Perl-Praxis

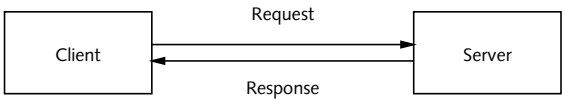
CGI-Skripte

Jörn Clausen
joern@TechFak.Uni-Bielefeld.DE

Übersicht

- WWW, Web-Server
- CGI-Skripte
- Parameterübergabe
- Web-Formulare
- `CGI.pm`

Datenaustausch im WWW



- Adressierung durch *Uniform Resource Locator* (URL)
- Anfrage und Antwort werden per HTTP transportiert
- HTTP-Nachricht besteht aus *header* und *body*
- body kann leer sein
- Antwort kann dynamisch generiert werden

ein kleiner Web-Server

- Übungsverzeichnis verwenden, **NICHT DAS HOMEDIRECTORY**
- Web-Server starten:

```
$ /vol/lehre/PerlPraxis/mini_httpd -C config -p 8088
```

- andere Port-Nummer wählen: `-p ...`
- Web-Browser starten
- `http://vino.TechFak.Uni-Bielefeld.DE:8088/`
- korrekten Rechner (vino/antipasto), korrekte Port-Nummer
- Datei `hello.html` auswählen

CGI-Skripte

- informeller Standard: *Common Gateway Interface*
- Web-Server führt externes Programm aus
- Ausgabe des Programms wird als Antwort zurückgeschickt
- enormes Gefahrenpotential:
  - Ausführung von (fehlerhaftem?) Code
  - Ausführung von beliebigem Code
- eigenen Code sorgfältig prüfen
- **Keine schnellen Hacks!!**

CGI-Skripte, cont.

- Aufbau einer Antwort: *header*, Leerzeile, *body*

```
Content-Type: text/html

<html>
  <head>
...
```

- Perl-Skript `helloworld.cgi`:

```
#!/vol/perl-5.8/bin/perl
print "Content-Type: text/plain\n\n";
print "hello world!\n";
```

- `http://vino.TechFak....:8088/helloworld.cgi`
- Perl-Interpreter direkt angeben, nicht mit `/usr/bin/env`

Aufgaben

- Schreibe ein CGI-Skript, das die aktuelle Uhrzeit zurückliefert. Zur Erinnerung:

```
($sec, $min, $hour) = localtime(time());
```

- aktuelle Uhrzeit:

```
#!/vol/perl-5.8/bin/perl

my ($sec, $min, $hour) = localtime(time());

print "Content-Type: text/plain\n\n";
print "It is $hour:$min:$sec\n";
```

redirects

- Skript verweist auf andere Ressource
- neuer URL wird im header zurückgeliefert

```
print "Location: http://www.Uni-Bielefeld.DE\n\n";
```

- body kann leer sein
- redirect verfolgen:

```
$ GET -S http://vino.TechFak....:8088/redirect.cgi | more
GET http://vino.TechFak....:8088/redirect.cgi --> 302 Found
GET http://www.Uni-Bielefeld.DE --> 200 OK
<!DOCTYPE html PUBLIC ...
```

- 302 und 200 Status Codes von HTTP

Der body kann zwar leer sein, muß aber vorhanden sein. Beachte die Leerzeile nach dem header.

Parameterübergabe

- Anfrage kann Daten enthalten (Google, Amazon, Wikipedia, ...)
- Schlüssel-Wert-Paare
- Datentransfer vom Client zum Web-Server:
  - `GET`: Parameter werden im URL kodiert

    `...google.de/search?q=boggit&ie=UTF-8&oe=UTF-8&...`
  - `POST`: Daten werden im body transportiert
- Datentransfer vom Web-Server zum CGI-Skript:
  - Environment
  - `STDIN`

Einschub: Environment

- Environment-Variablen
- mit Shell-Kommando env ansehen
- Zugriff in Perl: Hash %ENV

```
print "User: $ENV{USER}\n";
print "Suchpfad: $ENV{PATH}\n";
```

- schreibender Zugriff möglich
- Environment an Prozess gebunden

Aufgaben

- Schreibe ein CGI-Skript `showenv.cgi`, das das gesamte Environment des CGI-Prozesses anzeigt. Wie sieht das Environment aus, wenn Du `showenv.cgi` auf der Kommandozeile aufrufst?
- Hänge an den URL Parameter an:

  ```
  .../showenv.cgi?query=boggit
  ```

  Wie verändert sich das Environment?
- Gib mehrere Parameter an. Wie könntest Du diese Parameterliste mit Perl weiterverarbeiten?

- `showenv.cgi`:

```
print "Content-Type: text/plain\n\n";
foreach $var (sort(keys(%ENV))) { print "$var: $ENV{$var}\n"; }
```

Environment (gekürzt):

```
REMOTE_ADDR: 129.70.128.82
LD_LIBRARY_PATH: /usr/local/lib:/usr/lib
REQUEST_METHOD: GET
PATH: /usr/local/bin:/usr/ucb:/bin:/usr/bin
GATEWAY_INTERFACE: CGI/1.1
SERVER_SOFTWARE: mini_httpd/1.19 19dec2003
HTTP_HOST: teak.techfak.uni-bielefeld.de:8088
HTTP_USER_AGENT: Mozilla/5.0 (X11; U; SunOS sun4u; en-US; rv:1.6) Gecko/20040121
```

Das Environment der Shell hat mehr und längere Einträge. Vor allem ist der Suchpfad PATH deutlich verkürzt.

- Durch den angehängten Parameter wird eine weitere Environment-Variable erzeugt:

```
QUERY_STRING: query=boggit
```

- Die Liste der Parameter wird direkt in `QUERY_STRING` übernommen. Sie läßt sich mit Hilfe von `split('&', ...)` und `split('=', ...)` leicht weiterverarbeiten.

Web-Formulare

- Öffne die Datei `form.html` mit dem Web-Browser

```
<form action="showenv.cgi" method="GET">
  Anfrage: <input type="TEXT" name="query">
  <br>
  <input type="SUBMIT">
  <input type="RESET">
</form>
```

- Daten werden an `showenv.cgi` übergeben
- Methode `GET`, d.h. Parameter erscheinen im URL
- Eingabe testen: „Ernie & Bert“
- ersetze in `form.html` Methode `GET` durch `POST`

Die Variable `QUERY_STRING` ist verschwunden.

```
CONTENT_LENGTH: 10
REQUEST_METHOD: POST
CONTENT_TYPE: application/x-www-form-urlencoded
```

- Methode `POST`:

```
QUERY_STRING: query=Ernie+%26+Bert
```

- “Ernie & Bert”:

```
SCRIPT_NAME: /showenv.cgi
HTTP_REFERER: http://teak.techfak.uni-bielefeld.de:8088/form.html
REQUEST_METHOD: GET
QUERY_STRING: query=test
```

- Methode `GET` (gekürzt):

Auswertung von Web-Formularen

- immer wiederkehrende Aufgaben/Probleme:
  - Daten können auf verschiedene Arten übertragen werden
  - spezielle Zeichen müssen (de)kodiert werden
  - Antwort muß korrekten header und body enthalten
  - Eingaben müssen „vorsichtig“ ausgewertet werden
- universeller Helfer: `CGI.pm`

CGI.pm verwenden

- nochmal aktuelle Uhrzeit:

```
use CGI;

my ($sec, $min, $hour) = localtime(time());
my $query=CGI->new;
print $query->header("text/plain");
print "It is $hour:$min:$sec\n";
```

- weitere Informationen im header:

```
print $query->header(-type => "text/html",
                     -expires => "+3h");
```

- redirect:

```
print $query->redirect("http://www.Uni-Bielefeld.DE");
```

Aufgaben

- Schreibe mit Hilfe von `CGI.pm` ein CGI-Skript, das per Zufall einen redirect auf eine der folgenden Webseiten erzeugt:

  http://www.google.de/
  http://elgoog.rb-hosting.de/
  http://www.blug.linux.no/rfc1149/

- zufälliger redirect:

```
use CGI;

my @urls = ("http://www.google.de/",
            "http://elgoog.rb-hosting.de/",
            "http://www.blug.linux.no/rfc1149/");
my $rand = int(rand(@urls));

my $query=CGI->new;
print $query->redirect($urls[$rand]);
```

HTML mit `CGI.pm` erzeugen

- korrektes HTML, vor allem korrekte Tag-Klammerung:

```
use CGI ":standard";

print header("text/html"),
      start_html("Perl-Praxis"),
      h1("HTML mit CGI.pm"),
      p("CGI, das Common Gateway Interface..."),
      end_html;
```

- Import von Symbolnamen bei `use`
- prozedurale Verwendung von `CGI.pm`

Aufgaben

- Ändere das Skript zur Anzeige der aktuellen Uhrzeit so ab, daß mit Hilfe von `CGI.pm` HTML-Code erzeugt wird.
- Sieh Dir mit

  ```
  $ GET -e http://vino.TechFak....:8088/clock.cgi
  ```

  den header der Antwort an.
- Sorge dafür, daß die Antwort 10 Sekunden lang im Cache gehalten werden darf. Wie ändert sich der header?

  Was passiert, wenn Du im *Location Bar* des Browsers „Return“ drückst? Was passiert, wenn Du die „Reload“-Funktion des Browsers verwendest?

Formulare mit `CGI.pm` erzeugen

- weitere Eingabemöglichkeiten (`formcgi.cgi`):

```
print start_form,
      textfield(-name => "name",
                -default => "Joe User"),
      popup_menu(-name => "age",
                 -values => ["-17","18-25","25-35","36+"],
                 -default => "25-35"),
      scrolling_list(-name => "languages",
                     -values => ["Perl","Java","Haskell"],
                     -default => ["Java","Haskell"],
                     -multiple => "true"),
      br, submit, reset,
      end_form;
```

- Was passiert beim „submit“? Sieh Dir den (HTML-)Quelltext der Seite an.

- Beim submit wird das Formular erneut geladen. Die eingetragenen bzw. ausgewählten Werte bleiben aber erhalten. Im Quelltext sieht man, daß diese als default-Werte eingetragen sind:

```
<input type="text" name="name" value="Joe User" />
<select name="age">
  <option value="-17">-17</option>
  <option value="18-25">18-25</option>
  <option selected="selected" value="25-35">25-35</option>
  <option value="36+">36+</option>
</select>
<select name="languages" size="3" multiple="multiple">
  <option value="Perl">Perl</option>
  <option selected="selected" value="Java">Java</option>
  <option selected="selected" value="Haskell">Haskell</option>
</select>
```

Parameterübergabe

- Skript `showparams.cgi`:

```
my $query=CGI->new;
print $query->header("text/plain");
foreach $name ($query->param) {
    my $val = $query->param($name);
    print "$name: $val\n";
}
```

- probiere URL

```
.../showparams.cgi?query=boggit&answers=10
```

- trage `showparams.cgi` als `action` in `form.html` ein
- probiere `GET` und `POST` als `method`

iterative CGI-Skripte

- häufig Dialog von Fragen und Antworten
- komplexere Formulare, Shop-Systeme
- Zustand muß gespeichert werden
- ein CGI-Skript mit versteckten Status-Informationen
- leicht zu überlisten
- andere (bessere?) Verfahren: Cookies, Session-ID, ...

iterative CGI-Skripte, cont.

- Mail-Order-Animals: `animal.cgi`

```
my $animal = $query->param("animal");
my $color = $query->param("color");

if (!($animal and $color)) { print start_form;
    if (!$animal) {
        print "animal: ", textfield(-name => "animal",
                                    -default => "tiger");
    } elsif (!$color) {
        print "animal: $animal",br,
              "color: ", textfield(-name => "color",
                                   -default => "yellow"),
               hidden(-name => "animal");
    }
    print end_form; } else {
    print p("So you want a $color $animal");
}
```

Das CGI-Skript ruft sich jedesmal selber auf. Falls bereits Daten eingegeben wurden, werden diese in versteckten Formular-Feldern transportiert. Nachdem die Art angegeben wurde, sieht der HTML-Quelltext folgendermaßen aus:

```
<form method="post" action="/animal.cgi"
    enctype="application/x-www-form-urlencoded">
  animal: gnu<br />
  color: <input type="text" name="color" value="yellow" />
  <input type="hidden" name="animal" value="gnu" />
</form>
```

Aufgaben

- Schreibe ein CGI-Skript, das folgende Informationen abfragt:

Name	Textfeld	
Alter	Textfeld	optional
EMail	Textfeld	
Studiengang	NWI / MGS / BIG	optional

  Wiederhole die Eingabe solange, bis alle verpflichtenden Angaben (Name und EMail-Adresse) eingegeben wurden.

- Implementiere weitere Tests:
  - Das Alter muß eine Zahl zwischen 0 und 120 sein.
  - Die EMail-Adresse muß „glaubhaft“ aussehen, also z.B. ein at-Zeichen @ enthalten.

```
                                                    }
                                          print end_p;
                                  print " Du studierst $stud." if $stud;
                              print " Du bist $age Jahre alt." if $age;
                                    print "Hallo $name ($email).";
                                              print start_p;
                                                  } else {
                                          print end_form;
                                    print br,submit,reset;
                            print $query->hidden("reload", 1);
                                                -default => "");
'                                   -values => ["", "NWI", "MGS", "BIG"],
                        print br, "Studiengang: ", popup_menu(-name => "stud",
     print strong("Bitte EMail-Adresse eintragen!") if !$email and $reload;
                              print br, "EMail: ", textfield(-name => "email");
                                print br, "Alter: ", textfield(-name => "age");
        print strong("Bitte den Namen eintragen!") if !$name and $reload;
                                  print "Name: ",textfield(-name => "name");
                                          print start_form;
                                        unless ($name and $email) {

                                  my $reload = $query->param("reload");
                                      my $stud = $query->param("stud");
                                    my $email = $query->param("email");
                                      my $age = $query->param("age");
                                    my $name = $query->param("name");
```

- ein ganz einfacher Fragebogen: