1. WebSockets

Nature

Les "WebSockets" constituent une adaptation "HTTP" des classiques sockets "tcp/ip" .

C'est une sorte d'annexe/extension vis à vis du protocole HTTP .

Ayant leurs propres préfixes (schemes : ws : , wss :) , les "WebSockets" peuvent également être vues comme un nouveau protocole (avec même la notion de sous protocole possible tel que STOMP)

En tant que "sockets" , une "**WebSockets**" est un **canal de communication bi-directionnel établi durablement (tant que pas fermé)** entre un client et un serveur par exemple .

Ce canal bi-directionnel peut servir à spontanément envoyer (dans les 2 sens) des messages dans format quelconque (texte , json, ... ou binaire).

Fonctionnalités

Les "webSockets" sont surtout utilisés dans une logique de "**push**" ou "**subscribe/publish**" .

Une fois une connexion établie , le serveur peut spontanément envoyer de nouvelles valeurs (qui viennent de changer) vers le coté client/navigateur sans que celui-ci soit obligé d'effectuer une requête préalable d'actualisation .

En règle générale, de nombreux clients sont simultanément connectés à un même serveur .

Le serveur peut alors via une simple boucle diffuser une information vers tous les clients connectés via une websocket active .

Application classique :

- discussion en tant réel : "chat" , messagerie instantanée

- actualisation automatique de graphique (ex : SVG, canvas, ...) dès qu'une valeur change

- tableau (board) partagé en équipe

- toute autre communication en mode push (avec ou sans RxJs / mode réactif) .

Principe de fonctionnement

Une connexion "webSocket" s'effectue en partant d'une connexion "http" existante puis en "upgradant" celle-ci durant une phase d'échange d'informations appelée "**handshake** " .

Le client envoi une requête HTTP de ce type

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 6
```

Le serveur doit s'il accepte l'upgrade , renvoyer une réponse de ce type :

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Au niveau du paramètre du constructeur WebSocket() de l'api html5/javascript, l'url d'une websocket à préciser pour établir une connexion ne commence par par http:// ni par https:// mais par **ws://** ou **wss://** . Cependant , comme la connexion ws:// est un upgrade d'une connexion HTTP , le numéro de port utilisé par les "websockets" reste le standard **80** et donc pas de soucis en général pour que les requêtes/réponses puis passer à travers proxy ou firewall .

Une fois la connexion établie , Chaque protagoniste (client et serveur) voit l'autre coté du canal de communication comme un "**endpoint**" vers lequel on peut spontanément envoyer des messages via une méthode .**send**() .

Du coté réception , les méthodes "callback" suivantes seront automatiquement appelées :

- **`onopen`** : ouverture d'une WebSocket
- **`onmessage`** : réception d'un message
- **`onerror`** : erreur(s) survenue(s)
- **`onclose`** : fermeture de WebSocket (de l'autre coté)

Principales "API" et "implémentations" pour le coté serveur

Api java "JSR 356" supporté entre autres par Tomcat 7 , 8, ... (**JEE7**)

Intégration "Spring 4+" , ...

Equivalent dans la plupart des autres technologies du coté serveur (php , .net , nodeJs , ...)

Exemple de "chat"(webSocket) : code client "html5+javascript"

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
   <title>Apache Tomcat WebSocket Examples: Chat</title>
   <style type="text/css">...</style>
   <script type="application/javascript"><![CDATA[
      "use strict";
```

```
var Chat = {};  //custom literal js object

Chat.socket = null;

Chat.connect = (function(host) {
    if ('WebSocket' in window) {
        Chat.socket = new WebSocket(host);
    } else if ('MozWebSocket' in window) {
        Chat.socket = new MozWebSocket(host);
    } else {
        Console.log('Error: WebSocket is not supported by this browser.');
        return;
    }

    Chat.socket.onopen = function () {
        Console.log('Info: WebSocket connection opened.');
        document.getElementById('chat').onkeydown = function(event) {
            if (event.keyCode == 13) {
                Chat.sendMessage();
            }
        };
    };

    Chat.socket.onclose = function () {
        document.getElementById('chat').onkeydown = null;
        Console.log('Info: WebSocket closed.');
    };

    Chat.socket.onmessage = function (message) {
        Console.log(message.data);
    };
});

Chat.initialize = function() {
    if (window.location.protocol == 'http:') {
        Chat.connect('ws://' + window.location.host + '/examples/websocket/chat');
    } else {
```

```
            Chat.connect('wss://' + window.location.host + '/examples/websocket/chat');
        }
    };

    Chat.sendMessage = (function() {
        var message = document.getElementById('chat').value;
        if (message != '') {
            Chat.socket.send(message);
            document.getElementById('chat').value = '';
        }
    });

    var Console = {};  //custom literal js object

    Console.log = (function(message) {
        var console = document.getElementById('console');
        var p = document.createElement('p');
        p.style.wordWrap = 'break-word';
        p.innerHTML = message;
        console.appendChild(p);
        while (console.childNodes.length > 25) {
            console.removeChild(console.firstChild);
        }
        console.scrollTop = console.scrollHeight;
    });

    Chat.initialize();
]]></script>
</head>
<body>
    <p>
        <input type="text" placeholder="type and press enter to chat" id="chat" />
    </p>
    <div id="console-container">
        <div id="console"/>
    </div>
</div> </body> </html>
```