Fallstricke

Per Anhalter durch JavaScript

Oliver Pehnke, Benjamin Schmid

JavaScript ist die „erfolgreichste Programmiersprache im bekannten Universum". Glauben Sie nicht? Na dann zählen Sie mal allein die Browser-Installationen weltweit. Aber nicht nur im Netz, wo sie gepaart mit dem am Horizont stehenden HTML5 das Gelobte Land für mobile Überall-Anwendungen und Windows 8 ausruft, auch in vielen Java-Domänen greift sie massiv um sich. Höchste Zeit sich doch einmal näher mit diesem Programmier-Jargon auseinanderzusetzen.

„Anyway I know only one programming language worse than C and that is Javascript", sagt Robert Cailliau [Cail07], einer der Männer, die das Internet erfunden haben. Und er steht mit seiner Meinung nicht unbedingt alleine da, wie ein Blick in das Web offenbart.

Hätte man Brendan Eich 1995 vielleicht nur etwas mehr Zeit gegönnt: In nur zehn Tagen entwickelte er damals für Netscape den Kern von JavaScript: eine Skriptsprache aufbauend auf Konzepten aus Scheme, Lisp und Self, gepaart mit einer C-Syntax. Den Namenspräfix „Java" erhielt sie aus rein marketingtechnischen Gründen.

Kurze Zeit später „reverse-engineered" Microsoft JavaScript für seinen Internet Explorer (IE) unter dem Namen JScript. So gründlich, dass sie die Implementierungs-Bugs mit übernahmen. Netscape versuchte, JavaScript zu standardisieren: Die IEEE lehnte ab, die ANSI lehnte ab. Einzig die European Computer Manufactoring Association (ECMA) sagte zu. Und so wurde 1997 nach vielen Verhandlungen der Sprachstandard mit dem sperrigen Namen ECMAScript geboren. Ihren Schöpfer Eich erinnert der Name mehr an eine Hautkrankheit [Eich06]. Die Bugs überlebten: Auch heute ist für JavaScript `null` noch ein Objekt.

Ihren Geburtsschwierigkeiten und noch lebenden Montagsmacken zum Trotz: JavaScript erobert die Welt. Im Web drängt sie die Konkurrenz von Flash, JavaFX und Silverlight auf die hinteren Plätze. Hand in Hand mit HTML5 und seinen Features wie Canvas, Offline Storage und direkter Videounterstützung dürften hier bald auch die letzten Bastionen fallen. Google schiebt ebenfalls kräftig mit an. Und als größte Gemeinsamkeit ist HTML5 gepaart mit JavaScript auch kraftvolles Geschütz, um mit nur einem Wurf eine ganze Palette an mobilen Endgeräteplattformen zu adressieren.

Aber auch im Server-Bereich sind Sie nicht mehr sicher: Die NoSQL-Datenbank Apache CouchDB serviert in JavaScript Object Notation (JSON). Mit NodeJS lassen sich hoch-performante Webdienste ganz ohne nebenläufige Programmierung realisieren. Wie auch: JavaScript kennt keine Threads. Und spätestens seit Microsoft vor Kurzem verkündete, statt Silverlight nun HTML5 und JavaScript zu „First Class Citizens" für die UI-Entwicklung unter Windows 8 zu befördern, ist JavaScript endgültig zur neuen Lingua franca der Informationstechnik aufgestiegen.

Wie konnten Sie also nur diesen Edelstein in Ihrer bisherigen IT-Laufbahn so hartnäckig übergehen? Aber kein Grund zur Reue! Damit stehen Sie nicht allein da. Dieser Reiseführer möchte eine kleine Schützenhilfe für einen professionellen Entwickler-Start in die Sprache JavaScript geben. Eine Sprache, die so lange so unscheinbar erschien und sich inzwischen anstrengt, die Welt zu beherrschen.

Wie der Vogel zur Ente wird

JavaScript ist eine Sprache voller Missverständnisse. Eine erste Annäherung führt vermutlich zur Suchmaschine der Wahl. Doch was finden wir: Wikipedia? W3Schools? Download Java? Der Einstieg ist schwer, denn die Dokumentation ist verstreut und überdeckt durch Beispiele von Programmierlaien. Auch empfehlenswerte Bücher für den professionellen Einsteiger sind rar. Das Buch der JavaScript-Ikone Douglas Crockford, „JavaScript - The good parts", richtet sich eher an Fortgeschrittene. Viele andere Bücher mit „JavaScript" im Titel nutzen Sie besser als Briefbeschwerer. Dagegen sind die Seiten des Mozilla Developer Network [Moz] als weiterführende Lektüre durchaus empfehlenswert. Und, nein, diese finden Sie bislang leider nicht so ohne Weiteres an oberster Stelle Ihrer Suchanfrage.

Aber werfen wir zuerst gemeinsam einen ersten Blick auf das Kind. Obwohl der Name anderes erwarten lässt, hat JavaScript nur oberflächliche Gemeinsamkeiten mit Java. Auch der Suffix „Script" ist kein Indiz für eine „toy language". Dafür unterstützt sie verschiedene Programmierparadigmen von imperativ, funktional bis prototypisch objektorientiert und ist dynamisch, schwach und „duck" typisiert.

JavaScript braucht nur wenige selbsterklärende Datentypen: `Boolean`, `Number` (64 Bit Fließkomma-Zahlen) und `String` (Unicode Text). Die Typen `null` und `undefined` deklarieren leere bzw. unbekannte Werte.

Anders als stark typisierte Sprachen prüft JavaScript Datentypen erst zur Laufzeit und kommt ohne „Casts" aus. Variablen sind in JavaScript typenlos. Ihre Werte werden abhängig vom Kontext implizit konvertiert. Damit verstehen Sie auch, was Autoren der scheinbar sinnlosen Operationen `!!` und `*1` bezwecken: Dies sind quasi Schmalspur-Casts und erzwingen die boolesche bzw. numerischen Interpretation eines Wertes.

JavaScript kennt auch keine Klassen. Sie sind hier unnötig, denn die Objektorientierung erfolgt mittels sogenannter Prototypen. Jedes Objekt hält im Feld `prototype` eine Referenz dieser Blaupause. Prototypen vereinen in einem Objekt sowohl Struktur (also Methoden- und Feldnamen) als auch Werte (sprich Standardwerte und Methodenimplementierung). Sie sind reguläre Objekte und können somit auch zur Laufzeit geändert

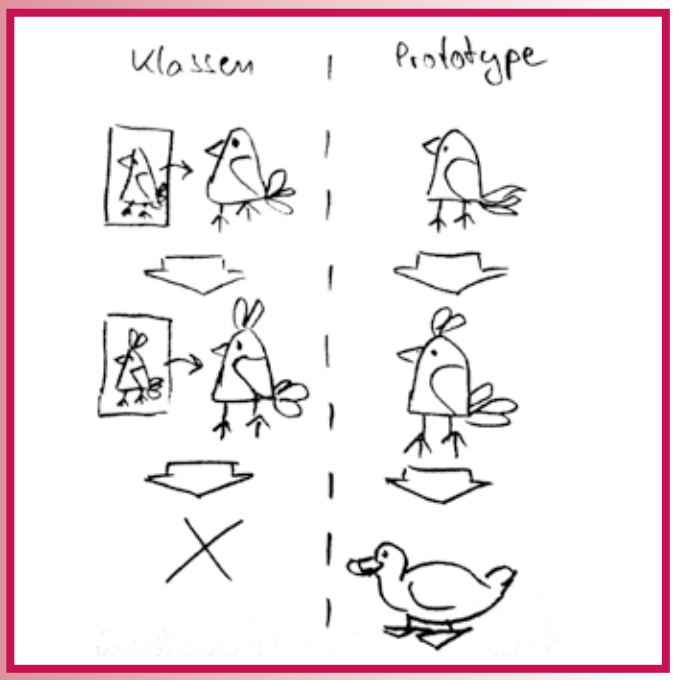

Abb. 1: Duck Typing

Abb. 2: Global Var

werden. So wird aus dem Vogel, welcher aussieht wie eine Ente, zur Laufzeit wirklich die Ente (duck typing).

Hier zeigen sich aber auch Konfliktpotenziale über die Vorherrschaft. Die Prototypen aller Typen – auch so elementarer wie String – sind frei zugänglich. Tauschen Sie die Implementierung der String-Funktion `length` aus, so ist dies zum einen möglich, zum anderen auch für alle wirksam.

Gleiches gilt auch für alle Werte, die Sie nicht explizit einer lokalen Variablen oder einem Objekt zuweisen, weil Sie z. B. das Schlüsselwort `var` vergessen haben. Diese landen im globalen Objekt. Der Letzte gewinnt. Neben dem Browser, der hier seine Einstiegs-Objekte `document` und `window` platziert, war z. B. jQuery schnell: Es deklariert ein globales Objekt `$`, welches als prägnanter Zugriffspunkt zur Framework-Funktionalität den typischen Charakter des verwendenden Codes prägt. Sie sind etwas spät dran: Meiden Sie also das Ändern bestehender Typen und suchen Sie sich für Einträge im globalen Scope lieber andere Namen.

Ein Koffer voller Bausteine

Ein Objekt ist in JavaScript lediglich eine Abbildung von Schlüsseln und Werten. Alle Schlüssel sind vom Typ String, die Werte können von beliebigem Typ sein. Objekte können andere Objekte beinhalten und so logische Strukturen abbilden.

```
var myObject = {
  name:  "Carrot",
  "for":    "Max",
  details: {
    color: "orange",
    size: 12,
    resize: function( grow ) {...}
  },
  doMagic: function() {...}
}
```

Listing 1: JavaScript Object Notation

Das Code-Beispiel in Listing 1 deklariert ein Objekt mit diversen Werten, einer Methode `doMagic` und einem verschachtelten Objekt `details`. Bis auf die Funktionen entspricht es zugleich auch dem Standard *JavaScript Object Notation* (JSON). Jetzt verstehen Sie auch, warum ihn Douglas Crockford 2001 kurzerhand aus der Taufe hob, als sein Kunde – wohl eher XML im Sinne – auf Standards pochte.

Funktionen sind „First Class Citizens“ und ebenfalls Objekte. Sie können also als Parameter und Rückgabewerte übergeben und in Objekten wie normale Werte deklariert und geändert werden. Sie bilden auch den alleinigen Gültigkeitsbereich (Scope) von Variablen. Anders als in Java können Sie also Deklarationen in noch so viele Klammern verpacken – sie bleiben trotzdem bis zum Ende der Funktion sichtbar und lebendig. Geschickt kombiniert mit einer sofortigen Funktionsausführung und den gleich näher erläuterten Closures, lässt sich damit auch eine Einschränkung und Kapselung von Sichtbarkeiten à la `private` in Java erreichen. Wer hat's erfunden? Ebenfalls Crockford!

Die Schokoladenseite von JavaScript

- *Funktionen sind first-class Objekte:* Sie können frei als Parameter-, Rückgabe- oder Variablenwerte genutzt werden.
- *Objekte sind einfach assoziative Arrays (Maps):* `obj.a` und `obj['a']` sind synonym. Dies erlaubt eine immense Flexibilität zur Laufzeit.
- *Schwache Typisierung:* Variablen und Felder haben keinen Typ. Ihr Wert impliziert diesen.
- *Closures:* Verschachtelt deklarierte Funktionen behalten die um sie herum sichtbaren Variablen.
- *Prototypische Vererbung:* Typen werden über veränderbare Vorlagenobjekte, nicht statische Klassen abgebildet.
- *Funktion können sich selbst ausführen:* Kombiniert mit Closures lassen sich somit mächtige Callback-Objekte einfachst erstellen.

Etwas spezieller verhält sich da schon das Schlüsselwort `this`. In Java ist die Welt noch einfach – hier liefert `this` den Eigentümer der Methode. In JavaScript ist sein Wert dagegen von der Art des Funktionsaufrufs abhängig. So kann es dort auch schon mal auf das globale Objekt oder bei Aufrufen mittels `call()` oder `apply()` sogar auf völlig fremde Objekte zeigen.

Eingeschlossen

Eines der mächtigsten Sprachmittel in JavaScript sind Closures. Nur was ist eine Closure? Kurz gesagt ist eine Closure ei-

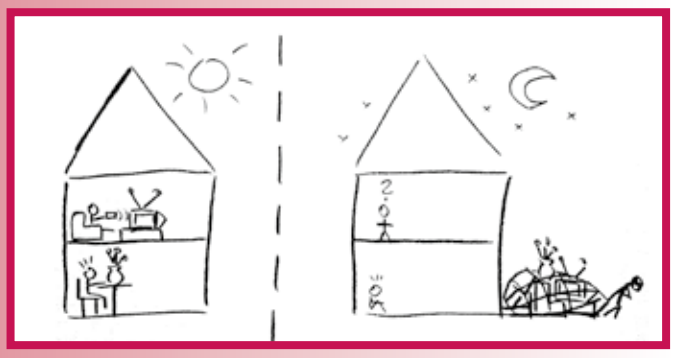

Abb. 3: „Closures“

ne Funktion, die sich zum Zeitpunkt ihrer Erstellung die Variablen um sich herum merkt.

Nicht verstanden? Da stehen Sie nicht allein da. Aber stellen Sie sich ein Haus vor mit einem glücklichem Paar, Ledercouch und Küchenvollausstattung (s. Abb. 3). Sonnenuntergang - beide gehen glücklich `schlafen()`. Mitten in der Nacht tauchen Diebe auf und nehmen alles mit. Wenn die beiden aufwachen, ist alles verschwunden. Sie können sich aber weiterhin sehr genau an alle Dinge erinnern. Das Paar bildet eine Closure.

Anders formuliert: Funktionen können innerhalb anderer Funktionen definiert werden. Die innere Funktion hat dann Zugriff auf alle Parameter und Variablen der äußeren Funktion, auch über die Lebensdauer der äußeren Funktion hinaus. Die in Listing 2 erstellte, innere Funktion bildet somit eine Closure: Sie hält den Zustand `zuhause` der äußeren Funktion `schlafen()` mit, bis sie selbst dereferenziert wird. Dieser Zustand kann auch explizit gehalten werden.

```
function schlafen(zuhause) {
  return function() {
     document.getElementById("label").innerHtml = zuhause;
  }
}
var paarNachDemAufstehen = schlafen("Unser schönes Heim")

// Wert von 'zuhause', obwohl schlafen() schon beendet ist
htmlButton.onClick = nachDemAufstehen
```

Listing 2: Closures

Die dunklen Seiten

Neben all den Missverständnissen hat JavaScript auch einige kantige Eigenheiten, die Ihnen kein noch so gewiefter Handelsvertreter als Vorteile verkaufen kann. Vom ersten Tag war JavaScript an die Entwicklung des Internets und seiner Browser gebunden. Es hat nie einen Testlauf gegeben und nur wenige Wartungs- oder Reparaturphasen. Über die Anfänge äußert sich selbst Schöpfer Eich reumütig: „Die erste Version war wirklich miserabel.“ Zeitdruck und Kompromisse in der Standardisierung sorgten dafür, dass Fehler und Eigenarten in der Sprache bis heute konserviert sind. ECMAScript 4 sollte aufräumen und statische Typisierung, Module und optionale Klassen bringen. Es scheiterte an politischen Spaltereien. Die Ersatz-Version 5 „Harmony“ möchte versöhnen und dafür nur die wichtigsten Punkte nachbessern. Geplante Deadline: 2013.

Listing 3 zeigt das hässliche Gesicht von JavaScript ganz ohne Kosmetik. Und Sie dachten, dass `null` ein Objekt ist. Wie

```
alert(typeof null); // 'object'
alert(null instanceof Object); // false

alert(typeof NaN);  // Number
alert(NaN == NaN); // false

alert(0.1 + 0.2  == 0.3) // false

var huh;
alert(huh === undefined); // true
undefined = "Surprise!";
alert(huh === undefined); // false!
```

Listing 3: JavaScript-Fallstricke

dumm von Ihnen! Dann versuchen Sie mal, mit der Idee von `NaN` (not a number, IEEE-Norm 754) umzugehen. Da JavaScript nur die Fließkomma-Darstellung erlaubt, kommen Sie daran nicht vorbei. Der Typ von `NaN` ist eine Zahl! Zudem ist sie auch nicht identisch zu sich selbst! Sie ist zu gar nichts identisch. Um also herauszufinden ob z. B. eine Division durch Null erfolgte, muss man die spezielle Funktion `isNaN()` bemühen. In Java erhält man für Ganzzahlen dagegen automatisch eine warnende Exception.

Schon Kopfschmerzen? Auch noch nicht, wenn Sie erfahren, dass

```
0.1 + 0.2 === 0.3
```

in JavaScript niemals wahr ist? Diese Eigenheit ist nicht einmal eine Spezialität der Sprache, sondern ein generelles Thema von Fließkommazahlen. Rundungsfehler führen dazu, dass die Summe in Wahrheit 0.30000000000000004 ergibt. Ganzzahlige Datentypen hat JavaScript nicht. Die Lösung hier heißt also, entweder vorübergehend mit 100 zu multiplizieren oder nur auf Bereichen zu vergleichen. Ernsthaft!

Oder, falls Sie mal Ihre Kollegen zur Verzweiflung bringen wollen: Durch Vergleich mit `undefined` kann geprüft werden, ob ein Wert oder Feld überhaupt definiert wurde. Allerdings handelt es sich dabei um eine Variable des globalen Kontexts. Ihr Wert ist änderbar! Auch versehentlich.

Abb.4: Überraschungspotenziale der Ziel-Plattformen

Von all den genannten Fallstricken, die tiefsten Fallgruben bereiten vor allem die unterschiedlichen Plattformen. Unangefochtener Spitzenreiter ist der trotz biblischen Alters im Unternehmensumfeld kaum tot zu kriegende IE6. Hegen Sie nicht zu viele Hoffnungen: Auch die neueren Ausgaben machen ihm die Spitzenstellung nicht unbedingt streitig. Dank Dritt-Bibliotheken wie jQuery, YUI, Prototype oder MooTools muss sich ein Entwickler heute zum Glück etwas seltener damit auseinandersetzen.

Überleben auf neuem Boden

Aber Zeit für praktische Tipps! Als wichtigster Punkt lässt sich nur anmerken: Nehmen Sie sich Zeit für den Einstieg! In keiner anderen Sprache wird so viel entwickelt, ohne diese überhaupt richtig zu verstehen, wie in JavaScript. Sie haben bereits einen ersten Überblick über ihre zentralen Konzepte und Eigenarten und kennen einige ihrer teils tückischen, teils kuriosen Fallgruben. Darüber hinaus ist es hilfreich und sinnvoll, das eigene Wissen laufend weiter zu vertiefen.

Der zweite Eckpfeiler ist ein von Beginn an sorgfältiges und strukturiertes Vorgehen. Schwach typisierte Sprache haben ihre Stärken in der schnellen und zwangloseren Lösung von Problemstellungen mit überschaubarer Komplexität, oft für reine Ein-Mann-Unterfangen (Programming-in-the-small). Im Enterprise-Umfeld haben Sie es in der Regel jedoch mit komplexen Problemstrukturen, ganzen Teams und längeren Zeiträumen zu tun. Ein sorgfältiger Designentwurf und Modularisierung sind hier essenzielle Bausteine, damit sich Ihr Projekt nicht bereits kurz nach dem Start hoffnungslos verzettelt. Als professioneller Entwickler aus dem Java-Umfeld haben Sie hier große Vorteile, da Sie ein strukturiertes und sorgfältiges Vorgehen bereits gewohnt sind.

Unterstützend können hier Bibliotheken wie CoreJS [Next] sein, die Mittel für eine Java-ähnlichere Strukturierung bieten. Scheuen Sie sich auch nicht, Ihren Code zwischen großen Mengen Zeilenkommentaren zu verstecken. Ein vor der Auslieferung empfehlenswerter JavaScript-Kompressor filtert diese heraus. Sie werden später für jeden Tipp dankbar sein, den Sie sich selber hinterlassen haben, was Sie sich denn nun eigentlich an genau *dieser* Stelle gedacht haben.

Und last but not least: Wenn Sie in Ihrem Code auf Felder, Methoden oder Variablen zugreifen, prüfen Sie immer zuerst deren Existenz ab! Als dynamische Sprache haben Sie keine Sicherheit, dass das, was Sie erhalten, Ihren Erwartungen entspricht. Wenn Sie dennoch auf undefinierte Elemente zugreifen, erhalten Sie nicht `null`, sondern lösen damit in JavaScript eine Exception aus.

Zu den Waffen

Die richtigen Entwicklungswerkzeuge sind von zentraler Bedeutung. Bis vor wenigen Jahren konnte man im direkten Vergleich mit Java leicht den Eindruck gewinnen, dass Notepad gepaart mit JavaScript-`alert()`-Popup die offizielle Entwicklungsumgebung ist. Dies hat sich inzwischen jedoch grundlegend geändert. Sie müssen auch bei JavaScript nicht länger auf Syntax-Highlighting, Code-Vervollständigung, Debugging-Funktionalität oder Performance-Profiling verzichten, vorausgesetzt, Sie kennen die richtigen Werkzeuge. Die kommerzielle IntelliJ IDEA-Variante, das freie Firefox-Plug-in Firebug sowie die Entwicklerwerkzeuge des Chrome- und des Safari-Browsers sind schon mal ein gutes Starter-Kit. Nicht zu vergessen ein vollständiges Spektrum der Zielbrowser, da ein Abtesten auf allen Zielplattformen unabdingbar ist.

Setzen Sie Bibliotheken mit Bedacht ein. Sie können signifikant helfen, die Produktivität zu verbessern. Ihr Umfang und ihre Performance-Kosten wiegen in JavaScript-Applikationen jedoch deutlich schwerer als in der Java-Welt. Ebenfalls nicht fehlen sollte in Ihrer Werkzeugbox Crockfords Codequalitäts-Werkzeug JSLint. Es hilft Ihnen, problematische Stellen früh zu erkennen und einen sauberen Programmierstil einzuhalten. So akzeptiert der Vergleichsoperator problemlos `"1" == 1`. Bevorzugen Sie daher den „Ternary Equals Operator“ `===`, welcher strikt ohne Typkonvertierung vergleicht und damit auch näher an der Ihnen aus Java bekannten Semantik liegt. Auch bei (ungewollt) eingeführten globalen Symbolen aufgrund fehlender `var`-Präfixe oder dem fehlenden Semikolon am Zeilenende klopfen Ihnen JSLint und IntelliJ zuverlässig auf die Finger.

Ihr persönlicher Survival Guide

- Lernen Sie Sprache und Grammatik. Das vermeidet Missverständnisse.
- Wählen Sie die richtigen Werkzeuge.
- Testen Sie umfangreich. Fehler werden erst zur Laufzeit sichtbar.
- Halten Sie Ordnung. Ihr Code sollte gut dokumentiert, strukturiert und getestet sein.
- Nutzen Sie Frameworks sparsam. Hier drohen Seiteneffekte und Performance-Kosten.

Fazit

JavaScript ist nicht die beste Programmiersprache, dies wird keine Sprache je sein. Sie ist in ihrer Bedeutung aber auch nicht mehr aufzuhalten. Das haben inzwischen alle Beteiligten am Tisch erkannt. Seitdem sind deutliche Bemühungen erkennbar, diesem Anspruch auch mehr gerecht zu werden: Interpreter, Werkzeuge, Bibliotheken und nicht zuletzt die Sprache selbst haben viele lang überfällige Verbesserungen erfahren. Was nun fehlt, sind mehr Leute, die mit ihr auch professionell umgehen können. Seien auch Sie einer davon.

P.S.: Actually we do like JavaScript!

Links

[Cail07] `http://en.wikinews.org/wiki/Wikinews:Story_preparation/Interview_with_Robert_Cailliau`
[Eich06] `https://mail.mozilla.org/pipermail/es-discuss/2006-October/000133.html`
[Moz] `https://developer.mozilla.org/en/JavaScript`
[Next] `http://echo.nextapp.com/site/corejs`

Oliver Pehnke ist Softwarearchitekt und JavaScript-Pioneer im Java-zentrischen Umfeld der eXXcellent solutions. Ohne Ermüdung singt er seinen Kollegen von seinen waghalsigen JavaScript-Abenteuern am Lagerfeuer vor.
E-Mail: O.Pehnke@exxcellent.de

Benjamin Schmid ist Technology Advisor bei der eXXcellent solutions und hat für die dortigen Projekte in allen technologischen und methodischen Fragen immer gute Ratschläge parat. Seine Schwerpunkte liegen in Architektur, Code-Qualitätssicherung und GUI-Technologien.
E-Mail: B.Schmid@exxcellent.de.