Einführung zu WebSockets: Sockets im Web

(/profiles/#malteubl)
Veröffentlicht **Okt. 20, 2010**

UNTERSTÜTZTE BROWSER: *Your browser appears to support the fu*

Das Problem: Client-Server- und Server-Client-Verbindungen mit geringer Latenz

Im Großen und Ganzen wurde das Internet um das sogenannte Anforderungs- und Antwortmuster von HTTP konstruiert. Ein Client lädt eine Webseite hoch und nichts passiert, bis der Nutzer auf die nächste Seite klickt. Ungefähr im Jahr 2005 sorgte AJAX dafür, dass das Internet dynamischer wirkte. Dennoch wurde die gesamte HTTP-Kommunikation vom Client gesteuert. Dieser benötigte zum Laden neuer Daten vom Server die Interaktion der Nutzer oder periodische Abrufaktionen.

Seit einiger Zeit gibt es Technologien, durch die der Server Daten im selben Moment an den Client sendet, in dem bekannt wird, dass diese verfügbar sind. Diese Technologien werden "Push" oder "Comet" (http://en.wikipedia.org/wiki/Comet_(programming)) genannt. Einer der häufigsten Hacks, der den Anschein einer vom Server initiierten Verbindung gibt, wird als lange Abfrage (long polling) bezeichnet. Bei langen Abfragen öffnet der Client eine HTTP-Verbindung mit dem Server. Sie bleibt geöffnet, bis die Antwort gesendet wird. Sobald der Server über neue Daten verfügt, sendet er die Antwort. Bei anderen Methoden sind Flash (http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/Socket.html) , XHR-Multipart (http://cometdaily.com/2007/12/27/a-standards-based-approach-to-comet-communication-with-rest/) -Anforderungen und sogenannte htmlfiles (http://cometdaily.com/2007/10/25/http-streaming-and-internet-explorer/) erforderlich. Lange Abfragen und die anderen Methoden funktionieren ziemlich gut. Sie kommen täglich zum Einsatz, beispielsweise beim Chat in Google Mail.

Alle diese Behelfslösungen beruhen jedoch auf demselben Problem: Sie nutzen das aufwendige HTTP, was sie nicht besonders für Anwendungen mit geringer Latenz qualifiziert. Denken Sie an Shooter-Spiele mit mehreren Mitspielern im Browser oder an andere Online-Spiele mit Echtzeitkomponenten.

Einführung zu WebSocket: Sockets im Web

Die WebSocket (http://dev.w3.org/html5/websockets/) -Spezifikation legt ein API fest, die "socket"-Verbindungen zwischen einem Webbrowser und einem Server herstellt. Kurz: Es besteht eine persistente Verbindung zwischen Client und Server und beide Parteien können jederzeit mit dem Senden von Daten beginnen.

Erste Schritte

Sie öffnen eine WebSocket-Verbindung, indem Sie ganz einfach den WebSocket-Konstruktor aufrufen:

```
var connection = new WebSocket('ws://html5rocks.websocket.org/echo', ['soap', 'xmpp']);
```

Beachten Sie das `ws:`. Das ist das neue URL-Schema für WebSocket-Verbindungen. Außerdem gibt es `wss:` für sichere WebSocket-Verbindungen, genau wie `https:` für sichere HTTP-Verbindungen.

Wenn Sie einige Ereignis-Handler sofort an die Verbindung anhängen, wissen Sie, wann die Verbindung geöffnet ist, eingehende Mitteilungen empfangen wurden oder wenn ein Fehler aufgetreten ist.

Das zweite Argument akzeptiert optionale Subprotokolle. Es kann sich um einen String oder um ein Array von Strings handeln. Jeder String sollte einen Subprotokollnamen darstellen. Der Server akzeptiert nur eines der weitergegebenen Subprotokolle im Array. Das akzeptierte Subprotokoll wird durch den Zugriff auf die `protocol`-Eigenschaft des WebSocket-Objekts festgelegt.

Die Subprotokollnamen müssen den registrierten Subprotokollnamen in der IANA-Registrierung (http://www.iana.org/assignments/websocket/websocket.xml) entsprechen. Zurzeit gibt es nur einen seit Februar 2012 registrierten Subprotokollnamen (soap).

```
// When the connection is open, send some data to the server
connection.onopen = function () {
  connection.send('Ping'); // Send the message 'Ping' to the server
};

// Log errors
connection.onerror = function (error) {
  console.log('WebSocket Error ' + error);
};

// Log messages from the server
connection.onmessage = function (e) {
  console.log('Server: ' + e.data);
};
```

Mit dem Server kommunizieren

Sobald eine Verbindung mit dem Server besteht, also das `open`-Ereignis ausgelöst wurde, können Sie beginnen, dem Server Daten zu senden. Verwenden Sie dazu die `send('your message')`-Methode im Verbindungsobjekt. Sie unterstützte bisher nur Strings, kann aber seit der neusten Spezifikation auch Binärmitteilungen senden. Zum Senden von Binärdaten können Sie entweder das `Blob`- oder das `ArrayBuffer`-Objekt verwenden.

```
// Sending String
connection.send('your message');

// Sending canvas ImageData as ArrayBuffer
var img = canvas_context.getImageData(0, 0, 400, 320);
var binary = new Uint8Array(img.data.length);
for (var i = 0; i < img.data.length; i++) {
  binary[i] = img.data[i];
}
connection.send(binary.buffer);

// Sending file as Blob
var file = document.querySelector('input[type="file"]').files[0];
connection.send(file);
```

Der Server kann Ihnen gleichermaßen jederzeit Mitteilungen senden. Sobald das der Fall ist, wird der `onmessage`-Rückruf ausgelöst. Der Rückruf empfängt ein Ereignisobjekt. Auf die eigentliche Nachricht kann über die `data`-Eigenschaft zugegriffen werden.

WebSocket kann in der letzten Spezifikation auch Binärmitteilungen empfangen. Binär-Frames können im `Blob`- oder `ArrayBuffer`-Format empfangen werden. Setzen Sie zum Angeben des Formats der empfangenen Binärmitteilung, die "binaryType"-Eigenschaft von WebSocket entweder auf "blob" oder auf "arraybuffer". Das Standardformat lautet "Blob". (Sie müssen den "binaryType"-Parameter beim Senden nicht abstimmen.)

```
// Setting binaryType to accept received binary as either 'blob' or 'arraybuffer'
connection.binaryType = 'arraybuffer';
connection.onmessage = function(e) {
  console.log(e.data.byteLength); // ArrayBuffer object if binary
};
```

Eine weitere neue Funktion von WebSocket sind Erweiterungen. Wenn Sie Erweiterungen verwenden, können Sie Frames komprimiert (http://tools.ietf.org/html/draft-tyoshino-hybi-websocket-perframe-deflate-05) , als Multiplex (http://tools.ietf.org/html/draft-tamplin-hybi-google-mux-02) usw. senden Erweiterungen, die vom Server akzeptiert werden, finden Sie, indem Sie die Erweiterungseigenschaft des WebSocket-Objekts nach dem "open"-Ereignis untersuchen. Es gibt zurzeit, im Februar 2012, noch keine offiziell veröffentlichten Erweiterungsspezifikationen.

```
// Determining accepted extensions
console.log(connection.extensions);
```

Ursprungsübergreifende Kommunikation

Als modernes Protokoll ist die ursprungsübergreifende Kommunikation direkt in WebSocket enthalten. Sie sollten sich immer noch vergewissern, dass Sie nur mit vertrauenswürdigen Clients und Servern kommunizieren. WebSocket ermöglicht jedoch die Kommunikation zwischen Parteien in jeder Domain. Der Server entscheidet, ob sein Service für alle Clients verfügbar wird oder nur für diejenigen, die auf einer Gruppe gut definierter Domains residieren.

Proxy-Server

Jede neue Technologie bringt neue Probleme mit sich. Im Fall von WebSocket ist es die Kompatibilität mit Proxy-Servern, die HTTP-Verbindungen in den meisten Unternehmensnetzwerken vermitteln. Das WebSocket-

Protokoll verwendet das HTTP-Upgradesystem, welches normalerweise für HTTP/SSL genutzt wird, für das "Upgrade" einer HTTP-Verbindung in eine WebSocket-Verbindung. Einige Proxy-Server können damit nicht umgehen und unterbrechen die Verbindung. Daher ist es eventuell, auch wenn ein bestimmter Client das WebSocket-Protokoll verwendet, nicht möglich, eine Verbindung herzustellen. Aus diesem Grund ist der nächste Abschnitt sogar noch wichtiger :)

WebSockets sofort verwenden

WebSocket ist noch eine junge Technologie und daher noch nicht in allen Browsern implementiert. Sie können WebSocket jedoch sofort mit Bibliotheken einsetzen, die eine der oben erwähnten Fallbacks verwenden, falls WebSocket nicht verfügbar ist. Eine in dieser Domain sehr beliebte Bibliothek ist socket.io (http://socket.io/) . Sie verfügt über eine Client- und Serverimplementierung des Protokolls und enthält Fallbacks. "socket.io" unterstützt zurzeit, im Februar 2012, keine Binärmitteilungen. Es gibt auch kommerzielle Lösungen, wie PusherApp (http://pusherapp.com/) , die sich einfach in jede Webumgebung integrieren lassen, indem sie ein HTTP-API zur Verfügung stellen, mit der WebSocket-Mitteilungen an Clients gesendet werden. Aufgrund der zusätzlichen HTTP-Anforderungen entsteht im Vergleich zur reinen WebSocket immer ein zusätzlicher Aufwand.

Serverseitig

Durch die Nutzung von WebSocket entsteht ein völlig neues Nutzungsmuster für serverseitige Anwendungen. Herkömmliche Serverstapel, wie LAMP, sind entsprechend des HTTP-Anforderungs-/Antwortzyklus konzipiert und funktionieren meist nicht gut mit einer großen Anzahl offener WebSocket-Verbindungen. Wenn Sie gleichzeitig eine große Anzahl von Verbindungen geöffnet haben, ist eine Architektur erforderlich, die eine hohe Gleichzeitigkeit bei möglichst geringen Einbußen bei der Leistung ermöglicht. Solche Architekturen werden meist um Threading oder sogenannte nicht-blockierende IOs konzipiert.

Serverseitige Implementierungen

- Node.js
  - Socket.IO (http://socket.io/)
  - WebSocket-Node (https://github.com/Worlize/WebSocket-Node)
  - ws (https://github.com/einaros/ws)
- Java
  - Jetty (http://www.eclipse.org/jetty/)
- Ruby
  - EventMachine (http://github.com/igrigorik/em-websocket)
- Python
  - pywebsocket (http://code.google.com/p/pywebsocket/)
  - Tornado (https://github.com/facebook/tornado)
- Erlang
  - Shirasu (https://github.com/michilu/shirasu)
- C++

    - libwebsockets (http://git.warmcat.com/cgi-bin/cgit/libwebsockets/)
- .NET
    - SuperWebSocket (http://superwebsocket.codeplex.com/)

Protokollversionen

Das Wire-Protokoll für WebSocket, ein Handshake und der Datentransfer zwischen Client und Server, lautet jetzt RFC6455 (http://tools.ietf.org/html/rfc6455) . Die letzte Version von Chrome und Chrome für Android sind vollständig kompatibel mit RFC6455, einschließlich der Binärmitteilungen. Firefox wird ab Version 11 ebenfalls kompatibel sein; Internet Explorer ab Version 10. Sie können zwar auch ältere Protokollversionen nutzen, es wird jedoch nicht empfohlen, da bekanntermaßen Schwachstellen vorliegen. Wenn Sie über Serverimplementierungen für ältere Versionen des WebSocket-Protokolls verfügen, wird empfohlen, dass Sie sie auf die neueste Version aktualisieren.

Einsatzbeispiele

Verwenden Sie WebSocket wenn Sie eine sehr geringe Latenz, also fast eine Echtzeitverbindung, zwischen Client und Server benötigen. Bedenken Sie, dass dadurch eventuell auch der Aufbau der serverseitigen Anwendungen neu überdacht werden muss. Hierbei sollten Technologien, wie Ereigniswarteschlangen, einbezogen werden. Einige Einsatzbeispiele lauten:

- Online-Spiele mit mehreren Mitspielern
- Chat-Anwendungen
- Live-Sport-Ticker
- Echtzeit-Aktualisierungen von sozialen Streams

Demos

- Plink (http://labs.dinahmoe.com/plink/)
- Paint With Me (http://paintwith.me/)
- Pixelatr (http://connorhd.co.uk/project/pixelatr/)
- Dashed (http://www.dashed.com/)
- Massively multiplayer online crossword (http://scrabb.ly/)
- Ping server (in den obigen Beispielen verwendet) (http://www.websockets.org/echo.html)
- HTML5demos – Beispiele (http://html5demos.com/web-socket)

Referenzen

- The WebSocket API (http://dev.w3.org/html5/websockets/)
- The WebSocket Protocol (http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-03)
- WebSockets - MDN (https://developer.mozilla.org/en/WebSockets)