**用StompJS/SpeechSynthesis实现前端消息实时语音播报**



**前言**

前端消息的实时推送我相信很多人不陌生，我们可以想到利用WebSocket，服务端主动向客户端推送数据，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，并进行双向数据传输。其优点有很多，能更好的节省服务器资源和带宽，并且能够更实时地进行通讯等等。语音播报则能够在人们视觉没有来的及关注时侯，通过听觉来获取需要信息。

这篇文章主要介绍的是基于websocket，利用Stomp.js以及HTML5语音Web Speech API——SpeechSynthesis来实现前端消息的实时推送与语音播报。

**StompJS**

让我们先了解一下**STOMP**（the Simple (or Streaming) Text Orientated Messaging Protocol）——面向消息（或流）的简单文本协议。它提供了一个可互操作的连接格式，允许STOMP客户端与任意STOMP消息代理（Broker）进行交互。

WebSocket的实现客户端看起来比较简单，但是需要与后台进行很好的配合和调试才能达到最佳效果。通过SockJS 、Stomp来进行浏览器兼容，可以增加消息语义和可用性。简而言之，WebSocket 是底层协议，SockJS 是WebSocket 的备选方案，也是底层协议，而 STOMP 是基于 WebSocket（SockJS）的上层协议。

**创建STOMP客户端**

下面简单的介绍一下常用的方法。在web浏览器中我们可以通过两种方式进行客户端的创建：1、使用普通的WebSocket

let url = "ws://localhost:61614/stomp";  
let client = Stomp.client(url);

2、使用定制的WebSocket如果需要使用其他类型的Websocket（例如由SockJS包装的Websocket），就利用下面的方式创建客户端

let url = "ws://localhost:61614/stomp";  
let socket = new SockJS(url);  
let client = Stomp.over(socket);

除上面的客户端创建方式不同外，后续的连接等操作都是一样的。

**连接服务端**

我们可以用client.connect()方法来连接服务端

client.connect(login,passcode,successCallback,errorCallback);

其中login和passcode都是字符串，相当于是用户的登录名和密码凭证。successCallback为连接成功的回调函数，errorCallback为连接失败的回调函数。还可以这样写：

client.connect({  
 login:'name',  
 passcode:'666',  
 'token':'2333'  
},successCallback,errorCallback);

断开连接：

client.disconnect(function(){console.log("再见")})

**Heart-beating（心跳）**

heart-beating也就是消息传送的频率，incoming是接收频率，outgoing是发送频率，其默认值都为10000ms，我们可以手动设置：

client.heartbeat.outgoing = 5000;   
client.heartbeat.incoming = 0;

**发送消息**

客户端向服务端发送消息利用send()方法，此方法有三个参数：第一个参数（string）必需，为发送消息的目的地；第二个参数（object）可选，包含了额外的头部信息；第三个参数（string）可选，为发送的消息。

client.send(destination, {}, body);

**订阅消息**

订阅消息也就是客户端接收服务端发送的消息，订阅消息可以利用subscribe()方法，此方法有三个参数：第一个参数（string）必需，为接收消息的目的地；第二个参数必需为回调函数；第三个参数{object}为可选，包含额外的头部信息。

client.subscribe(destination, callback, {});

取消订阅消息可以利用unsubscribe()方法：

let mySubscribe = client.subscribe;  
 mySubscribe.unsubscribe();

客户端订阅消息可以订阅广播，如下所示：

client.subscribe('/topic/msg',function(messages){  
 console.log(messages);  
})

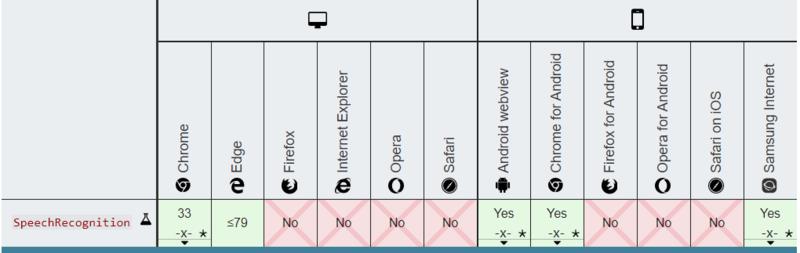
也可以进行一对一消息的接收：

//第一种方式  
const userId = 666;  
client.subscribe('/user/' + userId + '/msg',,function(messages){  
 console.log(messages);  
})  
//第二种方式  
client.subscribe('/msg',function(messages){  
 console.log(messages);  
}, {"userId ": userId })

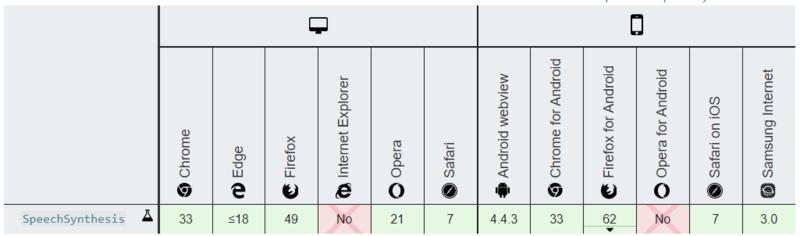
客户端采用的写法要根据服务端代码来做选择。

**Web Speech API**

在HTML5中，与语音相关的Web Speech API可以分为两种：一种为语音识别(Speech Recognition)，另一种为语音合成(Speech Synthesis)。他们的作用分别为“语音转文字”和“文字转语音”。既然是HTML5中的东西，我们还是要先看看他们的兼容性如何：Speech Recognition：



Speech Synthesis：



从上面的图中可以看出：语音识别(Speech Recognition)很惨烈，大部分浏览器还不支持。语音合成(Speech Synthesis)除开IE和Opera，基本上都支持了。本文要实现的是语音播报，就是要把文字消息，转成语音播报出来，而语音合成(Speech Synthesis)就是实现这样的功能，而且兼容性也是不错的，所以我们就能拿来使用啦~

**SpeechSynthesis**

语音识别(Speech Recognition)就不过多介绍了，我们来详细看看语音合成(Speech Synthesis)。我们可以先把下面这段代码打到浏览器的控制台上：

let speechInstance = new window.SpeechSynthesisUtterance('你好，可以交个朋友吗');  
window.speechSynthesis.speak(speechInstance);

不出意外，浏览器说话了，说明浏览器是支持这个API的。下面简单介绍一下相关的属性和方法：SpeechSynthesisUtterance对象的属性：

属性类型描述textstring需要要读的内容langstring使用的语言（比如："zh-CN"）volumenumber音量，值在0-1之间（默认是1）ratenumber语速的倍数，值在0.1-10之间（默认1倍）pitchnumber音高，值在0-2之间，（默认是1）voicestring指定希望使用的声音

SpeechSynthesisUtterance对象的方法：

方法描述onstart语音开始合成时触发onpause语音暂停时触发onresume语音合成重新开始时触发onend语音结束时触发

上述定义的speechInstance其实是我们创建的文本实例，而真实的语音是由speechSynthesis来创建的，其常用的方法如下：

方法描述speak()开始合成语音，将对应的实例添加到语音队列中cancel()停止合成语音，删除队列中所有的语音pause()暂停语音合成resume()恢复暂停后的语音getVoices()返回浏览器所支持的语音包数组

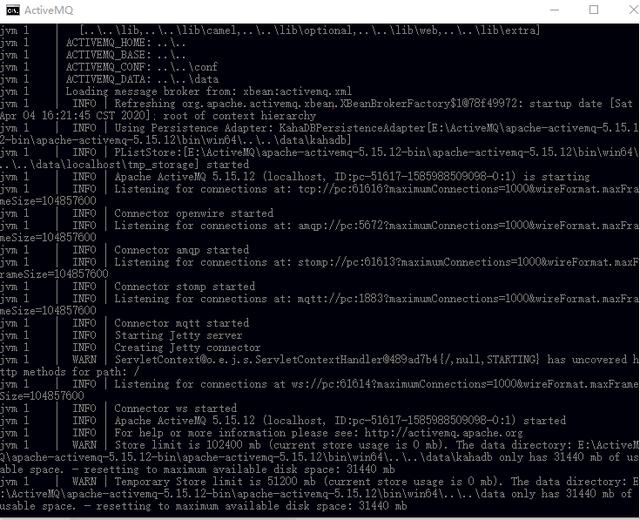
**实战环节**

上面介绍了StompJS和SpeechSynthesis常用的属性和方法，是时候动手码一码了。

**前端web页面消息实时接收**

**模拟服务端发送消息**

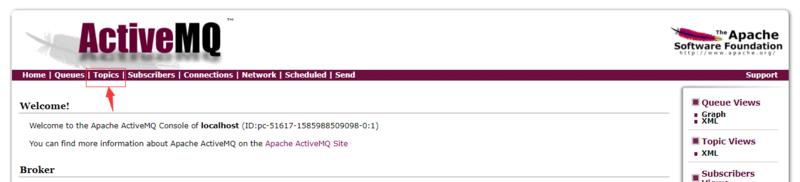
想要接收实时消息，我们当先然要有消息的来源对不对。消息是从后台发来的，一般是利用Java，然后结合ActiveMQ或者RabbitMQ等消息中间件，Java代码就不多说，我们接下来就利用ActiveMQ来模拟服务端向客户端发送消息。这里说一说windows环境下吧，首先要下载ActiveMQ，直接点击官网：http://activemq.apache.org/do...选择最新发布下来的压缩包，解压即可，然后进入解压后的bin目录，可以看见里面有两个文件夹，win32和win64，这个就根据自己电脑的操作系统来选择，点击进去，再双击activemq.bat启动，如果看见下面这样就说明启动成功：



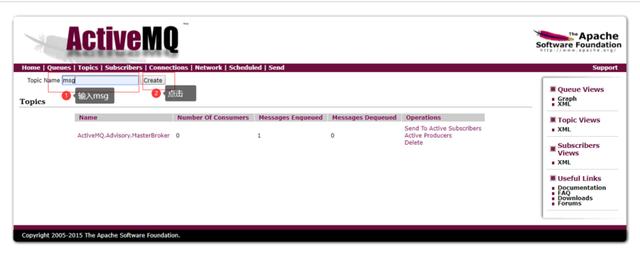
如果没有启动成功，那多半是因为没有jdk，点这里，跟着安装就欧克啦~安装完毕后，再双击activemq.bat，现在我们就启动成功了。在浏览器中输入：http://localhost:8161，可以看到：



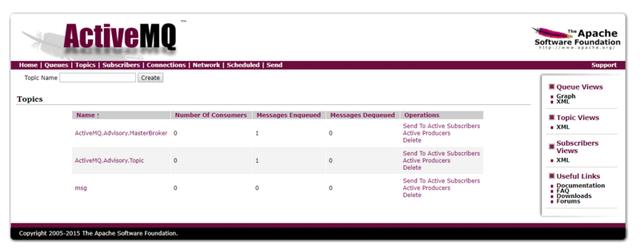
点击Manage ActiveMQ broker，用户名和密码都是admin，然后再点击Topics



在输入框中输入msg，然后点击create按钮



可以看见下方列表中多了一个Name为msg的Topics



其中需要注意的是下面几项，以msg为例：

* Number Of Consumers ：消费者数量，相当于连接服务端msg的客户端的数量；
* MessagesEnqueued：进入队列的消息，相当于服务端向客户端发送的消息数量；
* MessagesDequeued：出了队列的消息，相当于客户端消费（订阅）掉的消息数量。
* 其它想要多了解的可以搜索一波。

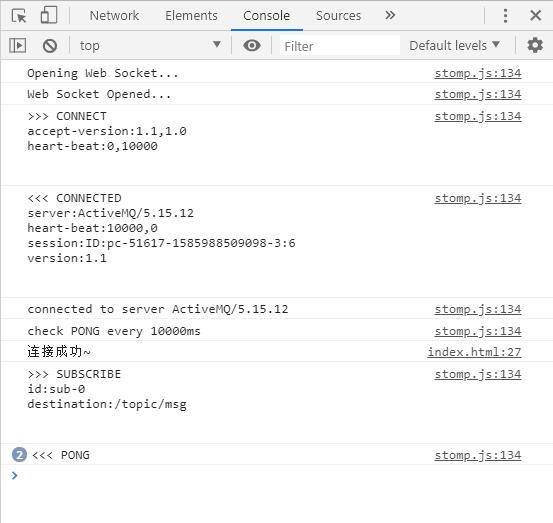
模拟“服务端”准备就绪。

**客户端消息接收**

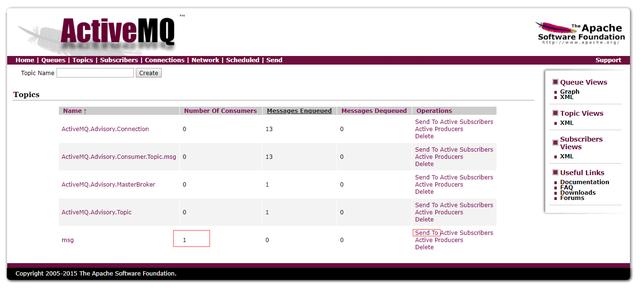
这“服务端”搞好了，接下来就是客户端的实现，代码贴出来index.html：

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>实时语音播报</title>  
</head>  
<body>  
 <script src="http://cdn.bootcss.com/stomp.js/2.3.3/stomp.js"></script>   
 <script>  
 window.onload = function() {  
 let data = '';  
  
 //建立连接  
 function connect(){  
 let client;  
 let url = 'ws:127.0.0.1:61614/stomp';  
 client = Stomp.client(url);  
 client.heartbeat.outgoing=0;  
 client.connect({},  
 //连接成功回调   
 function connectCallback() {  
 console.log("连接成功~");  
 //订阅消息  
   
 // 因为我们订阅的是topic下的msg，所以这里是'/topic/msg'  
 client.subscribe('/topic/msg', function(message){  
 if(message.body){  
 data = message.body;  
 console.log(message.body);  
 }  
 })  
 },  
 //连接失败回调  
 function errorCallBack(error){  
 console.log(error)  
 }  
 )  
 }  
 connect();  
 }  
   
 </script>  
</body>  
</html>

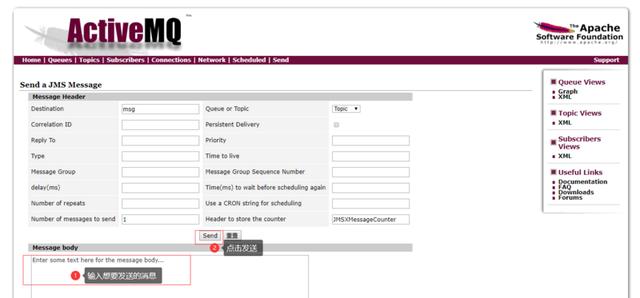
在浏览器中打开这个HTML文件，然后打开控制台，可以看见，我们已经连接服务端成功了：



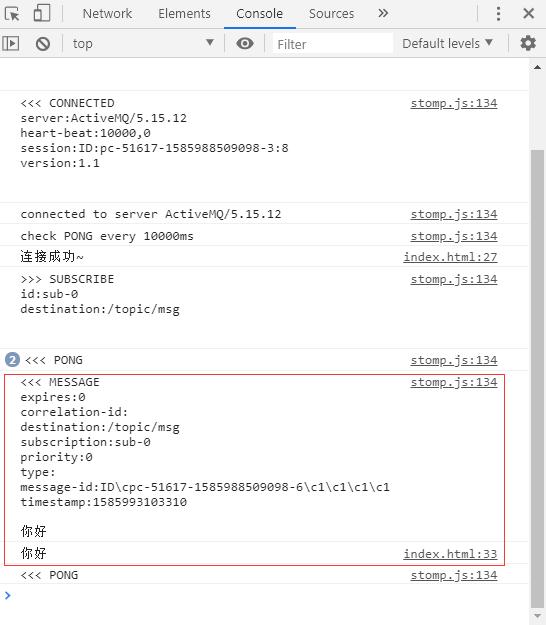
连接成功之后呢，我们就可以尝试在服务端向客户端发送消息，先切换到ActiveMQ 的页面：



可以看见我们的消费者的数量为1了，然后点击Send To，就可以开始发消息了：



比如我们发送一个“你好”，然后我们再切到index.html:



我们收到了来自服务端的问候~

**SpeechSynthesis语音播报**

消息已经能够实时接收了，现在就是需要把接收到的消息读出来，思路很简单，就是把语音合成相关API封装成一个函数，然后当我们服务端发送消息到客户端之后，把消息数据传到为我们定义好的语音播报函数里面，然后就能读出我们服务端发出的消息了，说干就干：

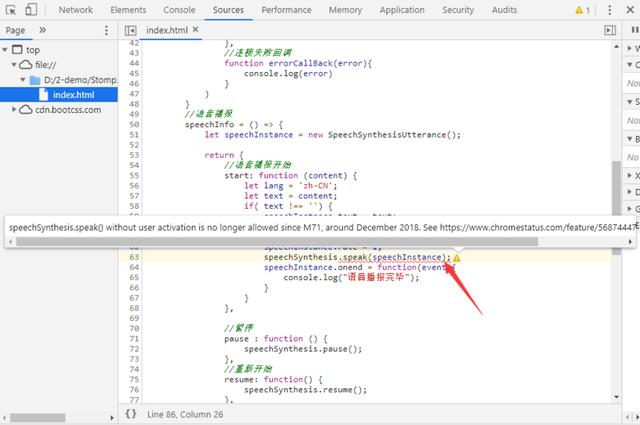
//语音播报  
speechInfo = () => {  
 let speechInstance = new SpeechSynthesisUtterance();  
  
 return {  
 //语音播报开始  
 start: function (content) {   
 let lang = 'zh-CN';  
 let text = content;  
 if( text !== '') {  
 speechInstance.text = text;  
 speechInstance.lang = lang;  
 speechInstance.volume = 1;  
 speechInstance.rate = 1;  
 speechSynthesis.speak(speechInstance);  
 speechInstance.onend = function(event){  
 console.log("语音播报完毕");  
 }  
 }  
 },  
  
 //暂停  
 pause : function () {  
 speechSynthesis.pause();  
 },  
 //重新开始  
 resume: function() {  
 speechSynthesis.resume();  
 },  
 //取消  
 cancel: function() {  
 speechSynthesis.cancel();  
 }  
 }  
};

那咱们调用一下，然后在ActiceMQ页面发送一条新消息，看是不是如我们所愿：

...  
  
client.subscribe('/topic/msg', function(message){  
 if(message.body){  
 data = message.body;  
 console.log(message.body);  
 //调用语音合成函数  
 speechInfo().start(data);  
 }  
})  
  
...

如果是火狐，360安全浏览器等浏览器，我们的消息和声音都如期而至。

但是如果用的是Chrome的话，很难受，并没有声音，难道是Chrome不支持了吗？但是我们之前那两行测试代码说明Chrome是支持SpeechSynthesis的，那是怎么回事？答案就在下面：



Chrome不再支持SpeechSynthesis.speak()的自动播放，要想用的话，必须用户手动去调用它。原因可以看这里被垃圾广告滥用后谷歌浏览器71将限制语音合成自动播放

**垃圾网站出来背锅！！！**

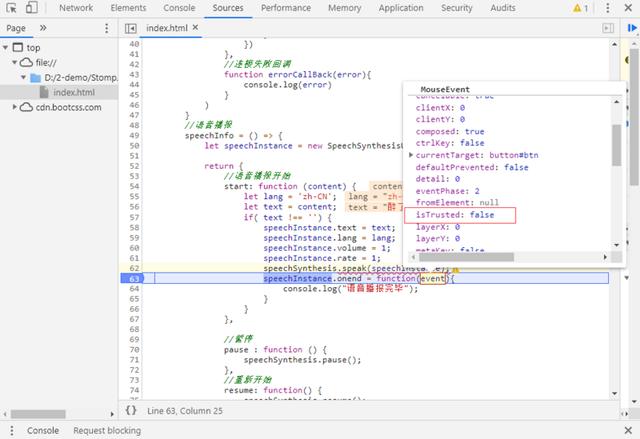
想不到吧，有一天需要去“兼容”Chrome了。语音不能实时的播报出来，我们看看有没有什么办法。我的思路是在页面加一个按钮，然后进行模拟人去点击，完整index.html代码：

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Document</title>  
</head>  
<body>  
 <div>  
 <button id="btn"> 点击</button>  
 </div>  
 <script src="http://cdn.bootcss.com/stomp.js/2.3.3/stomp.js"></script>   
 <script>  
 window.onload = function() {  
 let data = '';  
  
   
 //建立连接  
 function connect(){  
 let client;  
 let url = 'ws:127.0.0.1:61614/stomp';  
 client = Stomp.client(url);  
 client.heartbeat.outgoing=0;  
 client.connect({},  
 //连接成功回调   
 function connectCallback() {  
 console.log("连接成功~");  
 //订阅消息  
   
 client.subscribe('/topic/msg', function(message){  
 if(message.body){  
 data = message.body;  
 console.log(message.body);  
 if(navigator.userAgent.toLowerCase().indexOf("chrome") !== -1){  
 document.getElementById("btn").click();  
 } else {  
 speechInfo().start(data);  
 }  
 }  
 })  
 },  
 //连接失败回调  
 function errorCallBack(error){  
 console.log(error)  
 }  
 )  
 }  
 //语音播报  
 speechInfo = () => {  
 let speechInstance = new SpeechSynthesisUtterance();  
  
 return {  
 //语音播报开始  
 start: function (content) {   
 let lang = 'zh-CN';  
 let text = content;  
 if( text !== '') {  
 speechInstance.text = text;  
 speechInstance.lang = lang;  
 speechInstance.volume = 1;  
 speechInstance.rate = 1;  
 speechSynthesis.speak(speechInstance);  
 speechInstance.onend = function(event){  
 console.log("语音播报完毕");  
 }  
 }  
 },  
  
 //暂停  
 pause : function () {  
 speechSynthesis.pause();  
 },  
 //重新开始  
 resume: function() {  
 speechSynthesis.resume();  
 },  
 //取消  
 cancel: function() {  
 speechSynthesis.cancel();  
 }  
 }  
 };  
 document.getElementById("btn").onclick=function () {  
 console.log("触发成功")  
 speechInfo().start(data);  
 };  
 document.getElementById("btn").click();  
 connect();  
 }  
   
 </script>  
</body>  
</html>

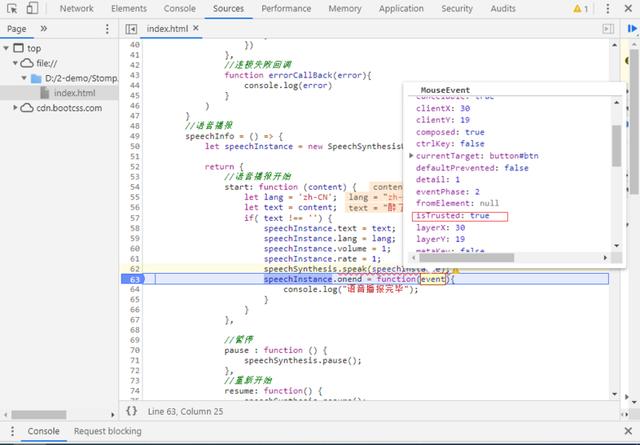
但是我们能想到，谷歌想不到？这里又涉及一个知识点：isTrusted

Event接口的isTrusted是一个Boolean类型的只读属性.当事件由用户操作生成时为true，由脚本创建或修改,或通过调用EventTarget.dispatchEvent生成,为false

我们在控制台代码中打个断点，然后在ActiveMQ 发条消息瞧一瞧，是不是与这个有关：



可以看见isTrusted的值为false，这个模拟点击事件是不被浏览器信任的，然后我们再手动点击一下我们写好的按钮：



isTrusted的值为true，我们的声音也出来了，当我们再在ActiveMQ 发送一条消息：



声音自动播放出来了。

**最后**

需要提一点的是，我在实际是在react中开发的，相关方法都和上述的写法类似，但是却不会触发Chrome对于SpeechSynthesis.speak()的限制，这个限制也是我在写这篇文章的时候，用原生js+HTML的时候发现的。总的来说用原生js+HTML实现的并不算完美，在Chrome下需要在页面加载完成后进行一次点击，才能把后续的语音实时的播报出来。