

David Herman

JavaScript effektiv

**68 Dinge, die ein guter JavaScript-Entwickler
wissen sollte**

Mit einem Geleitwort von Brendan Eich, dem Erfinder von JavaScript



dpunkt.verlag

Inhaltsverzeichnis

Geleitwort	xiii
Vorwort	xvii
Danksagung	xxi
1 Darf ich vorstellen: JavaScript	1
Thema 1 Welche Version von JavaScript verwenden Sie?	1
Thema 2 Fließkommazahlen in JavaScript sind anders	7
Thema 3 Vorsicht bei der impliziten Typumwandlung	10
Thema 4 Verwenden Sie primitive Datentypen statt Objektwrappern	16
Thema 5 Vergleichen Sie unterschiedliche Typen nie mit ==	18
Thema 6 Achtung: JavaScript ergänzt automatisch Semikolons	22
Thema 7 JavaScript nutzt 16-Bit-Unicode	29
2 Gültigkeitsbereich von Variablen	33
Thema 8 Verwenden Sie das globale Objekt so wenig wie möglich	33
Thema 9 Vergessen Sie nicht, lokale Variablen zu deklarieren	36
Thema 10 Vermeiden Sie das Schlüsselwort with	38
Thema 11 Keine Angst vor Closures	41
Thema 12 Das müssen Sie kennen: Hoisting	44
Thema 13 Schaffen Sie lokale Gültigkeitsbereiche durch IIFEs	47
Thema 14 Gültigkeit von benannten Funktionsausdrücken	49
Thema 15 Verlässliche Gültigkeitsbereiche von lokalen Funktions- deklarationen	53
Thema 16 Vermeiden Sie es, Variablen mit eval zu erstellen	56
Thema 17 Verwenden Sie eval lieber indirekt	57

3 Funktionen	61
Thema 18 Die Unterschiede zwischen Funktionen, Methoden und Konstruktoren	61
Thema 19 Keine Angst vor Funktionen höherer Ordnung	64
Thema 20 Rufen Sie Methoden mit benutzerdefiniertem Empfänger mit call auf	68
Thema 21 Rufen Sie variadische Funktionen mit apply auf	70
Thema 22 Erstellen Sie variadische Funktionen mit arguments	72
Thema 23 Ändern Sie niemals das arguments-Objekt	73
Thema 24 Speichern Sie Verweise auf arguments in einer Variable	75
Thema 25 Extrahieren Sie Methoden mit festem Empfänger per bind	77
Thema 26 Nutzen Sie bind beim Currying	79
Thema 27 Kapseln Sie Code mit Closures, nicht mit Strings	81
Thema 28 Verlassen Sie sich nicht auf die toString-Methode	83
Thema 29 Vorsicht, wenn Sie den Call Stack inspizieren!	84
4 Objekte und Prototypen	87
Thema 30 Achten Sie auf den Unterschied zwischen prototype, getPrototypeOf und __proto__	87
Thema 31 Verwenden Sie lieber Object.getPrototypeOf statt __proto__	91
Thema 32 Ändern Sie niemals __proto__!	92
Thema 33 Erstellen Sie Konstruktoren, die auch ohne new funktionieren	93
Thema 34 Speichern Sie Methoden mithilfe von Prototypen	96
Thema 35 Speichern Sie private Daten mithilfe von Closures	98
Thema 36 Speichern Sie den Instanzstatus nur in Instanzobjekten	100
Thema 37 this sollten Sie kennen!	103
Thema 38 Rufen Sie Superklassenkonstruktoren von Subklassenkonstruktoren aus auf	106
Thema 39 Eigenschaftsnamen aus der Superklasse sollten Sie niemals wiederverwenden!	110
Thema 40 Vermeiden Sie die Vererbung von Standardklassen	112
Thema 41 Prototypen sind »richtige« Implementierungen	114
Thema 42 Das brauchen Sie nicht: Unbesonnenes Monkey-Patching	115
5 Arrays und Dictionaries	119
Thema 43 Erstellen Sie schlanke Dictionaries mit Object	119
Thema 44 Schützen Sie sich mithilfe von Null-Prototypen vor einer Prototyp-Verunreinigung	123

Thema 45	Schützen Sie sich mit <code>hasOwnProperty</code> vor Prototyp-Verunreinigungen	124
Thema 46	Verwenden Sie für geordnete Collections lieber Arrays statt Dictionaries	129
Thema 47	Fügen Sie niemals aufzählbare Eigenschaften zu <code>Object.prototype</code> hinzu!	132
Thema 48	Ändern Sie Objekte nicht während einer Aufzählung	134
Thema 49	Verwenden Sie <code>for</code> -Schleifen statt <code>for...in</code> -Schleifen, wenn Sie über Arrays iterieren	139
Thema 50	Verwenden Sie lieber Iterationsmethoden als Schleifen	140
Thema 51	Generische Arraymethoden für arrayähnliche Objekte wiederverwenden	145
Thema 52	Verwenden Sie lieber Arraylitterale statt des Arraykonstruktors	148
6	Erstellung von Bibliotheken und APIs	149
Thema 53	Bemühen Sie sich um eine einheitliche Schreibweise	149
Thema 54	Behandeln Sie »undefined« als »nicht vorhanden«	151
Thema 55	Zu viele Parameter? Nutzen Sie Optionsobjekte!	155
Thema 56	Vermeiden Sie unnötige Zustände	160
Thema 57	Verwenden Sie strukturelle Typisierung für flexible Schnittstellen	164
Thema 58	Unterscheiden Sie Arrays und arrayähnliche Objekte	168
Thema 59	Vermeiden Sie übermäßige Typumwandlung	172
Thema 60	Unterstützen Sie Method Chaining	176
7	Nebenläufigkeit	179
Thema 61	Blockieren Sie die Event Queue nicht, wenn I/O stattfindet	180
Thema 62	Verwenden Sie verschachtelte oder benannte Callbacks für die asynchrone Abarbeitung	183
Thema 63	Denken Sie an die Fehlerbehandlung!	188
Thema 64	Nutzen Sie Rekursion für asynchrone Schleifen	191
Thema 65	Blockieren Sie die Event Queue bei längeren Berechnungen nicht	195
Thema 66	Steuern Sie nebenläufige Operationen mit einem Zähler	199
Thema 67	Rufen Sie asynchrone Callbacks niemals synchron auf!	204
Thema 68	Verwenden Sie Promises für eine sauberere asynchrone Logik	206
	Index	211