# 基于netty-socketio 进行全平台消息推送

官网地址：

<https://socket.io/>

<https://github.com/socketio/socket.io>

<https://github.com/mrniko/netty-socketio>

<https://github.com/mrniko/netty-socketio-demo>

## 基本使用

1. 服务器依赖文件

<!-- 服务端 -->

<dependency>

<groupId>com.corundumstudio.socketio</groupId>

<artifactId>netty-socketio</artifactId>

<version>1.7.7</version>

</dependency>

<!-- 客户端 -->

<dependency>

<groupId>io.socket</groupId>

<artifactId>socket.io-client</artifactId>

<version>1.0.0</version>

</dependency>

1. 服务端程序

package com.gwzx.gwzxyw.socket.test;

import org.apache.log4j.Logger;

import com.corundumstudio.socketio.AuthorizationListener;

import com.corundumstudio.socketio.Configuration;

import com.corundumstudio.socketio.HandshakeData;

import com.corundumstudio.socketio.SocketIOClient;

import com.corundumstudio.socketio.SocketIOServer;

import com.corundumstudio.socketio.listener.ConnectListener;

import com.corundumstudio.socketio.listener.DisconnectListener;

import com.gwzx.gwzxyw.socket.SocketManager;

public class ServerDemo {

private static Logger logger = Logger.getLogger(ServerDemo.class);

public static void main(String[] args) {

try {

Configuration config = new Configuration();

config.setHostname("localhost");

config.setPort(9093);

//此处做token认证

config.setAuthorizationListener(new AuthorizationListener() {

@Override

public boolean isAuthorized(HandshakeData data) {

System.out.println(data);

return true;

}

});

SocketIOServer server = new SocketIOServer(config);

CharteventListener listner = new CharteventListener();

listner.setServer(server);

// chatevent为事件名称

server.addEventListener("chatevent", ChatObject.class, listner);

// 连接监听器

server.addConnectListener(new ConnectListener() {

@Override

public void onConnect(SocketIOClient client) {

SocketManager.clietMap.put(client.getSessionId(), client);

logger.info("连接:SessionId=" + client.getSessionId() + ", 总数con:" + SocketManager.clietMap.size());

}

});

// 断开监听器

server.addDisconnectListener(new DisconnectListener() {

@Override

public void onDisconnect(SocketIOClient client) {

// TODO Auto-generated method stub

SocketManager.clietMap.remove(client.getSessionId());

logger.info("断开:SessionId=" + client.getSessionId() + ", 总数dis:" + SocketManager.clietMap.size());

}

});

// 启动服务

server.start();

Thread.sleep(Integer.MAX\_VALUE);

server.stop();

} catch (Exception e) {

e.printStackTrace();

}

}

}

1. Listener事件处理类

package com.gwzx.gwzxyw.socket.test;

import com.corundumstudio.socketio.AckRequest;

import com.corundumstudio.socketio.SocketIOClient;

import com.corundumstudio.socketio.SocketIOServer;

import com.corundumstudio.socketio.listener.DataListener;

public class CharteventListener implements DataListener<ChatObject> {

SocketIOServer server;

public void setServer(SocketIOServer server) {

this.server = server;

}

@Override

public void onData(SocketIOClient client, ChatObject data,

AckRequest ackSender) throws Exception {

// TODO Auto-generated method stub

// chatevent为 事件的名称，data为发送的内容

System.out.println("服务器收到 "+ data.getUserName() +" 消息:"+data.getMessage());

System.out.println("服务器广播消息给borad事件");

data.setMessage("server");

this.server.getBroadcastOperations().sendEvent("broad", data);

}

}

1. JAVA 客户端

package com.gwzx.gwzxyw.socket.test;

import io.socket.client.IO;

import io.socket.client.Socket;

import io.socket.emitter.Emitter;

import org.json.JSONException;

import org.json.JSONObject;

public class ClientDemo {

private static Socket socket = null;

public static void main(String[] args) {

try {

IO.Options opts = new IO.Options();

opts.query = "token=123456"; //传参数

socket = IO.socket("http://localhost:9093", opts);

socket.on(Socket.EVENT\_CONNECT, new Emitter.Listener() {

@Override

public void call(Object... args) {

System.out.println("客户端连接成功");

JSONObject json = new JSONObject();

try {

json.put("userName", "gmh");

json.put("message", "gmh");

} catch (JSONException e) {

e.printStackTrace();

}

System.out.println("客户端发送测试数据:" + json.toString());

socket.emit("chatevent", json);

}

}).on("broad", new Emitter.Listener() {

@Override

public void call(Object... args) {

JSONObject obj = (JSONObject)args[0];

System.out.println("客户端 收到 服务器broad事件数据:" + obj);

}

}).on(Socket.EVENT\_DISCONNECT, new Emitter.Listener() {

@Override

public void call(Object... args) {

System.out.println("disconnect");

}

});

socket.connect();

} catch (Exception e){

e.printStackTrace();

}

}

}

1. JS 客户端

下载socket.io.js 连接 <http://www.bootcdn.cn/socket.io/>

var socket = io.connect('locahost:9093',{query: 'token=123456'});

socket.on('connect', () => {

console.log("Client connect=====>socket id:" + socket.id);

socket.emit('chatevent', {

userName : "123",

message : "测试"

});

});

socket.on('broad', function (data) {

console.log(data);

});

socket.on('disconnect', function () {

console.log("Client disconnected!");

});

socket.on('reconnect', function () {

console.log("Client reconnected!");

});

socket.on('reconnecting', function () {

console.log("Client reconnecting'!");

});

1. 完整js方法

<script type="text/javascript">

var socket = io.connect('http://localhost:9092');

socket.on('connect',function() {

output('<span class="connect-msg">Client has connected to the server!</span>');

});

socket.on('chatevent', function(data) {

output('<span class="username-msg">' + data.userName + ' : </span>'

+ data.message);

});

socket.on('disconnect',function() {

output('<span class="disconnect-msg">The client has disconnected! </span>');

});

function sendDisconnect() {

socket.disconnect();

}

function sendMessage() {

var userName = $("#name").val()

var message = $('#msg').val();

$('#msg').val('');

socket.emit('chatevent', {

userName : userName,

message : message

});

}

function output(message) {

var currentTime = "<span class='time' >" + new Date() + "</span>";

var element = $("<div>" + currentTime + " " + message + "</div>");

$('#console').prepend(element);

}

</script>

## 使用Socket.IO实现websocket

1. java部分

package test;

import java.io.IOException;

import java.util.concurrent.CopyOnWriteArraySet;

import javax.websocket.\*;

import javax.websocket.server.PathParam;

import javax.websocket.server.ServerEndpoint;

/\*\*

\* @ServerEndpoint 注解是一个类层次的注解，它的功能主要是将目前的类定义成一个websocket服务器端,

\* 注解的值将被用于监听用户连接的终端访问URL地址,客户端可以通过这个URL来连接到WebSocket服务器端

\*/

@ServerEndpoint(value = "/websocket/{param}")//{}中的数据代表一个参数，多个参数用/分隔

public class WebSocketTest {

private String uname;

//

// 静态变量，用来记录当前在线连接数。应该把它设计成线程安全的。

private static int onlineCount = 0;

// concurrent包的线程安全Set，用来存放每个客户端对应的MyWebSocket对象。若要实现服务端与单一客户端通信的话，可以使用Map来存放，其中Key可以为用户标识

private static CopyOnWriteArraySet<WebSocketTest> webSocketSet = new CopyOnWriteArraySet<WebSocketTest>();

// 与某个客户端的连接会话，需要通过它来给客户端发送数据

private Session session;

/\*\*

\* 连接建立成功调用的方法

\*

\* @param session

\* 可选的参数。session为与某个客户端的连接会话，需要通过它来给客户端发送数据

\*/

@OnOpen

public void onOpen(@PathParam(value = "param") String uid, Session session) {

this.session = session;

this.uname = uid;

System.out.println(uid);

webSocketSet.add(this); // 加入set中

addOnlineCount(); // 在线数加1

System.out.println("有新连接加入！当前在线人数为" + getOnlineCount());

}

/\*\*

\* 连接关闭调用的方法

\*/

@OnClose

public void onClose() {

webSocketSet.remove(this); // 从set中删除

subOnlineCount(); // 在线数减1

System.out.println("有一连接关闭！当前在线人数为" + getOnlineCount());

}

/\*\*

\* 收到客户端消息后调用的方法

\*

\* @param message

\* 客户端发送过来的消息

\* @param session

\* 可选的参数

\*/

@OnMessage

public void onMessage(String message, Session session) {

System.out.println("来自客户端的消息:" + message);

// 群发消息

for (WebSocketTest item : webSocketSet) {

try {

if (item.uname.equals(this.uname)) {

item.sendMessage(item.uname + ":" + message);

}

} catch (IOException e) {

e.printStackTrace();

continue;

}

}

}

/\*\*

\* 发生错误时调用

\*

\* @param session

\* @param error

\*/

@OnError

public void onError(Session session, Throwable error) {

System.out.println("发生错误");

error.printStackTrace();

}

/\*\*

\* 这个方法与上面几个方法不一样。没有用注解，是根据自己需要添加的方法。

\*

\* @param message

\* @throws IOException

\*/

public void sendMessage(String message) throws IOException {

this.session.getBasicRemote().sendText(message);

// this.session.getAsyncRemote().sendText(message);

}

public static synchronized int getOnlineCount() {

return onlineCount;

}

public static synchronized void addOnlineCount() {

WebSocketTest.onlineCount++;

}

public static synchronized void subOnlineCount() {

WebSocketTest.onlineCount--;

}

}

1. js部分

<!DOCTYPE html>

<html>

<head>

<title>Java后端WebSocket的Tomcat实现</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

</head>

<body>

Welcome<br/><input id="text" type="text"/>

<button onclick="send()">发送消息</button>

<hr/>

<button onclick="closeWebSocket()">关闭WebSocket连接</button>

<hr/>

<div id="message"></div>

</body>

<script type="text/javascript">

var uid = "admin";

var websocket = null;

//判断当前浏览器是否支持WebSocket

if ('WebSocket' in window) {

websocket = new WebSocket("ws://localhost:8080/WebSocketTest/websocket/"+uid);

}

else {

alert('当前浏览器 Not support websocket')

}

//连接发生错误的回调方法

websocket.onerror = function () {

setMessageInnerHTML("WebSocket连接发生错误");

};

//连接成功建立的回调方法

websocket.onopen = function () {

setMessageInnerHTML("WebSocket连接成功");

}

//接收到消息的回调方法

websocket.onmessage = function (event) {

setMessageInnerHTML(event.data);

}

//连接关闭的回调方法

websocket.onclose = function () {

setMessageInnerHTML("WebSocket连接关闭");

}

//监听窗口关闭事件，当窗口关闭时，主动去关闭websocket连接，防止连接还没断开就关闭窗口，server端会抛异常。

window.onbeforeunload = function () {

closeWebSocket();

}

//将消息显示在网页上

function setMessageInnerHTML(innerHTML) {

document.getElementById('message').innerHTML += innerHTML + '<br/>';

}

//关闭WebSocket连接

function closeWebSocket() {

websocket.close();

}

//发送消息

function send() {

var message = document.getElementById('text').value;

websocket.send(message);

}

</script>

</html>

## netty-socketio1.7.15导致客户端无法连接服务器

1. 默认的是使用polling轮询进行连接，所以指定一下链接协议，直接使用websocket进行连接，连接成功。

Var socket= io.connect(‘http://localhost:9092?uuid=123456’,{transports:[‘websocket]});