I

Node + Express + Socket.io = chat

A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

Сокет – это конечная точка двусторонней связи между двумя программами, работающими в сети. Сокет привязан к номеру порта, так чтобы слой TCP мог идентифицировать приложение, для которого предназначены данные.

С точки зрения программы сокет – это окошко, через которое виден другой участник взаимодействия. То есть интерфейс к нему.

Событие 'connection' возбуждается когда создаётся очередной сокет. В его коллбэк передаётся информация о сокете - объект. Внутри можно использовать этот объект уже для отслеживания этого конкретного соединения, в том числе отсоединения.

```
io.on('connection', socket=>{
        console.log('a user connected!');
        socket.on('disconnect', ()=>{
            console.log('a user disconnected!');
        });
});
```

Для того, чтобы связь создалась, и это событие возбудилось, нужно, чтобы обратился веб-клиент по соответствующему протоколу. В терминах node/express это означает, что у нас должен быть какой-то маршрут типа /chat/, обращение к которому интерпретируется как обращение веб-клиента чата.

Что такое веб-клиент чата? Это веб-страница, содержащая интерфейс чата и сценарий для взаимодействия с сервером. С ней взаимодействует человеческий клиент чата.

В простейшем виде выглядит так: (first-step)

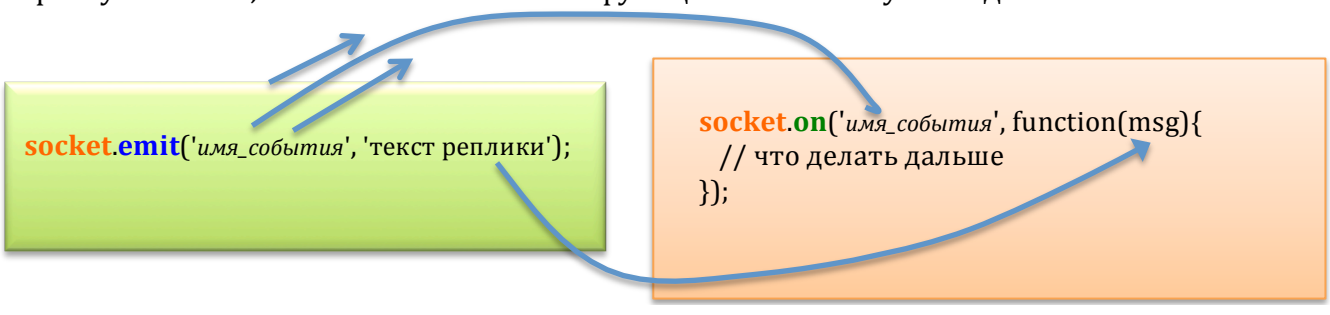
```
io.on('connection', socket => console.log('a user connected!') );

let client = `<script src="/socket.io/socket.io.js"></script><h1>Chat web-client!</h1>
        <script>var socket = io();</script>`;

app.get('/chat', (req, res) => {
    res.set({'Access-Control-Allow-Origin': '*', 'elias': 'goss'});
    res.send(client);
});
```

Адрес `/socket.io/socket.io.js` автоматически генерируется серверной стороной.

выделенная красным строка инициализирует взаимодействие, т.е. посылает запрос, который возбуждает событие **'connection'**.

Но в действительности, конечно, эти строки располагаются в отдельной статической веб-странице.

Далее, мы видим, что и в клиентском, и в серверном сценарии после разных действий появляется переменная с именем socket. На обеих сторонах там содержатся разные по факту объекты, но они оба – сокеты. И функциональность у них одинаковая:

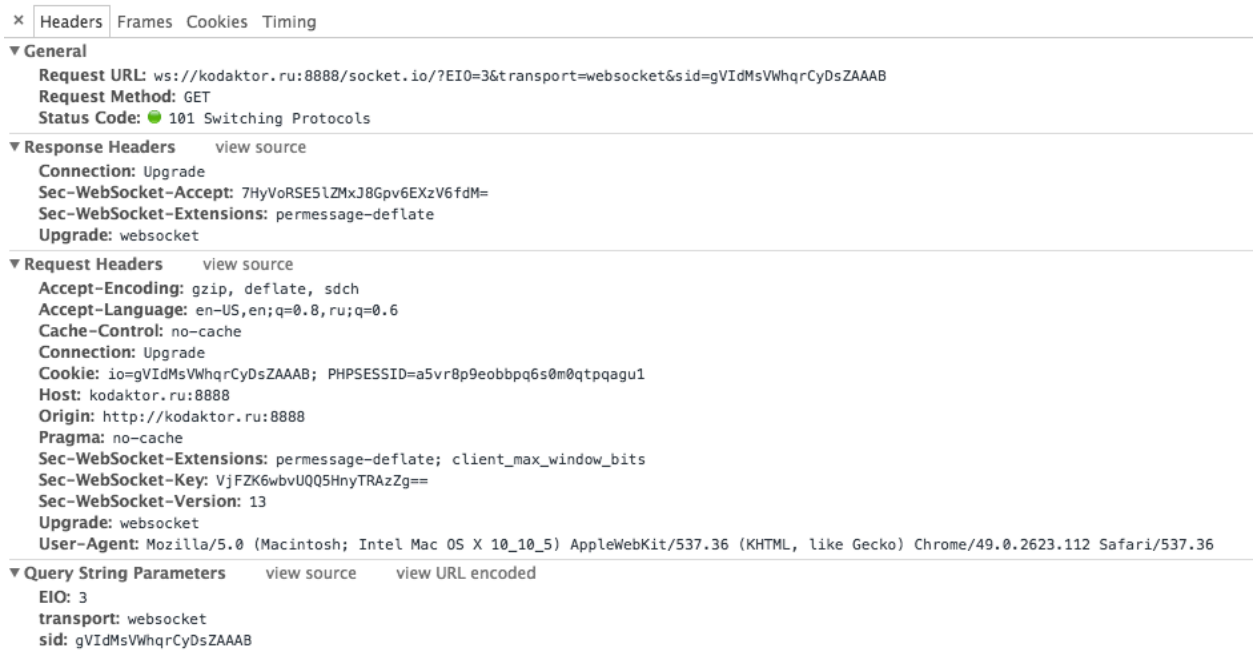


II

Итак, если мы даже встроили в каркас нашего приложения маршрут, который генерирует сокет, мы уже получили в первом приближении модель происходящего:

III

При посещении адреса /chat мы получаем запрос по протоколу websocket:



... и возбуждение события сокета `'connection'`

Далее нужно вывести клиентский код в отдельную страницу и дополнить работу с сокетами командами посылки сообщений и слушания их.

И, в общем-то теперь осталось параллельно добавить в клиентский и серверный код «зеркальные» команды слушания сокета и посылки в него реплики.

см. на следующей странице

IV

```
/*jshint esversion: 6 */
/*jshint -W058 */

const PORT = 5555;
var        express = require('express'),
           app = express(),

           http = require('http').Server(app),
          io = require('socket.io')(http);


module.exports = (()=>{
   function inner(){
      this.start = whatToDo=>{
                  app
                     .use(express.static(__dirname + '/public'))
                     .use((req, res, next)=>next())
                  .get('/', (req, res) => {
                           res.send('<h1>Welcome to Express!</h1>');
                  });


             io.on('connection', socket=>{
                  console.log('a user connected!');

                     socket.broadcast.emit('chat message push', '>> a user connected ');
                                   //everyone except the new one
                     socket.emit('chat message push', 'Welcome to Ilia Goss chat!');
                                   //only the newcomer

                   //message from client - recast to others
                  socket.on('chat message', msg=>{
                      console.log(`message: ${msg}`);
                      socket.broadcast.emit('chat message push', msg);
                      socket.emit('chat message push', `<strong>me:</strong>  ${msg}` );
                  });
                  socket.on('disconnect', ()=>{
                    console.log('a user disconnected!');
                    socket.broadcast.emit('chat message push', '>> a user disconnected ');
                  });
          });


          app

            .get('/chat', (req, res)=>{
                  res.redirect('/page.html');

           })

           .set('port',  process.env.port||PORT );

           http.listen( app.get('port') ,()=>console.log(`--> Port ${ app.get('port') } listening!!!`));
      };
    }
  return new inner;
})();
```

Это близкая к финальной версия server.js
На след.странице – аналогично для веб-клиента client.html

http://kodaktor.ru/express_clientchat

```
<!DOCTYPE html>
<html>
 <head>
  <title>Express Chat Form</title>
   <meta charset="utf-8"><style>* {font-family:sans-serif} span {position:fixed; right:20px; top:10px;border:double; padding: 15px; border-
radius:20px;} p {border: solid 1px silver}</style>
  <script src="//ajax.googleapis.com/ajax/libs/jquery/2.2.2/jquery.min.js"></script>
  <script src="/socket.io/socket.io.js"></script>
  <script>
     $(function(){
        var backs = [{"background":"green"}, {"background":"white"}];
        var socket = io();
        socket.on('chat message push', function(msg){
           $('div').append($('<p>').html(msg));
        });

       $('#b').on('click', ()=>{
              socket.emit('chat message', $('#i').val()); //послать сообщение серверной части
              $('#i').val('').css(backs[0]); //очистить поле ввода сообщения
              setTimeout(function(){  $('#i').css(backs[1]);  }, 500);
            });
     });

  </script>
 </head>
 <body>
    <h1>Содержимое чата:</h1>
    <div></div>
    <span><h2>Введите реплику:</h2>
      <input id="i"><button id="b">Написать!</button></span>
 </body>
</html>
```