



I WebSocket : Comunicazione Asincrona Full-Duplex Per Il Web

Programmazione *by Giuseppe Capodiceci -*



Condividi

E' un fatto ben noto che l' HTTP (Hypertext Transfer Protocol) è un protocollo di tipo request-response senza stato (stateless). La semplicità del protocollo HTTP lo rende molto scalabile, ma inefficiente e non adatto per le applicazioni web altamente interattive o in tempo reale, oggi sempre più diffuse. D'altronde l' HTTP è stato progettato per la condivisione di documenti e non per la creazione di questo tipo di applicazioni.

Prima della versione 1.1 del protocollo, ogni richiesta al server comportava una nuova connessione mentre dalla versione 1.1 la cosa è stata migliorata con l'introduzione delle connessioni persistenti che permettono al browser web di riutilizzare la stessa connessione per il caricamento di immagini, script, ecc

L'HTTP, inoltre, è stato progettato per essere half-duplex il che significa che la trasmissione dei dati è consentita in una sola direzione alla volta. Un walkie-talkie è un esempio di un dispositivo half-duplex dove una sola persona alla volta può parlare. Per superare queste lacune dell'HTTP gli sviluppatori hanno creato alcune soluzioni o hack tra questi il polling, il long polling (aka comet), e lo streaming.

Con il polling, il client effettua chiamate sincrone per ottenere informazioni dal server. Se il server dispone di nuove informazioni disponibili invierà i dati nella risposta. In caso contrario, nessuna informazione verrà inviata al client e il client farà una nuova connessione dopo un certo lasso di tempo per riverificare la disponibilità di nuovi dati. Questo meccanismo è molto inefficiente ma è un modo molto semplice per ottenere un comportamento simil real-time. Il long polling invece è un'altra soluzione in cui il client effettua una connessione al server e il server mantiene aperta la connessione fino a quando i dati sono disponibili oppure viene raggiunto un predefinito timeout. A causa dello squilibrio tra HTTP sincrono e queste applicazioni asincrone, anche questa soluzione tende ad essere complicata, non standard e inefficiente.

Per colmare questa sempre maggiore esigenza delle applicazioni web di poter effettuare comunicazioni bidirezionali in maniera standard è stato introdotto il concetto di WebSocket. Ogni linguaggio di programmazione ha la propria implementazione, in questo articolo, vedremo in particolare l'implementazione nella piattaforma java (JSR 356 API) e un client in javascript.

Prima di vedere l'utilizzo pratico dei WebSockets è bene prima capire cosa sono e come funzionano.

Indice



[Cosa sono i WebSocket?](#)

[Come funziona un WebSocket?](#)

[Vantaggi nell'uso dei WebSockets](#)

[A cosa mi possono servire i WebSocket?](#)

mer 03 febbraio 2021 - **Lo Sviluppatore** anno VI

Un esempio

WebSocket Java Server Endpoint

WebSocket Javascript client

Conclusioni

Cosa sono i WebSocket?

I WebSocket sono un protocollo di messaggistica che permette una comunicazione asincrona e full-duplex su connessione TCP. I WebSockets non sono connessioni HTTP anche se usano l'HTTP per avviare la connessione.

Un sistema full-duplex permette la comunicazione in entrambe le direzioni in maniera contemporanea, tipico esempio di questo tipo di comunicazione è quella telefonica dove gli interlocutori possono parlare ed essere ascoltati allo stesso tempo. I WebSocket sono stati inizialmente proposti come parte della specifica HTML5, che promette di portare la facilità di sviluppo e di efficienza della rete alle applicazioni web moderne, interattive, ma è stato successivamente spostato in un documento standard separato per mantenere la specifica focalizzata solo su WebSocket (RFC 6455 e WebSocket API JavaScript).

Come funziona un WebSocket ?

Ogni connessione WebSocket inizia la sua vita come una richiesta HTTP. Tale richiesta HTTP è molto simile a tutte le altre richieste salvo che nell'intestazione viene specificata un'operazione di tipo *Upgrade* che indica che il client vuole aggiornare la connessione ad un protocollo diverso, in questo caso a WebSocket. Ad aggiornamento avvenuto viene stabilita la connessione WebSocket tra client e server sfruttando la stessa connessione sottostante usata durante la fase iniziale della comunicazione (handshake) e la comunicazione in entrambe le direzioni può cominciare.

Sotto un esempio di handshake lato client:

```
GET /path/to/websocket/endpoint HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
Origin: http://localhost
Sec-WebSocket-Version: 13
```

mentre, lato server, in risposta ad una richiesta del tipo visto sopra, abbiamo una cosa del genere:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=
```

Il server applica una operazione nota (sia al cliente che al server) al valore della chiave *Sec-WebSocket-Key* presente nell'header della chiamata per generare il valore della *Sec-WebSocket-Accept*. Il client fa la stessa operazione sempre sul valore della stessa chiave *Sec-WebSocket-Key*, e, se il valore ricevuto dal server coincide con il valore calcolato dal client la connessione può considerarsi come stabilita con successo e client e server possono cominciare a comunicare. Tramite i WS è possibile scambiare sia messaggi di testo (codificati come UTF-8) sia messaggi in formato binario.

Un endpoint WebSocket è rappresentato da URI nel seguente formato:

```
ws://host:port/path?query
wss://host:port/path?query
```

Lo schema *ws* rappresenta le comunicazioni in chiaro mentre lo schema *wss* rappresenta le comunicazioni crittate. La porta è un dato facoltativo, si consideri che per le comunicazioni in chiaro si usa la porta 80 mentre per le comunicazioni crittate la 443, analogamente al protocollo l'http. La componente *path* rappresenta il path



I browser moderni implementano il protocollo WebSocket e forniscono una API JavaScript per la connessione agli endpoint, inviare messaggi, e assegnare i metodi di callback per gli eventi websocket (quali: connessioni aperte, messaggi ricevute, connessioni chiuse, ecc.).

Per avere informazioni sulla situazione di compatibilità dei vari browser seguire il seguente link:

<http://caniuse.com/#feat=websockets>

Vantaggi nell'uso dei WebSockets

1. I WebSocket sono più efficienti e performanti rispetto ad altre soluzioni, come il polling.
2. Richiedono meno banda e riducono la latenza.
3. Semplificano le architetture applicative in real-time.
4. I WebSocket non richiedono intestazione per inviare messaggi riducendo quindi la larghezza di banda richiesta.

A cosa mi possono servire i WebSocket?

Alcuni dei possibili casi d'uso di WebSocket sono :

- applicazioni di chat
- giochi multiplayer
- Stock trading o applicazioni finanziarie
- Editing cooperativo di documenti
- Applicazioni di social networking

WebSockets in Java

La JSR 356 è la specifica di riferimento dei WebSocket per la piattaforma Java che permette di creare, configurare e distribuire endpoint WebSocket in una web application, nonché implementare client remoti per poter accedere a tali endpoint da qualsiasi applicazione Java. L'API Java per WebSocket si compone dei seguenti package.

- *javax.websocket.server* che contiene le annotazioni, le classi e le interfacce per creare e configurare gli endpoint del server.
- *javax.websocket* che contiene annotazioni, classi, interfacce, e le eccezioni che sono comuni a client e server endpoint.

Gli Endpoint WebSocket sono istanze della classe *javax.websocket.Endpoint*. L'API Java per WebSocket consente di creare gli endpoint sia in maniera programmatica sia mediante l'uso di annotation. Per creare un endpoint in maniera programmatica, si estende la classe *Endpoint* e si fa l'override dei vari metodi che gestiscono il ciclo di vita di un server websocket. Per creare un endpoint mediante annotazioni, basta decorare una classe Java e alcuni dei suoi metodi con le annotazioni specifiche. Dopo aver creato l'endpoint, questo viene distribuito esponendo un URI specifico che i client remoti possono utilizzare per connettersi ad esso. L'API WebSocket è una libreria puramente event driven.

Un esempio

Le comunicazioni via websocket sono naturalmente indipendenti dalle varie implementazioni e consente la comunicazione tra entità eterogenee implementate in un qualsiasi linguaggio di programmazione. Come esempio vogliamo mostrare un caso del genere in cui abbiamo un server endpoint implementato in Java e un client endpoint implementato in javascript che permette di inviare e ricevere messaggi da una comunissima pagina HTML.

WebSocket Java Server Endpoint

```
1 package myfirstws;  
2  
3 import java.io.IOException;  
4  
5 import javax.websocket.OnClose;  
6 import javax.websocket.OnMessage;
```

mer 03 febbraio 2021 - Lo Sviluppatore anno VI

```

8  import javax.websocket.Session;
9  import javax.websocket.server.ServerEndpoint;
10 &nbsp;
11 /**
12  * @ServerEndpoint da un nome all'end point
13  * Questo può essere acceduto via ws://localhost:8080/myfirstws/echo
14  * "localhost" è l'indirizzo dell'host dove è deployato il server ws,
15  * "myfirstws" è il nome del package
16  * ed "echo" è l'indirizzo specifico di questo endpoint
17  */
18 @ServerEndpoint("/echo")
19 public class EchoServer {
20     /**
21     * @OnOpen questo metodo ci permette di intercettare la creazione di una nuova sessione.
22     * La classe session permette di inviare messaggi ai client connessi.
23     * Nel metodo onOpen, faremo sapere all'utente che le operazioni di handshake
24     * sono state completate con successo ed è quindi possibile iniziare le comunicazioni.
25     */
26     @OnOpen
27     public void onOpen(Session session){
28         System.out.println(session.getId() + " ha aperto una connessione");
29         try {
30             session.getBasicRemote().sendText("Connessione Stabilita!");
31         } catch (IOException ex) {
32             ex.printStackTrace();
33         }
34     }
35     &nbsp;
36     /**
37     * Quando un client invia un messaggio al server questo metodo intercetterà tale messaggio
38     * e compierà le azioni di conseguenza. In questo caso l'azione è rimandare una eco del messaggio indi
39     */
40     @OnMessage
41     public void onMessage(String message, Session session){
42         System.out.println("Ricevuto messaggio da: " + session.getId() + ": " + message);
43         try {
44             session.getBasicRemote().sendText(message);
45         } catch (IOException ex) {
46             ex.printStackTrace();
47         }
48     }
49     &nbsp;
50     /**
51     * Metodo che intercetta la chiusura di una connesione da parte di un client
52     *
53     * Nota: non si possono inviare messaggi al client da questo metodo
54     */
55     @OnClose
56     public void onClose(Session session){
57         System.out.println("Session " +session.getId()+" terminata");
58     }
59 }

```

WebSocket Javascript client

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Echo Web Socket</title>
5          <meta charset="UTF-8">
6          <meta name="viewport" content="width=device-width">
7      </head>
8      <body>
9          <div>
10             <input type="text" id="messageinput"/>
11          </div>
12          <div>
13             <button type="button" onclick="openSocket();" >Open</button>
14             <button type="button" onclick="send();" >Send</button>
15             <button type="button" onclick="closeSocket();" >Close</button>
16          </div>
17          <!-- la risposta del server viene scritta qui -->
18          <div id="messages"></div>
19
20          <!-- Script che utilizza i WebSocket -->
21          <script type="text/javascript">
22
23              var websocket;

```

mer 03 febbraio 2021 - **Lo Sviluppatore** anno VI

```

25
26
27     function openSocket(){
28         // Assicura che sia aperta un unica connessione
29         if(webSocket !== undefined && webSocket.readyState !== WebSocket.CLOSED){
30             writeResponse("Connessione WebSocket già stabilita");
31             return;
32         }
33         // Creiamo una nuova istanza websocket
34         webSocket = new WebSocket("ws://localhost:8080/EchoChamber/echo");
35
36         /**
37          * Facciamo il bind delle funzioni con gli eventi dei websocket
38          */
39         webSocket.onopen = function(event){
40             // quando viene aperta la connessione inviamo un messaggio di OK
41             // al server
42             // scrivo il messaggio nella textbox e lo invio
43             writeResponse("OK");
44             send();
45         };
46
47         webSocket.onmessage = function(event){
48             writeResponse(event.data);
49         };
50
51         webSocket.onclose = function(event){
52             writeResponse("Connection closed");
53         };
54     }
55
56     /**
57     * Invia il contenuto della text input al server
58     */
59     function send(){
60         var text = document.getElementById("messageinput").value;
61         webSocket.send(text);
62     }
63
64     function closeSocket(){
65         webSocket.close();
66     }
67
68     function writeResponse(text){
69         messages.innerHTML += "<br/>" + text;
70     }
71
72 </script>
73
74 </body>
75 </html>

```

Conclusioni

Abbiamo visto come i websocket ci possono venire in aiuto quando abbiamo l'esigenza di creare applicazioni in realtime che si devono cambiare messaggi. Per chi volesse approfondire segnalo alcune risorse utili sull'argomento:

- [Il sito "ufficiale"](#)
- [La specifica delle API \(client\)](#)
- [La specifica del protocollo \(server\)](#)
- [Java API for websocket](#)
- [How to build Java WebSocket Applications Using the JSR 356 API](#)

Potrebbe interessarti anche:



[< Articolo Precedente](#)[Java 8 – la Streaming API](#)[Articolo Successivo >](#)[Code comments](#)

Lascia un commento

☐

Do il mio consenso
affinché un cookie
salvi i miei dati
(nome, email, sito
web) per il prossimo
commento.

Questo sito usa Akismet per ridurre lo spam. [Scopri come i tuoi dati vengono elaborati.](#)

SEGUI LO SVILUPPATORE SU...



CITAZIONI

Fare il debugging è come
essere il detective in un
film giallo in cui tu sei
anche l'assassino

Debugging

LA VIGNETTA



LE RECENSIONI IMPROBILI



Deadline Driven Development

O RLY?

Mark Black

© 2021 **Lo Sviluppatore**

Powered by WordPress | Theme: AccessPress Mag - Child Theme by Giuseppe Capodici

[Home](#)

[Java](#)

[Programmazione](#)

[Design](#)

[Tools](#)

[Tips & Tricks](#)

[Curiosità e Humor](#)