Websprachen im Vergleich

ASPX ist die Technologie zukünftiger Internetseiten auf Microsoft-Servern und ersetzt oder ergänzt das bisherige ASP (Active Server Pages). Dieser Artikel zeigt Gemeinsamkeiten und Unterschiede interaktiver Internet-Programme in JavaScript, PHP und ASP/JavaScript im Vergleich mit einem baugleichen ASPX-Programm.

Franz Fiala

Zur Beschreibung aller Programmversionen wird ein Taschenrechner-Programm verwendet. Es gibt drei benannte Textfelder `x`, `y` und `z` für die Zahlen, ein Select-Control für die Rechenoperation und einen Button, um die Berechnung durchzuführen. Der Unterschied zwischen den Programmversionen ist die Art, wie die Aufteilung der Programmaktivität zwischen clientseitigem und serverseitigem Skript erfolgt. Auf alle Details, die für die Erklärung nicht unbedingt notwendig sind, wurde verzichtet (`HTML`-Header, `BODY`-Tag, Fehlerprüfung bei Parameterübergabe, Fehlerbehandlung von Benutzereingaben).

Programm am Client (Rechner0)

Interaktive Internet-Seiten verwenden JavaScript-Programme, um auf Useranfragen zu reagieren und bei Bedarf die Seite neu zu gestalten. Sie haben daher Dateiendung **`htm`** oder **`html`** und enthalten eingebettete Skripts in der Sprache JavaScript. Andere Skriptsprachen sind zwar möglich, werden aber kaum eingesetzt, weil JavaScript die einzige gemeinsame Clientsprache für die verschiedene Browser-Implementierungen ist. Die Angaben und die Ergebnisse bleiben Bestandteil derselben Seite, bei Benutzeraktivitäten baut sich die Seite lokal neu auf; die Seite wird nicht vom Server neu aufgebaut.

Die Textfelder und die Auswahl der Rechenoperationen werden mit `name=` benannt; damit kann der Inhalt im Rahmen einer Funktion ausgewertet werden. Das Event `onClick()` ruft die Funktion `rechne()`. `rechne()` benutzt die die Funktion `eval()` zur Berechnung des Ergebnisses. `eval()` erlaubt die Auswertung von Ausdrücken zur Laufzeit. Damit die Ausdrücke nicht so lang werden, wurde das Objekt `document.rechner` durch die Variable `r` ersetzt. Der Zugriff auf die Feldinhalte erfolgt ohne Parameterübergabe, um größtmögliche Ähnlichkeit zu den späteren Programmversionen herzustellen.

Diese Form des Taschenrechners wird gänzlich am Client ausgeführt. Beachten Sie, dass die am Schluss besprochene ASPX-Version eine große Ähnlichkeit mit dieser einfachsten Version **`rechner0.htm`** aufweist. Programmierer, die bisher mit Clientprogrammierung beschäftigt waren, werden die Umstellung auf ASPX-Programme daher als sehr einfach empfinden.

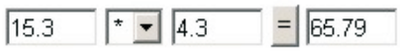
Datenübergabe zum Server (Rechner1)

Wenn Programmteile am Server ausgeführt werden sollen (Datenbankzugriffe, Berechnungen, Validierung...), muss man das Programm in zwei Teile aufteilen:

- **HTML-Teil** (Client-Teil **`rechner1.htm`**), der die Interaktion mit dem User besorgt. Beachten Sie, dass auf dieser Seite das Ergebnisfeld fehlt, weil die Auswertung und Anzeige durch eine andere Datei (**`rechner1.asp`**) erfolgt.
- **Skript-Teil** (Server-Teil **`rechner1.asp`**), der die Daten am Server verarbeitet und ausgibt. Dieses Skript muss daher die Angabe der HTML-Seite wiederholen und das Ergebnis anzeigen. Ein Link führt wieder zurück zur ersten Seite.

Die Auswertefunktion in der HTML-Datei verlagert sich bei einem Client-Server-Programm in das Server-Skript, mit der Kon-

Rechner0



Aufruf
`rechner0.htm`

rechner0.htm

```
<script language="JavaScript">
     function rechne() {
       r = document.rechner
       r.z.value = eval (r.x.value + r.op.value + r.y.value)
     }
</script>
<form name="rechner">
     <input type="text" name="x" size=3>
     <select name="op">
          <option value="+">+</option>
          <option value="-">-</option>
          <option value="*">*</option>
          <option value="/">/</option>
     </select>
     <input type="text" name="y" size=3>
     <input type="button" value="=" onClick="rechne()">
     <input type="text" name="z" size=3>
</form>
```

sequenz, dass dem Benutzer des Programms der Programmkode verborgen bleibt.

Die sendende HTML-Datei enthält nun kein Skript mehr. Man muss aber Maßnahmen zur Datenübergabe treffen. Das sind einmal die Attribute `action` und `method` im `form`-Tag, die das empfangende Skript und die Kommunikationsmethode angeben. Weiters muss das Steuerelement `button` in `submit` geändert werden, damit der Klick nicht eine Funktion anspricht sondern die Eingaben auf der HTML-Seite an das Serverskript sendet.

Als Übertragungsmethode kann `get` oder `post` gewählt werden. In unseren Beispielen wird `get` gewählt, weil dann die Parameter in der Kommandozeile übergeben werden und damit für Testzwecke verwendet werden können. (Diese Wahl ist bei ASPX-Skripts nicht mehr möglich.)

Das Serverskript arbeitet mit der Sprache VBScript und übernimmt die Daten aus der Kommandozeile mit der Methode `Request.QueryString()` in gleichnamige Variablen und benutzt die Funktion `eval()` zur Berechnung des Ergebnisses (`eval()` ist auch in VBScript verfügbar). Das Ergebnis wird angezeigt; der Benutzer bekommt mit einem Link die Möglichkeit, die vorige Seite wieder aufzurufen. (Wäre die Datenübergabe seitens der rufenden Datei `post` gewesen, müsste das Serverskript die Daten mit `Request.Form()` übernehmen.)

Der Funktionsaufruf wie im ersten Programm **`rechner0.htm`** fehlt scheinbar. Er entspricht praktisch dem Aufruf der Seite **`rechner1.asp`** durch den Submit-Button auf der Seite **`rechner1.htm`**.

Unbefriedigend bei diesem Programm ist, dass jede Wiederholung der Anwendung eine Neueingabe erfordert, was bei mehreren Eingaben unpraktisch ist. Daher wird in den folgenden Beispielen (Rechner2 und Rechner3) gezeigt, wie die Daten wieder in die ursprüngliche HTML-Datei zurückbefördert werden können.

Gegenseitige Datenübergabe (Rechner2)

Bei sich wiederholenden Aufgaben ist zweckmäßig, wenn eine Berechnung am Server gleich wieder als Input auf der HTML-Seite erscheint, daher muss das Server-Skript auch die HTML-Seite mit einem Parameter-Satz aufrufen. Diese Variante wird in den Dateien **`rechner2.htm`** und **`rechner2.asp`** demonstriert. Man sieht bei dieser Lösung, dass in der HTML-Seite ein erheblicher Aufwand mit der Parameteübergabe entsteht. Insgesamt ist diese Lösungsvariante sehr ineffizient, weil die Parameterübergabe in beiden Programmteilen, noch dazu in verschiedenen Sprachen zu implementieren ist.

Die HTML-Datei **`rechner2.htm`** hat gegenüber der Version in **`rechner1.htm`** die zusätzliche Aufgabe, die Paramter vom Server-Skript zu übernehmen. Da das Zerlegen des Komandozeilenstrings durch JavaScript nicht besonders unterstützt wird,

Rechner1

Aufruf von rechner1.htm
rechner1.htm

rechner1.asp

```
15.3*4.3=65,79
Weiter
```

Aufruf von rechner1.asp durch rechner1.htm
rechner1.asp?x=15.3&op=*&y=4.3&z=65.79

rechner1.htm

```
<form name="rechner" action="rechner1.asp" method="get">
     <input type="text" name="x" size=3>
     <select name="op">
          <option value="+">+</option>
          <option value="-">-</option>
          <option value="*">*</option>
          <option value="/">/</option>
     </select>
     <input type="text" name="y" size=3>
     <input type="submit" value="=">
</form>
```

rechner1.asp

```
<%
     x=Request.QueryString("x")
     op=Request.QueryString("op")
     y=Request.QueryString("y")
     z=eval(x+op+y)
%>
<%=x%><%=op%><%=y%>=<%=z%><br>
<a href="rechner1.htm">Weiter</a>
```

muss eine eigene Funktion `GetParValue()` geschrieben werden. Das Skript besteht aus zwei Teilen:

- der erste Teil übernimmt die Parameter von der Kommandozeile und speichert sie in gleichnamigen Variablen,
- der zweite Teil trägt die übernommenen Werte in die Steuerelemente ein.

Am Serverscript wird daher das Ergebnis nicht angezeigt sondern mit `Response.Redirect()` zum Client zurückgeschickt. Die Funktion `CStr()` wird benötigt, um die Zahlenwerte in Strings für die Rückgabe zu verwandeln. Am Client muss ein Skript diese Daten übernehmen und dann in die Steuerelemente eintragen.

`Response.Redirect()` ist eine effiziente Möglichkeit, zu einer weiteren Datei zu verzweigen und gleichzeitig über die Adresszeile die Parameter zu übergeben. Die Methode versagt allerdings, wenn die gerufene Datei die Daten nicht mit der Get-Methode in der Adresszeile sondern mit der Post-Methode übertragen werden, In diesem Fall müsste man ein Formular mit versteckten Feldern aufbauen, das an die empfangende Datei geschickt wird. Das genau macht ein ASPX-Programm aber ohne, dass der Benutzer sich darum kümmern muss.

Aus Platzgründen ist der Quellkode für dieses Programm bei der Webversion dieses Artikels zu finden.

Client-Server-Programm in derselben Datei (Rechner3)

Durch die Vereinigung der Eingabemaske und des Auswerteskripts in derselben Seite entfällt die verschiedenartige Behandlung der Übergabeparameter in JavaScript und VBScript, denn hier werden die Übergabeparameter immer in derselben Art übernommen.

Mit dieser Vorgangsweise sind wir schon nahe am Konzept von ASPX, denn auch bei ASPX-Skripts (besser ASPX-Programmen) ruft eine Seite immer sich selbst auf.

Der Vorteil: Programme und HTML-Elemente derselben Seite können direkt kommunizieren.

VBScript kann die Variablenwerte in einer sehr kompakten Form in den HTML-Kode einfügen; Beispiel:

```
<input type="text" name="x" size=3 value="<%=x%>">
```

Dieses Skript enthält gleichzeitig die Funktion zur Auswertung der Rechnung als auch das Formular, das der Client-Computer zu sehen bekommt. Dieser Kode ist aber nicht immer derselbe, er hängt von den Benutzereingaben ab, weil das ASP-Skript die konkreten Werte über die Kodefragmente `<%=x%>`, `<%=y%>` und `<%=z%>` in den HTML-Kode einfügt.

Beim Select-Steuerelement ist es aber nicht so einfach, weil man das Attribut `selected` in jenes Option-Element einfügen muss, dessen Rechenoperation der Benutzer ausgewählt hat. Durch das entsprechende Programmkonstrukt dann der HTML-Kode ziemlich zerhackt. Es gibt dafür auch eine alternative Lösungsmöglichkeit, indem man ein kurzes clientseitiges Skript lädt, das nach dem Laden des HTML-Kode die entsprechende Option einstellt.

Die serverseitige Skriptsprache kann die jeweils benötigten Skripts am Client generieren, das Clientscript muss daher nicht alle möglichen Fälle abdecken, sondern nur den für diese Seite zutreffenden. Hier ist es die Zeile

```
document.rechner.op.options[<%=i%>].selected = true
```

die je nach ausgewählter Rechenoperation einen anderen Wert für `i` einsetzt.

Der User sieht folgenden Kode:

```
<form name="rechner" action="rechner3.asp" method="get">
     <input type="text" name="x" size=3 value="15.3">
     <select name="op">
          <option value="+">+</option>
          <option value="-">-</option>
          <option value="*">*</option>
          <option value="/">/</option>
     </select>
     <input type="text" name="y" size=3 value="4.3">
     <input type="submit" value="=">
     <input type="text" name="z" size=3 value="65.79">
</form>
<script>
     document.rechner.op.options[2].selected = true
</script>
```

Auch der Dateiname des Skripts **`rechner3.asp`** wird durch `<%=Script%>` in das Action-Attribut durch das ASP-Skript eingetragen. Damit bleibt der Kode unverändert, auch wenn die Datei umbenannt werden sollte.

Die PHP-Version dieses Programms befindet sich als **`rechner3.php`** bei der Webversion dieses Artikels.

rechner3.asp

Erster Aufruf von rechner3.asp
rechner3.asp

Zweiter Aufruf von rechner3.asp ("="-Button)
rechner3.asp?x=15.3&op=*&y=4.3&z=0

Dritter Aufruf von rechner3.asp ("="-Button)
rechner3.asp?x=15.3&op=*&y=4.3&z=65.79

rechner3.asp

```
<%
     Script=Request.ServerVariables("SCRIPT_NAME")
     x=Request.QueryString("x")
     op=Request.QueryString("op")
     y=Request.QueryString("y")
     z=eval(x+op+y)
%>
<form name="rechner" action="<%=Script%>" method="get">
     <input type="text" name="x" size=3 value="<%=x%>">
     <select name="op">
          <option value="+">+</option>
          <option value="-">-</option>
          <option value="*">*</option>
          <option value="/">/</option>
     </select>
     <input type="text" name="y" size=3 value="<%=y%>">
     <input type="submit" value="=">
     <input type="text" name="z" size=3 value="<%=z%>">
</form>
<%
     Operatoren = "+-*/"
     i = Instr(Operatoren,op)-1
%>
<script>
     document.rechner.op.options[<%=i%>].selected = true
</script>
```

Rechner4

15.3 * 4.3 = 65.79

Aufruf von Rechner4 (erster Aufruf und folgende Aufrufe)
rechner4.aspx

rechner4.aspx

```
<script runat="server" language="JScript">
     function Rechne(o:Object, e:System.EventArgs)
     {
       z.Value=eval(x.Value+op.Value+y.Value);
     }
</script>
<form runat="server">
     <input type="text" id="x" size="3" runat="server">
     <select id="op" runat="server">
          <option value="+">+</option>
          <option value="-">-</option>
          <option value="*">*</option>
          <option value="/">/</option>
     </select>
     <input type="text" id="y" size="3" runat="server">
     <input type="button" value="="
          runat="server" OnServerClick="Rechne">
     <input type="text" id="z" size="3" runat="server">
</form>
```

ASPX-Programm (Rechner4)

Eine gleichartige ASPX-Datei ist von der Funktion her verwandt mit der zuletzt beschriebenen Version **rechner3.asp**, allerdings ohne die Notwendigkeit, dass sich der Programmierer darum kümmern muss, wie Server- und Clientteil zu trennen sind. Vom Aufbau her ähnelt das Programm aber eher der einfachsten Client-Version **rechner0.htm**.

Bei ASPX-Programmen gibt es nur eine Programmdatei, der Programmierer hat daher volle Übersicht; diese Datei wird aber in einen (unsichtbaren) serverseitigen Binärkode und einen zum Client geschickten HTML-Kode aufgeteilt. Bisher musste der Programmierer für diese Aufteilung sorgen, jetzt genügt das Attribut `runat=server`, um jene Programmteile zu kennzeichnen, deren Kenntnis für das Serverprogramm erforderlich ist. Über die Parameterübergabe zwischen diesen Teilen muss man sich nicht mehr kümmern, die wird von ASPX übernommen.

Aus dem Skript wird ein vollwertiges Programm, denn am Server arbeitet ein Kompiler. Der erste Aufruf einer Seite dauert daher etwas länger, weil der Kode kompiliert wird; alle folgenden Aufrufe arbeiten mit maximaler Geschwindigkeit. Der Programmierer genießt die volle Unterstützung der Fehlerprüfung durch den Kompiler.

Vergleich ASP und ASPX

Verglichen werden die Dateien **rechner3.asp** und **rechner4.aspx**.

- In ASPX-Dateien hat das `form`-Tag keine Attribute, weil die aktuelle Datei immer auch gleichzeitig das Ziel ist.
- Die Steuerelemente der HTML-Seite werden mit dem `id`-Attribut angesprochen (und nicht mit dem `name`-Attribut)
- Jedes HTML-Element, das vom Skript angesprochen werden soll, erhält das zusätzliche Attribut `runat="server"`
- Der Skript-Teil in ASPX ist eigentlich ein kompiliertes Programm, daher müssen alle verwendeten Variablen typisiert werden.

Hier wurde die Spache JScript eingesetzt; sie besitzt auch die für unser Beispiel benötigte Funktion `eval()` (übrigens als einzige der vier verfügbaren Sprachen).

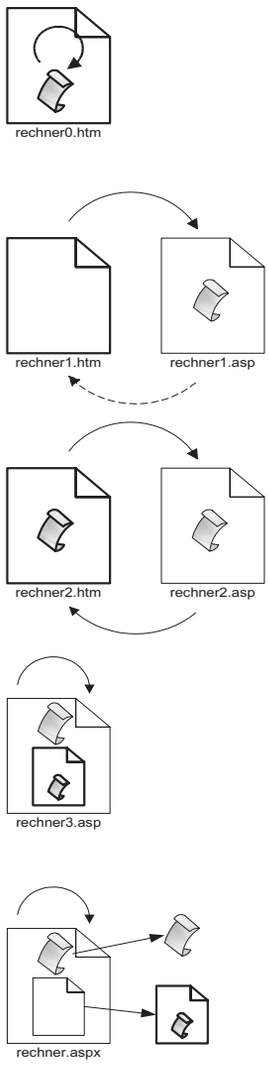
Während man beim Program **rechner3.asp** den zum User geschickten HTML-Teil aus der ASP-Datei extrahieren kann, ist das bei ASPX nicht möglich, denn die Generierung des HTML-Kodes liegt jetzt in der Verantwortung von ASPX. Hier ist der automatisch generierte Kode von **rechner4.aspx**:

```
<form name="_ctl0" method="post" action="rechner4.aspx" id="_ctl0">
     <input type="hidden" name="__EVENTTARGET" value="" />
     <input type="hidden" name="__EVENTARGUMENT" value="" />
     <input type="hidden" name="__VIEWSTATE"
          value="dDwyMzk0ODY5NjQ7Oz5TnIdTdTOGjaoWkbv3BaqeOxptDg==" />

     <script language="javascript">
     <!--
          function __doPostBack(eventTarget, eventArgument) {
               var theform = document._ctl0;
               theform.__EVENTTARGET.value = eventTarget;
               theform.__EVENTARGUMENT.value = eventArgument;
               theform.submit();
          }
     // -->
     </script>

     <input name="x" id="x" type="text" size="3" />
     <select name="op" id="op">
          <option value="+">+</option>
          <option value="-">-</option>
          <option value="*">*</option>
          <option value="/">/</option>
     </select>
     <input name="y" id="y" type="text" size="3" />
     <input language="javascript" onclick="__doPostBack('_ctl1','')"
          name="_ctl1" type="button" value="=" />
     <input name="z" id="z" type="text" size="3" />
</form>
```

Man sieht, dass das Skript sich selbst mit der Post-Methode aufruft und dabei mit versteckten Feldern arbeitet. Die Server-Eventfunktion `ServerOnClick` wurde in die clientseitige Eventfunktion `onclick()` verwandelt, die die Formulardaten ähnlich wie ein Submit-Button zum Server schickt.

ASP- und PHP-Programme transportieren Variablenwerte ohne Schutzmaßnahmen zwischen Client und Server, daher ist diese Schnittstelle ohne zusätzliche Maßnahmen leicht angreifbar, weil man auch andere als die dafür vorgesehenen HTML-Seiten als Input verwenden kann und die Parameter verändern kann.

Bei ASPX wird die Kommunikation durch die Variable `__VIEWSTATE` geschützt und daher ist das gemeinsame Programm kaum angreifbar.



fett: Teile, die am Client sichtbar sind

Der Weg zu ASPX

Client-Programm: Ein Anwenderprogramm vom Typ "Eingabe"->"Berechnung"->"Ausgabe wir in einer HTML-Seite durch eine JavaScript-Funktion realisiert, das Programm wird am Client wiederholt.

Client-Eingabe, Server-Berechnung: Durch Aufteilung des Programms in Eingabe und Berechnung kann die Berechnung zum Server verlagert werden. Für eine Wiederholung der Berechnung ist ein Link zurück zur Eingabemaske erforderlich.

Datenrückgabe zum Client: Damit die Eingabe auch nach der Berechnung wieder in der Eingabemaske sichtbar ist, müssen die Daten vom Serverscript zum Client zurückgeschickt werden.

Serverscript und Eingabemaske in einer Datei: Durch Vereinigung der Eingabemaske und Serverscript in einer Datei wird die Verarbeitung der Daten wesentlich vereinfacht, weil immer dasselbe Skript die Daten übernimmt. Das Programm wird leicht unübersichtlich, weil verschlachtelter HTML- und Skript-Kode die Lesbarkeit erschweren.

ASPX-Programme trennen Darstellung und Daten, weil alle HTML-Elemente als Variable benannt und daher vom Skript her parametrierbar sind. Die Aufteilung in Server-(Binär)-Skript und Eingabemaske erfolgt implizit.