## **1. socket.io**

Socket.IO是一个WebSocket库，包括了客户端的js和服务器端的nodejs，它的目标是构建可以在不同浏览器和移动设备上使用的实时应用。

## **2. socket.io的特点**

* 易用性：socket.io封装了服务端和客户端，使用起来非常简单方便。
* 跨平台：socket.io支持跨平台，这就意味着你有了更多的选择，可以在自己喜欢的平台下开发实时应用。
* 自适应：它会自动根据浏览器从WebSocket、AJAX长轮询、Iframe流等等各种方式中选择最佳的方式来实现网络实时应用，非常方便和人性化，而且支持的浏览器最低达IE5.5。

## **3. 初步使用**

### **3.1 安装部署**

使用npm安装socket.io

$ npm install socket.io

### **3.2 启动服务**

创建 app.js 文件

**var** express = require('express');**var** path = require('path');**var** app = express();

app.get('/', **function** (req, res) {

res.sendFile(path.resolve('index.html'));

});

**var** server = require('http').createServer(app);**var** io = require('socket.io')(server);

io.on('connection', **function** (socket) {

console.log('客户端已经连接');

socket.on('message', **function** (msg) {

console.log(msg);

socket.send('sever:' + msg);

});

});

server.listen(80);

### **3.3 客户端引用**

服务端运行后会在根目录动态生成socket.io的客户端js文件 客户端可以通过固定路径/socket.io/socket.io.js添加引用  
客户端加载socket.io文件后会得到一个全局的对象io  
connect函数可以接受一个url参数，url可以socket服务的http完整地址，也可以是相对路径，如果省略则表示默认连接当前路径

创建index.html文件

<script src="/socket.io/socket.io.js"></script><script>

window.onload = **function**(){

**const** socket = io.connect('/');

//监听与服务器端的连接成功事件

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

console.log('连接成功');

});

//监听与服务器端断开连接事件

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

console.log('断开连接');

});

};</script>

### **3.4 发送消息**

成功建立连接后，我们可以通过socket对象的send函数来互相发送消息 修改index.html

**var** socket = io.connect('/');

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

//客户端连接成功后发送消息'welcome'

socket.send('welcome');

});//客户端收到服务器发过来的消息后触发

socket.on('message',**function**(message){

console.log(message);

});

修改app.js

**var** io = require('scoket.io')(server);

io.on('connection',**function**(socket){

//向客户端发送消息

socket.send('欢迎光临');

//接收到客户端发过来的消息时触发

socket.on('message',**function**(data){

console.log(data);

});

});

## **4. 深入分析**

### **4.1 send方法**

* send函数只是emit的封装
* node\_modules\socket.io\lib\socket.js源码

**function** **send**(){

**var** args = toArray(arguments);

args.unshift('message');

**this**.emit.apply(**this**, args);

**return** **this**;

}

emit函数有两个参数

* 第一个参数是自定义的事件名称,发送方发送什么类型的事件名称,接收方就可以通过对应的事件名称来监听接收
* 第二个参数是要发送的数据

### **4.2 服务端事件**

| **事件名称** | **含义** |
| --- | --- |
| connection | 客户端成功连接到服务器 |
| message | 接收到客户端发送的消息 |
| disconnect | 客户端断开连接 |
| error | 监听错误 |

### **4.3 客户端事件**

| **事件名称** | **含义** |
| --- | --- |
| connect | 成功连接到服务器 |
| message | 接收到服务器发送的消息 |
| disconnect | 客户端断开连接 |
| error | 监听错误 |

## **5. 划分命名空间**

### **5.1 服务器端划分命名空间**

* 可以把服务分成多个命名空间，默认/,不同空间内不能通信 `js

io.on('connection', function (socket) { //向客户端发送消息 socket.send('/ 欢迎光临'); //接收到客户端发过来的消息时触发 socket.on('message',function(data){ console.log('/'+data); }); }); io.of('/news').on('connection', function (socket) { //向客户端发送消息 socket.send('/news 欢迎光临'); //接收到客户端发过来的消息时触发 socket.on('message',function(data){ console.log('/news '+data); }); });

### 5.2 客户端连接命名空间

```js

window.onload = function(){

var socket = io.connect('/');

//监听与服务器端的连接成功事件

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

console.log('连接成功');

socket.send('welcome');

});

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

console.log(message);

});

//监听与服务器端断开连接事件

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

console.log('断开连接');

});

var news\_socket = io.connect('/news');

//监听与服务器端的连接成功事件

news\_socket.on('connect',function(){

console.log('连接成功');

socket.send('welcome');

});

news\_socket.on('message',function(message){

console.log(message);

});

//监听与服务器端断开连接事件

news\_socket.on('disconnect',function(){

console.log('断开连接');

});

};

## **6. 房间**

* 可以把一个命名空间分成多个房间，一个客户端可以同时进入多个房间。
* 如果在大厅里广播 ，那么所有在大厅里的客户端和任何房间内的客户端都能收到消息。
* 所有在房间里的广播和通信都不会影响到房间以外的客户端

### **6.1 进入房间**

socket.join('chat');//进入chat房间

### **6.2 离开房间**

socket.leave('chat');//离开chat房间

## **7. 全局广播**

广播就是向多个客户端都发送消息

### **7.1 向大厅和所有人房间内的人广播**

io.emit('message','全局广播');

### **7.2 向除了自己外的所有人广播**

socket.broadcast.emit('message', msg);

socket.broadcast.emit('message', msg);

## **8. 房间内广播**

### **8.1 向房间内广播**

从服务器的角度来提交事件,提交者会包含在内

//2. 向myroom广播一个事件，在此房间内包括自己在内的所有客户端都会收到消息

io.in('myroom').emit('message', msg);

io.of('/news').in('myRoom').emit('message',msg);

### **8.2 向房间内广播**

从客户端的角度来提交事件,提交者会排除在外

//2. 向myroom广播一个事件，在此房间内除了自己外的所有客户端都会收到消息

socket.broadcast.to('myroom').emit('message', msg);

socket.broadcast.to('myroom').emit('message', msg);

### **8.3 获取房间列表**

io.sockets.adapter.rooms

### **8.4 获取房间内的客户id值**

取得进入房间内所对应的所有sockets的hash值，它便是拿到的socket.id

**let** roomSockets = io.sockets.adapter.rooms[room].sockets;

## **9. 聊天室**

* 创建客户端与服务端的websocket通信连接
* 客户端与服务端相互发送消息
* 添加用户名
* 添加私聊
* 进入/离开房间聊天
* 历史消息

app.js

//express+socket联合使用//express负责 返回页面和样式等静态资源，socket.io负责 消息通信**let** express = require('express');**const** path = require('path');**let** app = express();

app.get('/news', **function** (req, res) {

res.sendFile(path.resolve(\_\_dirname, 'public/news.html'));

});

app.get('/goods', **function** (req, res) {

res.sendFile(path.resolve(\_\_dirname, 'public/goods.html'));

});**let** server = require('http').createServer(app);**let** io = require('socket.io')(server);//监听客户端发过来的连接//命名是用来实现隔离的**let** sockets = {};

io.on('connection', **function** (socket) {

//当前用户所有的房间

**let** rooms = [];

**let** username;//用户名刚开始的时候是undefined

//监听客户端发过来的消息

socket.on('message', **function** (message) {

**if** (username) {

//如果说在某个房间内的话那么他说的话只会说给房间内的人听

**if** (rooms.length > 0) {

**for** (**let** i = 0; i < rooms.length; i++) {

//在此处要判断是私聊还是公聊

**let** result = message.match(/@([^ ]+) (.+)/);

**if** (result) {

**let** toUser = result[1];

**let** content = result[2];

sockets[toUser].send({

username,

content,

createAt: **new** Date()

});

} **else** {

io.in(rooms[i]).emit('message', {

username,

content: message,

createAt: **new** Date()

});

}

}

} **else** {

//如果此用户不在任何一个房间内的话需要全局广播

**let** result = message.match(/@([^ ]+) (.+)/);

**if** (result) {

**let** toUser = result[1];

**let** content = result[2];

sockets[toUser].send({

username,

content,

createAt: **new** Date()

});

} **else** {

io.emit('message', {

username,

content: message,

createAt: **new** Date()

});

}

}

} **else** {

//如果用户名还没有设置过，那说明这是这个用户的第一次发言

username = message;

//在对象中缓存 key是用户名 值是socket

sockets[username] = socket;

socket.broadcast.emit('message', {

username: '系统',

content: `<a>${username}</a> 加入了聊天`,

createAt: **new** Date()

});

}

});

//监听客户端发过来的join类型的消息,参数是要加入的房间名

socket.on('join', **function** (roomName) {

**let** oldIndex = rooms.indexOf(roomName);

**if** (oldIndex == -1) {

socket.join(roomName);//相当于这个socket在服务器端进入了某个房间

rooms.push(roomName);

}

})

//当客户端告诉服务器说要离开的时候，则如果这个客户端就在房间内，则可以离开这个房间

socket.on('leave', **function** (roomName) {

**let** oldIndex = rooms.indexOf(roomName);

**if** (oldIndex != -1) {

socket.leave(roomName);

rooms.splice(oldIndex, 1);

}

});

socket.on('getRoomInfo', **function** () {

console.log(io);

//let rooms = io.manager.rooms;

console.log(io);

});

});// io.of('/goods').on('connection', function (socket) {// //监听客户端发过来的消息// socket.on('message', function (message) {// socket.send('goods:' + message);// });// });

server.listen(8080);/\*\*

\* 1. 可以把服务分成多个命名空间，默认/,不同空间内不能通信

\* 2. 可以把一个命名空间分成多个房间，一个客户端可以同时进入多个房间。

\* 3. 如果在大厅里广播 ，那么所有在大厅里的客户端和任何房间内的客户端都能收到消息。

\*/

index.html

<!DOCTYPE html><html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://cdn.bootcss.com/bootstrap/3.3.1/css/bootstrap.css">

<style>

.user {

color: green;

cursor: pointer;

}

</style>

<title>聊天室</title></head>

<body>

<div class="container">

<div class="row">

<div class="col-md-8 col-md-offset-2">

<div class="panel panel-default">

<div class="panel-heading text-center">

<div>

<button class="btn btn-danger" onclick="join('red')">进入红房间</button>

<button class="btn btn-danger" onclick="leave('red')">离开红房间</button>

</div>

<div>

<button class="btn btn-success" onclick="join('green')">进入绿房间</button>

<button class="btn btn-success" onclick="leave('green')">进入绿房间</button>

</div>

<div>

<button class="btn btn-primary" onclick="getRoomInfo()">

获取房间信息

</button>

</div>

</div>

<div class="panel-body">

<ul class="list-group" id="messages" onclick="clickUser(event)">

</ul>

</div>

<div class="panel-footer">

<div class="row">

<div class="col-md-10">

<input id="textMsg" type="text" class="form-control">

</div>

<div class="col-md-2">

<button type="button" onclick="send()" class="btn btn-primary">发言</button>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<script src="/socket.io/socket.io.js"></script>

<script>

**let** socket = io('/');

**let** textMsg = document.querySelector('#textMsg');

**let** messagesEle = document.querySelector('#messages');

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

console.log('客户端连接成功');

});

socket.on('message', **function** (messageObj) {

**let** li = document.createElement('li');

li.innerHTML = `<span class="user">${messageObj.username}</span>:${messageObj.content} <span class="text-right">${messageObj.createAt.toLocaleString()}</span>`;

li.className = 'list-group-item';

messagesEle.appendChild(li);

});

**function** **send**() {

**let** content = textMsg.value;

**if** (!content)

**return** alert('请输入聊天内容');

socket.send(content);

}

**function** **join**(name) {

//向后台服务器发送一个消息，join name是房间名

socket.emit('join2', name);

}

**function** **leave**(name) {

//向后台服务器发送一个消息，离开某个房间

socket.emit('leave3', name);

}

**function** **getRoomInfo**() {

socket.emit('getRoomInfo');

}

**function** **clickUser**(event) {

console.log('clickUser', event.target.className);

**if** (event.target.className == 'user') {

**let** username = event.target.innerHTML;

textMsg.value = `@${username} `;

}

}

</script></body>

</html>

## **10. 聊天室**

### **10.1 app.js**

**let** express = require('express');**let** http = require('http');**let** path = require('path')**let** app = express();**let** mysql = require('mysql');**var** connection = mysql.createConnection({

host: 'localhost',

user: 'root',

password: 'root',

database: 'chat'

});

connection.connect();

app.use(express.static(\_\_dirname));

app.get('/', **function** (req, res) {

res.header('Content-Type', "text/html;charset=utf8");

res.sendFile(path.resolve('index.html'));

});

**let** server = http.createServer(app);//因为websocket协议是要依赖http协议实现握手的，所以需要把httpserver的实例的传给socket.io**let** io = require('socket.io')(server);**const** SYSTEM = '系统';//保存着所有的用户名和它的socket对象的对应关系**let** sockets = {};**let** mysockets = {};**let** messages = [];//从旧往新旧的 slice//在服务器监听客户端的连接

io.on('connection', **function** (socket) {

console.log('socket', socket.id)

mysockets[socket.id] = socket;

//用户名，默认为undefined

**let** username;

//放置着此客户端所在的房间

**let** rooms = [];

// 私聊的语法 @用户名 内容

socket.on('message', **function** (message) {

**if** (username) {

//首先要判断是私聊还是公聊

**let** result = message.match(/@([^ ]+) (.+)/);

**if** (result) {//有值表示匹配上了

**let** toUser = result[1];//toUser是一个用户名 socket

**let** content = result[2];

**let** toSocket = sockets[toUser];

**if** (toSocket) {

toSocket.send({

user: username,

content,

createAt: **new** Date()

});

} **else** {

socket.send({

user: SYSTEM,

content: `你私聊的用户不在线`,

createAt: **new** Date()

});

}

} **else** {//无值表示未匹配上

//对于客户端的发言，如果客户端不在任何一个房间内则认为是公共广播，大厅和所有的房间内的人都听的到。

//如果在某个房间内，则认为是向房间内广播 ，则只有它所在的房间的人才能看到，包括自己

**let** messageObj = {

user: username,

content: message,

createAt: **new** Date()

};

//相当于持久化消息对象

//messages.push(messageObj);

connection.query(`INSERT INTO message(user,content,createAt) VALUES(?,?,?)`, [messageObj.user, messageObj.content, messageObj.createAt], **function** (err, results) {

console.log(results);

});

**if** (rooms.length > 0) {

/\*\*

socket.emit('message', {

user: username,

content: message,

createAt: new Date()

});

rooms.forEach(room => {

//向房间内的所有的人广播 ，包括自己

io.in(room).emit('message', {

user: username,

content: message,

createAt: new Date()

});

//如何向房间内除了自己之外的其它人广播

socket.broadcast.to(room).emit('message', {

user: username,

content: message,

createAt: new Date()

});

});

\*/

**let** targetSockets = {};

rooms.forEach(room => {

**let** roomSockets = io.sockets.adapter.rooms[room].sockets;

console.log('roomSockets', roomSockets);//{id1:true,id2:true}

Object.keys(roomSockets).forEach(socketId => {

**if** (!targetSockets[socketId]) {

targetSockets[socketId] = true;

}

});

});

Object.keys(targetSockets).forEach(socketId => {

mysockets[socketId].emit('message', messageObj);

});

} **else** {

io.emit('message', messageObj);

}

}

} **else** {

//把此用户的第一次发言当成用户名

username = message;

//当得到用户名之后,把socket赋给sockets[username]

sockets[username] = socket;

//socket.broadcast表示向除自己以外的所有的人广播

socket.broadcast.emit('message', { user: SYSTEM, content: `${username}加入了聊天室`, createAt: **new** Date() });

}

});

socket.on('join', **function** (roomName) {

**if** (rooms.indexOf(roomName) == -1) {

//socket.join表示进入某个房间

socket.join(roomName);

rooms.push(roomName);

socket.send({

user: SYSTEM,

content: `你成功进入了${roomName}房间!`,

createAt: **new** Date()

});

//告诉客户端你已经成功进入了某个房间

socket.emit('joined', roomName);

} **else** {

socket.send({

user: SYSTEM,

content: `你已经在${roomName}房间了!请不要重复进入!`,

createAt: **new** Date()

});

}

});

socket.on('leave', **function** (roomName) {

**let** index = rooms.indexOf(roomName);

**if** (index == -1) {

socket.send({

user: SYSTEM,

content: `你并不在${roomName}房间，离开个毛!`,

createAt: **new** Date()

});

} **else** {

socket.leave(roomName);

rooms.splice(index, 1);

socket.send({

user: SYSTEM,

content: `你已经离开了${roomName}房间!`,

createAt: **new** Date()

});

socket.emit('leaved', roomName);

}

});

socket.on('getAllMessages', **function** () {

//let latestMessages = messages.slice(messages.length - 20);

connection.query(`SELECT \* FROM message ORDER BY id DESC limit 20`, **function** (err, results) {

// 21 20 ........2

socket.emit('allMessages', results.reverse());// 2 .... 21

});

});

});

server.listen(8080);

/\*\*

\* socket.send 向某个人说话

\* io.emit('message'); 向所有的客户端说话

\*

\*/

### **10.2 index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"

crossorigin="anonymous">

<style>

.user {

color: red;

cursor: pointer;

}

</style>

<title>socket.io</title>

</head>

<body>

<div class="container" style="margin-top:30px;">

<div class="row">

<div class="col-xs-12">

<div class="panel panel-default">

<div class="panel-heading">

<h4 class="text-center">欢迎来到珠峰聊天室</h4>

<div class="row">

<div class="col-xs-6 text-center">

<button id="join-red" onclick="join('red')" class="btn btn-danger">进入红房间</button>

<button id="leave-red" style="display: none" onclick="leave('red')" class="btn btn-danger">离开红房间</button>

</div>

<div class="col-xs-6 text-center">

<button id="join-green" onclick="join('green')" class="btn btn-success">进入绿房间</button>

<button id="leave-green" style="display: none" onclick="leave('green')" class="btn btn-success">离开绿房间</button>

</div>

</div>

</div>

<div class="panel-body">

<ul id="messages" class="list-group" onclick="talkTo(event)" style="height:500px;overflow-y:scroll">

</ul>

</div>

<div class="panel-footer">

<div class="row">

<div class="col-xs-11">

<input onkeyup="onKey(event)" type="text" class="form-control" id="content">

</div>

<div class="col-xs-1">

<button class="btn btn-primary" onclick="send(event)">发言</button>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<script src="/socket.io/socket.io.js"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"

crossorigin="anonymous"></script>

<script>

let contentInput = document.getElementById('content');//输入框

let messagesUl = document.getElementById('messages');//列表

let socket = io('/');//io new Websocket();

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

console.log('客户端连接成功');

//告诉服务器，我是一个新的客户，请给我最近的20条消息

socket.emit('getAllMessages');

});

socket.on('allMessages', function (messages) {

let html = messages.map(messageObj => `

<li class="list-group-item"><span class="user">${messageObj.user}</span>:${messageObj.content} <span class="pull-right">${new Date(messageObj.createAt).toLocaleString()}</span></li>

`).join('');

messagesUl.innerHTML = html;

messagesUl.scrollTop = messagesUl.scrollHeight;

});

socket.on('message', function (messageObj) {

let li = document.createElement('li');

li.className = "list-group-item";

li.innerHTML = `<span class="user">${messageObj.user}</span>:${messageObj.content} <span class="pull-right">${new Date(messageObj.createAt).toLocaleString()}</span>`;

messagesUl.appendChild(li);

messagesUl.scrollTop = messagesUl.scrollHeight;

});

// click delegate

function talkTo(event) {

if (event.target.className == 'user') {

let username = event.target.innerText;

contentInput.value = `@${username} `;

}

}

//进入某个房间

function join(roomName) {

//告诉服务器，我这个客户端将要在服务器进入某个房间

socket.emit('join', roomName);

}

socket.on('joined', function (roomName) {

document.querySelector(`#leave-${roomName}`).style.display = 'inline-block';

document.querySelector(`#join-${roomName}`).style.display = 'none';

});

socket.on('leaved', function (roomName) {

document.querySelector(`#join-${roomName}`).style.display = 'inline-block';

document.querySelector(`#leave-${roomName}`).style.display = 'none';

});

//离开某个房间

function leave(roomName) {

socket.emit('leave', roomName);

}

function send() {

let content = contentInput.value;

if (content) {

socket.send(content);

contentInput.value = '';

} else {

alert('聊天信息不能为空!');

}

}

function onKey(event) {

let code = event.keyCode;

if (code == 13) {

send();

}

}

</script>

</body>

</html>