Socket.io

<https://socket.io/docs/v3>

Socket.io提供了基于事件的实时双向通讯

Browser和WebServer间的实时数据传输, 最早只能通过AJAX轮询方式实现。

AJAX轮询原理是设置定时器，定时通过AJAX同步服务端数据。这种方式存在延时且对服务端造成很大负载

WebSocket 协议

WebSocket是HTML5新增的一种通信协议，其特点是服务端可以主动向客户端推送信息，客户端也可以主动向服务端发送信息，是真正的双向平等对话，属于服务器推送技术的一种。

在WebSocket API中，浏览器和服务器只需要做一个握手的动作，然后浏览器和服务端之间就形成了一条快速通道，两者之间就直接可以数据相互传送，带来的好处是：

* 相互沟通的Header很小，大概只有2Bytes。
* 服务器不再被动的接收到浏览器的请求之后才返回数据，而是在有新数据时就主动推送给浏览器。

建立websocket连接过程：

* 浏览器首先要向服务器发起一个HTTP请求，这个请求和通常的HTTP请求不同，包含了一些附加头信息，其中附加头信息Upgrade: WebSocket表明这是一个申请协议升级的HTTP请求。
* 服务端解析这些头信息，然后产生应答信息返回给客户端，连接即建立
* 双方自由通信，知道一方断开连接

Socket.io 应用

服务端：

服务端socket.io必须绑定一个http.Server实例，因为WebSocket协议是构建在HTTP协议之上的，所以在创建WebSocket服务时需调用HTTP模块并调用其下createServer()方法，将生成的server作为参数传入socket.io。

|  |
| --- |
| var httpServer = require('http').createServer();  var io = require('socket.io')(httpServer);  httpServer.listen(3000); |

**隐式绑定：**

方式一：

唯一参数：options

|  |
| --- |
| const options = { /\* ... \*/ };  const io = require('socket.io')(**options**);  io.on('connection', socket => { /\* ... \*/ });  io.listen(3000); |

方式二：

第一个参数：port，第二个参数options

|  |
| --- |
| const options = { /\* ... \*/ };  const io = require('socket.io')(**3000, options**);  io.on('connection', socket => { /\* ... \*/ }); |

隐式创建一个http server,可以通过**io.httpServer**访问到。

**显示绑定：**

绑定到一个已经存在http server上

参数一：server, 参数二：options

|  |
| --- |
| const server = require('http').createServer();  const options = { /\* ... \*/ };  const io = require('socket.io')(**server, options**);  io.on('connection', socket => { /\* ... \*/ });  server.listen(3000); |

**在Express中使用：**

|  |
| --- |
| const app = require('express')();  const server = require('http').createServer(app);  const options = { /\* ... \*/ };  const io = require('socket.io')(server, options);  io.on('connection', socket => { /\* ... \*/ });  server.listen(3000); |

建立连接：

当服务端和客户端连接成功时，服务端会监听到connection事件

客户端会监听connect事件和disconnect事件

服务端：

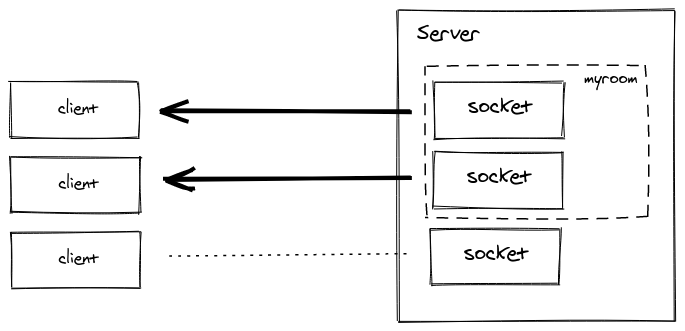
|  |
| --- |
| const { isObject } = require('lodash');  const app = require('express')();  let httpServer = require('http').createServer(app);  let io = require('socket.io')(httpServer, {cors: ['http://127.0.0.1:8000']});  // 服务端监听连接状态：io的connection事件表示客户端与服务端成功建立连接，它接收一个回调函数，回调函数会接收一个socket参数。  io.on('connection', (socket)=>{  console.log('client connect server, ok!');  // 加入到一个以socket.io命名的room里  socket.join(socket.id);  // 向socket.id命名的room下的所有socket触发事件：  io.to(socket.id).emit('server message', socket.id);  // io.emit()方法用于向服务端发送消息，发送所有命名空间"/"下的client, 参数1表示自定义的数据名，参数2表示需要配合事件传入的参数  io.emit('server message', {msg:'client connect server success'});  // socket.broadcast.emit()表示向除了自己以外的客户端发送消息  socket.broadcast.emit('server message', {msg:'broadcast'});  // 监听断开连接状态：socket的disconnect事件表示客户端与服务端断开连接  socket.on('disconnect', ()=>{  console.log('connect disconnect');  });    // 与客户端对应的接收指定的消息  socket.on('client message', (data, data2)=>{  console.log(data, data2);  });  // socket.disconnect();  });  httpServer.listen(8000, () => {  // console.log('ready');  });  app.get('/test', (req, res) => {  io.emit('test', 'I am testing');  }); |

客户端：

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="https://cdn.bootcdn.net/ajax/libs/socket.io/3.0.0-rc3/socket.io.js"></script>  <script>  // socket.io引入成功后，可通过io()生成客户端所需的socket对象。  // let socket = io('http://127.0.0.1:8000',{transports: ['websocket']});  // 不设置transports的话，默认会走polling模式，服务端需要配置cors跨域，若配置transprots: [‘websocket’]则不必配跨域；  let socket = io('http://127.0.0.1:8000');  socket.on('server message', (data) => {  console.log('server:', data);  })  socket.on('test', data => {  console.log(data);  })  // socket.emit()用户客户端向服务端发送消息，服务端与之对应的是socket.on()来接收信息。  socket.emit('client message', {msg:'hi, server'}, 'hahahah');  // socket.on()用于接收服务端发来的消息  socket.on('connect', ()=>{  console.log('client connect server');  });  socket.on('disconnect', ()=>{  console.log('client disconnect');  });  </script>  </head>  <body>    </body>  </html> |

socket.emit(event, data1,data2) 第二个参数，或者更多参数，可以通过socket.on(event, (data1, data2) => {}) 回调函数里的data1,data2获取。

**Room：**



可以加入一个房间，方便后续针对特定的房间广播

|  |
| --- |
| io.on('connection', socket => {  socket.join('some room');  });  io.to('some room').emit('some event');  或：  io.to('room1').to('room2').to('room3').emit('some event'); |

简单应用：

|  |
| --- |
| io.on('connection', async (socket) => {  const userId = await fetchUserId(socket);  socket.join(userId);  // and then later  io.to(userId).emit('hi');  }); |

**Namespace:**

Default namespace : / , 默认socket.io client 连接默认命名空间，服务器也默认监听这个默认命名空间；

|  |
| --- |
| // the following two will emit to all the sockets connected to `/`  会触发所有/命名空间下的socket连接  io.sockets.emit('hi', 'everyone');  **io.emit**('hi', 'everyone'); // short form |

Socket:

每个新连接，就会产生一个socket实例。