autodie - Fehlerbehandlung leicht gemacht

Fehlerbehandlung, Fehlerbehandlung, Fehlerbehandlung. Immer wieder stößt man auf ein Problem, bei dem man die Fehlerbehandlung vergessen hat. Und dann kommt nur Unsinn raus. Paul Fenwick hat in der Beschreibung seines Moduls `autodie` die richtige Beschreibung:

```
     bIlujDI' yIchegh()Qo'; yIHegh()!
It is better to die() than to return()
          in failure.
              -- Klingon programming proverb.
```

Genau um dieses Modul geht es auch in diesem Artikel.

Üblicherweise macht man bei Funktionen wie `open` eine Fehlerbehandlung mit `or die ...`

```
open my $fh, '<', $filename or die $!;
```

Auf die Dauer ist es aber ganz schön mühsam, immer den `or die`-Teil zu schreiben.

`autodie` ermöglicht es, built-in Funktionen zu verwenden, ohne dass man immer ein `or die "Fehlermeldung"` tippen muss. Man muss `autodie` nur sagen, welche Funktionen automatisch eine Fehlerbehandlung bekommen sollen. Ohne import-Liste werden alle ":default" Funktionen überwacht. Paul Fenwick hat alle Funktionen in Kategorien eingeteilt, die hierarchisch aufgebaut ist. Als oberstes Element gibt es `:all`, darunter die Kategorien `:default` und `:system`. Diese teilen sich weiter auf.

Fatal.pm

Zum gleichen Zweck wurde ursprünglich das Modul `Fatal` geschrieben, das auch mit dem Perl-Core mitgeliefert wird. So kann man mit `Fatal` alle `open`-Aufrufe überwachen

```
use Fatal qw(open);

open my $fh, '<', 'keine.datei';
```

Auch wenn jetzt keine `or die`-Anweisung hinter dem `open` zu finden ist, stirbt das Skript mit einer Fehlermeldung.

```
C:\>fatal.pl
Can't open(GLOB(0x225f90), <, keine.datei):
   No such file or directory at (eval 1)
     line 3
        main::__ANON__('GLOB(0x225f90)', \
             '<', 'keine.datei')
          called at C:\fatal.pl line 5
```

Es besteht auch die Möglichkeit, eigene Subroutinen mit dem `do or die`-Ansatz zu versehen.

```
use Fatal;

sub mysub {
    die 'test' if $_[0] == 3; $_[0]
}

import Fatal 'mysub';

mysub( @ARGV );
```

`Fatal` hat aber auch einige Schwächen:

Das Modul lässt sich nicht "ausschalten". Es geht also nicht, dass ein Fehlschlagen von `open` nur in einem bestimmten Bereich zu einem Skriptabbruch führt.

Eine weitere Schwäche ist, dass es keine Fehlerobjekte gibt. Fängt man die Fehler mit `eval` ab, so landet in `$@` wirklich nur der Fehlerstring und kein Objekt (siehe Artikel Try-Catch). So wird die Auswertung relativ umständlich, da man dann mit Regulären Ausdrücken und Ähnlichem arbeiten muss.

Zusätzlich muss man auf den Rückgabewert von Funktionen achten. In dem Beispiel mit der eigenen Subroutine funkti-

oniert das Ganze nicht, wenn man nichts oder eine 0 an das Skript übergibt. Da kein "wahrer" Wert ("wahr" im Verständnis von Perl) zurückgegeben wird, geht Fatal davon aus, dass ein Fehler vorliegt.

```
C:\>fatal.pl 0
Can't mysub(0), $! is "" at (eval 1) line 3
        main::__ANON__(0) called at
           C:\fatal.pl line 7
```

Möchte man viele Funktionen überwachen, wird die Einbindung von `Fatal` mühsam, da jede einzelne Funktion in der Importliste angegeben werden muss:

```
use Fatal qw/chdir open close print .../;
```

autodie

Paul Fenwick hat mit `autodie` ein Pragma geschrieben, das dem gleichen Zweck wie `Fatal` dient und als direkter Ersatz verwendet werden kann. Fenwick hat auch gleich darauf geachtet, die Schwächen von `Fatal` zu beseitigen.

Wie bei Pragmas üblich hat `autodie` einen lexikalischen Gültigkeitsbereich. Damit kann man das Pragma in einem Block einschalten und im Nächsten ist es nicht mehr gültig. Dieser Unterschied wird deutlich, wenn man die folgenden zwei Skripte ausführt:

```
#!/usr/bin/perl

{
  use Fatal qw(open);
  open my $fh, '<', 'existierende.datei';
}

open my $fh, '<', 'keine.datei';
```

und

```
#!/usr/bin/perl

{
  use autodie qw(open);
  open my $fh, '<', 'existierende.datei';
}

open my $fh, '<', 'keine.datei';
```

Mit `Fatal` stirbt das Skript, während das Skript mit `autodie` durchläuft. Durch die lexikalische Gültigkeit hat man vielmehr die Kontrolle, in welchen Code-Teilen das Skript abbrechen soll wenn ein Fehler auftritt.

Damit die Importliste bei `autodie` nicht so lang wird, hat Paul Fenwick die Funktionen in Kategorien eingeteilt. Ein kleiner Ausschnitt der Einteilung sieht so aus:

```
:all
      :default
          :io
              read
              seek
              sysread
              sysseek
              syswrite
```

Damit wird schon deutlich, was in der Importliste für `autodie` stehen kann. Die komplette Einteilung ist in der Dokumentation des Moduls zu finden. Wenn man `use autodie qw(:all)` angibt, wird für alle relevanten built-in-Funktionen die Überwachung gestartet. Allerdings geht es noch kürzer mit `use autodie`. Sollen nur die IO-Funktionen überwacht werden, muss man `use autodie qw(:io)` schreiben. Natürlich kann man auch einfach nur eine einzelne Funktion überwachen: `use autodie qw(open)`.

Tritt ein Fehler auf, liefert `autodie` nicht einfach nur einen String zurück, sondern echte Objekte. Das hat den Vorteil, dass man nicht mit regulären Ausdrücken überprüfen muss, welche Art von Fehler geworfen wurde. Mit den Objekten kann man einfach Methoden verwenden.

```
#!/usr/bin/perl

use strict;
use warnings;
use autodie;

eval {
    open my $fh, '<', 'keine.datei';
    1;
} or do {

    if( $@->matches( 'open' ) ){
        print "open schlägt fehl: $@\n";
    }
    else{
        print "unbekannter fehler: $@\n";
    }

};
```

Wie man sieht, steht in der Spezialvariablen `$@` (siehe auch `perldoc perlvar`) nicht nur ein einfacher String, sondern man hat ein echtes Objekt, das überladen ist. Im Stringkontext wird die Fehlermeldung ausgegeben, sonst kann man damit umgehen wie mit anderen Objekten auch.

Die `matches`-Methode verlangt einen String, der dem Funktions- oder Kategorienname entspricht. So kann man dann überprüfen, ob allgemein ein IO-Fehler passiert ist: `$@->matches( ':io' )`.

Schwäche von autodie

Die Integration in Perl 5.8.x ist noch nicht ganz vollständig, da bis Perl 5.10 keine geeignete API für eigene Pragmas existierte. Fenwick arbeitet aber daran, dass auch Programmierer, die noch nicht auf Perl 5.10 umsteigen konnten, das Modul verwenden können.

Probleme gibt es mit ein paar built-in-Funktionen mit speziellen Prototypen. Da `autodie` diese Funktionen umschreiben muss, ist es schwierig, das gleiche Verhalten der Funktion zu erlangen.

So gibt es Probleme mit `system`/`exec` in einer speziellen Form: `system { $cmd } @args`. Das wird als Syntaxfehler erkannt. Wer diese Form trotzdem verwenden will, sollte `CORE::system` bzw. `CORE::exec` verwenden oder vor dem Aufruf `autodie` mit `no autodie qw(system exec)` deaktivieren.

Renée Bäcker