基于websocket搭建在线聊天室

Web Sockets是一个在单独的持久连接上提供全双工、双向通信。在 JavaScript 中创建了 Web Socket 之后，会有一个 HTTP 请求发送到浏览器以发起连接。在取得服务器响应后，建立的连接会使用 HTTP 升级从 HTTP 协议交换为 Web Socket 协议。也就是说，使用标准的 HTTP 服务器无法实现 Web Sockets，只有支持这种协议的专门服务器才能正常工作。

使用自定义协议而非 HTTP 协议的好处是，能够在客户端和服务器之间发送非常少量的数据，而不必担心 HTTP 那样字节级的开销。由于传递的数据包很小，因此 Web Sockets 非常适合移动应用。

## 项目搭建

### 前端：

前端主要涉及到html，js，css。

### 服务端：

服务端采用nodejs 搭建，Nodejs是一个基于Chrome JavaScript运行时建立的平台， 用于方便地搭建响应速度快、易于扩展的网络应用。依赖其内置的http模块来搭建支持http请求的服务器，同时依赖ws模块来支持服务端的websocket请求接受和发送。

#### 安装：

下载nodejs安装包，下载地址：官网http://www.nodejs.org/download/

最新的nodejs安装包已经集成npm（nodejs的包管理工具），所以不需要另外再配置npm。

安装完成之后可以用 npm install moduleNames命令安装Node模块。Nodejs内置也一些基本的模块（类似http、url等）不需要通过npm安装就可以直接使用，其他模块则需要安装之后才可以使用。

安装完毕后会产生一个node\_modules目录，其目录下就是安装的各个node模块。

node的安装分为全局模式和本地模式。

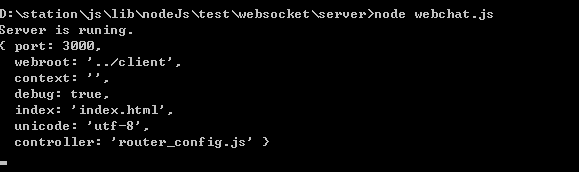
一般情况下会以本地模式运行，包会被安装到和你的应用程序代码的本地node\_modules目录下。

在全局模式下，Node包会被安装到Node的安装目录下的node\_modules下。

项目路径中如果有package.json文件时，直接使用npm install方法就可以根据dependencies配置安装所有的依赖包。

#### 运行

在控制台中输入命令 node 文件名，就可以运行js文件



## 基础代码

### 前端：

首先创建 Web Socket，先示例一个 WebSocket 对象并传入要连接的 URL。

var socket=new WebSocket("ws:"+ window.location.host);

ws标识是一个web socket协议，后面的window.location.host是websocket的服务器地址，该示例中websocket的服务器地址和http服务器地址是同一个，所以采用这种写法。

实例化了 WebSocket 对象后，浏览器就会马上尝试创建连接。与 XHR 类似， WebSocket 也有一个表示当前状态的 readyState 属性。不过，这个属性的值与 XHR 并不相同，而是如下所示。

❑ WebSocket.OPENING (0)：正在建立连接。

❑ WebSocket.OPEN (1)：已经建立连接。

❑ WebSocket.CLOSING (2)：正在关闭连接。

❑ WebSocket.CLOSE (3)：已经关闭连接。

在成功建立连接的时候，会触发一个open事件，我们可以监听这个事件，在事件触发的时候给服务器发送一个建立连接的信息。

socket.onopen = function() {

socket.send(JSON.stringify({

"data": {

"eventName":"connect",

"room": window.location.hash.slice(1),

"name":name

}

}));

};

用send方法就可以给服务端发送信息了。上面的代码用json格式的数据发送了三个主要信息（均为自定义的，web socket没有限制数据格式）给服务端。以该项目为例，eventName标识置为“connect”标识是一个新的用户尝试与聊天室连接，room标识是哪个聊天室，name标识是哪个用户。

服务端在成功收到请求后，如果它认为要响应你的请求，就会回发一个消息给你，如果不响应，前端不会受到任何信息。

接受服务端的响应通过给web socket实例添加message事件实现。

socket.onmessage = function(message) {

console.log("receive message");

var json = JSON.parse(message.data);

if (json.eventName==="sys\_message") {

windowNode.innerHTML+=json.message+"<br>";

}else{

windowNode.innerHTML+=json.message+"<br>";

}

};

上面代码中只是简单的把服务端回发的信息打印到div上。

除了open，和message事件外，websocket还支持error和close事件，分别在发生错误时触发，连接不能持续和在连接关闭时触发。

要关闭一个websocket可以直接调用close方法。

到这里，前端的代码就基本满足实现一个简单的公共聊天室了。

### 服务端：

Webchat.js是项目的主要代码文件。

path=require('path');

ws=require("ws").Server;

var s2server=require(path.resolve("s2"));

s2server.setWebConfig("config.js");

s2server.setControllers("router\_config");

s2server.listen();

首先我们要搭建一个http服务器来满足网站正常的http请求。S2就是我们本地主机实现的一个基于http模块的服务器。我们首先在http模块基础上构建出一个自己的服务器组件。该代码中我把它命名为s2，（实际在项目搭建中我们可以去借助现成的http服务器，比方说express，去满足相对比较复杂的业务需求）。

下面看下s2这个包的核心代码：

s2 = {

http: require('http'),

path: require('path'),

url: require('url'),

fs: require('fs'),

util: require("util"),

querystring:require("querystring")

};

s2.config = require("./config/config");

s2.request = require("./request");

s2.mine = require("./config/mine\_config");

s2.error = require("./error");

s2.controllers = require("./config/router\_config");

s2.routers = require("./router");

这个是s2这个包所有的依赖。Config用来配置网站，request处理前端请求，mine\_config定义文件格式，error处理错误信息，router\_config配置路由文件，router实现路由。

我们重点关注s2 http服务器创建的核心代码部分

var server = s2.http.createServer(function(req, res) {

try {

var request =req;

var result = s2.routers.route(request, res);

} catch(e) {

s2.error.type404(req, res, e);

}

});

module.exports = {

server:server,

setWebConfig: function(path) {

s2.config = require(s2.path.resolve(path));

},

setMine: function(path) {

s2.mine = require(s2.path.resolve(path));

},

setControllers: function(path) {

s2.controllers = require(s2.path.resolve(path));

},

setError: function(path) {

s2.mine = require(s2.path.resolve(path));

},

listen: function() {

server.listen(s2.config.port);

console.log("Server is runing.");

console.log(s2.util.inspect(s2.config));

},

}

createServer方法就会返回一个server实例，该方法接受一个回调函数用来处理http请求收到的时候要如何处理，上面代码中就交给我了router这个模块去处理具体的请求。

然后在暴露出来的方法中我们重点关注listen这个方法，server.listen之后服务器就会正式开始监听http请求。

到这里http服务器搭建就基本完成了。

Websocket服务器部分，我们主要依赖ws这个外置的nodejs模块。（上面s2部分用到的包都是nodejs内置的模块）。

var wsserver= new ws({server: s2server.server});

把s2server.server这个作为参数传入ws，创建一个服务端的websocket实例。

之后就是设置他的监听事件。

wsserver.on('connection',function(ws){

console.log('new connection founded successfully');

ws.on('message',function(data){

data=JSON.parse(data).data;

if(data.eventName=="connect"){

console.log('join'+data.room);

if(!rooms[data.room]){

rooms[data.room]=[];

}

rooms[data.room].push(ws);

console.log(rooms[data.room].length);

for(var i =0;i<rooms[data.room].length;i++){

rooms[data.room][i].send(JSON.stringify({

"eventName":"sys\_message",

"message":data.name +" joins "

}));

}

}else{

for(var i =0;i<rooms[data.room].length;i++){

rooms[data.room][i].send(JSON.stringify({

"eventName":"message",

"message":data.name+":"+data.message

}));

}

}

});

ws.on('close',function(){

for(var o in rooms){

for(var i =0;i<rooms[o].length;i++){

if(ws==rooms[o][i]){

rooms[o].splice(i,1);

return;

}

}

}

});

})

Ws在收到客户端的ws请求之后，首先会触发connection事件，在回调函数中会传入新创建的web socket链接。在服务端我们需要将每个有效的web socket链接存储起来。这里我们用rooms这个对像去存储，简单的用房间号作为key，value是一个为此房间号的websocket组成的数组。代码如下：

if(data.eventName=="connect"){

console.log('join'+data.room);

if(!rooms[data.room]){

rooms[data.room]=[];

}

rooms[data.room].push(ws);

之后就可以根据前端的房间号来选择给哪些web socket发生消息：

for(var i =0;i<rooms[data.room].length;i++){

rooms[data.room][i].send(JSON.stringify({

"eventName":"message",

"message":data.name+":"+data.message

}));

}

再给每个websocket添加一个close事件监听，在这个websocket关闭的时候就可以从rooms中将其移除。

ws.on('close',function(){

for(var o in rooms){

for(var i =0;i<rooms[o].length;i++){

if(ws==rooms[o][i]){

rooms[o].splice(i,1);

return;

}

}

}

});

这样整个服务器端就基本搭建完成。

## 项目运行

在nodejs环境配置好之后，在server文件下双击start.bat，启动服务端。

在浏览器中输入ip:port#roomnumber 就可以进行实时通信，

例如：<http://10.223.51.192:3000/#1>，示例中端口号为3000