Html-Formulare

Michael Dienert

19. Februar 2019

1 Das http-Protokoll

1.1 Ursprünglicher GET-Request

Ursprünglich war die Idee von http, mit einem sog. *GET-Request* eine Datei im html-Format von einem entfernten Server auf den eigenen Rechner zu übertragen und dort in einem Browser anzuzeigen. Die Übertragung findet dabei nach wie vor mit tcp statt.
Ein typischer GET-Request sieht z.B. so aus:

```
GET /index.html HTTP/1.1 CRLF
Host: www.heise.de CRLF CRLF
```

Mit dieser Anfrage wird die Datei `index.html` relativ zum Verzeichnis `DocumentRoot` auf dem Host `www.heise.de` in den eigenen Browser übertragen.
Welches Verzeichnis `DocumentRoot` auf dem Server ist, wird auf der Serverseite konfiguriert und spielt für den Client-Rechner keine Rolle.

1.2 Common Gateway Interface

Mit GET konnte man zunächst nur Dateien vom Server zum User-Agent (aka Browser) auf dem Client-Rechner übertragen.
Damit man aber auch Daten vom User-Agent zum Server **hoch**laden kann, wurde das
Common Gateway Interface - CGI
eingeführt: Die Daten, die man zum Server hochladen möchte, werden einfach an die URL des GET-Requests angehängt.
Beispiel 1:

```
GET /delete?item=hosen HTTP/1.1 CRLF
Host: localhost:8000 CRLF CRLF
```

Beispiel 2:

```
GET /create?item=hosen&price=44.33 HTTP/1.1 CRLF
Host: localhost:8000 CRLF CRLF
```

Beispiel 3:

```
GET /create?item=leber+wurst&price=4.00 HTTP/1.1 CRLF
Host: localhost:8000 CRLF CRLF
```

Beispiel 4:

```
GET /create?item=leber%2Bwurst&price=4.00 HTTP/1.1 CRLF
Host: localhost:8000 CRLF CRLF
```

Als Trennzeichen werden dabei benutzt:

? : Das Fragezeichen steht zwischen der angefragten Datei und dem sog. `QUERY_STRING`

= : In Beispiel 1 besteht der `QUERY_STRING` aus einem Parameter=Werte-Paar

+ : sind irgendwo Leerzeichen, werden diese durch ein +-Zeichen oder mit `%20` codiert.

%FF : Sonderzeichen werden mit ihrem ASCII/UTF8-Zeichencode mit vorangestelltem %-Zeichen übertragen

Die Länge des QUERY_STRINGS ist begrenzt. Die Maximale Zeichenzahl hängt vom verwendeten User-Agent ab.

1.3 Umgebungsvariablen bei CGI

Kommt nun so ein GET-Request mit einem angehängten QUERY_STRING auf der Serverseite an, wird der QUERY_STRING vom Webserver in einer *Umgebungsvariablen* gespeichert.

Diese Variable heisst sinnigerweise QUERY_STRING und kann von einem Programm in beliebiger Weise ausgewertet werden.

1.4 Methode POST

Neben der Methode GET, erlaubt http auch eine Methode POST. Wird eine Seite mit POST angefragt, wird der QUERY_STRING nicht an die URL angehängt, sondern im sog. Message-Body des Requests übertragen. Der Message Body folgt nach einer Leerzeile nach dem Header (deshalb oben CRLF CRLF).

Da Header und Message-Body mit tcp übertragen werden, kann der QUERY_STRING theoretisch bis zu 4GB lang werden.

Vom http-Server werden die Daten des QUERY_STRINGS in die Standardeingabedatei geschrieben. Programme, die den QUERY_STRING verarbeiten, lesen ihn aus der Standardeingabe aus.

2 html-Formulare

Damit man den QUERY_STRING nicht mit der Hand am Arm an die URL anhängen muss, wurden html-Formulare eingeführt. Ein html-Formular erzeugt im Browser-Fenster Textfelder und Buttons, mit denen man die Inhalte der Textfelder absenden kann.

3 Fragen zu html-Formularen

1. Welches html-Element erzeugt ein Formular?
2. Welche html-Elemente können Kinder eines Formular-Elements sein?
3. Welche Attribute kann das Formular-Element besitzen und was bewirken diese auf der Serverseite?
4. Welche Attribute kann das Element `input` besitzen und welche Funktion haben diese?

4 html-Formulare und go

Eine go-http-Handler-Funktion sieht z.B. so aus:

```
func foo(w http.ResponseWriter, r *http.Request) {

  //....

  t, _ := template.ParseFiles("form.html")
  t.Execute(w, nil)
}
```

4.1 Testseite erstellen

Erstelle die Datei `form.html` mit 5 Formular-Textfeldern und einem Absende-Button.
Von den 5 Textfeldern sollen 3 den selben Namen besitzen.
Notiere den QUERY_STRING, der sich beim Abschicken der Formulare ergibt.

4.2 Formulardaten mit go auswerten

1. Wann benötige ich den Aufruf `r.ParseForm()`?
2. Welche Werte liefert `r.Method`?
3. Welchen Datentyp hat `r.Form`?
4. Was passiert, wenn der QUERY_STRING viele Paare mit gleichem Key-Namen enthält? Wie kann ich auf die Werte zugreifen?
5. Wie kann ich die Methode `r.FormValue("<...>")` verwenden? Benötige ich hier auch `r.ParseForm()`?
6. Erweitere die CRUD-Aufrufe auf die Datenbank des Mikro-Shops um Formulare zur Eingabe der Daten. Verwende dabei Templates.