

ClassLoader

Frank Seitz

Lade Klassen automatisch, wenn sie benötigt werden

```
#!/usr/local/bin/perl

use strict;
use warnings;

use Sbit;
use Sbit::I18n;
use Sbit::Function;
use Sbit::Web::Cgi;
use Sbit::Web::Browser;
use Sbit::Web::Session;
use Sbit::Web::Html::Form;
use Sbit::Web::Html::Popup;
use Sbit::Web::Html::Progress;
use Sbit::Web::Html::Subpage;
use Sbit::Web::Html::Table;
use Sbit::Web::Html::Widget;
use Sbit::Dbms::Database;
use Sbit::Dbms::Table;
use Sbit::Image::Gd;
use Sbit::Image::Magick;
use Sbit::Plot::Diagram;
use Sbit::Plot::Diagram::Colorbar;
use Sbit::Io::Import;
use Sbit::Io::Export;
use Sbit::Exception;
# ... noch mehr Module

# das Programm
...
```

Automatisches Laden

```
#!/usr/local/bin/perl

use strict;
use warnings;

use ClassLoader;

# das Programm
...
```

- keine use-Aufrufe mehr schreiben
- Klassen werden erst geladen, wenn sie benötigt werden

- keine use-Aufrufe mehr schreiben
- Klassen werden erst geladen, wenn sie benötigt werden
- die Startzeit des Programms verkürzt sich

- keine use-Aufrufe mehr schreiben
- Klassen werden erst geladen, wenn sie benötigt werden
- die Startzeit des Programms verkürzt sich
- ggf. weniger Speicherbedarf

- keine use-Aufrufe mehr schreiben
- Klassen werden erst geladen, wenn sie benötigt werden
- die Startzeit des Programms verkürzt sich
- ggf. weniger Speicherbedarf

Wann wird eine Klasse "benötigt"?

Antwort: Beim ersten Methodenaufruf.

Wann wird eine Klasse "benötigt"?

Antwort: Beim ersten Methodenaufruf.

Aufruf einer Klassenmethode

```
my $obj = My::Class->new; # loads My/Class.pm
```

Aufruf einer Objektmethode

```
1 my $obj = bless {a=>1, b=>2}, 'My::Class';  
2 my $val = $obj->get('a'); # loads My/Class.pm
```

Voraussetzungen für Autoladbarkeit

Klasse ist nach Perl-Konventionen implementiert, d.h.

- ① jede Klasse in einem eigenen Modul (.pm-Datei)
- ② Modulpfad entspricht Klassenname

Voraussetzungen für Autoladbarkeit

Klasse ist nach Perl-Konventionen implementiert, d.h.

- ① jede Klasse in einem eigenen Modul (.pm-Datei)
- ② Modulpfad entspricht Klassenname
- ③ die Klasse lädt ihre Basisklassen selbständig

Klasse ist nach Perl-Konventionen implementiert, d.h.

- ① jede Klasse in einem eigenen Modul (.pm-Datei)
- ② Modulpfad entspricht Klassenname
- ③ die Klasse lädt ihre Basisklassen selbständig

Beispiel: Definition einer autoladbaren Klasse

My/Class.pm

```
1 package My::Class;  
2 use base qw/<BASECLASSES>/;  
3  
4 <METHODS>  
5  
6 1;
```


Wie funktioniert das?

- 1 die Klasse selbst
- 2 alle Basisklassen (Tiefensuche über @ISA)

- 1 die Klasse selbst
- 2 alle Basisklassen (Tiefensuche über @ISA)
- 3 UNIVERSAL

- ① die Klasse selbst
- ② alle Basisklassen (Tiefensuche über @ISA)
- ③ UNIVERSAL
- ④ AUTOLOAD() in 1-3

- 1 die Klasse selbst
- 2 alle Basisklassen (Tiefensuche über @ISA)
- 3 UNIVERSAL
- 4 AUTOLOAD() in 1-3

Implementierung Klasse ClassLoader

Definition

```
1 package ClassLoader;  
2 push @UNIVERSAL::ISA, 'ClassLoader';  
3  
4 sub AUTOLOAD {  
5     ...  
6 }  
7  
8 1;
```

Implementierung AUTOLOAD-Methode

```
1 sub AUTOLOAD {  
2     my $this = shift;  
3     # @_: Methodenargumente
```

Implementierung AUTOLOAD-Methode

```
1 sub AUTOLOAD {  
2     my $this = shift;  
3     # @_: Methodenargumente  
4  
5     my ($class,$sub) = our $AUTOLOAD =~ /^(.*)::(\w+)$/;
```


Implementierung AUTOLOAD-Methode

```
1 sub AUTOLOAD {  
2     my $this = shift;  
3     # @_: Methodenargumente  
4  
5     my ($class,$sub) = our $AUTOLOAD =~ /^(.*)::(\w+)$/;  
6     return if $sub !~ /^[^A-Z]/; # return if DESTROY
```

Implementierung AUTOLOAD-Methode

```
1 sub AUTOLOAD {  
2     my $this = shift;  
3     # @_: Methodenargumente  
4  
5     my ($class,$sub) = our $AUTOLOAD =~ /^(.*)::(\w+)$/;  
6     return if $sub !~ /^[^A-Z]/; # return if DESTROY  
7  
8     eval "use $class";
```

Implementierung AUTOLOAD-Methode

```
1  sub AUTOLOAD {  
2      my $this = shift;  
3      # @_: Methodenargumente  
4  
5      my ($class,$sub) = our $AUTOLOAD =~ /^(.*)::(\w+)$/;  
6      return if $sub !~ /^[^A-Z]/; # return if DESTROY  
7  
8      eval "use $class";  
9      if ($@) {  
10         <EXCEPTION: Modul kann nicht geladen werden>  
11     }
```

Implementierung AUTOLOAD-Methode

```
1  sub AUTOLOAD {  
2      my $this = shift;  
3      # @_: Methodenargumente  
4  
5      my ($class,$sub) = our $AUTOLOAD =~ /^(.*)::(\w+)$/;  
6      return if $sub !~ /^[^A-Z]/; # return if DESTROY  
7  
8      eval "use $class";  
9      if ($@) {  
10         <EXCEPTION: Modul kann nicht geladen werden>  
11     }  
12  
13     unless ($this->can($sub)) {  
14         <EXCEPTION: Methode existiert nicht>  
15     }
```

Implementierung AUTOLOAD-Methode

```
1  sub AUTOLOAD {  
2      my $this = shift;  
3      # @_: Methodenargumente  
4  
5      my ($class,$sub) = our $AUTOLOAD =~ /^(.*)::(\w+)$/;  
6      return if $sub !~ /^[^A-Z]/; # return if DESTROY  
7  
8      eval "use $class";  
9      if ($@) {  
10         <EXCEPTION: Modul kann nicht geladen werden>  
11     }  
12  
13     unless ($this->can($sub)) {  
14         <EXCEPTION: Methode existiert nicht>  
15     }  
16  
17     return $this->$sub(@_);  
18 }
```

- abweichender Modulpfad \Rightarrow per use laden
- mehrere Klassen per Modul \Rightarrow per use laden

- abweichender Modulpfad \Rightarrow per use laden
- mehrere Klassen per Modul \Rightarrow per use laden
- Module aus Funktionen \Rightarrow per autouse oder use laden

- abweichender Modulpfad \Rightarrow per `use` laden
- mehrere Klassen per Modul \Rightarrow per `use` laden
- Module aus Funktionen \Rightarrow per `autouse` oder `use` laden
- autom. Laden per `import()` nicht möglich

- abweichender Modulpfad \Rightarrow per `use` laden
- mehrere Klassen per Modul \Rightarrow per `use` laden
- Module aus Funktionen \Rightarrow per `autouse` oder `use` laden
- autom. Laden per `import()` nicht möglich

- sonstige AUTOLOAD-Methoden stören *nicht*
- keine Performance-Einbuße

- sonstige AUTOLOAD-Methoden stören *nicht*
- keine Performance-Einbuße

<http://search.cpan.org/~fseitz/ClassLoader/>