INET` gedacht. Ein paar Inkompatibilitäten gibt es, die in der Dokumentation von `IO::Socket::IP` aufgelistet sind. In den meisten Fällen sollte ein `s/IO::Socket::INET/IO::Socket::IP/g` reichen um seine Anwendungen IPv6-fähig zu machen.

Die ganzen Updates werden hier nicht im Detail dargestellt, da alleine die Auflistung der aktualisierten Module mehrere Seiten in Anspruch nehmen würde. Eine Aktualisierung aber doch ins Auge gesprungen: Die Ausgabe von `Data::Dumper` hat sich - je nach Daten und Einstellungen - geändert. Wer also auf die "Exaktheit" der Ausgabe vertraut, sollte sich den Code nochmal anschauen.

Mit der Version 1.35 von `List::Util` kommen einige nützliche Funktionen in den Kern: `any`, `all`, `none`, `notall` und `product` wurden neu hinzugefügt und die Funktionen `reduce` und `first` werden implementiert auch wenn *MULTICALL* nicht gesetzt ist.

Die hier gezeigten Änderungen sind nur ein Teil dessen was sich in der neuen Perl-Version alles getan hat. Es zeigt auch, dass die Perl-Entwicklung nicht stehen bleibt. Gerade für größere Features wäre es gut, wenn möglichst viele Programmiererinnen sich auch mal Entwicklerversionen von Perl installieren und die neuen Features ausprobieren. Dank *perlbrew* bzw. *plenv* ist das kein Problem.

Mehr Informationen zu den ganzen Neuerungen finden sich in der *perldelta*-Dokumentation.