Thomas Fahle

Boilerplatecode mit Import::Into reduzieren

Wenn man mal so darüber nachdenkt, wie oft man als Perlprogrammierer (m/w) etwas wie

```
package Foo;
use strict;
use warnings;
use Package::A qw/x y z/;
use Package::B qw/u v w/;
```

in den Kopfteil eines Moduls per copy and paste eingefügt hat, dann fragt man sich, ob sich dieser Vorgang nicht deutlich vereinfachen lässt.

Leider ist das nicht ganz so einfach.

Import Basics

Wenn ein Modul mit `use` eingebunden wird, dann wird zunächst das Modul mit `require` geladen und anschließend die Methode `import` des Moduls aufgerufen.

```
use Foo LIST;
```

ist also äquivalent zu

```
BEGIN {
   require Foo.pm;
   Foo->import( LIST );
};
```

Die Methode `import` ist für das Ein- oder Ausschalten verschiedener pragmas bzw. Compileroptionen und das Kopieren von Modulen, Funktionen und Variablen in den Namensraum (namespace) des Aufrufers zuständig.

Keine Import Standards

Es gibt keine Standards für die Implementierung der Methode `import` - man kann z.B. `Exporter`, `Sub::Exporter`, `Exporter::Declare` oder `Moose::Exporter` verwenden.

Und dann gibt es ja auch noch pragmas.

Das Schreiben eines eigenen Importers kann daher schwierig bis schmerzhaft sein.

Import::Into als einheitliche Schnittstelle

Import::Into von Matt S. Trout, Graham Knop und Christian Walde bietet eine einheitliche und einfach zu bedienende Schnittstelle für den Import von Packages und pragmas in andere Packages.

Der Vorgang ist nicht ganz trivial, die Autoren erklären die technischen Details aber ausführlich in der Dokumentation des Moduls.

Import::Into Beispiele

Nachfolgend zwei einfache Beispiele, die den Einstieg in **Import::Into** erleichtern sollen.

Weitere Funktionen verrät ein Blick in die Dokumentation des Moduls.

Beispiel Pragmata importieren

In diesem Beispiel soll die Pragmata `strict` und `warnings` aktiviert werden. Weiterhin soll der Pfad zu lokal installierten CPAN-Modulen via `lib` erweitert werden.

Die Funktionalität wird in einem eigenen Modul zwecks Wiederverwendung gekapselt.

Zunächst werden die gewünschten Pragmata geladen und ggf. initialisiert.

Anschließend importiert eine **eigene** `import` Funktion diese mittels `import::into` in den Namensraum des Aufrufers.

```
package MySetup;
use strict;
use warnings;

use lib '/opt/extlib/perl/lib/perl5';

use Import::Into;

sub import {
   my $target = caller;

   strict   ->import::into( $target );
   warnings ->import::into( $target );
   lib      ->import::into( $target );
}

# Return true - it's a package
1;
```

Dieses Modul wird einfach in ein Programm oder Modul mittels `use` eingebunden.

```
#!/usr/bin/perl

use MySetup;
```

oder

```
package MyApp;

use MySetup;
```

Die Pragmata `strict`, `warnings` und `lib` sind nun aktiviert, wie das folgende simple Beispiel zeigt.

```
#!/usr/bin/perl

use MySetup;

print "Perl version $]\n\n";

foreach my $inc (@INC) {
       print "$inc\n";
}

# Yields a warning
my $x = "2:" + 3;
```

Ausgabe des Beispielprogramms :

```
Perl version 5.018002

/opt/extlib/perl/lib/perl5/
   x86_64-linux-gnu-thread-multi
/opt/extlib/perl/lib/perl5
/etc/perl
/usr/local/lib/perl/5.18.2
/usr/local/share/perl/5.18.2
/usr/lib/perl5
/usr/share/perl5
/usr/lib/perl/5.18
/usr/share/perl/5.18
/usr/local/lib/site_perl
.

Argument "2:" isn't numeric in addition
 (+) at ...
```

Beispiel Module importieren

Um Module und deren Funktionen mittels **Import::Into** zu importieren, werden die Module zunächst geladen.

Module, die Funktionen exportieren, werden mit einer **leeren** Importliste geladen. Die leere Importliste soll verhindern, dass Funktionen in das `Setup` Modul importiert werden.

Anschließend werden die gewünschten Funktionen explizit in den Namensraum des Aufrufers importiert.

Objektorientierte Module, die ja keine Funktionen exportieren, werden einfach komplett importiert.

```
package MySetup;
use strict;
use warnings;

use lib '/opt/extlib/perl/lib/perl5';

use Import::Into;
use List::MoreUtils ();
use Encode ();
use Moose;

sub import {
   my $target = caller;

   strict     ->import::into( $target );
   warnings   ->import::into( $target );
   lib        ->import::into( $target );

   List::MoreUtils->import::into( $target,
           'any', 'all', 'apply', 'uniq' );

   Encode         ->import::into( $target,
           'encode', 'decode' );

   Moose          ->import::into( $target );

}

# Return true - it's a package
1;
```

Jetzt sind, wie bereits oben gezeigt, die Pragmata `strict`, `warnings` und `lib` aktiviert.

Weiterhin wurden das Modul `Moose` und die Funktionen `any`, `all`, `apply`, `uniq`, `encode` und `decode` in den Namensraum des Aufrufers importiert.

Erheblich weniger Boilerplatecode

Der am Anfang des Artikels gezeigte Code lässt sich nun zur Wiederverwendung kapseln

```
package MySetup;
use strict;
use warnings;

use Import::Into;

use Package::A qw//;
use Package::B qw//;

sub import {
   my $target = caller;

   strict     ->import::into( $target );
   warnings   ->import::into( $target );

   Package::A ->import::into( $target,
                'x', 'y', 'z') ;
   Package::B ->import::into( $target,
                'u', 'v', 'w') ;
}

# Return true - it's a package
1;
```

und deutlich kürzer anwenden

```
package Foo;
use MySetup;
```

Bei verbesserter Wartbarkeit des Codes sind statt fünf Zeilen nur noch zwei Zeilen Code erforderlich.