

# WebSockets

Moderne HTML5-Echtzeitanwendungen entwickeln

Bearbeitet von  
Peter Leo Gorski, Luigi Lo Iacono, Hoai Viet Nguyen

1. Auflage 2015. Buch. 281 S.  
ISBN 978 3 446 44371 6  
Format (B x L): 18 x 24,5 cm  
Gewicht: 626 g

[Weitere Fachgebiete > EDV, Informatik > Programmiersprachen: Methoden > Webprogrammierung](#)

schnell und portofrei erhältlich bei

  
DIE FACHBUCHHANDLUNG

Die Online-Fachbuchhandlung [beck-shop.de](http://beck-shop.de) ist spezialisiert auf Fachbücher, insbesondere Recht, Steuern und Wirtschaft. Im Sortiment finden Sie alle Medien (Bücher, Zeitschriften, CDs, eBooks, etc.) aller Verlage. Ergänzt wird das Programm durch Services wie Neuerscheinungsdienst oder Zusammenstellungen von Büchern zu Sonderpreisen. Der Shop führt mehr als 8 Millionen Produkte.



Leseprobe

Peter Leo Gorski, Luigi Lo Iacono, Hoai Viet Nguyen

WebSockets

Moderne HTML5-Echtzeitanwendungen entwickeln

ISBN (Buch): 978-3-446-44371-6

ISBN (E-Book): 978-3-446-44438-6

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44371-6>

sowie im Buchhandel.

# Inhalt

<b>Vorwort</b> .....	<b>IX</b>
<b>Danksagung</b> .....	<b>XI</b>
<b>1 Einleitung</b> .....	<b>1</b>
1.1 Wie ist das Buch strukturiert? .....	3
1.2 Wer sollte das Buch lesen? .....	3
1.3 Wie sollte mit dem Buch gearbeitet werden? .....	4
<b>2 HTTP: Hypertext Transfer Protocol</b> .....	<b>5</b>
2.1 Grundlegendes .....	5
2.1.1 ISO/OSI-Referenzmodell für Kommunikationssysteme .....	6
2.1.2 Vereinfachtes OSI-Modell .....	8
2.1.3 Dienst .....	9
2.1.4 Protokoll .....	10
2.1.5 Kommunikationsfluss .....	11
2.2 HTTP en détail .....	13
2.2.1 Kommunikationsablauf .....	13
2.2.2 Textbasiert .....	14
2.2.3 Aufbau von Request-Nachrichten und Protokollelemente .....	16
2.2.4 Aufbau von Response-Nachrichten und Statuscodes .....	20
2.2.5 Zustandslos .....	24
<b>3 Höhere Interaktivität und Echtzeitfähigkeit</b> .....	<b>25</b>
3.1 XMLHttpRequest (XHR) .....	25
3.2 Polling .....	27
3.3 Long-Polling .....	28
3.4 Comet .....	29
3.5 Server-Sent Events .....	31
3.6 Bewertung der Verfahren .....	33

<b>4</b>	<b>Die Leitung: Das IETF WebSocket-Protokoll</b>	<b>35</b>
4.1	Was bisher geschah	35
4.2	Opening-Handshake – WebSocket-Verbindung aufbauen	36
4.3	WebSocket-Frames	39
4.4	Fragmentierung	42
4.5	Maskierung	43
4.6	Datenframes	45
4.6.1	Frames mit Textdaten	45
4.6.2	Frames mit Binärdaten	47
4.7	Control-Frames	48
4.7.1	Ping-Frame	48
4.7.2	Pong-Frame	49
4.7.3	Close-Frame	49
4.8	Closing-Handshake – WebSocket-Verbindung schließen	53
4.9	Tools zur Protokollanalyse	54
4.9.1	Wireshark	54
4.9.2	Fiddler	60
4.9.3	Chrome Developer Tools	68
4.9.4	Chrome Network Internals	70
<b>5</b>	<b>Der Client: Die W3C WebSocket-API</b>	<b>75</b>
5.1	Was bisher geschah	75
5.2	Browserunterstützung	76
5.3	Namensschema	79
5.4	Objekterzeugung und Verbindungsaufbau	80
5.5	Zustände	81
5.6	Event-Handler	83
5.7	Erstes vollständiges Programmchen	86
5.8	Attribute	89
5.9	Datenübertragung	92
5.9.1	Übertragung textbasierter Daten	93
5.9.2	Übertragung binärer Daten	94
5.10	Verbindungsabbau	96
5.10.1	Verbindung beenden	96
5.10.2	Close-Event	96
5.11	Ausblick: HTTP 2.0/SPDY	97

<b>6</b>	<b>Der Server: Sprachliche Vielfalt</b> .....	<b>101</b>
6.1	Übersicht verfügbarer WebSocket-Implementierungen .....	101
6.2	Node.js .....	102
6.2.1	Installation von Node.js .....	104
6.2.2	WebSocket.io .....	106
6.2.3	Socket.io .....	109
6.2.4	WebSocket-Node .....	113
6.3	Vert.x.....	119
6.4	Play Framework .....	122
6.4.1	Installation .....	122
6.4.2	Anlegen eines neuen Play-Projekts .....	123
6.4.3	Anatomie einer Play-Anwendung.....	123
6.4.4	Die Play-Konsole .....	124
6.4.5	Controller in Play.....	124
6.4.6	Views in Play .....	125
6.4.7	Routes in Play .....	126
6.4.8	Vom Controller zur View .....	127
6.4.9	Erstellen eines Echo-Servers in Play .....	129
6.5	JSR 356 .....	132
6.5.1	JSR 356-basierender WebSocket-Server .....	136
6.5.2	JSR 356-basierender WebSocket-Client .....	140
<b>7</b>	<b>WebSockets in der Praxis</b> .....	<b>147</b>
7.1	Performance und Skalierbarkeit .....	147
7.1.1	Test-Echo-Server .....	148
7.1.2	Testclient für die Auslastung.....	151
7.1.3	Testclient für die Zeitmessung .....	152
7.1.4	Einrichtung der Testumgebung .....	153
7.1.5	Ergebnisse.....	157
7.2	Sicherheit .....	158
7.2.1	Same Origin Policy .....	158
7.2.2	Vertrauliche Kommunikation .....	159
7.2.3	Authentifizierung .....	162
7.2.4	Firewalls und Proxys .....	183
7.2.5	Mögliche Gefährdungen .....	186
7.2.6	Bewertung der Sicherheitslage .....	195
7.3	Mit Haken und Ösen .....	196
7.3.1	Pings und Pongs .....	196
7.3.2	Das Cache-Problem im Internet Explorer 10 .....	197

<b>8</b>	<b>Beispielanwendungen</b> .....	<b>201</b>
8.1	Fernbedienung von Webanwendungen mit einer Smartphone-/Tablet-Fernbedienung .....	202
8.2	Chat-System .....	211
8.3	Heatmap für Usability-Tests .....	216
8.4	Überwachungskamera per Webcam .....	221
	<b>Schlusswort</b> .....	<b>231</b>
<b>A</b>	<b>XHR-Objekt</b> .....	<b>233</b>
<b>B</b>	<b>Chrome Developer Tools auf Android</b> .....	<b>235</b>
<b>C</b>	<b>Umgebungsvariablen definieren</b> .....	<b>241</b>
C.1	Mac OS/Linux mit Bash .....	241
C.2	Windows .....	242
<b>D</b>	<b>Express.js</b> .....	<b>247</b>
D.1	Anatomie einer Express.js-Anwendung .....	248
D.2	Die Anwendungslogik in der Datei app.js .....	250
D.3	Die Jade-Template-Engine .....	252
D.4	Express.js mit WebSocket-Servern verbinden .....	254
	<b>Literatur</b> .....	<b>257</b>
	<b>Stichwortverzeichnis</b> .....	<b>263</b>

# Vorwort

Netzwerke umgeben uns quasi überall. Viele Dinge des täglichen Geschäfts- und Privatlebens sind ohne Kommunikation über ein digitales Datennetz nicht mehr vorstellbar. Zur Lingua franca der Protokolle der Anwendungsschicht hat sich in den letzten Jahren das HTTP gemausert. Angeschoben durch den immensen Erfolg des Web bedienen sich heute eine Vielzahl anderer Anwendungen dem HTTP. Darunter z.B. die Lokalisierung und Ansteuerung von Geräten im Hausnetz via UPnP, die Speicherung von Daten in der Cloud alla Dropbox und vermehrt auch das TV. Die Gründe hierfür sind vielfältig. Der simple Aufbau von HTTP und die zahlreich verfügbaren Implementierungen spielen dabei sicherlich eine Hauptrolle. In den Nebenrollen finden sich Darsteller wie die durchgängige Firewall-Freundlichkeit und neuerlich auch die ubiquitäre Verbreitung von HTTP in Geräten jeden Couleurs. Ist man mit seinen Anwendungen und Diensten an Reichweite interessiert, führt derzeit kein Weg an HTTP vorbei.

Neben all dieser Euphorie ist es ratsam, alle Eigenschaften des neuen Gefährten zu kennen, da sich daraus auch seine Grenzen ableiten lassen. An der Zustandslosigkeit von HTTP lässt sich dies schön verdeutlichen. Erlaubt es auf der einen Seite ein vereinfachtes Caching und globale Skalierbarkeit durch Content Distribution Networks (CDN), erschwert es die Implementierung von Warenkörben in E-Commerce Anwendungen. Letzteres hat man im Laufe der Zeit gut in den Griff bekommen. Für andere Nachteile haben sich dagegen noch keine Patentrezepte etabliert. Hierzu zählt z.B. die unidirektional ausgelegte Kommunikation. Gemäß der Protokollspezifikation meldet sich immer der Client mit einer Anfrage an den Server, der auf diese mit einer entsprechenden Antwort reagiert. Dass sich der Server selbstinitiativ bei einem Client meldet, ist nicht vorgesehen. Dies liegt auch darin begründet, dass die Verbindung zwischen Client und Server nicht dauerhaft, sondern nur für die Zeit des Austausches von Anfragen und ihrer Antworten besteht. In modernen Anwendungen ist das Pushen von Nachrichten oder Daten vom Server zum Client eine häufige Anforderung, die es technisch umzusetzen gilt. Genau an dieser Stelle setzt die WebSocket-Technologie an.

Dem Wildwuchs an Ansätzen, die zum Aufbrechen der Kommunikationseinbahnstraße von HTTP vorgeschlagen wurden und uneinheitlich zur Verwendung gekommen sind, wird mit WebSockets eine standardisierte Lösung entgegengesetzt. Der Hunger nach einem derartigen Standard war groß. Dies lässt sich daran ablesen, dass die Unterstützung von WebSockets durch alle einschlägigen Industrie Größen bekannt gegeben wurde, als sich die Standardisierung noch in den Kinderschuhen befand. Bereits heute lässt sich kein Webbrowser vermissen, der nicht WebSockets mit an Bord hat. Obwohl die Standardisierung

zum aktuellen Zeitpunkt noch nicht abgeschlossen ist, kann davon ausgegangen werden, dass es sich hierbei in nicht allzu langer Zeit um einen etablierten Standard handeln wird.

Das vorliegende Buch stellt diesen neuen Grundpfeiler für moderne Anwendungen auf Grundlage von HTTP vor. Dabei geht es auf alle Aspekte der WebSocket-Technologie in angemessener Tiefe ein. Meine vorweggenommene Motivation wird zu Beginn des Buches, angereichert mit dem notwendigen Background, aufgegriffen und somit auf das eigentliche Thema hingeführt. Dann stehen die WebSockets im Rampenlicht. Mit dem unter den Fittichen der IETF stehendem WebSocket-Protokoll geht es los. Hier wird Bit genau beleuchtet, wie sich die verschiedenen Frames zwischen Client und Server bewegen. Das zeigt unter anderem, wie viel effektiver die Übertragung mittels WebSockets im Vergleich zu HTTP ist, wenn der Overhead der HTTP-Header wegfällt. Im Anschluss lernt man die von der W3C verantwortete JavaScript API kennen, mit der sich die Clientseite einer verteilten WebSocket-Anwendung in einem Browser implementieren lässt. Das ist allerdings noch nicht alles, was für ein ganzheitliches Projekt benötigt wird. Die Serverseite fehlt und hier sieht das Bild nicht ganz so einheitlich aus. Fehlende Standards machen es erforderlich, sich die vom ausgewählten Framework entsprechend bereitgestellten APIs anzueignen. Diesem Umstand Rechnung tragend, werden in dem Buch verschiedene serverseitige Frameworks behandelt, die für Java und JavaScript bereitstehen. Abgerundet wird das Ganze durch zahlreiche Beispiele, die auch größere Zusammenhänge nachvollziehbar und verständlich machen.

Mir hat die Lektüre das notwendige Know-how und das Verständnis in die neuen Möglichkeiten vermittelt, mit dem ich nun spannenden neuen Projekten unter Verwendung von WebSockets entgegenblicke. Ich wünsche Ihnen ebenso viel Erkenntnisgewinn beim Lesen, Ihr

*Alexander Leschinsky*  
Geschäftsführer der G&L Systemhaus GmbH

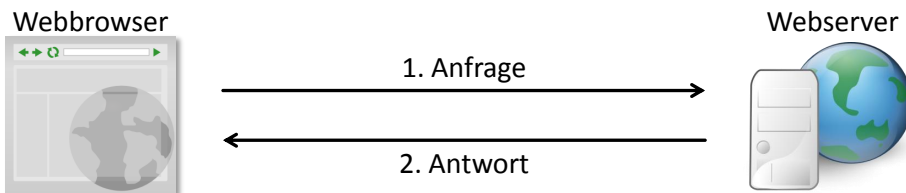
Köln, im November 2014



# 1

## Einleitung

Einer der technischen Hauptakteure im Web ist HTTP, das *Hypertext Transfer Protocol* [FGM<sup>+</sup>99], und das ist so, weil HTTP den genauen Kommunikationsablauf zwischen Webbrowser und Webserver festlegt. Genau wie ein Protokoll für eine königliche Gala zu Hofe alle Regeln in Bezug auf Kleidung, Etikette und Umgangsformen zusammenfasst, so legt ein Kommunikationsprotokoll alle Regeln zum Nachrichtenaustausch fest. Anders als bei Hofe sind das bei einem Kommunikationsprotokoll aber vielmehr Regeln in Bezug auf den Aufbau, das Format und die Codierung der ausgetauschten Nachrichten. Außerdem muss in einem Protokoll festgelegt sein, wer die Kommunikation beginnt und wie diese dann Nachricht für Nachricht bis zum Kommunikationsende abläuft. HTTP verwendet hier einen einfachen Request-Response-Nachrichtenaustausch. Dieser wird immer vom Webclient durch eine Anfrage (engl. *Request*) initiiert und entsprechend vom Webserver mit einer Antwort (engl. *Response*) beantwortet. Sobald Sie also z. B. eine Webadresse – die URL – in die Adresszeile eines Webbrowsers eingeben oder einen Bookmark bzw. Link anklicken, setzen Sie damit eine Anfrage an einen Server ab, der diese dann bearbeitet und die Antwort daraufhin zurücksendet (siehe [Bild 1.1](#)).



**Bild 1.1** Grundlegender HTTP-Nachrichtenaustausch

In diesem einfachen Request-Response-Nachrichtenaustausch liegen viele Gründe für den großen Erfolg von HTTP und nicht zuletzt des Webs. Dazu zählen u. a. die guten Skalierungseigenschaften, denen wir die Größe des Webs verdanken. Wie Sie sich aber sicher jetzt schon denken, bringt das Nachrichtenpaar nicht nur Vorteile mit sich. Und tatsächlich ist die Protokollregel, dass der Request immer vom Client ausgehen muss, eine Zwangsjacke, die bestimmte Bewegungsfreiheiten in Form von Kommunikationsmustern nicht zulässt. Der Teil einer Webanwendung, der auf dem Server ausgeführt wird, kann sich nämlich nicht von sich aus an einen Client wenden. Diese Eigenschaft des Protokolls schränkt folglich Anwendungsfälle ein, in denen die Serverseite Statusänderungen an die betroffenen Clients weiterreichen können muss. Typische Beispiele hierfür sind Chatanwendun-

gen, Finanzdatenströme, Webmail, Sportticker, das Monitoring von Haushalt und Industrie oder auch Internetauktionen. All diese Anwendungen teilen die Gemeinsamkeit, dass (häufig in relativ kurzen Zeitabschnitten) Daten auf dem Server hinzukommen bzw. sich ändern und dies unmittelbar an die angeschlossenen Clients bekannt gegeben werden muss. Liegt Ihnen eine derartige Anforderung auf dem Tisch, dann würden Sie sich eine standardisierte Technik wünschen, mit der Sie die Schranken der Serverseite öffnen und die entsprechenden Clients ansprechen können.

HTTP bietet für derartige Kommunikationsmuster keine native Unterstützung. Wenn Sie sich in den letzten Jahren aus dieser Zwangsjacke hätten befreien und die Kommunikationseinbahnstraße zur Umsetzung von Serverbenachrichtigungen hätten aufbrechen wollen, so wären Sie auf Grundlage des Verfügbaren auf eine (selbst) konstruierte Lösung angewiesen gewesen. Die Konstruktionen, die findige Webentwickler dabei gefunden haben, sind vielfältig. Prinzipiell können Sie den Standardfunktionsumfang durch Browser-Plug-ins und serverseitige Erweiterungen aufbohren und damit proprietäre Lösungen zur Serverbenachrichtigung implementieren. Der erste Ansatz überhaupt kam tatsächlich auch aus dieser Technologiegattung und bediente sich Java Applets und der LiveConnect-Schnittstelle zur Anbindung an JavaScript-Funktionen im Browser. Seit HTML5 besteht allerdings ein starker Trend weg von Browsererweiterungen à la Plug-in oder Add-on. Schützenhilfe kam zudem aus dem Umfeld der Smartphones und Tablets, das sich gegen eine Unterstützung von Flash und Silverlight entschieden hat, was schließlich selbst Adobe und Microsoft dazu bewegt hat, die Entwicklungen ihrer Technologien für mobile Plattformen einzustellen. Daher wollen wir auch nicht weiter auf das Auslaufmodell der Plug-in-basierten Ansätze eingehen. Weitere Konstruktionen basieren auf Ansätzen, die den Webbrowser in regelmäßigen Zeitabständen beim Webserver nach Veränderungen fragen lassen. Diese aktuell weit verbreiteten Verfahren haben Nachteile, was das Kommunikationsaufkommen anbelangt. Dennoch haben sie unter Umständen ihre Berechtigung. Sie lernen diese Ansätze daher kennen und insbesondere diese zu bewerten, um die richtige Technologie für Ihr Entwicklungsvorhaben auswählen zu können. Nichtsdestotrotz handelt es sich aber weitestgehend um Notlösungen, die durch die in der HTML5-Standardisierung befindlichen alternativen Lösungswege abgelöst werden. Dazu gehören zum einen *Server-Sent Events* (SSE) [Hic12a] und zum anderen *WebSockets* (WS) [FM11a]. Die verschiedenen Bestandteile des Standards, die HTML5 einführt, sind thematisch gruppiert und mit einem Logo versehen worden [W3C14]. SSE und WS sind in der sogenannten Connectivity-Gruppe. Das Logo sehen Sie in einer leicht abgewandelten Form in Bild 1.2.



**Bild 1.2** Das Logo für die Connectivity-Standards Server-Sent Events und WebSockets, platziert auf dem HTML5-Schild [W3C14]

WebSockets stehen im Fokus dieses Buches, da Sie damit neben den Serverbenachrichtigungen eine Vielzahl weiterer Anwendungsfälle realisieren können, zu denen z. B. Spiele, jegliche Art von Konversationen (Text, Sprache, Sprache mit zusätzlichem Video), Kollaborationen sowie das Benutzermonitoring im Rahmen von Usabilitystudien zählen.

## ■ 1.1 Wie ist das Buch strukturiert?

WebSockets und die damit einhergehenden neuen Entwicklungsmöglichkeiten stehen somit im Zentrum des vorliegenden Buches und an diese wollen wir uns mit Ihnen Schritt für Schritt ranpirschen. In [Kapitel 2](#) führen wir uns dazu zunächst die notwendigen Grundlagen in Bezug auf HTTP zu Gemüte. Kennen Sie diese schon, können Sie getrost die Abkürzung zu [Kapitel 3](#) nehmen, in dem wir uns mit den Mechanismen befassen, mit denen eine höhere Interaktivität und Echtzeitfähigkeit bei Webanwendungen erreicht werden kann. In [Kapitel 4](#) widmen wir uns dann dem WebSocket-Protokoll und in [Kapitel 5](#) geht es anschließend ans Eingemachte. Hier lernen wir die WebSocket API kennen und programmieren mit JavaScript erste Beispiele für WebSocket-Clientanwendungen in Webbrowsern. In [Kapitel 6](#) wenden wir uns den WebSocket Implementierungen auf der Serverseite sowie gebräuchlichen Frameworks zu. Weitere wichtige Aspekte folgen in [Kapitel 7](#), das das Testen von verteilten WebSocket-basierten Applikationen beschreibt und auf Eigenschaften der Performance eingeht. Was niemals vergessen werden darf, sind Sicherheitsaspekte, insbesondere wenn die Anwendung aus verteilten Komponenten zusammengesetzt ist, die über offene Netze miteinander gekoppelt sind, wie das im Web quasi immer der Fall ist. Daher wollen wir uns damit ebenfalls in [Kapitel 7](#) auseinandersetzen und die Besonderheiten von WebSockets in diesem Kontext herausstellen. In [Kapitel 8](#) werden wir dann das Gelernte einsetzen und verschiedene „größere“ und vollständige Anwendungen implementieren. Diese haben wir dabei so gewählt, dass damit ein möglichst großes Spektrum an Möglichkeiten aufgezeigt wird. Wir spannen den Bogen von einer generischen Fernsteuerung für Webanwendungen, über ein klassisches Chatsystem über eine Heatmap für Usability-Tests bis hin zu einer Überwachungskamera per Webcam.

## ■ 1.2 Wer sollte das Buch lesen?

In erster Linie richtet sich das vorliegende Buch an den Entwickler in Ihnen. Möchten Sie mehr über WebSockets wissen und verstehen, was Sie mit diesen so alles in Ihren Webprojekten anstellen können, dann sollten Sie hier fündig werden. Da der Koffer an Werkzeugen, Technologien und Sprachen kaum unterschiedlicher gefüllt ist als bei Webentwicklern, stellen wir auf keine spezifische Umgebung ab, sondern versuchen dieser farbenfrohen Vielfalt gerecht zu werden. So finden sich Beispiele auf Basis von Node.js, vertx und JSR 356 im Buch wieder. Als Programmiersprachen verwenden wir Java und JavaScript. Neben der gängigen Vorstellung und Erläuterung der Buchinhalte sind die zahlreichen Beispielanwendungen in [Kapitel 8](#) eine Stärke des vorliegenden Buches, mit denen das Erlernte geübt und nachvollzogen werden kann. Webentwickler sollten somit in die Lage versetzt werden,

einen umfassenden Einstieg in die Thematik zu bekommen und anhand der exemplarischen Anwendungen einen tief gehenden Einblick in den Einsatz von WebSockets zu erhalten.

Neben Webentwicklern eignet sich das Buch auch für Lehrveranstaltungen an Hochschulen und damit für Studierende verschiedener Fachrichtungen, die mit WebSockets innovative Anwendungen im Web entwickeln wollen. Zudem profitieren Informations- und Softwarearchitekten sowie Konzepter und (technische) Projektleiter vom vorliegenden Buch, da ein Grundverständnis der Kommunikationsmuster, die mit WebSockets realisiert werden können, für Architekturen und Konzepte von Bedeutung sein können. Auch hierfür haben wir versucht, mit den in [Kapitel 8](#) beschriebenen Beispielanwendungen den Horizont aufzuzeichnen.

## ■ 1.3 Wie sollte mit dem Buch gearbeitet werden?

Das Buch führt sukzessive in die Thematik ein und setzt nur wenige Vorkenntnisse voraus. Wenn Sie über kaum bis wenige Erfahrungen in der Webentwicklung verfügen, raten wir Ihnen, sich an diesem Aufbau zu orientieren und sich Kapitel für Kapitel einzuarbeiten. Nach den Grundlagen werden in den einführenden Kapiteln erste kleinere Beispielanwendungen gezeigt und erläutert. Wir empfehlen, diese nachzuprogrammieren und die Quelltexte im Detail nachzuvollziehen. Ist das Fundament gelegt, geht es mit weiterführenden Aspekten weiter, die selektiv durchgearbeitet werden können. Das Durcharbeiten der Beispielanwendungen ist wiederum sehr empfehlenswert. Es trainiert auf der einen Seite die im Buch vermittelten Inhalte. Bei Bedarf können Unklarheiten nochmals dediziert nachgelesen werden. Auf der anderen Seite zeigen sie die Potenziale von WebSockets auf. Leser mit fundiertem Vorwissen können sicherlich die Grundlagenkapitel überspringen und sich direkt mit den originären Inhalten und den Beispielanwendungen befassen.



**<http://websocket101.org/>**

Wir freuen uns von Ihnen zu hören! Ob Lob oder Kritik, ob Frage oder Anregung, ob Hinweis oder Verbesserung, wir glauben, dass das Buch von all dem profitieren kann, wenn es konstruktiv eingebracht wird. Dazu wollen wir unser Nötigstes tun und freuen uns auf Ihr Mitwirken. Aktuelles wird dabei zeitnah auf der buchbegleitenden Webseite bereitgestellt. Schauen Sie doch häufiger mal auf

*<http://websocket101.org/>*

vorbei. Dort finden Sie auch unsere Kontaktinformationen.

An dieser Stelle bleibt uns nur noch, Ihnen viel Spaß beim Lesen und Nachprogrammieren sowie viel Erfolg bei den eigenen Entwicklungen zu wünschen. ■

# Stichwortverzeichnis

## Symbole

100Base-TX [11](#)  
802.11 [11](#)  
802.3 [11](#)

## A

ABNF → angereicherte Backus-Naur-Form  
Add-on [2](#)  
Administrationsseite [219](#)  
Ajax [26](#), [172](#)  
aktive Inhalte [158](#)  
Alexa Top [187](#)  
Amazon Web Services [148](#), [153](#)  
Android [235](#)  
angereicherte Backus-Naur-Form [79 f.](#)  
Anmeldedialog [162](#)  
Annotation [181](#)  
Anwendungsschicht [6 f.](#), [12](#)  
Apache-Webserver [185](#)  
Application Layer ⇒ Anwendungsschicht  
Applikationsserver [134](#)  
ArrayBuffer [92](#), [94 f.](#)  
ArrayBufferView [92](#), [94](#)  
ASCII-Zeichen [57](#)  
Auslastung [147 f.](#), [151](#)  
Authentifizierung [162](#)  
Authentizität [159](#)  
AWS → Amazon Web Services

## B

Base64-Codierung [16](#), [37](#), [163](#)  
Bash [241](#)  
Bayeux-Protokoll [30](#)  
Betaversion [134](#)  
BiDirectional or Server-Initiated HTTP [36](#), [78](#),  
[91](#)  
bidirektional [35](#)  
Bild [15](#), [116](#)  
Binärdaten [16](#), [47](#), [76](#), [94](#), [113](#)

binäre Codierung [14](#)  
Binary Frame [119](#)  
Binary Large Object → BLOB  
Bitübertragungsschicht [6](#)  
BLOB [92](#), [94](#), [116](#), [226](#)  
Bösartige Services [194](#)  
Bootstrap [219](#)  
Botnetz [190](#)  
Browserunterstützung [76](#)  
Buffer [91](#), [93](#)  
BufferedReader [15](#)  
Button [115](#), [207](#), [213](#)

## C

CA → Zertifizierungsstelle  
Cache [22](#), [163](#), [198](#)  
Caching-Poisoning [184](#), [187](#)  
Callback-Funktion [103](#), [149](#), [151 f.](#), [176](#)  
canvas [95](#)  
<canvas>-Tag [206](#)  
Cascading Style Sheets [158](#), [209](#)  
Chat [211](#)  
Chrome Developer Tools [68](#), [117](#), [235](#)  
– Analyse [69](#)  
– Installation [68](#)  
Chrome Network Internals [70](#)  
Clamp [96](#)  
Client [13](#), [75](#)  
Close-Event [96](#)  
Close-Frame [41](#), [49](#), [53](#), [97](#)  
– Statuscode [51](#), [96](#)  
– Statuscodes [51](#)  
Closing-Handshake [53](#)  
CoffeeScript [119](#)  
Comet [29](#)  
Connectivity-Gruppe [2](#), [30](#)  
Continuation Frames [119](#)  
Control-Frames [48](#)  
Cookie [122](#), [166 f.](#), [172](#), [176](#), [193](#)

CORS → Cross-Origin Resource Sharing  
 Cross-Origin-Angriff 192  
 Cross-Origin Resource Sharing 29  
 Cross-Origin-Zugriffe 159  
 Cross-Site-Request-Forgery 159, 193  
 Cross-Site-Scripting 158 f., 188, 190, 194  
 Cross-Site-Tracing 19  
 Cross-Site WebSocket Hijacking 193  
 CSRF → Cross-Site-Request-Forgery  
 CSS → Cascading Style Sheets

## D

Darstellungsschicht 6 f.  
 Data Link Layer ⇒ Sicherungsschicht  
 Datei-Handler 155  
 Datenübertragung 92  
 Datenframes 40, 45  
 Datenkompression 91  
 Datenvolumen-Overhead 16  
 Deflate-Algorithmus 91  
 Demaskierung 44  
 Denial of Service 187, 188, 195  
 Dienst 9  
 DNS 11  
 DNS-Anfragen 71  
 Domain 16

## E

Early Draft 132, 134  
 EC2 153  
 echo 242  
 Echo-Server 58, 147  
 Echtzeit-Kommunikation 202  
 Echtzeit-Remote-Shell 191  
 Echtzeitfähigkeit 25  
 Eclipse 127  
 ECMAScript → JavaScript  
 Endpunkt 149 f., 205  
 Entführung der Client-Authentifizierung 193  
 Erweiterungen → Extensions  
 Escape-Sequenz 15  
 Ethernet 11  
 Event-Server 202  
 Executive Committee 132  
 Express.js 216, 247  
 – app.js 250  
 – Formularbasierte Authentifizierung 168, 172, 176  
 – Instanz 254  
 – Layout 253  
 – Methode  
 – cookie.parse() 174

– listen() 254  
 – Modul  
 – session 168  
 – store 168  
 – Ordnerstruktur 248  
 – package.json 249  
 – Request-Handler 251  
 – Route-Handler 169 f.  
 – Webserver 205  
 – WebSocket-Server 254  
 Extended payload length 41, 47, 188  
 Extensions 91

## F

Fallback 33, 79, 109, 204  
 Fehlerbegründung 85  
 Fehlercode 85  
 Fernbedienung 202  
 Fiddler 60  
 – Analyse 66  
 – Einrichtung 61  
 – Installation 61  
 – Protokollierung 65  
 FIN-Flag 40, 42  
 Final Release 133  
 Firewall 183 f.  
 Formularbasierte Authentifizierung 166  
 Fragment 17, 40  
 Fragmentierung 42  
 Framegröße 113 f.  
 FTP 11

## G

GlassFish 133, 135  
 Globally Unique Identifier 38  
 grep 157  
 Groovy 119  
 GUID → Globally Unique Identifier  
 gzip 15

## H

Hashfunktion 164  
 Header 59  
 Heartbeats 110  
 Heatmap 216  
 Hexdump 57  
 HTML-Content 125  
 HTML-Datei 107  
 HTML-Element 89  
 HTML-Formular 166  
 HTML-Seite 150  
 HTML5 2, 76

- HTTP → Hypertext Transfer Protocol
- HTTP 2.0 97
- HttpOnly 172
- HyBi → BiDirectional or Server-Initiated HTTP
- Hypertext Transfer Protocol 1, 5, 13, 97
  - Anfrage 1, 12 ff., 22, 25
  - Anfragezeile 16
  - Antwort 1, 12, 24, 60
  - Authentifizierung 162
  - Basic Authentication 163
  - Content-Type 32, 104
  - Digest Authentication 164
  - Header
    - Acces-Conroll-Allow-Origin 192
    - Allow 18
    - Authorization 163
    - Cache-Control 199
    - Connection 29, 36
    - Origin 36, 159, 190, 194
    - Sec-WebSocket-Accept 37 f.
    - Sec-WebSocket-Key 36 f.
    - Sec-WebSocket-Protocol 37, 81, 184
    - Sec-WebSocket-Version 36
    - Upgrade 21, 36, 161
    - WWW-Authenticate-Header 164
  - Methode
    - CONNECT 184
    - DELETE 18
    - GET 17, 25, 32
    - HEAD 18
    - OPTIONS 18
    - PATCH 19
    - POST 17
    - PUT 18
    - TRACE 19
  - Reason 20
  - Status 20
  - Status-Line 20
  - Statuscode 20, 104, 163
  - Streaming 30
  - Version 17, 20 f.
- I
- IANA 81
- ICMP 11
- IDS → Intrusion Detection Systems
- IETF → Internet Engineering Task Force
- ImageData-Objekt 95
- IMAP 11
- <img>-Tag 206
- Injection 159
- Instanzen 103
- Integer Codierung 50
- Interaktivität 25
- International Organization for Standardization 6
- Internet Engineering Task Force 35, 75
- Interpreter 119
- Intranet 191
- Intrusion Detection Systems 184
- Intrusion Prevention Systems 184
- IP 9, 11
- IP-Hijacking 184, 187
- IPS → Intrusion Prevention Systems
- IPSec 11
- ISO → International Organization for Standardization
- ISO/OSI-Layers 6, 57
- ISO/OSI-Referenzmodell 6
- J
- Jade 204, 251
- Jade-Template 206
- Jade-Template-Engine 251 f.
- JAR-Datei 136
- Java 119 f., 122 f.
- Java API for WebSocket 133
- Java Community Process 132
- Java Enterprise 132
- Java Specification Request 132
- Java Virtual Machine 119
- JavaScript 25, 29, 75, 103, 109, 158
  - canvas-toBlob.js 224
  - getImageData() 95
  - Konsole 86 ff., 182
  - Objekt 213
  - sendFile() 115
  - takeSnapshot() 226
- JavaScript-API ⇒ W3C WebSocket-API
- JCP → Java Community Process
- JDK 122
- jQuery 206, 212
- JSON 209
- JSON-Objekt 214, 249
- JSON-Parser 251
- JSONP 29, 172
- JSR → Java Specification Request
- JSR 356 132, 136
  - Annotation 138
  - API 139
  - Konfigurationsklasse 139
  - Session 138
  - WebSocket-Client 140
  - WebSocket-Server 136

JVM → Java Virtual Machine

## K

Kommunikation 10, 13  
 Kommunikationsendpunkt 9  
 Kommunikationsfluss 11  
 Kommunikationskanal 10  
 Komprimierungsverfahren 15  
 kryptografische Schlüssel 160

## L

Ladezeit 98  
 Landing-Page 187  
 Lasttest 153  
 Latenz 48  
 Login-Formular 167, 170  
 Login-Maske 179  
 Login-Seite 170  
 Long-Polling 28, 110

## M

MAC 7  
 Malware 191  
 Man in the Middle 187, 195  
 Mask-Bit 59  
 MASK-Flag 41  
 Maskierung 43, 184  
 Maskierungsschlüssel → Masking-Key  
 Masking-Key 39, 41, 43  
 Mausevents 218  
 Mausposition 92  
 Medienquellen 225  
 Methode 16  
 Mikro-Instanz 153  
 MIME 11  
 MIME-Typ 32  
 Mobilfunknetze 161  
 Modulo-Operator 44  
 Mouse-Tracking 216  
 Multiplexing Extension for WebSockets 91

## N

Nachrichtengröße 113 f., 222  
 Namensschema 79  
 NetBeans 134 f.  
 netstat 157  
 Netty 122  
 Network Layer ⇒ Vermittlungsschicht  
 Network Scan 191  
 Netzwerkanalysen 191  
 Netzwerkschicht 12  
 Netzwerkschnittstelle 55

nginx 165, 185  
 Node.js 102, 148, 151, 204  
 – Echo-Server 107  
 – Installation 104  
 – Modul  
 – Express 204  
 – Forever 156  
 – Modulmanager 103, 105  
 – Webserver 103  
 Non-Control-Frames 40  
 nonce 164

## O

öffentlicher Schlüssel 160  
 Opcode 40, 42, 48, 59  
 Open Systems Interconnection 6  
 Opening-Handshake 36, 60, 70  
 – Request 36, 67, 72, 167  
 – Response 37, 72  
 Origin-Host 194  
 OSI → Open Systems Interconnection  
 OSI-Modell 6, 8, 11  
 Overhead 27, 42

## P

Paket 56  
 Paketfilter 183  
 Path 17  
 Payload 19, 125  
 Payload len 41  
 Performance 98, 147  
 Performancetest 157  
 Persistent Connection 29  
 Phishing-Mails 190  
 Physical Layer ⇒ Bitübertragungsschicht  
 Ping 113, 186, 196  
 Ping-Frame 41, 48  
 Ping-Intervall 197  
 Play Framework 122  
 – Action 125, 130, 179  
 – activator 122, 124  
 – Controller 124, 127, 178  
 – Echo-Server 129  
 – Event-Handler 129  
 – Formularbasierte Authentifizierung 178  
 – Helper-Klasse 179  
 – Installation 122  
 – Konsole 124  
 – Layout-Template 125  
 – Methode  
 – close() 130  
 – invoke() 130



- onClose() 130
- onMessage() 130
- onReady() 130
- render() 125 f.
- validate() 180
- WebSocket.reject 183
- write() 130
- Ordnerstruktur 123
- Projekt 123
- readyState 182
- Result 125
- Route 126, 128, 181
- Typparameter 130
- View 125, 127, 131, 179
- WebSocket.In 130
- WebSocket.Out 130
- Plug-In 2
- Polling 27
- Pong 186, 196
- Pong-Frame 41, 49
- POP3 11
- Port 7, 17, 183, 192
- Port Scanner 187, 192, 195
- PPPoE 11
- Presentation Layer ⇒ Darstellungsschicht
- PrintWriter 15
- privater Schlüssel 160
- Protokoll 1, 9 f.
- Protokollanalyse 54
- Proxy-Server 60, 159, 165, 183
- Public Draft 132
- Python 119

## Q

QR-Code 204, 206 f.  
Query 17, 79

## R

Referenzmodell 6, 8  
Remote-Debugging 235  
Remote Shell 187, 191, 195  
Request → Hypertext Transfer Protocol  
Anfrage  
Request For Comments 35  
Response → Hypertext Transfer Protocol  
Antwort  
Response-Code 164  
Response-Header 24  
Ressource 16, 22  
REST-Konzept 19  
RFC → Request For Comments  
RFC 6455 78, 91, 159, 162

Round-Trip-Zeit 147, 153, 157  
Route-Matcher 212  
RS-232 11  
RSV-Flag 40, 42, 59  
Ruby 119

## S

Same Origin Policy 29, 158, 193  
Scala 122, 125  
Script Tag Long Polling 29  
SCTP → Stream Control Transmission Protocol  
Secure Socket Layer 11, 159  
Server 101  
Server-Push 29, 31  
Server-Sent Events 2, 31, 203

- EventSource-Objekt 31
- Handler
  - onerror 31
  - onmessage 31
  - onopen 31
- Kanal 31

Serverbenachrichtigung 33  
Serverfehler 23  
Session 168  
Session-ID 166 f., 174, 176, 204, 206  
Session Layer ⇒ Sitzungsschicht  
Session-Token 194  
SFTP 11  
SHA-1 37  
Sicherheit 158  
Sicherungsschicht 6  
Sichtbare Proxys 184  
Signierungsschlüssel 174  
Sitzungsschicht 6 f.  
Skalierbarkeit 147  
Smart-TV 202  
SMTP 11  
SOAP 80  
Sockets 14  
Socket.io 109, 204, 216

- Echo-Server 110
- Event
  - connect 112
  - disconnect 112
  - error 112
  - message 112
- Formularbasierte Authentifizierung 172
- Methode
  - emit() 112
  - io.connect() 112, 206
  - send() 112

SOP → Same Origin Policy

SPDY 97  
 SPDY-Session 71  
 SSE → Server-Sent Events  
 SSH 11  
 SSL → Secure Socket Layer  
 Startskript 153  
 Stream Control Transmission Protocol 98  
 String 93  
 Subprotocol Name Registry 81  
 Subprotokoll 39, 76, 80, 91  
 Supervisor 154  
 Switching Protocols 21, 37, 60

## T

TCP 9, 11 f., 14, 53, 60, 73, 160  
 TCP-Timeout 71  
 Technology Classes 30  
 Telnet 11, 14  
 Testclient 151  
 Testszenario 148  
 Testumgebung 153  
 textbasiert 14  
 textbasierte Daten 93  
 Textdaten 45  
 Textframe 59  
 Thread 103  
 Timeout 54  
 TLS → Transport Layer Security  
 Token 166  
 Touch-Events 207 f.  
 Transfervolumen 15, 197  
 Transparente Proxys 184  
 Transport Layer ⇒ Transportschicht  
 Transport Layer Security 160 f., 185  
 – Handshake 160  
 – Zertifikat 160  
 Transportschicht 6, 12, 187  
 Tyrus 133

## U

UDP 9, 11  
 Umgebungsvariablen 241  
 Umleitung 22  
 unidirektional 33  
 Unverfälschtheit 159  
 URL 1, 16 f., 79 f., 116  
 URL-Schema 16, 79  
 – https 160  
 – ws 79  
 – wss 80, 160, 185  
 Usability-Test 216  
 User 16

UTF-8 93

## V

V8-Engine 103  
 Validierung 219  
 Verbindungsabbau 10, 13, 96  
 Verbindungsaufbau 10, 13  
 Vereinfachtes OSI-Modell 8  
 Vermittlungsschicht 6 f.  
 Vert.x 119, 148, 211  
 – Closed-Handler 150  
 – Echo-Server 120, 149  
 – Installation 120  
 – Paketmanager 120  
 – WebSocket-Handler 121, 149  
 Vertrauliche Kommunikation 159  
 Vertraulichkeit 184  
 virtueller Rechner 148, 153

## W

W3C 30, 75  
 W3C WebSocket-API 75, 151, 196  
 – Attribut 89  
 – binaryType 89, 92, 94  
 – bufferedAmount 89, 91  
 – code 96 f.  
 – extensions 89, 91  
 – protocol 89, 91  
 – readyState 81, 88  
 – reason 96 f.  
 – url 89  
 – wasClean 97  
 – Candidate Recommendation 76  
 – Editor's Draft 76  
 – Event-Handler 83  
 – onclose 83, 85, 87 f., 97, 108  
 – onerror 83 f., 87  
 – onmessage 83, 85, 87  
 – onopen 83 f., 86 f., 89  
 – Event-Objekt 97  
 – Instanz 80, 87  
 – Konstruktor 80, 87  
 – Methode  
 – addEventListener() 86  
 – close() 82, 88, 96  
 – send() 82, 92, 95  
 – Objekt 80  
 – Objekterzeugung 80, 88  
 – Protokollversion 78  
 – Verbindungsaufbau 80  
 – Working Draft 75  
 – Zustand

- CLOSED 81 f., 85, 87
- CLOSING 81 f.
- CONNECTING 81
- OPEN 81 f., 87

Webcam 221

WebIDL 80, 96

WebRTC 222

WebSocket-Client 75, 86

WebSocket-Datenverkehr 66

WebSocket-Element 76

WebSocket-Endpoint 139, 178

WebSocket-Frame 39, 42, 59, 70, 93, 118, 188

WebSocket-Handshake 36, 67, 159

WebSocket-Header 39 f.

WebSocket-Node 113, 151, 196

- Event
  - connect 151
  - onmessage 119
- Formularbasierte Authentifizierung 176
- message-Objekt 114
- Methode
  - connect() 151
  - send() 116
  - sendBytes() 115
- Server 114

WebSocket-Payload 39

WebSocket Per-frame Compression 91

WebSocket-Protokoll 35

WebSocket-Proxys 185

WebSocket.io 106

- Echo-Server 107

window-Objekt 76

Wireshark 54

WLAN 11

WLAN-Router 12

word count 157

## X

XHR → XMLHttpRequest

XHR-Objekt 233

XMLHttpRequest 25, 233

- Methode
  - send() 26
  - setTimeout() 27
- Objekt 26

XOR 43

XSS → Cross-Site-Scripting

XST → Cross-Site-Tracing

## Z

Zeitmessung 148, 152, 157

Zeitstempel 152

Zertifikat 185

Zertifikatswarnung 185

Zertifizierungsstelle 160, 185

Zufallszahl 43, 167

Zugriffskontrolle 167 f.

Zustandsdiagramm 82

Zustandsinformation 168

zustandslos 24