5/98

LINUX MAGAZIN

Das einzige deutschsprachige Linux-Magazin

LINUX MAGAZIN

ISSN 1432 - 640 X DM 9,80 ÖS 75,- SFR 9,80

5/98

Ferngesteuert

Programmierung verteilter Anwendungen mit RPC

Linux News
Neues aus dem Internet

Entwicklungsumgebung
WipeOut

Bericht
Linux im Krankenhaus

Sicherheit
Die Secure Shell

www.linux-magazin.de

Perl Snapshot

Aus der CGI-Trickkiste III

von Michael Schilli

Client und Server verständigen sich über das CGI-Protokoll normalerweise ohne daß einer der beiden Partner sich den Zustand des anderen merkt. War derselbe Kunde schon mal da? Keine Ahnung! Aber spätestens falls jemand eine Bestellung in einem Ordersystem abgibt, wäre es vielleicht sinnvoll, den Auftrag einer vorher eingegebenen Kundennummer zuzuordnen - es sei denn, man möchte nichts verkaufen.

Um diese Problematik anzugehen, braucht das CGI-Protokoll ein wenig Unterstützung, ein *Zustand* muß her: Der Kunde gibt die Kundennummer ein - *schnipp!* - bestellt etwas - *schnipp!* - bestellt noch etwas - *schnipp, schnipp, schnipp!* - alles Zustände, die gespeichert und später wieder abholt werden wollen! Bloß wo?

Hidden Fields

Hidden Fields sind HTML-Formular-Felder, die Key/Value-Paare enthalten und Zustandsinformationen einer solchen CGI-Session dauernd zwischen Client und Server hin- und hertragen. Der Server legt zu sichernde Information in einem versteckten Feld eines Formulars ab, das er sowieso dem Client schickt. Der Browser des Clients schickt es beim nächsten 'Submit' unbewußt wieder mit. Der kleine Online-Shopper im letzten Beitrag zeigte, wie das funktioniert.

Cookies

Ein Cookie enthält eine kleine Informationsmenge, die der Server dem Client über einen Header-Eintrag zusteckt. Einmal geschluckt, bleibt es im Browser, der es beim nächsten Zugriff auf die gleiche URL (oder eine andere URL in derselben Domain, je nach Einstellung) wieder mitschickt, ohne daß der Benutzer sich darum kümmern muß, ja, oftmals sogar ohne daß der Benutzer davon weiß. Der Browser muß hierzu Cookies unterstützen, der Netscape-Navigator fing damit an, der Internet-Explorer zog bald nach, heute ist es Standard. Na, beinahe.

Server-Dateien

Paßt die zu speichernde Information nicht mehr in ein oder mehrere *Hidden Fields* oder *Cookies*, kann der Server sich den Zustand einer Session auch in einer Datei (oder auch in einer Datenbank) merken. Anhand eines per *Hidden-Field/Cookie* übermittelten eindeutigen Schlüssels identifiziert er einen einmal dagewesenen Besucher beim nächsten Mal sofort und kann sich auf der Festplatte z.B. alle Waren merken, die in dessen digitalem Einkaufswagen liegen.

Das Skript *cookie.pl* identifiziert Kunden anhand einer ID, die es beim ersten Besuch eindeutig aus der aktuellen Uhrzeit und der Nummer des ausführenden Prozesses - modulo 256 - generiert. Vor der Ausgabe des 'Willkommen!'-Textes setzt es den *Set-Cookie*-Header des übermittelten Dokuments, jubelt so dem Browser das Cookie unter, der es bei allen folgenden Besuchen wieder herausrückt und dem Skript so gedächtnismäßig auf die Sprünge hilft.

Cookies enthalten Name/Value-Paare, im vorgestellten Beispiel steht unter dem Schlüssel *ID* eine eindeutige Hex-Zahl. Die *expire*-Option bestimmt ein Verfallsdatum, nach dessen Ablauf der Browser die Information wieder vergißt. In *cookie.pl* passiert dies mit

Listing 1: cookie.pl

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

use CGI qw/:standard/;

if(defined ($id=cookie(-name => 'ID'))) { # Cookie gesetzt!
   print header();
   print b("Nummer $id! Was für eine Freude!");

} else {                                  # Neuer Kunde
   $id = unpack ('H*', pack('Nc', time, $$ % 0xff));

   $cookie = cookie('-name'    => 'ID',
                    '-value'   => $id,
                    '-expires' => '+1h',
                    '-domain'  => '.gauner.com');
   print header('-cookie' => $cookie);
   print b("Willkommen bei der Cookie-Mafia, Nummer $id!");
}
```

„+*1h*“ nach einer Stunde, andere mögliche Werte wären z. B. „+*1d*“ für einen Tag oder „+*10y*“ für zehn Jahre. Ist noch eine Domain angegeben, wie *.gauner.com* im vorliegenden Beispiel, liefert der Browser das Cookie nicht (nur) an die URL aus, hinter der das Skript hängt, sondern fügt es allen Aufrufen aus der gleichen Domain bei. So kommen unter Umständen mehrere Skripts in den Genuß des Wiedererkennungs-Effekts. Stimmt die Domain hingegen nicht, gibt's auch kein Cookie. Ähnlich funktioniert die in *cookie.pl* nicht verwendete *-path*-Option, die den Browser das Cookie nur an Skripts unter dem eingestellten Pfad (z. B. *cgi-bin/mydir*) übermitteln läßt.
Abbildung 1 zeigt den Browser beim ersten Aufruf von *cookie.pl*, Abbildung 2 bei allen folgenden. Bei gesetzter *-expire*-Option bleibt das Cookie auch über das Programmende des Browsers hinaus in dessen „Magen“ und ist bei einem Neustart sofort wieder präsent.

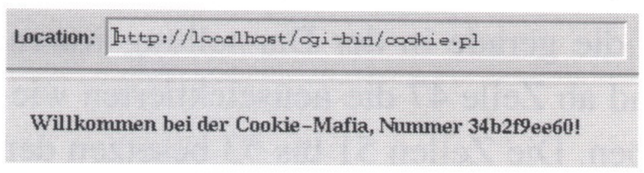

Abb.1: Heute als Fremder ...

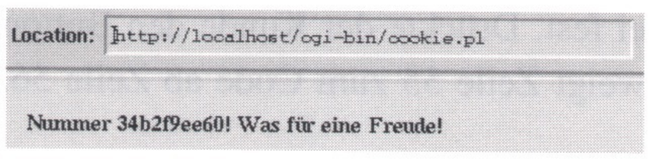

Abb.2: ... nächstens als Freund!

Der Server kriegt Zustände

Erinnert sich noch jeder an die erste Folge dieser CGI-Reihe? Dort war davon die Rede, daß das CGI-Modul normalerweise mit CGI-Objekten arbeitet und man sich mit dem *:standard*-Tag darum drücken kann - aber jetzt trifft's uns: Die nachfolgende *save*-Methode braucht tatsächlich das doofe Objekt.

```
use CGI;
$q = new CGI;
$q->save(STDOUT);
```

Alles klar? Die *save*-Methode schreibt alle hereinkommenden CGI-Parameter im Format *key=value* in Richtung des angegebenen File-Handles, im Beispiel in die Standard-Ausgabe. Ruft man das Skript oben von der Kommandozeile auf und tippt nach dem Prompt

```
(offline mode: enter name=value pairs on
standard input)
```


die Werte (CTRL-D ist „Control“ und „D“)

```
a='Hallo Test'
b=(Klammer)
CTRL-D
```

ein, erscheint

```
a=Hallo%20Test
b=%28Klammer%29
=
```

save schreibt also alle Eingabeparameter weg, wobei es Sonderzeichen sauber nach dem URL-Encoding-Verfahren umsetzt und schließt die Folge mit einer Zeile, die nur = enthält, abschließt. Andersherum erzeugt die Konstruktion

```
$q = new CGI(FILE);
```

ein neues CGI-Objekt, dessen Eingabedaten keineswegs über die CGI-Schnittstelle eintrudeln, sondern aus einer Datei stammen, die mit dem File-Handle *FILE* verbandelt ist. So darf ein Skript durchaus mit mehreren CGI-Objekten jonglieren, die entweder aus dem CGI-Umfeld oder aus der Konserve stammen.
Listing *cart.pl* zeigt die ultimative Anwendung: Ein simples Shopping-Cart, einen virtuellen Einkaufswagen. Erst trägt der potentielle Kunde seinen Namen mit Adresse in ein Formular ein (Abb. 3) und dann bekommt er aus einem Katalog von (für das Beispiel) durchnumerierten Produkten pro Seite jeweils zehn zur Auswahl dargestellt. Er kann einzelne Artikel auswählen, vor- und zurückblättern (Abb. 4), bis er sich schließlich dazu durchringt, zur Kasse zu fahren und die Bestellung abzuschicken (Abb. 5).
Zeile 5 definiert das Verzeichnis für Dateien, in denen er sich jeweils den Zustand der einzelnen Kunden

merkt. Im Beispiel muß das Verzeichnis *customers* unterhalb von *cgi-bin* bereits existieren und für den Eigentümer des Web-Servers beschreibbar sein, sonst kracht's.

$items_total ist die Gesamtzahl der zur Verfügung stehenden Artikel, eine Anzahl *$items_perpage* von Produkten stellt das Skript jeweils auf einer Seite zur Auswahl dar. Für das Testbeispiel generieren die Zeilen 10 bis 12 den Hash *%merchandise*, der jeder Artikel-Nummer 1 bis 100 den beschreibenden Text *Artikel Nummer X* zuordnet.

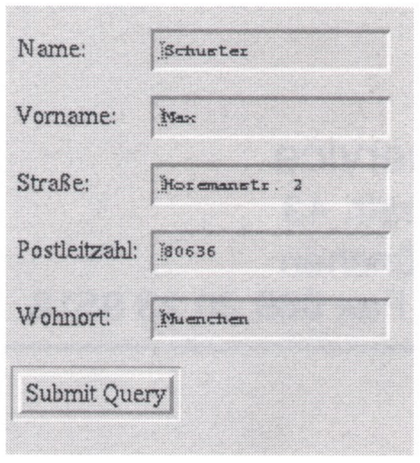

Abb.3: Adresse eintragen ...

Zeile 14 liest ein eventuell schon übermitteltes Cookie ein, für Neukunden ist *$id* demnach nicht gesetzt. In diesem Fall erzeugt *cart.pl* in Zeile 18 eine eindeutige Identifikationsnummer, indem es die aktuelle Uhrzeit in Sekunden und das letzte Byte (*% 256*) der Nummer des laufenden Prozesses (*$$*) als Hex-Zahlen hintereinanderhängt.

Diese ID verpackt Zeile 21 in ein Cookie und sendet es dem Browser, der es mangels gesetztem *-expire*-Datum nur im Rahmen seiner Laufzeit im Gedächtnis behält. Bei einem Neustart ginge der Reigen von vorne los. Die anschließend aufgerufene Funktion *print_address_form* gibt das Eingangsformular (Abb. 3) aus - Ende der ersten Runde.

Füllt der Kunde das Formular aus und klickt auf den *„Submit Query"*-Button, kommt die Logik ab Zeile 28 zum Zug, die, falls noch keine Kundendatei existiert, ein CGI-Objekt erzeugt und eine neue Datei, die den Namen der Kunden-ID trägt, anlegt. War der Kunde schon mal da, existiert die Datei schon und Zeile 31 liest den aktuellen Zustand in das CGI-Objekt *$q* ein. Die Funktionen *save_cgi* und *restore_cgi* sind ab den Zeilen 96 bzw. 104 definiert und stützen sich auf die *save*-Methode und den Konstruktor von *CGI.pm*. Falls in dem übermittelten Cookie nicht die erwartete Hex-Zahl, sondern irgendwelche Schweinereien stehen, bricht Zeile 106 das Skript ab.

Zeile 34 prüft, ob der Kunde auch alle Felder des Adressformulars ausgefüllt hat. Ist dies nicht der Fall, zeigt *chart.pl* wieder das Adressformular, diesmal mit einer roten Fehlermeldung.

Der in Zeile 41 ausgelesene CGI-Parameter *$offset* gibt an, an welcher Stelle des Katalogs der Kunde beim letzten Aufruf schmökerte. Die Zeilen 43 bis 54 korrigieren den Wert des CGI-Parameters *items*, der eine Liste ausgewählter Artikelnummern enthält. Hierzu legt der *grep*-Befehl aus Zeile 43 einen Array *@selected* an, der die Nummern bisher selektierter Artikel kopiert, aber die im vorher dargestellten Fenster ausläßt. Warum? Das Formular aus Abbildung 4 übermittelt im CGI-Parameter *items* nur *gesetzte* Artikel. Hat sich der Kunde umentschieden und einen vorselektierten Eintrag wieder deselektiert, kommt für den entsprechenden Knopf aus dem Formular kein CGI-Parameter herein, folglich entfernt *cart.pl* vorsorglich Werte im *@select*-Array, die gerade in der Darstellung waren, um anschließend ab Zeile 47 die neuselektierten wieder reinzupumpen. Die Zeilen 51 bis 53 besetzen den CGI-Parameter *items*, der die Liste der selektierten Einträge enthält und halten den Zustand der Session in der Server-Datei fest. Drückte der Kunde den Button *Bestellen*, verzweigt Zeile 55 zum Code ab Zeile 56,

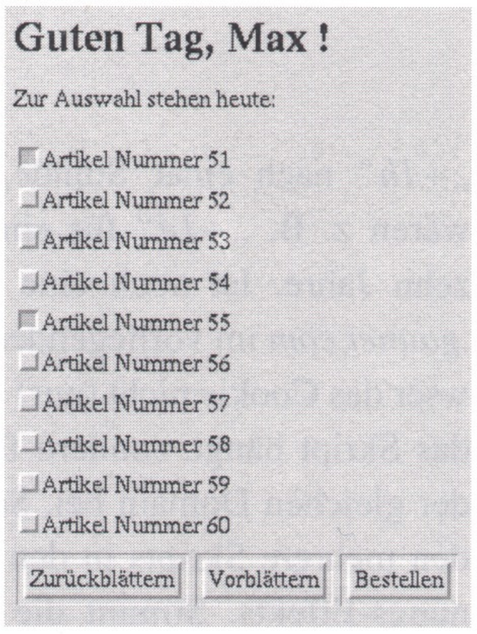

Abb.4: ... Produkte auswählen ...

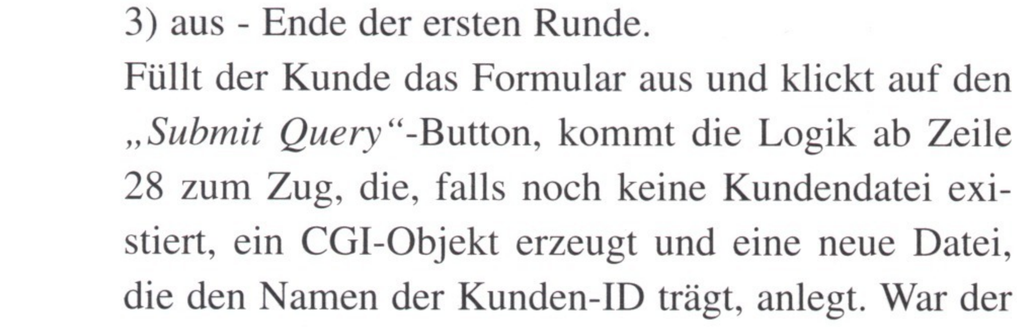

Abb.5: ... und zur Kasse!

Listing 2: cart.pl

```
1  #!/usr/bin/perl -w
2
3  use CGI qw/:standard :html3/;
4
5  $cudir          = „customers";   # Verzeichnis für
6                                     # temporäre Dateien
7  $items_total    = 100;           # Gesamtzahl Artikel
8  $items_perpage = 10;             # Dargestellt pro Seite
9
10 foreach $i (1..$items_total) { # Pseudo-Artikel-Hash
11      $merchandise{$i} = „Artikel Nummer $i";
12 }
13
14 $id = cookie(-name => 'ID');    # Cookie entgegennehmen
15
16 if(!defined $id) {               # Kein Cookie - neuer
Kunde!
17                                   # Neue ID erzeugen
18      $id = unpack ('H*', pack('Nc', time, $$ % 0xff));
19
20                                   # Adressformular
schicken
21      print header('-cookie' => cookie('ID' => $id));
22      print_address_form();
23      exit 0;
24 }
25
26 print header, start_html;
27
28 if(! -f „$cudir/$id") {
29      save_cgi($q = new CGI);     # Neue Kundendatei anlegen
30 } else {
31      $q = restore_cgi();         # Kundendatei einlesen
32 }
33                                   # Adressinformation kom-
plett?
34 if(!$q->param('name') || !$q->param('vorname') ||
35     !$q->param('strasse') || !$q->param('plz') ||
36     !$q->param('wohnort')) {
```

Listing 2: cart.pl

```
37      print_address_form(„Bitte füllen Sie alle Felder
aus.");
38      exit 0;
39 }
40
41 $offset = ($q->param('offset') || 0);
42
43 @selected = grep { $_ <= $offset ||
44                    $_ > $offset+$items_perpage }
45                ($q->param('items'));
46
47 foreach $item (param('newitems')) {
48     push(@selected, $item);
49 }
50
51 $q->delete('items');          # CGI-Parameter zurücksetzen
52 $q->param('items', @selected);
53 save_cgi($q);                 # Neu gesetzt und gesichert
54
55 if(param('Bestellen')) {     # Ausgang
56     print $q->param('vorname'), „ „, $q->param('name'),
br,
57           $q->param('strasse'), br,
58           $q->param('plz'), „ „, $q->param('wohnort'),
59           br, br, h1(„Bestellung");
60     foreach $item ($q->param('items')) {
61         print „1 Stueck $merchandise{$item}", br;
62     }
63     print p, b(„Ihre Bestellung ist schon unterwegs!"),
end_html;
64
65 } else {                      # Warenliste anzeigen
66
67     if($offset >= $items_perpage) {
68         $offset -= $items_perpage if
param(„Zurückblättern");
69     }
70     if($offset < $items_total - $items_perpage) {
71         $offset += $items_perpage if
param(„Vorblättern");
72     }
73
74     $q->param('offset', $offset);
75     save_cgi($q);
76
77     @subset = sort {$a <=> $b} keys %merchandise;
78     @subset = splice(@subset, $offset, $items_perpage);
79
80                              # Neue Warenliste
81     print h1(„Guten Tag, „, $q->param('vorname'), „!"),
82           p, „Zur Auswahl stehen heute:",
83           start_form(),
84           $q->checkbox_group(
85               '-name'       => 'newitems',
86               '-values'     => [@subset],
87               '-default'    => [$q->param('items')],
88               '-linebreak'  => 'true',
89               '-labels'     => \%merchandise),
90           submit('Zurückblättern'),
submit('Vorblättern'),
91           submit('Bestellen'),
92           end_form, end_html;
93 }
94
95
############################################################
96 sub save_cgi {
97
############################################################
98     open(FILE, „>$cudir/$id") || die „Can't open
$cudir/$id";
99     $_[0]->save(FILE);
100    close(FILE);
101 }
102
103
############################################################
104 sub restore_cgi {
105
############################################################
106     die „Get off, hacker!" if $id !~ /^[0-9a-f]+$/;
107     open(FILE, „<$cudir/$id") || die „Can't open
$cudir/$id";
108     my $q = new CGI(FILE);
109     close(FILE);
110     return $q;
111 }
112
113
############################################################
114 sub print_address_form {
115
############################################################
```

Listing 2: cart.pl

```
116     my $msg = (shift || „");
117     print start_html,
118           tt(CGI::font({color => 'red'}, $msg)),
119           start_form,
120           table(
121             TR(td(„Name:"), td(textfield('name'))),
122             TR(td(„Vorname:"), td(textfield('vorna-
me'))),
123             TR(td(„Straße:"), td(textfield('strasse'))),
124             TR(td(„Postleitzahl:"),
td(textfield('plz'))),
125             TR(td(„Wohnort:"), td(textfield('wohn-
ort')))),
126           submit, end_form, end_html;
127 }
```

der die Bestellungs-Bestätigung ausdruckt. Hier müsste ein reales Order-System dann den Auftrag in die Wege leiten, z.B. eine Email an einen Bearbeiter schicken und nach getaner Arbeit die Zustandsdatei auf dem Server löschen.

Falls der Kunde *Vorblättern* oder *Zurückblättern* wählte, verschrauben die Zeilen 68 und 71 den Wert der *$offset*-Variablen um eine Seitenlänge, und 74-75 speichern den neuen Wert in der Zustandsdatei.

Die Zeilen 77 und 78 legen in den Array *@subset* die Nummern der im aktuellen Durchgang zur Auswahl gestellten Artikel ab, indem sie den *%merchandise*-Hash nach numerischen Schlüsseln sortieren und mit *splice* ein Segment herausschneiden.

Die Zeilen 81 bis 92 zeichnen für die Ausgabe des Auswahlformulars verantwortlich.

Der zentrale *checkbox_group*-Befehl erzeugt den HTML-Code für die aufgereihten Knöppe, die *default*-Option selektiert die irgendwann vorher ausgewählten praktischerweise gleich vor.

Zu beachten ist die unterschiedliche Notation der Funktionen aus dem CGI-Modul: Da *$q* vorher mit dem eingefrorenen Zustand aus einer Dateien-Konserve initialisiert wurde, holt *$q->param()* Werte aus dem in der Serverdatei gesicherten CGI-Objekt, während *param()* (ohne Objekt) sich auf die aktuell hereinkommenden Parameter bezieht.

Nächstes Mal gibt's als Abschluß *Non-parsed-Header*-Skripts — ohne Netz und doppelten Boden. Bleibt am Ball!

DER AUTOR

Michael Schilli arbeitet als Web-Engineer für AOL Productions, San Mateo. Er ist Autor von *„Effektives Programmieren mit Perl 5"* und unter *mschilli1@aol.com* oder *http://mission.base.com/mschilli* zu erreichen.

Autorenliste

Oliver Bildesheim
Mathias Brandstetter
Falko Bräutigam
Marco Budde
Kalle Dalheimer
Dr. R. Hermans
Karsten Kankowski
Bernhard Kuhn
Michael Lupp
Joachim Schaaf
Michael Schilli
Tom Schwaller
Reginald Stadlbauer
Thomas Trüten
Mark Vogelsberger
Peter Wächtler

Service für DeLUG-Mitglieder

Aktuelle Software

Krypto-Info:

DFN-PCA Schlüssel: 2048/35DBF565 1997/04/16 DFN-PCA,
CERTIFICATION ONLY KEY (Low-Level: 1997-1998) <not-for-mail>
Key fingerprint = 09 7C 09 19 D3 C3 86 DC 7A 30 15 11 12 95 8D E3
IN-Root-CA Schlüssel: 2048/19980101 SIGN EXPIRE:1999-12-31
Key fingerprint = B3 06 9A 8D 38 04 3C 75 41 32 EE DC 8B 7D 61 0D
http://www.in-ca.individual.net/
Hashwert des Ewigen Logfiles vom 02.04.1998 14:46:18
sha1=db31b22f3317f25ed38d025b91c46e3a3f97f8b3
https://www.iks-jena.de/mitarb/lutz/logfile/

Inserentenverzeichnis

IMPRESSUM

Informationen, Berichte und Neuigkeiten aus der Linux-Welt

Herausgeber: Rudolf Strobl
Redaktion:
Email: redaktion@linux-magazin.de
Post: **Linux - Magazin Verlag**
Stefan-George-Ring 23
81929 München
Tel.: 089/99315-310
Fax: 089/99315-329
WWW: http://www.linux-magazin.de
ISSN: 1432 - 640 X
Chefredakteur: Tom Schwaller,
tom.schwaller@linux-magazin.de (ts)
Coverdesign: Tilo Wondollek, TW-Design
Layout : C.&D. Elsäßer
Mitarbeiter: siehe Autorenliste
Vertriebsmarketing: info@linux-magazin.de
Anzeigen: Agentur Neumann
Kalvarienbergstraße 31
Postfach 1352
87503 Immenstadt
Tel: 08323 / 98100
Fax: 08323 / 963226
Email: ThomasNeumann@t-online.de

Anzeigenpreise: Es gilt die Anzeigenpreisliste vom 01.12.1997

Das Linux - Magazin erscheint monatlich.
Einzelheft: **DM 9,80**
Preis des Jahresabonnements:
Incl. DeLUG-Mitgliedschaft **DM 180,-**
Ohne Mitgliedschaft **DM 103,-**
(Ausland: 115,-)

Für Schüler und Studenten 20% Ermäßigung (Vorlage Schüler- oder Studentenausweis)