

Einführung

HTML::Template ist ein Modul, das hauptsächlich für die Entwicklung von Perl-Webanwendungen benutzt wird. Allerdings kann man das Modul auch für weitere Template-basierte Anwendungen benutzen.

Ein großer Vorteil durch die Verwendung dieses Moduls besteht darin, dass das Design unabhängig vom Programmcode gestaltet werden kann. So kann das Design von einem Grafiker und der Code von einem Programmierer getrennt entwickelt werden. Man braucht also nicht mehr einen programmierenden Grafiker oder einen designenden Programmierer.

Beispiele für die Verwendung des HTML::Template-Moduls sind das neue Perl-Community.de-Board (<http://develop.perl-community.de>) und das Informationsportal Urlaub-im-Ferienpark.de (<http://www.urlaub-im-ferienpark.de>).

Einführungsbeispiel

Template

```
<html>
  <head>
    <title>
      <!-- TMPL_VAR NAME=WILLKOMMEN -->
    </title>
  </head>
  <body>
    <h1><!-- TMPL_VAR NAME=WILLKOMMEN --></h1>
  </body>
</html>
```

Skript

```
#!/usr/bin/perl
use strict;
use warnings;
use HTML::Template;

print Content-type: "text/html\n\n";

my $template = HTML::Template->new(filename =>
                                   './template.tpl');
my $willkommen = 'Willkommen bei der Perl-Community';

$template->param(WILLKOMMEN => $willkommen);

print $template->output();
```

Ergebnis



Tags

Das Modul verwendet für die Parameter Tags, die den „normalen“ HTML-Tags ähneln. Diese können so aussehen:

```
<TMPL_VAR NAME=NAME_DES_PARAMETERS>
```

Die Tags können aber auch in Form von HTML-Kommentaren geschrieben werden. Diese sehen dann so aus:

```
<!-- TMPL_VAR NAME=NAME -->
```

Diese Schreibweise sollte man verwenden, wenn man validen HTML-Code in seinem Template haben will.

In dem Modul stehen die folgenden Tags zur Verfügung:

TMPL_VAR

TMPL_VAR ist der einfachste Tag, der für Templates vorgesehen ist. Für jeden Parameter muss im Skript eine Ersetzung stattfinden. Bei einem fehlendem Wert, wird der Tag einfach weggelassen (an dieser Stelle ist im Output nichts zu sehen).

Bei *TMPL_VAR* hat man die Möglichkeit, vor der Ersetzung HTML-Zeichen im Wert zu ersetzen. Dies sollte man machen, wenn die Ausgabe kritisch werden könnte, so z.B. wenn die Ersetzung innerhalb eines HTML-Tags vorgenommen werden soll:

```
<input name="<!-- TMPL_VAR NAME=NAME -->">
```

sollte dann so aussehen:

```
<input name="<!-- TMPL_VAR NAME=NAME ESCAPE=HTML -->">
```

Hier kann ein `>` im Wert eine fehlerhafte Ausgabe bedeuten.

TMPL_VAR besitzt kein schließendes Tag.

TMPL_LOOP

TMPL_LOOP ist sehr gut zur Darstellung von Tabellen geeignet. So muss man dann nicht für jede Zeile alle Zellen mit einem eigenen Parameter füllen, sondern gestaltet eine „Beispielzeile“ und lässt es dann durch die Schleife ersetzen.

Innerhalb von *TMPL_LOOP* benutzt man dann *TMPL_VAR*, um Werte darzustellen. Eine Schachtelung von *TMPL_LOOP* ist möglich. Damit kann man z.B. mehrere Hobbies von Personen anzeigen lassen. Im Beispiel wird allerdings nur eine einzelne Schleife mit zwei Parametern eingesetzt.

Zum Ersetzen bekommt das Objekt ein Arrayreferenz übergeben, in der Hashreferenzen enthalten sind. In diesen Hashreferenzen sind dann die Werte für die *TMPL_VAR* enthalten.

Um die Gestaltung der Tabelle noch flexibler zu machen, gibt es die Laufvariablen `__first__`, `__inner__`, `__last__` und `__odd__`. Hierzu wird nur der Template-Text angegeben. Mit diesen Laufvariablen kann man die Darstellung von Tabellenkopf (z.B. Überschriften), Tabellenkörper (z.B. Daten) und Tabellenfuß (z.B. Copyrightangabe) unterschiedlich gestalten.

Diese Laufvariablen können mit *TMPL_IF*, *TMPL_ELSE* und *TMPL_UNLESS* benutzt werden. Im Beispieltemplate wird nur *TMPL_IF* benutzt.

Um dies nutzen zu können, muss bei den Optionen `loop_context_vars` auf 1 gesetzt werden (siehe **Methoden** `new()`).

Den Template-Text finden Sie nach dem Beispiel.

Template

```
<html>
  <head>
    <title>
      MITGLIEDER
    </title>
  </head>
  <body>
    <table border=1>
      <!-- TMPL_LOOP NAME=WILLKOMMEN -->
      <tr>
        <td><!-- TMPL_VAR NAME=VORNAME --></td>
        <td><!-- TMPL_VAR NAME=NACHNAME --></td>
      </tr>
      <!-- /TMPL_LOOP -->
    </table>
  </body>
</html>
```

Skript

```
#!/usr/bin/perl
use strict;
use warnings;
use HTML::Template;

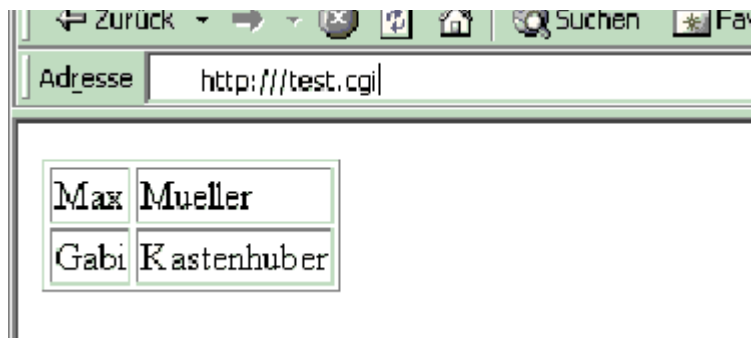
print Content-type: "text/html\n\n";

my $template = HTML::Template->new(filename =>
                                   './template.tpl');
my $willkommen = [{Vorname => 'Max',
                   Nachname => 'Mueller'},
                  {Vorname => 'Gabi',
                   Nachname => 'Kastenhuber'}];

$template->param(WILLKOMMEN => $willkommen);

print $template->output();
```

Ergebnis



Template (Laufvariablen)

```
<html><body><table border=1>
  <!-- TMPL_LOOP NAME=WILLKOMMEN -->
  <!-- TMPL_IF NAME=__first__ -->
  <tr>
    <td><!-- TMPL_VAR NAME=TITLE1 --></td>
    <td><!-- TMPL_VAR NAME=TITLE2 --></td>
  </tr>
  <!-- /TMPL_IF -->

  <!-- TMPL_IF NAME=__inner__ -->
  <tr>
    <td><!-- TMPL_VAR NAME=VORNAME --></td>
    <td><!-- TMPL_VAR NAME=NACHNAME --></td>
  </tr>
  <!-- /TMPL_IF -->
  <!-- /TMPL_LOOP -->
</table></body></html>
```

TMPL_INCLUDE

Mit *TMPL_INCLUDE* kann man weitere Dateien in das Template einfügen. Man kann es als SSI (Server Side Includes)-Ersatz nehmen, oder weitere HTML-Templates einfügen.

Die Parameter des eingefügten Templates werden dann wie „eigene“ Parameter behandelt. So muss man nicht zwei verschiedene Objekte erzeugen, um die Parameter zu ersetzen, sondern man ersetzt alles in einem Objekt.

Um Endlosschleifen zu vermeiden – wenn sich ein Template immer selbst includen würde – ist die maximale Tiefe der Includes 10. Das heißt, dass das Template sich maximal 10 mal selbst einbinden kann.

Diese Tiefe ist frei konfigurierbar (siehe **Methoden** Abschnitt *new()*).

Auch das *TMPL_INCLUDE* hat kein abschließendes Tag. Der Dateiname der einzubindenden Datei muss mit einem Dateipfad angegeben werden, also nicht mit einer URL.

Template1

```
<html>
  <head>
    <title>
      TMPL_INCLUDE - Beispiel
    </title>
  </head>
  <body>
    <!-- TMPL_INCLUDE NAME=Template2.tpl -->
  </body>
</html>
```

Template2

```
<h2>
  <!-- TMPL_VAR NAME=WILLKOMMEN --> Welt!
</h2>
<br />
Dies ist eine <!-- TMPL_VAR NAME=VARIABLE -->.
```

Skript

```
#!/usr/bin/perl
use strict;
use warnings;
use HTML::Template;

print Content-type: "text/html\n\n";

my $template = HTML::Template->new(filename =>
                                   './template.tpl');

my $willkommen = "Hallo";
my $scalar = "Ersetzung";

$template->param(WILLKOMMEN => $willkommen,
                 VARIABLE => $scalar);

print $template->output();
```

Ergebnis



TMPL_IF

Das erste „Logiktag“ in HTML::Template-Modul ist der *TMPL_IF*-Tag. Hier wird der Text zwischen den Tags nur ausgegeben, wenn die Variable im Perlskript einen Wahrheitswert liefert.

Allgemein sollten die „Logiktags“ nur verwendet werden, wenn im Skript nicht schon die entsprechende Logik verwendet wird, da sonst bei einer Änderung die entsprechenden Stellen in zwei verschiedenen Dateien vorgenommen werden müssen.

Wichtig bei *TMPL_IF* ist, dass am Ende das abschließende Tag kommt.

Template

```
<html>
  <head>
    <title>
      TMPL_IF - Beispiel
    </title>
  </head>
  <body>
    <!-- TMPL_IF NAME=BOOL -->
      Text zwischen den If-Tags
    <!-- /TMPL_IF -->
  </body>
</html>
```

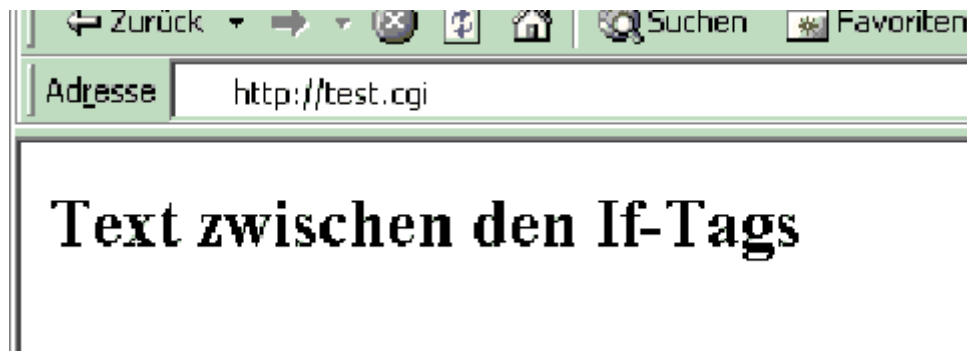
Skript

```
#!/usr/bin/perl
use strict;
use warnings;
use HTML::Template;

print Content-type: "text/html\n\n";

my $template = HTML::Template->new(filename =>
                                   './template.tpl');
my $willkommen = 1; # setzten des Wahrheitswertes
$template->param(BOOL => $willkommen);
print $template->output();
```

Ergebnis



TMPL_ELSE

TMPL_ELSE ist der „Alternativ“-Block zu *TMPL_IF* bzw. *TMPL_UNLESS* und entspricht der „normalen“ Programmierlogik. Allerdings sind if- (bzw. unless-)Block und else-Block nicht getrennt, sondern verschachtelt. Dem entsprechend gibt es auch kein schließendes Tag für *TMPL_ELSE*.

Der Text nach dem *TMPL_ELSE*-Tag wird dann ausgegeben, wenn der vorherige Parameter nicht zutrifft. Das heißt, wenn *TMPL_IF* und *TMPL_ELSE* kombiniert sind, dann wird der Text nach dem *TMPL_ELSE*-Tag nur dann ausgegeben, wenn das if unwahr ist.

Template

```
<html>
  <head>
    <title>
      TMPL_IF - Beispiel
    </title>
  </head>
  <body>
    <!-- TMPL_IF NAME=BOOL -->
      Text zwischen den If-Tags
    <!-- TMPL_ELSE -->
      Sonst kommt dieser Text
    <!-- /TMPL_IF -->
  </body>
</html>
```

Skript

```
#!/usr/bin/perl
use strict;
use warnings;
use HTML::Template;

print Content-type: "text/html\n\n";

my $template = HTML::Template->new(filename =>
                                   './template.tmpl');

my $willkommen = 0;

$template->param(BOOL => $willkommen);

print $template->output();
```

Ergebnis



TMPL_UNLESS

Das TMPL_UNLESS entspricht dem unless() von Perl. Das heißt, dass der Text zwischen den TMPL_UNLESS-Tags ausgegeben wird, wenn die Variable false zurückgibt oder nicht definiert ist. Auch hier kann ein TMPL_ELSE eingesetzt werden.

Template

```
<html>
  <head>
    <title>
      TMPL_UNLESS - Beispiel
    </title>
  </head>
  <body>
    <!-- TMPL_UNLESS NAME=BOOL -->
      Text zwischen den Unless-Tags
    <!-- /TMPL_UNLESS -->
  </body>
</html>
```

Skript

```
#!/usr/bin/perl
use strict;
use warnings;
use HTML::Template;

print Content-type: "text/html\n\n";

my $template = HTML::Template->new(filename =>
                                   './template.tpl');
my $willkommen = 0;

$template->param(BOOL => $willkommen);

print $template->output();
```


Für `new()` gibt es noch eine ganze Menge an Optionen, die man einstellen kann. Hier wird allerdings nur auf die wichtigsten Optionen eingegangen. Alle weiteren Optionen können in der Dokumentation zu `HTML::Template` nachgelesen werden (<http://search.cpan.org/~samregar/HTML-Template-2.6/Template.pm>).

Fehlererkennung

Die Optionen zur Fehlererkennung sind defaultmäßig alle auf 1 gestellt (außer der `vanguard_compatibility_mode`), um es während der Entwicklung dem Programmierer leichter zu machen, Fehler zu finden..

- `die_on_bad_params`
 - 0: Das Programm wird nicht beendet, wenn ein falscher oder nichtexistierender Parametername angegeben wird.
 - 1: Das Programm wird sofort beendet, wenn ein falscher Parametername bei der Ersetzung angegeben wird.
- `strict`
 - 0: Das Skript wird nicht beendet, auch wenn ein Templateähnliche Tags auftaucht.
Bsp.: `<!-- TMPL_HUH NAME=ZUH -->`
 - 1: Lässt nur ursprüngliche Template-Tags zu. Sonst wird das Skript sofort beendet.
- `vanguard_compatibility_mode`
 - 0: Es wird nur die original Syntax der Tags zugelassen
 - 1: Statt `<!-- TMPL_VAR NAME=WAS -->` ist auch die Notation in dieser Form erlaubt:
`%NAME%`

Caching

- `cache`

Wird diese Option eingeschaltet, wird das geparste Template in den Cache gespeichert. Dies funktioniert allerdings nur bei Templates, deren Objekte mit der `filename`-Option erzeugt wurden (siehe `new()`).

Bei jedem Aufruf der Seite wird dann überprüft, ob in der Zwischenzeit etwas am Template geändert wurde. Ist dies nicht der Fall, wird die gecachte Version ausgegeben, sonst wird das Template neu geladen.

Dies wird hauptsächlich in der Apache/mod_perl-Umgebung eingesetzt, da hier eine persistente Umgebung vorhanden ist. Bei Perl/CGI wird der Prozess sowieso bei jedem Aufruf neu erzeugt, so dass hier ein cachen nichts bringen würde.

Bei Tests wurde festgestellt, dass die Performance unter mod_perl um bis zu 90% verbessert wird.

- shared_cache

Diese Option kann auch bei Perl/CGI-Skripten benutzt werden. Hier wird das Modul IPC::SharedCache (auf CPAN erhältlich) benutzt. In diesem Skript wird auf die Funktionsweise dieses Moduls nicht eingegangen. Die ist in der Dokumentation zu dem Modul nachzulesen (<http://search.cpan.org/~samregar/IPC-SharedCache-1.3/SharedCache.pm>).

- double_cache

Ist diese Option eingeschaltet, wird eine Kombination von cache und shared_cache benutzt für das bestmögliche Caching. Standardmäßig ist diese Option ausgeschaltet.

Verschiedene

- associate

Diese Option erlaubt es, ein anderes Objekt einzubinden. Das andere Objekt muss aber eine param()-Methode besitzen. Wenn diese Option eingeschaltet ist, „erbt“ das Template-Objekt die Parameterwerte des assoziierten Objekts.

Diese Option wird häufig verwendet, um ein CGI-Objekt an ein Template-Objekt zu binden. So ist es einfach möglich, Werte, die in einem Formular eingegeben wurden in das Template zu übernehmen.

- case_sensitive

Ist diese Option eingeschaltet, wird bei der Parameterersetzung auf die Groß- und Kleinschreibung der Parameter geachtet. Dann ist NAME nicht gleich Name oder name.

Dies ist Standardmäßig ausgeschaltet.

- loop_context_vars

Möchte man die Schleifenvariablen benutzen, muss man diese Option einschalten.

- no_includes

Mit dieser Option verbietet man das einbinden von anderen Dateien.

- max_includes

Mit max_includes wird die maximale Tiefe der Einbindungen angegeben.

- global_vars

Hiermit kann man Parameter, die außerhalb einer Schleife deklariert wurden auch innerhalb der Schleife benutzen.

param()

Mit der Methode `param()` kann man verschiedene Sachen über die Parameter des Templates erfahren.

1. Abfragen, welche Parameter sind im Template enthalten

Hierbei gibt die Methode eine Liste mit allen Namen der im Template enthaltenen Parametern zurück. Dafür wird die Methode ohne Parameter aufgerufen.

```
my @parameter = $template->param();
```

2. Abfragen, welchen Wert ein Parameter hat

```
my $value = $template->param('NAME_DES_PARAMETERS');
```

3. Setzen des Wertes eines Parameters

```
$template->param(NAME => 'Wert');
```

4. Setzen von Werten von mehreren Parametern

```
$template->param(NAME1 => 'Wert1',  
                NAME2 => 'Wert2');
```

clear_param()

`clear_param()` setzt alle Parameter auf undef. Ein Anwendungsbeispiel, das man sich gut vorstellen kann ist, dass man versuchen möchte, mehrere statische HTML-Seiten zu erstellen und dafür ein und das selbe Template zu benutzen. Dann braucht man nicht mehrere Objekte von `HTML::Template` zu erzeugen. Ein einziges Objekt reicht aus. Man ersetzt für jede HTML-Seite, die man erstellen möchte die Parameter, speichert die HTML-Seite ab, setzt alle Parameter mit `clear_param()` wieder auf undef und fängt für die nächste Seite wieder von vorne an.

So kann man ein und das selbe Objekt für alle HTML-Seiten benutzen.

query()

Diese Methode gibt Informationen über die Struktur des Templates aus. Hiermit ist es möglich abzufragen, ob ein Parameter überhaupt im Template enthalten ist, und von welchem Typ dieser Parameter ist.

Für die Abfrage wird die name-Option verwendet.

Dabei werden die Namen der Parameter in Kleinbuchstaben zurückgegeben, die Typbezeichnungen in Großbuchstaben. Als Typbezeichnungen gelten VAR, LOOP, IF usw. - also die Tagbezeichnungen ohne das führende TMPL_

```
my $template = HTML::Template->new(filename =>
    './template.tpl', options => 'value');
if($template->query(name => 'foo')){
    # mach was, wenn der Parameter existiert
}

if($template->query(name => 'foo') eq 'VAR'){
    # mach was, wenn der Parameter vom Typ 'VAR' ist
}
```

output()

output() gibt den gesamten Template-Text zurück. Hierbei sind die Parameter bereits ersetzt.

```
my $inhalt = $template->output();
```