Schleifen in Javascript

Jobst-Hartmut Lüddecke

3. April 2013

Zusammenfassung

In dieser Lektion geht es um *Schleifen* (engl. *loop*). Diese *Schleifen* sind in jeder Programmiersprache das beste Werkzeug für *Wiederholungen*. Diese *Wiederholungen* finden entweder für eine *bestimmte Anzahl* von Durchläufen statt, oder solange eine *Bedingung erfüllt* ist. Dabei gibt es noch die Variationen, dass die Bedingung *vor* oder *nach* dem Schleifendurchlauf geprüft wird.

Die Beispiele wurde für HTML5 angepasst.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Struktogramm

Ein *Struktogramm* – auch *Nassi-Schneidermann-Diagramm* – ist eine Form der grafischen Darstellung der Struktur eines Programms. Gleiches tut ein *Flußdiagramm*. Weiterhin gebräuchlich sind *Hierarchiediagramme*, *Automaten* und *Petrinetze* um die Ebenen der Funktionsaufrufe und gewisse *Übergangsfunktionen*[1] darstellen zu können. All dies füllt ganze Regalmeter mit Büchern, ist aber zwingendes *Werkzeug* für eine ernsthafte Software-Entwicklung. Die Verbindung zu diesen Möglichkeiten der Darstellung sind dann die Kommentare im Programm selbst.

An dieser Stelle nun ein *Struktogramm*. Ein *Flußdiagramm* haben Sie bereits an anderer Stelle kennen gelernt. Warum nun jetzt ein *Struktogramm*?

Ganz einfach, mit einem *Struktogramm* kann man keine Struktur von *Spaghetti-Programmen* darstellen und da Sie die Struktur *vorher* festlegen sollen, können sie damit auch keine *Spaghetti-Programme* programmieren. Dies ist mit einem *Flußdiagramm* eines der leichtesten Übungen. Jetzt will ich Ihnen aber nicht zeigen wie man es *nicht* machen soll, sondern ein *Werkzeug* an die Hand geben, mit dem man saubere Strukturen festlegen kann. Hat man so eine Struktur erst einmal festgelegt, ist es völlig egal in welcher Programmiersprache jetzt diese Struktur umgesetzt und programmiert wird.

Dazu müssen Sie aber eine gewisse *Abstraktionsebene* einhalten und keine direkten Befehle einer bestimmten Programmiersprache verwenden. Gerade dies fällt Anfängern besonders schwer.

Sie werden jetzt Stück für Stück mit einigen Darstellungsmöglichkeiten so eines Struktogramms konfrontiert. Die Abarbeitung der Struktur geschieht von Oben nach Unten. Es ist also ein *Top-Down* Design. Wo es *Top-Down* gibt, gibt es an anderer Stelle noch ein *Bottom-Up*.

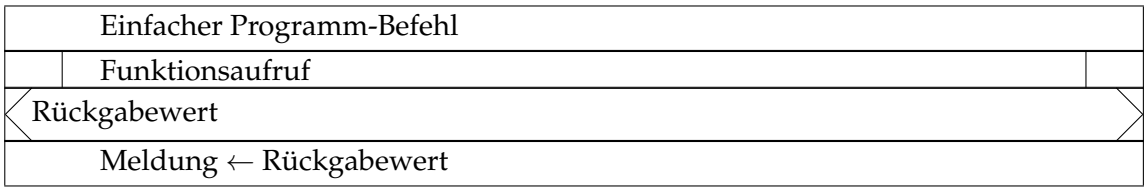

Abbildung 1: Struktogramm mit einfachen Anweisungen

Obiges Beispiel enthält nun die ersten 4 Konstrukte:

- Die erste *Schachtel* ist ein einfacher Befehl.
- Die zweite Schachtel ist der Aufruf einer anderen *Funktion*. In anderen Programmiersprachen gibt es noch *Unterprogramme* und *Prozeduren*. Die *Schachtel* sieht gleich aus.
- Die überwiegende Zahl der *Funktionen* liefert etwas zurück, was weiter verarbeitet werden soll. Die entsprechende Darstellung dafür ist die dritte *Schachtel*. Erwartet man nichts zurück, kann man diese Schachtel weglassen.
- Die vierte Schachtel ist dann wieder ein einfacher Befehl. In diesem Fall eine Wertzuweisung.

2 Pre-checked Loops

Dies sind Schleifen, bei denen die Bedingung für den Durchlauf der Schleife **vor** dem Durchlauf der Schleife geprüft wird. Ist die Bedingung nicht erfüllt, wird die Schleife erst gar nicht durchlaufen.

[1] Wichtig wenn man selbst Betriebssysteme oder Compiler programmiert

2.1 For-Schleife

Das Struktogramm einer *Pre-checked loop*, und damit auch einer *For-Schleife* hat auf der linken Seite einen *Haken*, der alle Befehle, die innerhalb der Schleife liegen umfasst. Diese Befehle können dann wiederum alle gültigen Konstrukte sein und aus mehreren Anweisungen bestehen.

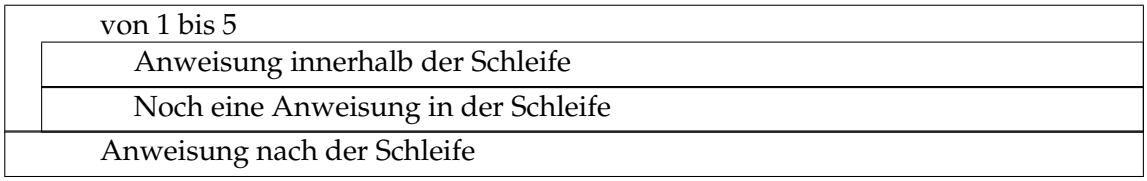

Abbildung 2: Struktogramm einer einfachen *for*-Schleife

Bei der *For*-Schleife wird eine vorher *deklarierte*[2] *Zählvariable* (z.B. *i*) initialisiert[3], eine Bedingung definiert und die Zählvariable inkrementiert[4], oder dekrementiert[5]. Die Schleife wird solange durchlaufen, solange die Bedingung *wahr*, bzw. der boolsche Wert des Vergleiches *true* ergibt.

```
var i;
for ( i=1; i<5; i++){
    <Anweisungen innerhalb der Schleife>
    }
<Anweisung nach der Schleife>
```

Abbildung 3: Prinzip der *for-Schleife*

Eine *for-Schleife* gibt es in jeder mir bekannten Programmiersprache, allerdings können die Parameter und Ausdrücke anders aussehen, machen aber immer das Gleiche. In **Javascript** besteht der *Parameter* der *for-Schleife* immer aus **3 Teilen** und stehen in **runden Klammern** und zwar aus:

- Der Zählvariablen mit dem Startwert.
- Der Endbedingung.
- Dem Zählausdruck (rauf oder runter).

Beispiel: Der Anfangswert der Zählvariablen *i* wird auf den Wert *1* gesetzt. *1* ist kleiner als *5*, die Bedingung ist also *true* und die Schleife wird durchlaufen und die Zählvariable *i* mit *i*++[6] inkrementiert. Jetzt ist der Wert der Zählvariablen *2*, die Bedingung ist erfüllt und der nächste Durchlauf wird gemacht usw.

Sobald die Zählvariable *i* den Wert *5* erreicht hat, ist die Bedingung *false*, denn *5* ist *nicht kleiner* als *5*, sondern *gleich* und die Schleife wird nicht mehr durchlaufen, sondern mit der Anweisung nach der Schleife das Programm fortgesetzt.

Achtung: Alles zwischen den **geschweiften Klammern nach** der Schleifendefinition in **runden Klammern** steht, wird in der Schleife wiederholt.

Im *Struktogramm* entspricht dies dem Teil, der vom linken Haken umfasst wird.

[2] siehe letztes Skript. In diesem Beispiel: *var i;*

[3] auf einen Anfangswert gesetzt

[4] bei der Zählvariablen z.B. mit dem Namen *i* wird mit Ausdruck *i*++ der Wert der Variablen *i* um 1 erhöht. Der Ausdruck *i*++ entspricht einem *i=i+1* und schon sind wir bei den Schreibweisen für Schreibfaule.

[5] den Wert z.B. mit *i* - - um 1 verkleinert. Das *i* - - entspricht also einem *i=i-1*

[6] eine der *speziellen* Notationen

Der Begriff <*Anweisungen innerhalb der Schleife*> ist wieder ein *Dummy*. Es ist *kein* Javascript-Befehl, sondern ein *Platzhalter* für ein *Programmsegment* das in der *Schleife* wiederholt abgearbeitet werden soll. In der Praxis besteht so ein mehrfach abgearbeitetes *Programmsegment* aus einigen *Anweisungen*, oder kann auch eine *Funktion* sein.

Frage: Wie viele Schleifendurchläufe gibt es bei diesem Beispiel?

```
<!DOCTYPE html>
<html lang="de" xml:lang="de">
<head>
<title>for-Schleife mit Javascript</title>
<script language="javascript" src="zaehler.js">
</script>
</head>
<body onload=zaehler()>

</body>
</html>
```

Abbildung 4: HTML5-Seite, die die Funktion *zaehler()* im Javascript *zaehler.js* mit einer Zählschleife mit *for* und direkter Ausgabe der Werte aufruft.

```
function zaehler(){
    var i=0;
    var schluss=10;
    document.write("hier geht es los: ");
    document.write("<br>");

    for (i=1;i<=schluss;i++){
        document.write(i);
        document.write(",");
    }
}
```

Abbildung 5: Javascript zaehler.js mit Zählschleife mit *for* und direkter Ausgabe der Werte.

2.2 While-Schleife

Die *While*-Schleife wird solange durchlaufen, solange die *Bedingung* erfüllt, also *true* ist. Dabei kann die *Bedingung* eine *boolsche Variable*, ein *logischer Ausdruck*, oder der *boolsche Rückgabewert* einer Funktion sein.

Diese Art der Schleife ist einfach zu programmieren, will aber genau bedacht werden. Leicht ist es möglich, dass die Bedingung *nie* erfüllt ist und damit die Schleife nie durchlaufen wird. Das Gegenteil wäre die *Endlosschleife*, bei der die Bedingung *immer* erfüllt ist und das Programm *nie zum Ende kommt*.

Achtung: <*Bedingung*> und <*Schleifeninhalt*> sind wieder *Platzhalter* für eine *beliebige Bedingung* und eine *beliebige Aktion* innerhalb der Schleife. Folglich sollten Sie bei Ihren Übungen dann die jeweils *passenden Befehle* in Javascript einsetzen!!!

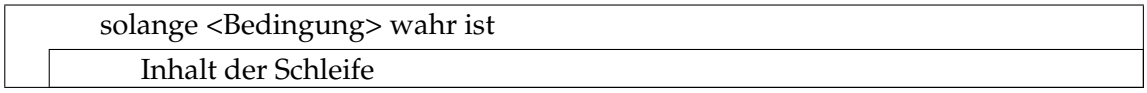

Abbildung 6: Struktogramm einer einfachen *while*-Schleife

```
while( <Bedingung> ){
    <Schleifeninhalt>
    }
```

Abbildung 7: Prinzip der *while*-Schleife

```
function zaehler(){
    var i=0;
    var schluss=10;
    document.write("hier geht es los: ");

    while (i < schluss ){
        i++;
        document.write(i);
        document.write(",");
    }
}
```

Abbildung 8: Version von *zaehler.js* mit einer Zählschleife die jetzt *while* verwendet.

3 Post-checked Loops

Dies sind Schleifen, bei der die Schleife zunächst erst einmal durchlaufen wird und **am Ende**, praktisch als Abbruchbedingung geprüft wird.

3.1 Do-While-Schleife

Die *Do-While*-Schleife entspricht der *While*-Schleife, nur wird die Bedingung erst am *Ende* der Schleife geprüft. D.h. die Schleife wird mindestens einmal und damit *einmal ungeprüft* durchlaufen.

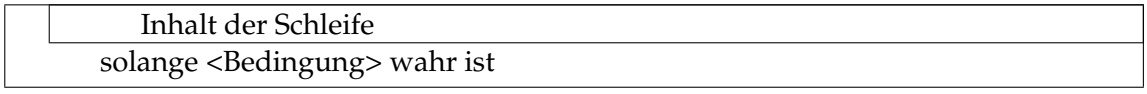

Abbildung 9: Struktogramm einer einfachen *do-while*-Schleife

```
do{
    <Schleifeninhalt>
    } while( <Bedingung> )
```

Abbildung 10: Prinzip der *do-while*-Schleife

```
function zaehler(){
    var i=0;
    var schluss=10;
    document.write("hier geht es los: ");

    do {
        i++;
        document.write(i);
        document.write(",");
    } while(i<schluss);
}
```

Abbildung 11: Zählschleife mit *do-while* und direkter Ausgabe der Werte.

4 Schachteln von Schleifen

Selbstverständlich kann im *Schleifeninhalt* selbst wieder eine Schleife stehen. Man muss sich nur darüber im Klaren sein, dass die äußere Schleife erst weiterarbeitet, nachdem die innere Schleife fertig abgearbeitet ist.

5 Aufgaben

- Bringen Sie die Zählschleife mit *for* in ihrem Bereich zum Laufen. Nicht nur abtippen, sondern auch mitdenken!
- Nun kommt die Zählschleife mit *while* dran.
- Selbstverständlich sollen Sie auch die Zählschleife mit *do-while* üben.
- Programmieren Sie eine Funktion, die die Zahlen von 0 bis 99 in 10 Zeilen mit jeweils 10 Spalten als Tabelle ausgibt. Mir kommt es dabei nicht auf eine besonders *hübsche* Formatierung, oder der Verwendung von *html-Tabellen* an. Ich möchte lediglich sehen, ob Sie 2 ineinander geschachtelte Schleifen richtig hinkriegen können.

  Tipp: Schauen Sie sich genau die bisherigen Beispiele an. Sie müssen lediglich eines dieser Beispiele erweitern und noch eine Schleife darum herum legen. Beachten Sie, dass *document.write* dynamischen html-Code ausgibt und folglich **Zeilenumbrüche** mit **html**-Code gemacht werden müssen.[7]

  Tipp 2: Sie machen sich die Sache leicht, wenn Sie 3 Zählvariablen verwenden. Eine für die **Zeilen**, eine für die **Spalten** und eine für den **Inhalt** der Tabelle. Es ist auch eine gute Idee seine Variablen entsprechend zu benennen.[8]

  Tipp 3: Papier und Bleistift ist das **beste Entwicklungswerkzeug**. Versuchen Sie – bevor Sie anfangen zu Programmieren – ein Struktogramm für diese Aufgabe zu entwickeln. Sie machen sich das Leben damit wirklich leichter!

- Programmieren Sie ein Tabellen-Werk aus 10 untereinander stehenden Tabellen (mit Abstand zwischen den Tabellen) aus jeweils 10 Zeilen mit jeweils 10 Spalten mit den Zahlen von 0 bis 999 als Inhalt. Verwenden Sie als äußere Schleife eine *while*-Schleife.

  Tipp: Sie machen sich die Sache leicht, wenn Sie jetzt 4 Zählvariablen verwenden. Eine für die Zeilen, eine für die Spalten, eine für die Tabellen und eine für den Tabelleninhalt.

  Alternative: Sie verwenden die o.a. Tabelle und setzen die *Abbruchbedingung* einer *while-Schleife* geschickt auf den Endwert der Tabelleninhalte.

  Tipp 2: Dies ist jetzt die Fortschreibung in **3 geschachtelte** Schleifen. Wenn dies bei Ihnen läuft, habe Sie die heutige Lektion verstanden.

  Tipp 3: Haben Sie für die letzte Aufgabe ein Struktogramm gemacht, dann sollte es jetzt kinderleicht sein um diese Struktur noch eine sinnvolle Schleife zu legen.

- Was in der Übungszeit nicht geschafft wird, wird zur Hausaufgabe zum nächsten Praktikumstermin. Es ist sehr wichtig, dass Sie dieses Lernziel erreicht haben. Es sind Grundlagen auf die wir aufbauen werden!

[7] siehe letzte Lektion.
[8] Leitsatz in der Informatik: Benutze **sprechende Namen**!