5 JScript

JScript ist die wichtigste Alternative zu VBScript. Auch wenn die Bedeutung der Sprache nicht die Größenordnung hat wie VBScript, sind die sprachlichen Möglichkeiten doch so verschieden, dass der Einsatz manchmal elegante Lösungen hervorbringt – zumal sie jederzeit verfügbar ist.

5.1 Einführung in JScript

Dieses Kapitel richtet sich an alle, die schon Erfahrung mit Java, C oder C++ haben. Es baut auf das auf, was Sie bereits wissen, und zeigt, worin sich diese Sprachen von JScript unterscheiden.

Voraussetzungen

Mit Ihrem Basiswissen und diesen Empfehlungen werden Sie in sehr kurzer Zeit ASP-Skripte schreiben können. Falls Sie über dieses Wissen noch nicht verfügen, so denke ich, dass Sie VBScript vielleicht einfacher erlernen werden und es zum Schreiben von Skripten schneller verwenden können.

Wenn Sie bereits JavaScript verwendet haben, zum Beispiel zum Erstellen von Navigationshilfen in HTML-Seiten oder für DHTML, dann kennen Sie bereits JScript, viel Neues werden Sie dann hier nicht finden. JScript ist mit JavaScript identisch und Sie können wahrscheinlich das meiste in diesem Kapitel überblättern. Sie sollten dennoch Abschnitt 5.2 *JScript im Einsatz mit ASP* lesen, denn dieser enthält ASP-spezifische Informationen.

Was ist JScript?

JScript ist die Microsoft-Implementierung von JavaScript – Implementierung ist treffender als Version, weil Syntax und Funktionalität von JScript *identisch* mit der Syntax von JavaScript ist. Ein Programm, welches in JavaScript geschrieben ist, wird immer auch in JScript laufen und umgekehrt (zumindest theoretisch). Da die ursprüngliche JavaScript-Sprache von Netscape entwickelt wurde, musste Microsoft eine eigene Implementierung von JavaScript als Sprache herausbringen, sodass diese in den Microsoft-Produkten verwendet werden konnte. Das Ergebnis ist JScript.

JavaScript versus Java

JavaScript und Java haben, bis auf den ähnlich klingenden Namen und einige gleich aussehende syntaktische Elemente nichts miteinander zu tun.

Java wurde von Sun Microsystems erfunden. JavaScript wurde von Netscape entwickelt und hieß anfänglich LiveScript. Wegen der vertrauten Beziehungen von Netscape zu Sun und auf Grund der Popularität von Java, wurde LiveScript erst in der letzten Minute zu JavaScript, mit dem Segen von Sun.

JavaScript teilt sich die gleiche Syntax und die gleichen Basisbefehle mit Java und auch mit C und C++. Das macht Java etwas einfacher für all jene zu lernen, die schon C- und C++-Entwickler sind. Aber die Art, wie JavaScript Variablen handhabt, unterscheidet sich doch sehr von Java, C oder C++. Dazu mehr in Abschnitt 5.2.3 *Was Sie mit JScript nicht können* und den folgenden.

5.2 JScript im Einsatz mit ASP

In diesem Abschnitt geht es um die ASP-spezifischen JScript-Informationen. Auch wenn Sie JavaScript kennen, können Sie hier wertvolle Informationen finden.

5.2.1 Trennzeichen

Sie verwenden die Trennzeichen <% und %>, um Ihren Code von den HTML-Tags zu trennen. Sie verwenden also die Trennzeichen unabhängig von der Skriptsprache, die Sie verwenden.

Trennzeichen für mehrere Zeilen

Sie können Trennzeichen auf jeder einzelnen Zeile individuell verwenden, wie in einigen Beispielen in Kapitel 4, oder Sie können verschiedene Zeilen mit einem Satz von Trennzeichen einschließen, wie das folgende Beispiel zeigt:

```
<% var zaehler
zaehler = 15 %>
```

Wenn Sie oft mehr als eine Zeile von Code in Trennzeichen einschließen müssen, können Sie das Aussehen lesbarer gestalten, indem Sie den Trennzeichen jeweils eine eigene Zeile zu Verfügung stellen:

```
<%
var zaehler
zaehler = 15
%>
```

Trennzeichen und geschweifte Klammern

Ihre Trennzeichen und Ihre geschweiften Klammern arbeiten unabhängig voneinander. Innerhalb eines Blocks darf also HTML-Code stehen, zum Beispiel:

```
<%
if (zaehler > 15)
{
   Warnung = True
%>
Zähler Überlauf!<p>
<%
}
%>
```

In diesem Beispiel ist eine HTML-Zeile in den Körper eines `If`-Befehls gestellt. Beachten Sie, wie Sie die Trennzeichen verwenden können, um den JScript-Code überall einzuschließen. Hier eine kompaktere Methode:

```
<% if (zaehler  > 15) {
  Warnung = True %>
Zähler Überlauf!<p> <% } %>
```

Wie auch immer Sie sich entscheiden, Ihre geschweiften Klammern und Ihre Trennzeichen zu organisieren, Sie sollten hier nur innerhalb des Ganzen konsistent bleiben, damit der Code gut lesbar ist.

Trennzeichen für die Anzeige von Variablenwerten

Verkürzte Ausgaben von Variableninhalten

Es gibt eine spezielle Syntax, die es erlaubt, direkt im HTML den Wert einer JScript-Variable auszugeben. Diese Syntax sieht folgendermaßen aus:

```
Die Antwort ist <% = result %>.<p>
```

In der HTML-Zeile werden die Trennzeichen und der Code durch das Ergebnis des Variablenwertes ersetzt, wenn HTML an den Browser zurückgesendet wird. Sie können jede beliebige Variable und ihren Wert anzeigen – ob numerisch oder als Zeichenkette –, indem Sie diese Notation verwenden. Wenn der Wert des Ergebnisses 5 ist, sieht die zurückgesendete HTML folgendermaßen aus:

```
Die Antwort ist 5.<p>
```

Sie können sogar diese Notation innerhalb eines HTML- Tags verwenden:

```
<font size=<% = fontsize %>>
```

Falls die JScript-Variable *fontsize* den Wert 4 an diesem Punkt des Skripts hat, sieht der zurückgesendete HTML-Code folgendermaßen aus:

```
<font size=4>
```

5.2.2 Auswahl der Standardsprache

Obwohl beide, VBScript und JScript, mit Ihrem Webserver ausgeliefert werden, kann nur eine von ihnen die bevorzugte Sprache sein, und die ist bei Lieferung VBScript. Sie können diese Voreinstellung auf verschiedene Weise verändern.

Die Auswahl von JScript als Ihre bevorzugte Sprache

Wenn Sie Windows 2000/NT/XP und den IIS verwenden und JScript als die bevorzugte Sprache für alle ASP-Skripte auf Ihrer gesamten Site einstellen wollen, nehmen Sie bitte diese Schritte vor:

1. Starten Sie den INTERNET DIENSTE MANAGER (erreichbar über SYSTEMSTEUERUNG | VERWALTUNG).
2. Öffnen Sie den Ordner: INTERNET INFORMATIONSDIENSTE UND DEN NAMEN DES COMPUTERS.
3. Suchen Sie den Eintrag STANDARDWEBSITE und klicken Sie mit der rechten Maustaste. Wählen Sie aus dem Menü den Eintrag EIGENSCHAFTEN.
4. Klicken Sie auf die Registerkarte BASISVERZEICHNIS.
5. Klicken Sie auf KONFIGURATION... Der Dialog ANWENDUNGSKONFIGURATION wird angezeigt.
6. Klicken Sie die Registerkarte ANWENDUNGSOPTIONEN (siehe Abbildung 5.1).
7. Löschen Sie VBScript und tippen Sie dafür JScript im Feld ASP-STANDARDSPRACHE ein.

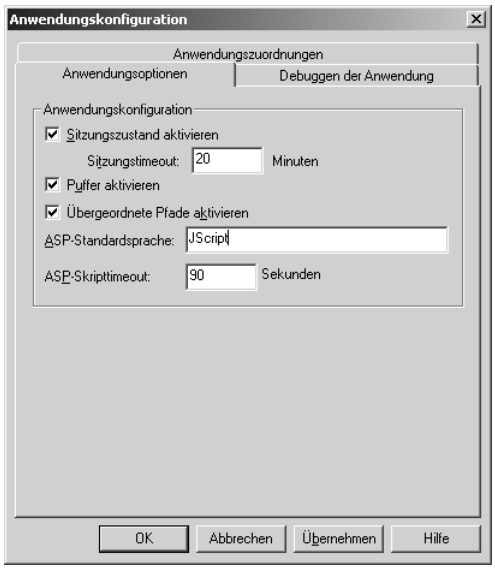

Abbildung 5.1: Einstellung von JScript als neue Standardskriptsprache

Wählen einer anderen Sprache für nur eine Seite

JScript im Skript wählen

Wenn Sie JScript als Sprache für ASP Seite für Seite wählen wollen, müssen Sie nur die folgende Zeile an den Anfang des Skripts setzen:

```
<%@ LANGUAGE = "JScript" %>
```

Falls aber JScript die bevorzugte Sprache für Ihre Website ist und Sie VBScript nur für eine Seite verwenden möchten, schließen Sie einfach diese Zeile als oberste mit ein:

```
<%@ LANGUAGE = "VBScript" %>
```

5.2.3 Was Sie mit JScript nicht können

Einschränkungen

Manchmal ist es einfacher zu verstehen, was die Sprache nicht kann, bevor Sie lernen, was sie alles kann. Wenn Sie ein C- oder C++-Programmierer sind, werden Sie mit Sicherheit ein paar Dinge finden, die JScript nicht bietet. Die folgende Übersicht zeigt diese Besonderheiten auf einen Blick.

Kein Compiler

JScript wird interpretiert

JScript ist eine vollständig interpretierte Sprache. Es ist nicht im traditionellen Sinne wie C und C++ kompiliert. Und JScript ist auch nicht kompiliert, sondern wird von einem reinen Interpreter ausgeführt.

JScript läuft im Webbrowser, wenn es in clientseitigen Skripten verwendet wird. Und es läuft auf einem Webserver, wenn Sie ASP-Skripte verwenden. Die Skripte werden beim Ablaufen interpretiert.

Der größte Nachteil interpretierter Sprachen ist, dass sie gewöhnlich langsamer als kompilierte Sprachen laufen. Aber Sie werden kaum bemerken, wie langsam ein kleines Skript läuft, nachdem Sie mehrere Minuten dazu gebraucht haben, dieses mit Ihrem 56 K-Modem einfach nur herunterzuladen!

Kein Präprozessor

Keine globale Konfiguration

Sowohl C als auch C++ besitzen einen Präprozessor, der durch den Quellcode durchgeht, bevor er kompiliert wird und auf die Direktiven des Compilers im Code reagiert, wie beispielsweise die `#define`- und `#include`-Direktiven. JScript hat keinen Präprozessor. Trotzdem helfen einige Eigenschaften in JScript, die Dinge zu gestalten, die Sie bisher mit einem Präprozessor zu tun gewöhnt waren.

Keine Zeiger

Keine Zeiger

C- und C++-Programmierer argumentieren oft, dass Zeiger kraftvolle und notwendige Programmierbefehle sind. Aber sogar der erfahrene Entwickler muss zugeben, dass sie oftmals die Quelle von schwer zu findenden Programmierfehlern sind.

Auf Grund der Fehler, die oft aus der Verwendung von Zeigern resultieren, besitzt JScript keine Zeiger. Es gibt jedoch *References* (dt. Verweise). *References* sind vorsichtigere Befehle und werden offener von der Sprache gehandhabt als Zeiger und machen deshalb auch Anfängern weniger Probleme.

Keine Structures oder Unions

Vereinfachtes Typkonzept

Haben Sie schon darüber nachgedacht, dass es konzeptionell eine Menge Überlappungen zwischen Klassen, `Structures` und `Unions` gibt? Warum sollte man es nicht zu in einer Sache verbinden? Das ist genau das, was in JScript getan wurde. In JScript ist alles ein Array – ob Sie es glauben oder nicht. Sehen Sie in Abschnitt 5.3.5 nach, um mehr Informationen über JScript-Objekte und -Arrays zu erhalten.

Keine Konstanten

Keine echten Konstanten

In C kommt es vor, dass Sie den Befehl `#define` verwenden, um einer Zahl, die dann immer und immer wieder verwendet wird, einen Namen geben zu. Auf diese Zahl kann dann innerhalb des Programms hingewiesen werden. Tatsächlich wurde diese Arbeitsweise so populär, dass der ANSI-Standard für C diese Fähigkeit in die Sprache mit dem Schlüsselwort `const` eingebaut hat. Java verwendet das Schlüsselwort `final`, um das Gleiche zu tun.

Leider unterstützt JScript noch keine Konstanten; `const` ist bereits ein reserviertes Schlüsselwort in JScript, obwohl es noch nicht implementiert ist.

Keine Klassen oder Vererbungen

Objekte ohne eigene Klassen

C ist keine objektorientierte Sprache. Also werden Sie als reiner C-Entwickler dieses nicht vermissen, aber C++- und Java-Entwickler werden dies wohl. JScript unterstützt keine Klassen oder Vererbungen. Aber es besitzt Objekte. Die Art, wie mit Objekten umgegangen wird, ist jedoch in der Programmierwelt einzigartig. Mit dem fehlen von echter objektorientierter Programmierung gehen einige andere Effekte einher. So gibt es keine Operatorenüberladung. Operatorenüberladung ist eine Eigenschaft, an die sich nur C++-Entwickler schon gewöhnt haben werden. Sie ist sehr sinnvoll, aber nicht zwingend notwendig und auch nicht Teil von JScript.

Keine unterschiedliche Anzahl von Argumenten

Wenig Flexibilität bei Funktionsdeklarationen

In C und C++ können Sie Funktionen erstellen, die zu unterschiedlichen Zeiten eine unterschiedliche Anzahl von Argumenten akzeptieren, in Abhängigkeit davon, wie die aufrufende Prozedur die Funktion verwenden will. JScript bietet diese Funktionalität nicht.

Keine goto-Befehle

Keine direkten Sprünge

Der `goto`-Befehl war immer und überall das Gegenteil strukturierter Programmierung. Zu einer Zeit, als Computersprachen noch viel primitiver waren, hatten Sie auch wenig Möglichkeiten die Verwendung des `goto`-

Befehls für bestimmte Situationen zu vermeiden. Aber die heutigen modernen Sprachen bieten alle nötigen Schleifen- und Verzweigungsstrukturen. Deshalb gehören in JScript `goto`-Befehle längst zur Geschichte.

5.2.4 Schwerpunkt: Zeichenketten

Die Unterstützung von Zeichenketten ist vollständig und komplett. Dies ist ein wesentlicher Unterschied zu C. Gegenüber VBScript ist der Umgang mit Zeichenketten konsequenter gelöst (auch wenn Sie das als VBScript-Profi als gewöhnungsbedürftig empfinden mögen). Auf Grund der sehr unterschiedlichen Art und Weise, wie die Datentypen im Allgemeinen von JScript behandelt werden, achten Sie bitte in den kommenden Abschnitten genau darauf, wie man Zeichenketten am besten verwendet.

Abschnitt 5.4.4 macht Sie mit den Methoden des String-Objekts vertraut, durch das jede Zeichenkette repräsentiert wird.

5.3 Allgemeine Anmerkungen zu JScript

Dieser Abschnitt behandelt die elementaren Eigenschaften von JScript, wie die Notation der Variablen, Kommentare und Funktionen.

5.3.1 Grundlagen in JScript

Dieser Abschnitt behandelt die Grundlagen der Sprache JScript. Dies geschieht nur soweit, dass Sie erste Skripte schreiben und mit der Webserverprogrammierung anfangen können. Eine vollständige Darstellung der Sprache soll dies nicht ersetzen. Nehmen Sie immer auch die Online-Referenz zur Hand.

Kommentare

Kommentare in JScript werden auf exakt die gleiche Weise behandelt, wie in den modernen Sprachen C, C++ und Java:

- `//`
  Hiermit beginnt eine Zeilenanmerkung, die am Ende der Zeile aufhört.
- `/*` und `*/`
  Dies beschreibt Beginn und Ende von mehrzeiligen Anmerkungen.

Diese Anmerkungen, genauso wie der Rest Ihres JScript-Codes, erscheinen nicht auf der Seite, wenn der Nutzer im Internet Explorer auf ANSICHT | QUELLTEXT ANZEIGEN klickt. Hier arbeitet die ASP-Engine ebenso wie bei

VBScript. Wenn Sie Anmerkungen ergänzen wollen, die auf der Seite zu sehen sein sollen, verwenden Sie die HTML-Anmerkungen Tags <!-- und -->.

JScript berücksichtigt Groß- und Kleinschreibung

JScript ist *abhängig* von Groß- und Kleinschreibung. Diese Abhängigkeit bedeutet für Sie verschiedenes:

1. Schlüsselwörter, wie zum Beispiel Kommandos, müssen genau so eingegeben werden, wie sie in der Dokumentation vorkommen – üblicherweise bedeutet das in der Kleinschreibung.
2. Objektnamen müssen auch so eingegeben werden, wie sie in ihrer Dokumentation erscheinen. Möglicherweise wird dort der erste Buchstabe großgeschrieben und der Rest klein. Wie auch immer, Eigenschafts- und Methodenbezeichnungen können in jeder Schreibweise vorkommen.
3. Variablen werden so benannt, wie Sie diese bestimmen. Also denken Sie sich einen Standard aus und bleiben Sie dabei!

Achten Sie darauf, dass Sie die Schreibweisenempfindlichkeit von JScript im Kopf behalten – besonders wenn Sie mit Variablen arbeiten –, denn anders als bei den anderen Sprachen, mit denen Sie bisher schon umgegangen sind, bekommen Sie keine Fehlermeldung, wenn Sie zufällig die Schreibweise einer Variable auf halbem Wege ändern. JScript nimmt einfach nur an, dass Sie eine neue Variable erstellen, und macht weiter. Diese Annahme kann Fehler erzeugen, die sehr schwierig zu finden sind.

Behandlung des Zeilenendes

Die Markierung des Zeilenendes

In C, C++ und Java müssen Sie jede Zeile mit einem ; beenden. In JScript ist dies immer noch eine gute Idee, aber es ist nicht erforderlich. Wenn Sie eine Zeile nicht mit einem ; beenden, nimmt das Programm an, dass die Zeile an der Stelle endet, an der Sie die Enter-Taste drücken. Dies ist ein Zugeständnis an die VBScript-Experten, die eine Zeilenendemarkierung nicht kennen. Und es ist ein Zugeständnis an C-Programmierer, die gern eine solche Markierung setzen.

5.3.2 Variablen

Unabhängig von Ihrem bisherigen sprachlichen Wissen ist es sehr nützlich, die folgenden Abschnitte über Variablen sehr aufmerksam zu lesen. Denn die Art, wie sie in JScript funktionieren, ist ungewöhnlich.

Variablen und Datentypen

Anders als in C, C ++ und Java sind Variablen in JScript nicht streng typisiert. Tatsächlich sind sie, wie in VBScript auch, überhaupt nicht auf einen Typ festgelegt! Dies ist im Übrigen eine der typischen Eigenschaften jeder

Skriptsprache im Unterschied zur Programmiersprache. Sie erstellen Variablen einfach durch die Verwendung des Schlüsselwortes `var`:

```
var name
var zaehler, maximum
```

Es gibt keinen Hinweis auf den Typ der Variable. Tatsächlich können Sie einen permanenten Typ einer Variable erstellen, wenn Sie diese erzeugen. Die Variable nimmt den Typ der Daten an, die sie beinhaltet (also den Datentyp) aber es ist auch vollständig akzeptabel, einer Variable einen Wert in Form einer Zeichenkette zuzuweisen und später der gleichen Variable einen numerischen Wert zu geben. Dies ist aber nicht die beste Programmierpraxis.

Wie auch bei anderen Sprachen können Sie die Variablen mit einem Wert belegen, wenn Sie diese erstellen.

```
var name = "JScript"
var counter = 100
```

Interne Datentypen

Obwohl Variablen in JScript keine festgelegten inneren Variablentypen haben, wenn Sie diese erstellen, nehmen sie doch den Datentyp an, der ihnen zugewiesen wird. Und diese Arten der Werte, die Sie ihnen zuweisen können, entsprechen in JScript etwa jenen, die Sie in jeder anderen Sprache finden:

1. Zeichenketten werden entweder von einfachen oder doppelten Anführungszeichen umgeben.
2. Zahlen kommen entweder in der Schreibweise mit Gleitkomma oder als ganze Zahlen vor. Ganze Zahlen können in dezimaler, oktaler (durch die Verwendung einer führenden 0) oder hexadezimaler Form (durch die Verwendung eines führenden 0x) repräsentiert werden. Zahlenvariablen können ebenso einige der folgenden Werte enthalten: `NaN` (steht für »Not a Number« – keine Zahl) `positive infinity`, (dt. positiv unendlich) `negative infinity (dt. negativ unendlich)`, `positive 0` und `negative 0`. Beachten Sie, dass Sie diese Werte nicht zuweisen können, sie können nur als Ergebnis einer mathematischen Operation entstehen und eine spezielle Eigenschaft einer Variablen darstellen.
3. Boolesche Variablen mit der Aussage richtig oder falsch sind spezielle Werte und sie setzen den Wert auch nicht auf 1 oder 0. Wenn Sie einen numerischen Ausdruck verwenden, wo ein Boolescher Wert erwartet wird, wird 0 als falsch verstanden und alles andere wird als richtig interpretiert.

Wie JScript Datentypen erkennt

Schließlich können Variablen auch noch zwei weitere Werte enthalten, die nicht wirklich unter eine bestimmte Datentyp-Form fallen:

1. Eine Variable beinhaltet einen undefinierten Wert, nachdem sie erstellt und bevor ihr ein Wert zugewiesen wurde. Wenn Sie versuchen, so eine Variable mit einem undefinierten Wert zu verwenden, erhalten Sie eine Fehlermeldung.
2. Eine Variable kann auch den Wert `Null` (dt. nichts) enthalten. Dies unterscheidet sich von der undefinierten Variable und auch von der 0. Die Variable mit dem Wert `Null` ist eine Variable, die wirklich nichts enthält.

Gültigkeitsbereich

Lokale und globale Variablen

Wenn Sie eine Variable in einem Skript mit Hilfe des Schlüsselwortes `var` außerhalb jeglicher Funktionen erstellen, bezieht sich die Variable global auf dieses Skript. Wenn Sie eine Variable innerhalb einer Funktion erstellen, so gilt diese Variable nur in dieser Funktion, das heißt, sie ist automatisch lokal.

Wenn Sie einer Variablen einen Wert zuweisen, die nicht mit dem Schlüsselwort `var` erstellt wurde, so wird sie automatisch erzeugt. Und unabhängig davon, ob die Zuweisung innerhalb einer Funktion oder außerhalb eine Funktion geschieht, gilt diese Variable global.

Wenn Sie versuchen, eine Variable zu verwenden, die zuvor nicht erstellt wurde, erhalten Sie eine Fehlermeldung. Diese Verhaltensweise entspricht der Nutzung der Anweisung `OPTION EXPLICIT` in VBScript.

Lokale Variablen arbeiten immer schneller als globale Variablen. Die langsamsten aller globalen Variablen sind jene, die nicht mit dem Schlüsselwort `var` erstellt wurden, bevor ihnen ein Wert zugewiesen wurde.

Umwandlung von internen Variablentypen

Automatische Umwandlung

JScript erledigt die Umwandlung meistens automatisch für Sie. Das folgende Beispiel zeigt dies:

```
<%@ LANGUAGE = JScript %>
<%
var netto, brutto, ausgabe
netto = 93.36
brutto = netto * 1.16
ausgabe  = "Der Nettopreis des Buches beträgt " + netto + " DM, "
ausgabe += "der Bruttobetrag " + brutto + "DM."
%>
<% = ausgabe %>
```

Listing 5.1: Automatische Typkonvertierung (js_convert.asp)

Wenn die Zahl mit dem Gleitkomma mit einer ganzen Zahl multipliziert wird, wandelt JScript den Variablentyp so um, wie er gebraucht wird. Wenn die Zahlen innerhalb einer Zeichenkette kombiniert werden, werden die Zahlen in eine Zeichenkette umgewandelt (siehe Abbildung 5.2).

Abbildung 5.2: Die Zahlen werden zur Anzeige in Zeichenketten verwandelt (Ergebnis von Listing 5.1)

parseInt
parseFloat

Wenn Sie Zahlen und Zeichenketten zusammen verwenden, nimmt JScript immer an, dass Sie die Zahlen in Zeichenketten umwandeln wollen. Für die umgekehrte Wandlung bietet JScript zwei Funktionen, die eine Zeichenkette entweder in eine ganze Zahl oder eine Gleitkommazahl umwandeln: `parseInt()` und `parseFloat()`.

```
<%@ LANGUAGE = JScript %>
<%
var byte, bit
byte = "2 Bytes"
bit = parseInt(byte) * 8
%>
<% = byte %> entsprechen <% = bit %> Bit.<p>
```

Listing 5.2: Anwendung der Funktion parseInt (js_parseint.asp)

`Die parse`-Funktionen bieten Ihnen noch eine weitere Eigenschaft an: Sie ignorieren jeglichen nicht numerischen Bestandteil nach der Zahl (siehe Abbildung 5.3).

Abbildung 5.3: Eine Zeichenkette wurde für die Berechnung in eine Zahl umgewandelt.

5.3.3 Operatoren

Die Ausstattung von JScript mit Operatoren ist im üblichen Rahmen. Viele lassen sich sogar zusammenfassen, sodass der Einsatz flexibler als in VBScript möglich ist.

Allgemeine Operatoren

Die Operatoren in JScript kommen Ihnen vielleicht bekannt vor. Die Operatoren sind eigentlich alle identisch mit denen in C. In Tabelle 5.1 finden Sie eine Liste der JScript-Operatoren.

Tab. 5.1: Mathematische Operatoren

Operator	Bedeutung
=	Zuweisung
= =	Gleichheit
+, -	Addition, Subtraktion
*, /	Multiplikation, Division
%	Modulus (Rest ganzzahliger Division)
++, --	Inkrement und Dekrement
-	Unäre Negation

Tab. 5.2: Bitweise Operatoren

Operator	Bedeutung
&	AND; Und
|	OR; Oder
^	XOR; exklusives Oder
<<	Bitweise links schieben
>>	Bitweise rechts schieben
>>>	Vorzeichenlos rechts schieben
~	NOT; Negation

Tab. 5.3: Logische Operatoren

Operator	Bedeutung
<, >	Kleiner als, größer als
< =	Kleiner als oder gleich
> =	Größer als oder gleich
! =	Ungleich
&&, | |	Logisches Und, Logisches Oder
!	Logisches Nicht
?:	Trinärer bedingter Operator
= =	Gleichheit

Operator	Bedeutung
= = =	Identität
! = =	Keine Identität
,	Sequenz

Tab. 5.3: Logische Operatoren (Forts.)

Verbundoperatoren

Kombinationen aus Zuweisung und Operator

Bislang mag Ihnen noch kein Unterschied zwischen VBScript und JScript aufgefallen sein. JScript kennt aber weitere Operatoren, mit denen sich die Programmierung teilweise vereinfachen lässt. Den wichtigsten Teil nehmen die Verbundoperatoren ein. Dies sind Kombinationen aus einem mathematischen Operator und dem Gleichheitszeichen. Das sieht zum Beispiel so aus:

```
zaehler += 10
```

Ohne diesen Operator müssten Sie den Ausdruck folgendermaßen schreiben:

```
zaehler = zaehler + 10
```

Die folgende Tabelle zeigt alle Verbundoperatoren und deren Bedeutung:

Verbundoperator	Bedeutung
+=	Addition
-=	Subtraktion
*=	Multiplikation
/=	Division
%=	Modulus
&=	Und
|=	Oder
^=	Exklusives Oder
>>=	Rechtsschieben
<<=	Linksschieben
>>>=	Vorzeichenlos rechts schieben

Tab. 5.4: Kombinierte Zuweisungsoperatoren

Funktionsoperatoren

JScript kennt noch einige weitere Operatoren, die in logischen Ausdrücken verwendet werden können.

delete
instanceof
typeof
void

Der unäre Operator `delete` löscht ein Element oder die Eigenschaft eines Objekts. Der Ausdruck, auf den sich der Operator bezieht, sollte ein Element oder eine Eigenschaft sein. `instanceof` stellt fest, ob ein Objekt Bestandteil einer Klasse ist. Mit `typeof` stellen Sie den Typ einer Variablen fest. Wann

immer Sie einen Operator aus syntaktischen Gründen einsetzen müssen und eigentlich keine Operation wünschen, setzen Sie `void` ein.

new Ein anderer Operator ist dagegen nicht unbedingt als solcher zu erkennen. `new` erstellt ein neues Objekt:

```
mein_objekt = new Array()
```

5.3.4 Bedingungen und Schleifen

Die Befehle, die Bedingungen und Schleifen in JScript erstellen, nutzen alle die gleichen Schlüsselwörter und arbeiten mehr oder weniger auf die gleiche Weise wie die in C, C++ und Java.

Bedingungen mit If...else erstellen

if else Der `if...else`-Befehl wird verwendet, um Fragen zu stellen und dann, ausgehend von der Antwort, zu verzweigen.

```
if (child == 0 && woman == 1)
{
   dinky = true
   family = false
} else {
   family = true
}
```

Sie können die geschweiften Klammern verwenden, um entweder im `if`- oder `else`-Bereich einen Block von Befehlen zu erstellen. Beachten Sie, dass es in JScript kein Schlüsselwort »then« gibt.

Mehrfache Verzweigungen mit switch...case

switch case Sie verwenden den Befehl `switch...case`, um eine Variable oder einen Ausdruck mit einer Zahl auf den möglichen Wert zu vergleichen. Er arbeitet genauso, wie der Befehl `switch...case` in C, C++ und Java mit den folgenden Ausnahmen:

- In JScript können Sie zur Verzweigung auch Zeichenketten nutzen.
- Die `switch`-Variable und der Wert, mit dem sie verglichen wird, müssen nicht vom gleichen Typ sein, da JScript mit Variablentypen sehr locker umgeht.
- C und C++ verlangen von dem Wert in jedem `case`, dass er einzigartig ist. JScript setzt das nicht voraus.
- Die `case`-Werte, die Sie miteinander vergleichen, können auch Variablen sein.

Wenn Sie also einen Code interpretieren wollten, könnten Sie folgendermaßen vorgehen:

```
<%@ LANGUAGE = JScript %>
<%
var sprache, nachricht
sprache = "VBScript"
switch (sprache)
{
case "VBScript":
   nachricht = "VBScript ist eine Skriptsprache"
   break
case "PHP":
   nachricht = "PHP ist eine Skriptsprache"
   break
case "C":
   nachricht = "C ist eine Programmiersprache"
   break
default:
  interpret = "Diese Sprache kenne ich nicht"
}
%>
<% = nachricht %>
```

Listing 5.3: Anwendung von switch...case in JScript (js_switch.asp)

Fehlerbehandlung im eigenen Code

try...catch...throw

Für die Behandlung »privater« Fehler steht eine besondere Anweisung zur Verfügung: `try...catch...throw`. Diese Anweisungen sind neu in Version 5 von JScript.

```
<%
function TryCatchDemo(x) {
  try {
    try {
    if (x == 0)  // Argument auswerten.
       throw "x gleich null";        // Fehler auslösen.
    else
      throw "x ungleich null";        // Anderen Fehler auslösen.
    }
    catch(e) {      // Hier "x = 0"-Fehler behandeln.
      if (e == "x gleich null")        // Überprüfen, ob Fehler
                                  //hier behandelt wird.
        return(e + " wird hier behandelt.");
        // Fehlermeldung für das Objekt zurückgeben.
      else      // Fehler kann hier nicht behandelt werden.
        throw e;      // Fehler für nächste Fehler-
    }                      // Behandlung neu auslösen.
}
  catch(e) {            // Andere Fehler hier behandeln.
    return(e + " wird auf einer höheren Stufe behandelt.");
```

```
      // Fehlermeldung zurückgeben.
  }
}
%>
0: <% = TryCatchDemo(0) %><BR>
1: <% = TryCatchDemo(1) %>
```

Listing 5.4: Fehlerbehandlung mit try...catch (js_trycatch.asp)

Schleifen mit for definieren

for

JScript unterstützt alle drei Schleifenbefehle, die auch von C++ und Java unterstützt werden: Die Zählschleife `for`, die am Anfang getestete `while`-Schleife und die am Ende getestete `do...while`-Schleife. Sie arbeiten genauso, wie Sie es von ihnen erwarten.

```
<%@ LANGUAGE = JScript %>
<%
var zweierpotenz
for (var x = 1; x <= 8; x++)
{
   zweierpotenz = Math.pow(2, x)
%>
Die Zweierpotenz von <% = x %> ist <% = zweierpotenz %>!
<P>
<%
}
%>
```

Listing 5.5: Einfache for-Schleife (js_for.asp)

while
do...while

Schleifen mit while und do

Die Schleifen `while` und `do...while` laufen so lange, bis eine Bedingung erfüllt ist.

```
<%@ LANGUAGE = "JScript" %>
<%
var quadrat = 0, x = 1
while (quadrat <= 100)
{
   x++
   quadrat = x * x
   %>
   Das Quadrat von <% = x %> ist gleich <% = quadrat %>
   <p>
   <%
}
%>
<% = x %> zum Quadrat ist größer als 100.<p>
```

Listing 5.6: while-Schleife (js_while.asp)

Der Unterschied zwischen der Schleife `while` und `do...while` besteht darin, dass die Bedingung in der `while`-Schleife zuerst kontrolliert wird. In der `do...while`-Schleife wird die Bedingung erst am Ende überprüft. Das bedeutet, dass der Code innerhalb einer `do...while`-Schleife immer wenigstens einmal ausgeführt wird. Der Code innerhalb einer `while`-Schleife wird niemals ausgeführt, wenn die Bedingung schon beim ersten Mal nicht erfüllt wird.

Expliziter Schleifenabbruch

break
continue

Die Befehle `break` und `continue` arbeiten genauso wie ihre Pendants in C++ und Java.

Der Befehl `break` stoppt die Ausführung einer Schleife oder des Befehls `switch...case` und springt vollständig aus ihnen heraus, beginnend mit der Zeile, die direkt der Schleife oder der `switch...case`-Kombination folgt.

Der Befehl `continue` stoppt die Ausführung des Inneren einer Schleife und beginnt die Schleife noch einmal von vorn.

Besondere Schleifen für Array

for...in

In Abschnitt 5.3.5 *Arrays* lernen Sie die Behandlung von Datenfeldern kennen. Für den unkomplizierten Umgang dient eine weitere Schleife: `for...in`. Hier ein einfaches Beispiel:

```
<%
var a, schluessel, s = "";
// Initialisiert das Objekt.
a = {"a" : "Athen", "b" : "Belgrad", "c" : "Chicago"}
// Durchläuft die Eigenschaften.
for (schluessel in a) {
    s += a[schluessel] + "<BR>";
}
%>
<% = s %>
```

Listing 5.7: for...in-Schleife zum Durchlaufen von Datenfeldern (js_forin.asp)

Dieses Skript gibt die Elemente der Liste *a* untereinander aus.

5.3.5 Arrays

Arrays sind ein Standardinstrument zum Speichern umfangreicher Daten. JScript bietet eine exzellente Unterstützung dafür.

Arrays erzeugen

Array()

In vielen Dingen sind Arrays in JScript frei von Begrenzungen jeglicher Art. Damit können sie für sehr viele Fälle eingesetzt werden, für die Sie in anderen Programmiersprachen umfangreiche Codes schreiben müssten. Es gibt nur eine wesentliche Einschränkung: In JScript sind Arrays auf eine Dimension begrenzt.

```
<%
titel = new Array(4)
titel[0] = "Active Server Pages"
titel[1] = 99.90
titel[2] = "Jörg Krause"
titel[3] = 965
%>
Titel: <% = titel[0] %><br>
Preis: <% = titel[1] %><br>
Autor: <% = titel[2] %><br>
Seiten: <% = titel[0] %>
<p>
```

Listing 5.8: Umgang mit einfachen Arrays (js_array.asp)

Arrays nehmen viele Datentypen auf

Arrays füllen

Das bedeutet aber auch, das Arrays mehr als einen Datentyp aufnehmen kann. Ein einziges Array kann verschiedene Datentypen in seinen unterschiedlichen Elementen aufnehmen, dies entspricht etwa dem Befehl `struct` in C oder C ++.

Vielleicht wird dies als schlechte Programmiererpraxis bezeichnet und als eines der Dinge, die Sie nicht tun sollten. Andererseits sind unglaublich elegante Programmierlösungen möglich. Den Designern der Sprache ging es tatsächlich um etwas anderes. Die Idee dahinter ist, zum einen die Sprache deutlich einfacher zu halten als »große« Programmiersprachen, zum anderen die Einschränkungen nicht so rigide zu halten, dass Probleme unlösbar werden.

In diesem Fall ist es sogar so, dass die Designer von JScript es sehr klar herausgearbeitet haben, dass sie tatsächlich wollen, dass Sie diese Fähigkeiten nutzen können. Tatsächlich haben sie sogar weitere Eigenschaften in das Programm eingebaut, um es Ihnen einfacher zu machen:

```
<%
titel = new Array()
titel["Titel"] = "Active Server Pages"
titel["Preis"] = 99.90
titel["Autor"] = "Jörg Krause"
titel["Seiten"] = 965
%>
Titel: <% = titel["Titel"] %><br>
Preis: <% = titel["Preis"] %><br>
Autor: <% = titel["Autor"] %><br>
Seiten: <% = titel["Seiten"] %>
<p>
```

Listing 5.9: Umgang mit indizierten Arrays (js_array2.asp)

Beachten Sie, dass diesmal keine Größe für das Array angegeben wurde. Dies ist nicht notwendig, weil jetzt noch nicht einmal mehr eine Zahl als Referenz auf das Array-Element benutzt wurde. Das nächste Beispiel zeigt eine noch faszinierendere Eigenschaft eines Arrays:

```
<%
titel = new Array(4)
titel.Titel = "Active Server Pages"
titel.Preis = 99.90
titel.Autor = "Jörg Krause"
titel.Seiten = 965
%>
Titel: <% = titel.Titel %><br>
Preis: <% = titel.Preis %><br>
Autor: <% = titel.Autor %><br>
Seiten: <% = titel.Seiten %>
<p>
```

Listing 5.10: Arrays als Objekte betrachtet (js_arrayo.asp)

Offensichtlich sind in JScript die Arrays sehr flexibel und es gibt ziemlich bizarre Anwendungen. Die in Listing 5.10 gezeigte Syntax offenbart die innere Struktur: Arrays (und im übrigen auch Zeichenketten) sind Objekte.

Arrays indizieren

Array-Indizes müssen einander nicht folgen.

Sogar wenn Sie Zahlen zur Indizierung der Elemente eines Arrays verwenden, müssen es keine aufeinander folgenden Zahlen sein. Das folgende Beispiel ist in JScript erlaubter Code:

```
<%
kunde = new Array ()
kunde (4) = "Müller Investitionen GmbH"
kunde (23) = "Versicherungen Sicher AG"
kunde (9384) = "Reifendienst Marquardt"
% >
```

Listing 5.11: Arrays als Objekte betrachtet (js_arrayi.asp)

Die Nummern der Arrayelemente in dem Array *kunde* werden dazu verwendet, um die Kundennummer zu bestimmen. Speicher wird dadurch nicht verschwendet, denn JScript setzt komprimierende Arrays ein. Der Code in Listing 5.11 hat nur drei Elemente und verbraucht auch nur dafür Platz.

5.3.6 Eigene Funktionen erzeugen

In C und C++ können Sie Funktionen erzeugen, die unabhängig sind und von jeder Stelle in Ihrer Applikation aufgerufen werden können. Java, in dem Versuch etwas mehr objektorientiert zu sein, fordert von Ihnen immer

die Erzeugung von Funktionen als Teil von Objekten. JScript arbeitet in dieser Hinsicht stärker wie C und C++. Sie können normale, einzeln agierende Funktionen kreieren, auf die von irgendwo auf der Seite zugegriffen werden kann.

function Sie erstellen eine Funktion mit Hilfe des Schlüsselwortes `function` und bestimmen den Namen, die Argumente und den Körper der Funktion.

```
<%
function quadriere(argument)
{
   var quadrat;
   quadrat = argument * argument;
   return quadrat;
}
%>
```

Listing 5.12: Aufbau einer selbst definierten Funktion

return Variablen, die innerhalb der Funktion erstellt werden, gelten lokal für diese Funktion. Sie verwenden den Befehl `return`, um den Wert zu bestimmen, der als Ergebnis der Funktion zurückgesendet wird.

Um diese Funktion aufzurufen, erwähnen Sie einfach ihren Namen, senden die richtigen Argumente und achten darauf, dass Ihnen der Wert zurückgesendet wird, wenn Sie einen Wert erwarten.

```
<% = quadriere(15) %>
<%
var ergebnis
ergebnis = quadriere(15)
%>
```

Listing 5.13: Aufruf der selbst definierten Funktion aus Listing 5.12

Werte und Referenzen

In C, C++ und Java legen Sie üblicherweise explizit fest, wie Argumente an eine Funktion gegeben werden – entweder durch den Wert oder durch Verweise. JScript versucht stattdessen allerlei Annahmen und macht dies immer auf die gleiche Weise.

Zahlen und Boolesche Werte werden immer nach ihrem Wert übergeben. Objekte, Felder und Funktionen werden immer durch die Verweisung übergeben.

Diese Regeln gelten nicht nur für Argumente in einer Funktion, sondern auch dann, wenn eine Variable einer anderen zugewiesen oder mit ihr verglichen wird.

Zeichenketten sind ein spezieller Fall. Sie werden kopiert und an die Funktion durch Verweisung übergeben, aber wenn sie verglichen werden, nutzt man wiederum ihren Wert – so wie Sie es sicher erwartet haben.

5.4 Eingebaute Methoden und Objekte

JScript ist ähnlich wie VBScript um bestimmte Funktionen erweitert worden. Im Gegensatz dazu sind die Erweiterungen stärker objektorientiert.

5.4.1 Das Objekt global

In JScript ist alles ein Objekt. Beim Programmieren macht sich das aber nicht immer bemerkbar. Das interne Objekt `global` nimmt Funktionen auf, die sonst nicht klar zugeordnet werden können. Die Anwendung entspricht tatsächlich auch der einer Funktion, denn Sie müssen nicht erst ein entsprechendes Objekt instanziieren. Hier eine Liste der Methoden, die wichtigsten werden nachfolgend kurz vorgestellt:

- `escape`
- `eval`
- `isFinite`
- `isNaN`
- `parseFloat`
- `parseInt`
- `unescape`

Die Methoden escape und unescape

escape unescape

Mit `escape` können Sie die Sonderzeichen einer Zeichenkette in das für die Übertragung per URL (siehe dazu auch Kapitel 4) nötige kodierte Format übertragen. Die Methode tauscht jedes Sonderzeichen gegen die Zeichenfolge %XX aus, wobei XX der Hex-Code des Zeichens ist. `unescape` wandelt eine entsprechend kodierte Zeichenkette wieder zurück.

Anwendungsbeispiel

Die folgende Zeichenkette enthält Umlaute:

```
var name
name = "Jörg Krause"
name = escape(name)
```

Die Zeichenkette enthält jetzt:

```
J%F6rg%20Krause
```

Die Methode `eval`

eval

Die `eval`-Funktion hat in JScript eine große Bedeutung. Immer wieder lassen sich Programmierprobleme damit sehr elegant lösen. `eval` ist die Art einer Funktion, die Sie wahrscheinlich nur in einer Interpretersprache finden, und sie bietet einige Vorteile, die in einer kompilierten Sprache wie C oder C ++ nur sehr schwer umzusetzen sind. Sie können eine Formel in der Form einer Zeichenkette nehmen und sie evaluieren, als wäre es eine Zeile des Codes. Die Methode führt also Code aus, der in einer Variablen steht.

```
<%@ LANGUAGE = JScript %>
<%
var formula, result
formula = "15 * (300 / 7)"
result = eval (formula)
%>
<% = formula %> ist gleich <% = result %>
```

Listing 5.14: So wird eval eingesetzt (js_eval.asp)

Das Ergebnis wird berechnet, als wäre es direkt in dem Code erschienen. Das folgende Beispiel gibt 64 zurück:

```
<%@ LANGUAGE = Jscript %>
<%
var formula, result
function quadriere(wert)
{
   var wert
   quadriere = wert * wert
   return quadriere
}
formula = " quadriere(8)"
result = eval (formula)
%>
<% = formula %> ist gleich <% = result %>
```

Listing 5.15: Nutzung der Funktion eval (js_eval2.asp)

Sie können JScript-Operatoren verwenden, Sie können auch eingebaute JScript-Funktionen und sogar Funktionen, die Sie selbst erstellen, verwenden. Das Aufrufen der `eval`-Funktion entspricht genau der Verarbeitung des Codes innerhalb der Zeichenkette durch den Interpreter an diesem Punkt im Skript. Jetzt können Ihre Programme beginnen, sich selbst zu schreiben!

5.4.2 Das ActiveX-Objekt

Mit dem ActiveX-Objekt erhalten Sie Zugriff auf die bereits bei VBScript beschriebenen Komponenten und Webserverobjekte. Die Anwendung unterscheidet sich nicht von der bereits beschriebenen.

```
var newObject = new ActiveXObject("MSWC.Browsercap")
```

Übersicht ActiveX-Objekte

Die folgenden Objekte stehen als ActiveX-Objekt zur Verfügung. Dadurch können Sie von allen Skriptsprachen unter ASP darauf zugreifen. Die Anwendung unterscheidet sich nicht oder nur marginal von der am Beispiel VBScript gezeigten. Folgende Objekte stehen zur Verfügung:

- `Dictionary`
- `Error`
- `FileSystemObject`

5.4.3 Das Array-Objekt

Übersicht

Das Array-Objekt wird mit dem folgende Befehl erstellt. Arrays selbst – und der Umstand, dass es sich dabei offensichtlich um Objekte handelt – wurden bereits in Abschnitt 5.3.5 behandelt.

Ein neues Array erstellen Sie folgendermaßen:

```
var neuesArray = new Array();
```

Methoden

Arrays werden in JScript mit verschiedenen Methoden bearbeitet, die im Folgenden übersichtsweise gezeigt werden. Beachten Sie bei der Interpretation der Arbeitsweise, dass es nur eindimensionale Arrays gibt, was die Anwendung bestimmter Funktionen überhaupt erst möglich macht:

- Die `concat`-Methode verbindet zwei Arrays miteinander:
- `Datenfeld1.concat(Datenfeld2)`
- Die `join`-Methode verbindet die Elemente eines Arrays mit einem Trennzeichen:
- `Datenfeld.join(Trennzeichen)`
- Die `reverse`-Methode dreht die Reihenfolge der Elemente eines Arrays um:
- `Datenfeld.reverse( )`
- Die `slice`-Methode gibt einen Teil eines Arrays zurück:
- `Datenfeld.slice(Beginn[, Ende])`

- Die `sort`-Methode dient der Sortierung eines Arrays. Die Sortierfunktion ist eine selbstdefinierte Funktion, die folgende Werte zurückgeben muss:
  - Einen negativen Wert, wenn das erste übergebene Argument kleiner als das zweite übergebene Argument ist.
  - Null, wenn beide Argumente gleich sind.
  - Einen positiven Wert, wenn das erste Argument größer als das zweite ist.

  `Datenfeld.sort(Sortierfunktion)`
  Die Sortierfunktion ist optional. Ohne Angabe wird eine interne Funktion verwendet.
- `toString`-Methode
  Wandelt das Array in eine Zeichenkette um. Die Elemente der Zeichenkette werden durch Kommata getrennt.
- `valueOf`-Methode
  Entspricht im Kontext eines Arrays der Methode `toString`. Dieselbe Methoden wird auch für andere Objekte verwendet und zeigt dann ein anderes Verhalten.

5.4.4 Das String-Objekt

Am Anfang des Kapitels wurde bereits erwähnt, dass auch Zeichenketten Objekte sind. Entsprechend finden Sie die Möglichkeiten zur Bearbeitung von Zeichenketten nicht unter dem Stichwort Funktionen, sondern Methoden. Das folgende Beispiel zeigt die Anwendung:

```
var strMessage, strError;
strMessage = new String("Dies ist eine Zeichenfolge.");
strError = new String("Die Umwandlung konnte nicht erfolgen.");
```

Methoden

Methoden zur Verarbeitung von Zeichenketten

Das `String`-Objekt kennt viele Methoden, die in der folgenden Tabelle überblicksartig vorgestellt werden.

Tab. 5.5: Zeichenketten-Methoden

Methode	Beschreibung
`anchor(anker)`	Setzt einen HTML-Anker: `<a name="anker">Zeichenkette</a>`
`big`	Setzt das HTML-Attribut `<big>`
`blink()`	Setzt das HTML-Attribut `<blink>`
`bold()`	Setzt das HTML-Attribut `<b>`
`charAt(Index)`	Gibt das Zeichen am `Index` zurück

Methode	Beschreibung
`charCodeAt(Index)`	Gibt den Unicode des Zeichens am `Index` zurück
`concat(Kette)`	Verknüpft die Zeichenkette mit `Kette`
`fixed()`	Setzt das HTML-Attribut `<TT>`
`fontcolor(Color)`	Setzt das HTML-Attribut `<font color="Color">`
`fontsize(Size)`	Setzt das HTML-Attribut `<font size="Size">`
`fromCharCode(CL)`	Gibt eine Zeichenliste zurück, die aus der Unicode-Liste `CL` stammt
`indexOf(Teil, Start)`	Durchsucht die Zeichenkette nach *Teil*, ab Position *Start*
`italics()`	Setzt das HTML-Attribut `<I>`
`lastIndexOf(Teil, Start)`	Gibt das letzte Auftreten von `Teil` ab Position `Start` zurück
`link(anker)`	Setzt einen HTML-Link: `<A HREF="anker">Zeichenkette</A>`
`match(Regex)`	Sucht nach dem regulären Ausdruck `Regex`
`replace(Regex, Ersatz)`	Sucht nach dem regulären Ausdruck `Regex` und ersetzt ihn durch `Ersatz`
`search(Regex)`	Sucht nach einem weiteren Auftreten des Ausdrucks `Regex`
`slice(Beginn, Ende)`	Gibt eine Teilzeichenkette als String-Objekt zurück
`small()`	Setzt das HTML-Attribut `<SMALL>`
`split(Trennung)`	Teilt eine Zeichenkette am Zeichen `Trennung` und gibt ein Array zurück.
`strike()`	Setzt das HTML-Attribut `<STRIKE>`
`sub()`	Setzt das HTML-Attribut `<SUB>`
`substr(Beginn, Länge)`	Gibt eine Teilzeichenkette zurück
`substring(Beginn, Ende)`	Gibt eine Teilzeichenkette zurück
`sup()`	Setzt das HTML-Attribut `<SUP>`
`toLowerCase`	Wandelt eine Zeichenkette in Kleinbuchstaben
`toString`	Gibt ein Objekt als Zeichenkette zurück
`toUpperCase`	Wandelt eine Zeichenkette in Großbuchstaben
`valueOf`	Die Zeichenfolge selbst

Tab. 5.5: Zeichenketten-Methoden (Forts.)

Anwendungsbeispiel

Um eine normale Zeichenkette mit den gezeigten Methoden zu bearbeiten, sind keine besonderen Maßnahmen notwendig:

```
var klein
klein = "Das ist ein Test".toLowerCase
```

Das ist eine der vorn bereits angesprochenen »bizarren« Notationen, nichtsdestoweniger gültig und ausgesprochen einfach handhabbar. Wer es etwas exakter mag, kann auch Folgendes schreiben:

```
var strObject, klein
strObject = new String("Das ist ein Test ")
klein = strObject.toLowerCase
```

Viele der Methoden erzeugen HTML-Code, was ausgesprochen praktisch ist.:

```
var mylink = "Homepage des Autors"
mylink = mylink.link("http://www.joerg.krause.net")
response.write(mylink)
```

Dieses Skript erzeugt die folgende Ausgabe:

```
<A HREF="http://www.joerg.krause.net">Homepage des Autors</A>
```

5.4.5 Das Date-Objekt

Das Date-Objekt erlaubt den Zugriff auf Zeit- und Datumsinformationen.

Erzeugung

Umgang mit Zeiten und Daten

Dieses Objekt behandelt Datums- und Zeitwerte. Folgende Syntax dient der Erzeugung eines `Date`-Objekts:

```
var neuesDatum = new Date()
var neuesDatum = new Date(DatumWert)
var neuesDatum = new Date(Jahr, Monat, Datum
                          [, Stunden [, Minuten
                          [, Sekunden [,ms] ] ] ]
                         )
```

Methoden

Auf das so erzeugte `Date`-Objekt können Sie folgende Methoden anwenden:

- `get`-Methoden, beispielsweise `getMonth` oder `getHour`, geben den entsprechenden Bestandteil des Datumswerts zurück.
- `set`-Methoden, wie `setMilliseconds` oder `setMinutes`, setzen den entsprechende Teil des Datumswerts.

- `toLocaleString` gibt den Datumswert im Format der im Betriebssystem gewählten Ländereinstellung zurück.

5.4.6 Das Enumerator-Objekt

Aufzählungen

Das `Enumerator`-Objekt dient der Realisierung von Aufzählungen. Auf die Elemente der Aufzählung kann nur über entsprechende Methoden zugegriffen werden. Dadurch unterscheidet es sich von Arrays. Das Objekt ersetzte ursprünglich die Anweisung `for...in`, die erst ab JScript 5.0 (ab Windows 2000/IIS5/IE5) verfügbar ist.

Das folgende Beispiel zeigt die Liste der Laufwerke des Servers:

```
{
  var fso, s, n, e, x;
  fso = new ActiveXObject("Scripting.FileSystemObject");
  e = new Enumerator(fso.Drives);
  s = "";
  for (;!e.atEnd();e.moveNext())
  {
      x = e.item();
      s = s + x.DriveLetter;
      s += " - ";
      if (x.DriveType == 3)
        n = x.ShareName;
      else if (x.IsReady)
        n = x.VolumeName;
      else
        n = "[Laufwerk nicht bereit]";
      s +=  n + "<br>";
  }
  return(s);
}
```

Listing 5.16: Verwendung der Laufzeitbibliothek in JScript

Methoden

Die einsetzbaren Methoden auf einen Blick:

- `atEnd`-Methode; gibt `true` zurück, wenn der Zeiger am Listenende steht.
- `item`-Methode; gibt den aktuellen Eintrag zurück.
- `moveFirst`-Methode; setzt den Zeiger an den Listenanfang.
- `moveNext`-Methode; setzt den Zeiger eine Position weiter.

Der Zeiger ist ein internes Element, das die jeweils aktuelle Position speichert.

5.4.7 Die Zahlenobjekte in JScript

Die Zahlenobjekte erlauben den Umgang mit Zahlenwerten und mathematischen Berechnungen.

Math-Objekt

Mathematische Funktionen

Alle mathematischen Funktionen stehen als Methoden des `Math`-Objekts zur Verfügung. Insofern unterscheidet sich lediglich die Syntax von der in VBScript, nicht jedoch das verfügbare Funktionsspektrum.

Methoden

Hier eine Liste der Methoden des Objekts `Math`:

Tab. 5.6: Methoden des Objekts Math

Methode	Beschreibung
`abs`	Absoluter Betrag
`acos`	Arcus Kosinus
`asin`	Arcus Sinus
`atan`	Arkus Tangens
`atan2`	Winkel zwischen x-Achse und Punkt x,y
`ceil`	Kleinste Ganzzahl des Arguments
`cos`	Cosinus
`exp`	e^x
`floor`	Größte Ganzzahl
`log`	Natürlicher Logarithmus
`max`	Maximum
`min`	Minimum
`pow`	Potenz
`random`	Zufallszahl
`round`	Rundungsmethode
`sin`	Sinus
`sqrt`	Quadratwurzel
`tan`	Tangens

Eigenschaften

Diese Eigenschaften sind Konstanten mit häufig benötigten mathematischen Werten:

- `E`
  Die Eulersche Zahl e (2,718), Basis des natürlichen Logarithmus

- `LN10`
  Der natürliche Logarithmus der Zahl 10
- `LN2`
  Der natürliche Logarithmus der Zahl 1
- `LOG10E`
  Der dekadische Logarithmus der Zahl e
- `LOG2E`
  Der natürliche Logarithmus der Zahl e
- `PI`
  Die Konstante π (3,141592653589793)
- `SQRT1_2`
  Die Wurzel aus ½ (0,707)
- `SQRT2`
  Die Wurzel aus 2 (1,414)

Denken Sie daran, das JScript Groß- und Kleinschreibung unterscheidet. Die Eigenschaften, die hier in Großbuchstaben dargestellt sind, müssen auch genauso geschrieben werden.

Anwendung

Die Anwendung ist einfach, wie das folgende Listing zeigt:

```
var absolut, zahl, mypi
zahl = -34
absolut = Math.abs(zahl)
mypi = Math.PI
```

Listing 5.17: Methoden und Eigenschaften des Math-Objekts nutzen (js_math.asp)

Alle anderen Methoden werden entsprechend verwendet. `pow` hat zwei Argumente (Basis und Exponent), alle anderen haben ein Argument. Ausnahme ist die gezeigte Methode `abs`, die keine weiteren Argumente benötigt.

Number-Objekt

Sonderzustände von Zahlen

Das `Number`-Objekt wird nur selten direkt erstellt, JScript betrachtet jede Zahl intern ohnehin als Objekt. Einige spezielle Eigenschaften können aber angewendet werden, die manchmal die Erstellung sinnvoll erscheinen lassen:

- `MAX_VALUE`
  Eine Konstante: der größte darstellbare Wert.
- `MIN_VALUE`
  Eine Konstante: der kleinste darstellbare Wert.

- `NaN` (Not a Number)
  Konstanter Code für den Zustand »keine Zahl«.
- `NEGATIVE_INFINITY`
  Ein Zustand: negativ unendlich.
- `POSITIVE_INFINITY`
  Ein Zustand: positiv unendlich.

Hier ein weiteres Beispiel:

```
if (ergebnis.NEGATIVE_INFINITY) {
   response.write ("Minus Unendlich erreicht")
}
```

5.5 Reguläre Ausdrücke in JScript

Die Unterstützung regulärer Ausdrücke unterscheidet sich nur in wenigen Details von VBScript. Der wesentlichste Unterschied besteht darin, dass die Unterstützung in JScript nicht neu ist, sondern schon seit Version 3.0 existiert. Einen Überblick über die Methoden und Eigenschaften der entsprechenden Objekte finden Sie in diesem Abschnitt.

5.5.1 Objekte für reguläre Ausdrücke

Reguläre Ausdrücke werden in Abschnitt 4.5 *Reguläre Ausdrücke* ab Seite 224 ausführlich vorgestellt. Die dortigen Ausführungen zu VBScript gelten auch für JScript. Auf Abweichungen wird gesondert eingegangen.

Überblick der Methoden und Eigenschaften

Auf einen Blick

Die folgende Liste zeigt einen Überblick über die verfügbaren Methoden und Eigenschaften der beiden Objekte:

- Eigenschaften des `RegEx`-Objekts:
  - `$1`...`$9`-Eigenschaften
  - `index`-Eigenschaft
  - `input`-Eigenschaft
  - `lastIndex`-Eigenschaft
- Eigenschaften des `Regular Expression`-Objekts:
  - `lastIndex`-Eigenschaft
  - `source`-Eigenschaft
- Methoden des `Regular Expression`-Objekts:
  - `compile`-Methode

- exec-Methode
- test-Methode

5.5.2 Die RegExp-Objekte

Auch JScript verfügt über zwei Objekte: RegExp und das »virtuelle« Regular-Expression-Objekt. Letzteres wird nicht explizit erzeugt, sondern implizit durch die Angabe eines Suchmusters. Zwangsläufig muss der Interpreter das Suchmuster erkennen. Dies geschieht durch entsprechende Literale, die glücklicherweise der Syntax in Perl entsprechen. Es ist deshalb mit JScript verhältnismäßig einfach, sich durch Funde von interessanten Mustern in der Literatur komplexeren Lösungen zu nähern.

Ein RegExp-Objekt anlegen

RegExp

Mit dem folgenden Code legen Sie ein RegExp-Objekt an:

```
var myregex = new RegExp("Suchmuster", "Schalter")
```

Alternativ ist auch die Perl-Schreibweise möglich:

```
var myregex = /Suchmuster/Schalter
```

Die Suchmuster wurden bereits in Abschnitt 10.1 und 10.3 ausführlich beschrieben. Die Möglichkeiten in JScript unterscheiden sich nicht von denen in VBScript. Die Schalter entsprechen den Kodierungen in Perl, beschränkt auf zwei Optionen:

Optionen

- i ignoriere Groß- und Kleinschreibung
- g erlaubt die globale Suche (greedy) des Musters über die Suchzeichenkette

Reguläre Ausdrücke anwenden

exec

Um den Suchvorgang auszuführen, wird die Methode exec verwendet:

```
var myresult = myregex.exec("Suchzeichenkette")
```

Die Variable *myresult* ist nun ein Array mit den gefundenen Teilzeichenketten.

Die Eigenschaften des RegExp-Objekts

Das RegExp-Objekt verfügt über zwei Eigenschaften, mit denen das Verhalten vor der Suche verfeinert werden kann:

- lastIndex
  Dieser Wert gibt den Index in der Zeichenkette an, ab dem gesucht werden soll. Ohne Angabe wird die gesamte Zeichenkette durchsucht.

- `source`
  Diese Eigenschaft enthält den regulären Ausdruck (das Suchmuster) selbst. Der Wert ist schreibgeschützt.

5.5.3 Beispiele

Zwei Beispiele zeigen, wie reguläre Ausdrücke in JScript verwendet werden.

E-Mail-Adresse prüfen

Häufig werden in Formularfeldern E-Mail-Adressen eingetragen. Auch dieser Ausdruck ist noch nicht perfekt, für die meisten Fälle aber hinreichend gut und schnell. Das folgende Skript zeigt eine Anwendung mit JScript:

```
<%
var checkmail = /^[_a-zA-Z0-9-]+(\.[_a-zA-Z0-9-]+)* _
                 @[a-zA-Z0-9-]+\.([a-zA-Z]{2,3})$/ig;
var mail = Request.Form("EMAIL");
if (!checkmail.test(mail)) {
   %>
   <span style="color:red">Fehler in der Adresse</span>
   <%
} else {
   %>
<span style="color:red">Adresse ist zulässig</span>
   <%
}
%>
<form method="post" action="J10_401.asp">
E-Mail:      <input type="text" size="50" name="email"
                    value="<% = Request.Form("email") %>">
<br>
<input type="submit">
</form>
```

Listing 5.18: E-Mail-Adresse mit regulären Ausdrücken validieren (js_regexform.asp)

5.5.4 Die Eigenschaften des Objekts RegExp

JScript greift auf die Elemente des Ausdrucks und die Ergebnisse der Auswertung mit Eigenschaften des `RegularExpression`-Objekts zu. Dieses Objekt entsteht aus dem `RegExp`-Objekt, das intern verfügbar ist und nicht gesondert angelegt werden braucht. Die Eigenschaften sind aber erst dann mit einem Wert belegt, wenn ein Ausdruck erfolgreich untersucht wurde.

$1...$9 In Abschnitt 4.5.6 *Teilmuster und Wiederholungen* ab Seite 242 wurde bereits eine wesentliche Eigenschaft regulärer Ausdrücke gezeigt: die Zerlegung des Ausdrucks in Gruppen. Dies erfolgt mit runden Klammern, wobei die

Zuordnung einfach der Nummer der linken (öffnenden) Klammer folgt. In JScript können Sie über die Eigenschaften `$1` bis `$9` auf solche Teile des Ausdrucks zugreifen. Die Schreibweise mag abenteuerlich anmuten, Perl-Profis erkennen sie aber sicher wieder. In Perl werden genauso benannte Variablen für diesen Zweck erzeugt.

Das folgende Skript in Listing 5.19 verwendet einen geringfügig modifizierten Ausdruck. Zusätzlich wurden bestimmte Bereiche in Klammern gesetzt. Auf die Funktionalität des Ausdrucks hat dies keinen Einfluss, wohl aber auf die »Füllung« der entsprechenden Eigenschaften `$1` usw.

```
<%
var checkmail = new RegExp("^([_a-zA-Z0-9-]+(\.[_a-zA-Z0-9-]+)*)
                   @([a-zA-Z0-9-]+)\.([a-zA-Z]{2,3})$", "ig");
var mail = Request.Form("EMAIL");
var result = checkmail.exec(mail);
if (!result) {
   %>
   <span style="color:red">Fehler in der Adresse</span>
   <%
} else {
   %>
   <span style="color:red">Adresse ist zulässig</span>
   <br>
   <%
   Response.Write("<br>Mailname: ");
   Response.Write(RegExp.$1);
   Response.Write("<br>Domain: ");
   Response.Write(RegExp.$3);
   Response.Write("<br>Toplevel: ");
   Response.Write(RegExp.$4);
}
%>
```

Listing 5.19: Modifizierte E-Mail-Kontrolle mit Zugriff auf die Bestandteile (Ausschnitt aus js_regexmail.asp)

Da vier öffnende Klammern vorhanden sind (drei neu gesetzte und eine bereits vorhandene), sind die Werte `$1` bis `$4` belegt. Die aus funktionaler Sicht erforderliche dritte Klammer (von links zählend) wird nicht verwendet. Die anderen drei geben die Bestandteile der E-Mail-Adresse zurück (siehe Abbildung 5.4).

RegExp.Index

Die Eigenschaft `Index` enthält die Zeichenposition, an der die erste Übereinstimmung des regulären Ausdrucks in der Suchzeichenkette gefunden wurde. Alle weiteren Übereinstimmungen spielen keine Rolle. Das erste Element hat den Index 0. Die Eigenschaft bezieht sich wieder auf das eingebaute `RegExp`-Objekt:

```
Response.Write(RegExp.Index)
```

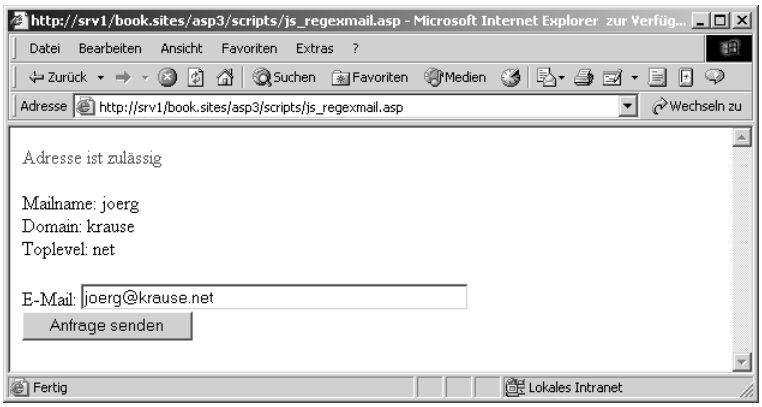

Abbildung 5.4: Auswertung und Zerlegung einer E-Mail-Adresse

RegExp.LastIndex Die Position nach der Übereinstimmung wird mit `LastIndex` ermittelt. Wurde keine Übereinstimmung gefunden, steht der Wert auf -1. Der Wert ist ebenso wie `Index` nullbasiert.

RegExp.Input Auf die Quelldaten des regulären Ausdrucks wurde bereits hinwiesen, diese stehen in der Eigenschaft `source`. Auch das Suchgebilde selbst ist so gespeichert, die entsprechende Eigenschaft heißt `input` und ist ebenso wie `source` schreibgeschützt.

5.6 Spezielle Spracheigenschaften

Dieser Abschnitt ist zum Verständnis der Sprache JScript nicht unbedingt notwendig, zeigt aber einige elementare Eigenschaften für den interessierten Leser.

5.6.1 Intrinsische Objekte

Intrinsisch = Eingebaut In JScript werden alle eingebauten Objekte als intrinsische Objekte bezeichnet. Damit wird klar gemacht, dass das explizite Ableiten der Objekte aus Klassen mit `new` nicht notwendig – und in der Regel auch nicht möglich – ist.

Die intrinsischen Objekte in JScript sind `Array`, `Boolean`, `Date`, `Function`, `Global`, `Math`, `Number`, `Object`, `RegExp`, `Regular Expression` sowie `String`.

5.6.2 Prototypen

prototype Als Prototypen werden die internen Klassendefinitionen der intrinsischen Objekte bezeichnet. Die `prototype`-Eigenschaft gibt einen Verweis auf den Prototyp einer Objektklasse zurück. Mit der `prototype`-Eigenschaft können Sie einer Objektklasse eine Basismenge von Funktionen bereitstellen. Neue Instanzen eines Objekts erben das Verhalten des Prototypes, der diesem

Objekt zugewiesen ist. Dies ist grundsätzlich eine Verhaltensweise, die JScript mangels Unterstützung für selbstgebaute Objekte vermissen lässt. Immerhin gibt es interessante Anwendungen dafür. Das folgende Skript erweitert die Eigenschaft `max` des intrinsischen Objekts Array. Das funktioniert, weil diese Eigenschaft eine `prototype`-Eigenschaft des Objekts ist. Nicht alle Eigenschaften und Methoden sind Prototypen. Auf welche das zutrifft, finden Sie in der Online-Dokumentation. Intern hängt es davon ab, wie das Objekt konstruiert wurde.

```
<%
function feld_max( ) {
  var i, max = this[0];
  for (i = 1; i < this.length; i++) {
     if (this[i] > max)
        max = this[i];
  }
  return max;
}
Array.prototype.max = feld_max;
var x = new Array(45, 3, 11, 7, 99, 9);
var y = x.max( );
%>
Größter Wert des Arrays: <% = y %>
```

Listing 5.20: Überschreiben eingebauter Funktionen mit prototype (js_proto.asp)

Dieses Beispiel gibt die Zahl 99 zurück; die Funktion ermittelt das größte Element eines Arrays. Die Methode `max` des Array-Objekts wurde dabei durch die Methode *feld_max* überschrieben. Die neue, veränderte Methode fügt sich nahtlos in den Kontext des Objekts ein, weil sie selbst durch die Eigenschaft `prototype` die Eigenschaften des Objekts geerbt hat.

5.6.3 Interne Konstruktoren

constructor

Mit der Eigenschaft `constructor` wird die Methode ermittelt, die ein Objekt erstellt. Angewendet werden kann `constructor` auf alle intrinsischen Objekte außer `Global` und `Math`.

Angenommen, Sie wissen nicht, ob eine bestimmte Variable ein String-Objekt ist, dann könnten Sie das folgendermaßen prüfen:

```
x = new String("Hallo");
if (x.constructor == String) {
   // Ja, es ist ein String-Objekt
}
```

Listing 5.21: Anwendung der Eigenschaft constructor