

## Dynamische Webseiten mit Perl

Um die Möglichkeiten von Perl<sup>1</sup> in der Webprogrammierung nutzen zu können, wird Webspaces mit Perl-Interpreter benötigt. Die meisten Provider bieten diese Möglichkeit schon zu relativ geringen Preisen an.

Perl kann man auf zwei verschiedene Wege für die Programmierung von dynamischen Webseiten benutzen. Die erste Variante dürfte die bekanntere sein – die CGI<sup>2</sup>-Skripte. Eine weitere Variante, die allerdings nicht so verbreitet ist, sind die Skripte mit mod\_perl.

Bei CGI-Skripten wird für jeden Request ein Prozess generiert, bei mod\_perl wird das Skript einmal geladen und bleibt dann im Speicher. Dies bringt enorme Geschwindigkeitsvorteile gegenüber der normalen CGI-Variante.

Den meisten dürften CGI-Skripte bei der Auswertung von Kontaktformularen oder bei Gästebüchern über den Weg laufen.

In den kommenden Wochen werde ich noch auf verschiedene Sachen eingehen wie:

- Template-Systeme
- Erstellung von Microsoft-Formaten
- Verbindung zu Datenbanken
- usw.

In diesem Teil des Tutorials geht es um grundlegende Sachen, bei der Programmierung von dynamischen Webseiten mit Perl. Und ich möchte zeigen, wie man sich als Programmierer viel Arbeit ersparen kann und dabei auch noch etwas für die Sicherheit der Skripte tun kann. Das ganze will ich an dem Beispiel Auswertung eines Kontaktformulars besprechen. Mit dem Skript, das wir hier schreiben, wollen wir überprüfen, ob der Benutzer alle nötigen Eingaben gemacht hat. Ist dies der Fall, wollen wir dem Betreiber der Homepage eine Benachrichtigung per Mail schicken. Ist dies nicht der Fall, wollen wir den Benutzer auf die fehlerhaften Eingaben aufmerksam machen.

Grundkenntnisse der Perlprogrammierung sollten aber schon vorhanden sein. Für Anfänger kann ich das Tutorial von Martin Fabiani<sup>3</sup> empfehlen.

Das Formular<sup>4</sup> ist folgendes:

```
<form action=" ./cgi-bin/kontakt.cgi" method="post">
<center><h2>Informationen anfordern...</h2></center>
<table>
<tr><td>Vorname: *</td>
<td><input type="text" name="vorname" size="15" /></td></tr>
<tr><td>Nachname: *</td>
<td><input type="text" name="nachname" size="15" /></td></tr>
<tr><td>E-Mail: *</td>
<td><input type="text" name="email" size="15" /></td></tr>
</table>
<hr width="100%">
```

1 <http://perl.com> und für Windows <http://activestate.com>

2 CommonGatewayInterface

3 Zu finden unter <http://fabiani.net> -> Vorträge -> Perl für Einsteiger

4 Von <http://hess-innenausbau.de>

```

Ich w&uuml;nsche<br>
<input type="Checkbox" name="Info" value="checked">Material zu<br>
<select name="materialien" size="3" multiple>
  <option>Kork</option>
  <option>Parkett</option>
  <option>Laminat</option>
  <option>Holzdecken</option>
</select><br>
<input type="checkbox" name="Besuch" />einen Besuch <br>
Hier können Sie Ihre Meinung loswerden und Fragen stellen:<br>
<textarea rows="5" cols="30" name="sonstiges"></textarea><br>
<hr width="100%">
<input type="submit" value="Abschicken" />
<input type="reset" value="L&ouml;schen" />
</form>

```

Beim Abschicken des Formulars wird das Perl-Programm kontakt.cgi aufgerufen. Zur besseren Unterscheidung bekommen im Normalfall CGI-Skripte die Endung .cgi und Perl-Programme für die Konsole die Endung .pl .

Das Programm bekommt die Daten aus dem Formular als Parameter gesendet. Um diese in unserem Programm verwenden zu können, müssen diese geparkt werden. Doch schon hier können die Schwierigkeiten beginnen. Im Internet findet man viele Skripte, die das Parsen mit Regulären Ausdrücken machen, aber davon rate ich ab.

Weiterhin sollte immer die Grundregel lauten „Traue keiner Benutzereingabe“. Benutzereingaben können gefährlich sein, wenn man bestimmte Sachen nicht beachtet.

So sollte man auf keinen Fall eine Datei öffnen, deren Namen man vom Benutzer bekommen hat. Dies kann vor allem böse Auswirkungen haben, wenn man seine CGI-Skripte als root laufen lässt. Stellen Sie sich vor, ein Besucher gibt `rm -rf /` ein und sie versuchen dies als Datei zu öffnen. Außerdem sollte man bei jedem CGI-Skript den Taint-Modus<sup>5</sup> wählen, der alle Benutzereingaben als gefährlich einstuft, bis der Programmierer das „Ok“ gegeben hat. Der Taint-Modus wird über den Schalter `-T` in der Shebang aktiviert. Dies sieht dann so aus:

```
#!/usr/bin/perl -T
```

Eine der größten Hilfen ist das Modul CGI.pm<sup>6</sup> von Lincoln D. Stein<sup>7</sup>, das zu jeder Standarddistribution von Perl gehört. Es nimmt uns die Arbeit des Parsens von Parametern ab, was nicht nur viele Fehler vermeidet, sondern auch die Anfälligkeit des Skripts stark herabsetzt. Zusätzlich wird der HTML-Quelltext innerhalb des CGI-Skripts auf ein Minimum reduziert!

Eines der Schlagworte der Programmierung der letzten Jahre war „Objektorientierung“. Das können wir auch mit Perl machen. Um das Modul CGI.pm und dessen Methoden im Skript nutzen zu können, muss das Modul in das Skript eingebunden werden.

```
use CGI;
```

Für die weitere Verwendung der CGI.pm-Methoden erzeugen wir ein Objekt von CGI.

```
my $cgi = CGI->new();
```

<sup>5</sup> FAQ zu Taint-Mode: <http://gunther.web66.com/FAQS/taintmode.html>

<sup>6</sup> CGI.pm ist auf CPAN (<http://cpan.org>) erhältlich. Dort findet man auch eine ausführliche Dokumentation.

<sup>7</sup> <http://search.cpan.org/~lds>

Damit eine Seite angezeigt wird, muss diese einen Header bekommen. Dieser enthält unter anderem den Content-type. Meine Erfahrungen, die ich in diversen Foren gesammelt habe, haben gezeigt, dass gerade hier ein häufiger Fehler liegt, wenn ein CGI-Skript nicht funktioniert. Das CGI-Modul bietet eine bequeme Möglichkeit den Header zu senden:

```
print $cgi->header(-type => 'text/html');
```

Für das Parsen der Parameter gibt es die Methode Vars(). Diese Methode liefert einen Hash, in dem die Namen der Formularfelder als Schlüssel dienen und die Benutzereingaben sind die dazugehörigen Werte.

Das sieht dann folgendermaßen aus:

```
my %params = $cgi->Vars(); # parsen der Parameter
print $params{vorname};
```

Die Benutzereingaben stehen uns jetzt also in dem Hash zur Verfügung. Nun wollen wir überprüfen, ob diese auch mögliche wahre Werte haben. Ob diese wirklich stimmen, kann man nicht überprüfen, da man sonst die Person direkt fragen müsste.

Was können wahre Werte sein?

- Vorname – darf keine Zahlen oder Satzzeichen oder ähnliches enthalten
- Nachname – wie Vorname
- E-Mail-Adresse – muss ein @ enthalten und noch weitere Kriterien

für den Vornamen und den Nachnamen kann man den gleichen regulären Ausdruck verwenden. Der Ausdruck für die E-Mail-Adresse ist äußerst kompliziert. Auch hier können wir wieder auf den Modulreichtum von CPAN<sup>8</sup> zurückgreifen. Hier benutzen wir das Modul Mail::RFC822::Address von Paul Warren<sup>9</sup>. Doch dazu kommen wir später.

Die oben genannten Bedingungen führen zu folgendem Regulären Ausdruck für den Vor-/Nachnamen. Das Problem ist, dass man extrem viele Zeichen ausschließen müsste. Ich beschränke mich hier auf ein paar Sonderzeichen.

```
/^[^\d\.\,\?\=]$/
```

```
my $vorname = $params{vorname};
if($vorname =~ /[\d\.\,\?\=]/){
    print $vorname.' ist ungültig';
}
```

Die gleiche Überprüfung muss für den Nachnamen stattfinden!

Das Modul Mail::RFC822::Address enthält schon den komplizierten Regulären Ausdruck. Damit lässt sich sehr einfach die syntaktische Korrektheit der Mail-Adresse überprüfen. Das ist allerdings keine Garantie für die wirkliche Existenz der Mail-Adresse! Der Test sieht dann folgendermaßen aus:

```
use Mail::RFC822::Address; # einbinden des Moduls
my $mail = $params{email};
```

<sup>8</sup> CPAN (Comprehensive Perl Archive Network) – größtes Archiv an Perl-Modulen <http://www.cpan.org>

<sup>9</sup> <http://search.cpan.org/~pwarren>

```
print „E-Mail-Adresse ungültig“ unless(valid($mail));
```

Jetzt haben wir alle Überprüfungen hinter uns und können uns jetzt damit beschäftigen, eine Mail an den Betreiber der Webseite zu schicken. In diesem Teil des Tutorials, benutzen wir noch sendmail, aber in den nächsten Teilen werden wir weitere Module von CPAN benutzen. Die Module haben auch den Vorteil, dass sendmail nicht unbedingt installiert sein muss. Vielmehr kann man damit andere SMTP-Server benutzen.

Als erstes erstellen wir den Text, den der Empfänger bekommen soll. Dieser soll die Daten, die der Besucher der Seite eingegeben hat, beinhalten. Hier werden wir den Text aber sehr einfach halten.

```
my $mailtext = „Hallo Betreiber der Webseite,\n\nheute war $vorname $nachname auf Deiner Seite und möchte Kontakt zu Dir aufnehmen. $vorname $nachname möchte Informationsmaterialien über\n\". map{„* $_\n\"}split(/\0/, $params{Info}. „bekommen.\n\nDein Kontaktbot“;
```

Zunächst muss eine Verbindung zu sendmail hergestellt werden. Da wir dem Programm noch ein paar Parameter übergeben müssen, öffnen wir eine Pipe zum Schreiben.

```
my $sendmail = '/Pfad/zu/sendmail';  
open(MAIL, "| $sendmail -oit") or die $!;
```

Danach müssen die Parameter wie Empfänger, Absender, Betreff und natürlich der Mailtext übergeben werden. Dazu müssen diese in die Pipe ausgegeben werden. Wichtig ist dabei, dass nach dem Header der Mail eine Leerzeile kommt!

```
print MAIL 'To: betreiber@webseite.tld'. "\n";  
print MAIL 'From: kontakt@webseite.tld'. "\n";  
print MAIL 'Subject: Anfrage über das Webformular'. "\n\n";  
print MAIL $mailtext;
```

Danach muss die Pipe wieder geschlossen werden.

```
close MAIL;
```

Das komplette Skript sieht jetzt folgendermaßen aus:

```
#!/usr/bin/perl -T

# um Flüchtigkeitsfehler zu vermeiden und vereinfacht Fehlersuche:
use strict;
use warnings;
use CGI; # CGI-Modul einbinden
# Fehler im Browser ausgeben:
use CGI::Carp qw(fatalsTo Browser warningsToBrowser);
use Mail::RFC822::Adress; # einbinden des Moduls

my $cgi = CGI->new(); # neues CGI-Objekt erzeugen
print $cgi->header(-type => 'text/html'); # Header ausgeben
print $cgi->start_html(-title => 'Formularauswertung');

warningsToBrowser(1); # Warnungen für Browser aktivieren

my %params = $cgi->Vars(); # Parameter auslesen

if(valid_params(\%params)){
    danke(\%params);
}
else{
    print $cgi->b('Leider sind ihre Daten fehlerhaft!');
}

print $cgi->end_html();

sub danke{
    my ($paramref) = @_ ;
    my %params = %$paramref;
    my $mailtext = „Hallo Betreiber der Webseite,\n\nheute war
        $params{vorname} $params{nachname} auf Deiner
        Seite und möchte Kontakt zu Dir aufnehmen.\n\n
        $params{vorname} $params{nachname}
        möchte Informationsmaterialien über\n".
        map{„* $_\n"}split(/\0/, $params{Info}).
        "bekommen.\n\nDein Kontaktbot";

    my $sendmail = '/Pfad/zu/sendmail';

    # Mail schreiben
    open(MAIL, "| $sendmail -oit") or die $!;
    print MAIL 'To: betreiber@webseite.tld'."\n";
    print MAIL 'From: kontakt@webseite.tld'."\n";
    print MAIL 'Subject: Anfrage über das Webformular'."\n\n";
    print MAIL $mailtext;
    close MAIL;

    # Ein Dankeschön an den Besucher
    print $cgi->b('Sehr geehrte/r ');
    print $cgi->b($params{vorname}.' '.$params{nachname}.'.');
    print $cgi->br;
    print $cgi->br;
    print „wir werden ihre Anfrage schnellstmöglich bearbeiten.“;
}
```

```
sub valid_params{
    my ($paramref) = @_ ;
    if($paramref->{vorname} =~ /\d\.\.\,?\=\/){
        return 0;
    }
    unless($paramref->{nachname} =~ /\d\.\.\,?\=\/){
        return 0;
    }
    unless(valid($paramref->{email})){
        return 0;
    }
    return 1;
}
```

Links für Perlprogrammierer:

<http://www.perl.com> – Perl

<http://www.perl-community.de> – das große deutschsprachige Perl-Forum

<http://www.perlmonks.org> – englischsprachiges Perl-Forum

<http://www.cpan.org> – Archiv mit Perl-Modulen

<http://www.perldoc.com> – Online-Version der perldocs (Dokumentation)

<http://www.activestate.com> – Perl für Windows

<http://www.oreilly.com> – Perl-Bücher

<http://regenechsen.de> – Tutorial zu Regulären Ausdrücken

Der Autor:

Renee Bäcker ist Gesellschafter der Smart-Websolutions Windolph, Morgenroth und Bäcker GbR.

Er programmiert seit ein paar Jahren Perl und ist Moderator im Perl-Community-Forum. Er ist

hauptsächlich im Bereich der Bioinformatik und auch im Webbereich tätig. ([http://www.renee-](http://www.renee-baecker.de)

[baecker.de](http://www.renee-baecker.de)) < [perl@renee-baecker.de](mailto:perl@renee-baecker.de) >