

HTML-Seiten parsen mit HTML::Parser (Einführung)

Warum sollte ich eine HTML-Seite parsen wollen?

Dafür gibt es mehrere Gründe. Der eine will Google-Ergebnisse parsen, der andere hat sich eine Forumssoftware geschrieben, bei der Teile von HTML wieder in BBCode umgewandelt werden müssen.

Oder einfach, weil es ein Kunde gewünscht hat oder man einfach mal etwas Probieren möchte.

Warum ein so komplexes Modul wie HTML::Parser?

Viele versuchen, mit eigenen Regulären Ausdrücken eine HTML-Seite zu parsen. Die meisten davon scheitern, weil es sehr komplex ist, einen geeigneten Regulären Ausdruck zu finden, der alle Eventualitäten behandelt. Wer weiß schon, dass die Seite, die er parsen möchte, sich auch 100%ig an den Standard von W3C hält? Keiner. Auf der einen Seite sind die Tags groß, auf der anderen Seite klein geschrieben. Auf der einen Seite, kommt bei einem Link erst das href-Attribut und danach ein alt-Attribut, auf der anderen Seite ist es gerade umgekehrt.

Mit HTML::Parser hat man diese Probleme nicht. Das Modul ist sehr zuverlässig, wenn es um Parsen von HTML-Seiten geht.

Bei „einfachen“ Aufgaben scheint das Modul etwas überdimensioniert und am Anfang blickt man vielleicht noch nicht so ganz durch, wann ich wie was einsetzen muss. Aber nach ein paar kleinen Übungen oder bei einer schwierigen Aufgabe lernt man das Modul zu schätzen.

Methoden von HTML::Parser

Das Modul stellt ein paar Methoden zur Verfügung, wobei eigentlich drei/vier Methoden für den Anfang ausreichen:

- `new`
- `parse`
- `handler`

Mit `new` erzeugt man ein neues Objekt von HTML::Parser. Mit diesem Objekt arbeiten wir dann weiter.

Mit `parse` lassen wir das Objekt die Datei oder den String parsen. Hierbei kann man den Dateinamen oder ein Skalar mit dem Text übergeben.

Mit `handler` wird auf verschiedene „Ereignisse“ reagiert. Die wichtigsten sind

- `start`
- `end`
- `text`

Mit `start` reagiert man auf den Beginn eines Tags. Nehmen wir als Beispiel einen Link:

```
<a href="http : //www.ziel.example">Ziel</a>
```

`start` reagiert jetzt auf das `<a>`, `text` auf 'Ziel' und `end` auf ``. Tritt jetzt eines dieser Ereignisse auf, so wird die Methode aufgerufen, die bei `handler` eingetragen ist.

Wenn auf ein solches Ereignis reagiert wird, kann man der Methode auch ein paar Parameter übergeben:

- `tagname`
- `attr`
- `self`
- `text, dtext`
- und noch weitere

`tagname` ist noch selbstredend. Hiermit wird übergeben, um welchen Tag es sich handelt (a, img, input etc.). Mit `attr` werden die Attribute übergeben. In unserem Beispiel wäre das `href` und `class`. `self` ist ein Verweis auf das eigene Objekt von `HTML::Parser`. Mit `text` wird der normale „sichtbare Text (hier: Ziel) übergeben.

Natürlich kann man alles aus einer HTML-Seite parsen. Man sollte aber erstmal mit kleinen Aufgaben anfangen, damit man mal ein Gefühl dafür bekommt, wie man die `handler` schachteln muss und welche Parameter man übergeben muss, damit man an sein Ziel kommt.

Beispiel1: Alle Links aus einer HTML-Seite parsen

```
#!/usr/bin/perl

use strict;
use warnings;
use HTML::Parser;

my @links;
my $string = qq~<a href="url1">linktext1</a> Ein anderer Text
             <a href="url2">linktext2</a> text~;
my $p = HTML::Parser->new();
$p->handler(start => \&start_handler, "tagname,attr,self");
$p->parse($string);

foreach my $link(@links){
    print "Linktext: ",$link->[1],"\tURL: ",$link->[0],"\n";
}

sub start_handler{
    return if(shift ne 'a');
    my ($class) = shift->{href};
    my $self = shift;
    my $text;
    $self->handler(text => sub{$text = shift;},"dtext");
    $self->handler(end => sub{push(@links,[$class,$text]) if
(shift eq 'a')},"tagname");
}
```

Beispiel2: Umwandlung von HTML in BBCode

```
#!/usr/bin/perl

use strict;
use warnings;
use HTML::Parser;

my ($author,$text);
my $p = HTML::Parser->new();
$p->handler(start => \&start_handler,"tagname,attr,self");
$p->parse_file('html.txt');

sub start_handler{
    return if(shift ne 'div');
    my ($class) = shift->{class};
    my $self = shift;
    if($class eq 'bbcode_quote_header'){
        $self->handler(text => \&get_author,"dtext");
    }
    elsif($class eq 'bbcode_quote_body'){
        $self->handler(text => sub{$text = shift;},"dtext");
    }
    $self->handler(end => sub{print
' [quote='.$author.']' .$text.' [/quote]' if($class eq
'bbcode_quote_body')});
}

sub get_author{
    my ($test) = @_ ;
    $test =~ s/(.*?)\s+schrieb:\s*?$/ $1/;
    $author = $test;
}
```

Inhalt von html.txt:

```
<div class="bbcode_quote_header">sammler schrieb:</div><div
class="bbcode_quote_body">blablabla...</div>
<div class="bbcode_quote_header">autor schrieb:</div><div
class="bbcode_quote_body">das ist ein text.</div>
```

Lesehinweise

Reguläre Ausdrücke

perldoc perlre

perldoc perlretut

perldoc perlrequick

<http://www.regenechsen.de>

HTML::Parser

<http://search.cpan.org/~gaas/HTML-Parser-3.45/Parser.pm>

und einige Artikel bei <http://board.perl-community.de> und <http://wiki.perl-community.de>