

УДК 004.7

Реализация сервера для установления соединения и непрерывного обмена сообщениями между парами компьютер-мобильное устройство

Павлов М.С., студент

*Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»*

Научный руководитель: Гапанюк Ю. Е., к.т.н, доцент

Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана

gapyu@bmstu.ru

Введение:

На данный момент существует значительное число технологий и подходов для передачи данных между двумя устройствами. Все эти решения можно разделить на 2 типа: зависящие от платформы (нативные) и кроссплатформенные (веб-решения). Достаточно часто стоит проблема синхронизации мобильного устройства и компьютера. Цель статьи - описание одного из оптимальных способов такого связывания этих устройств для дальнейшей передачи большого количества сообщений, которые обязательно должны передаваться, если существует синхронизация между устройствами.

В статье для наглядности рассмотрен пример клиент-сервер-клиентского приложения, обе клиентские части которого можно заменить на обработчики каких-либо событий, происходящих на каждом из клиентов. Таким образом, такое приложение может служить либо сервисом для обмена текстовыми сообщениями вне зависимости от того, как эти сообщения отправляются: человеком или обработчиками событий.

Задачи системы:

1. Организация постоянного (непрерывного) соединения между сервером и клиентами
2. Удобное связывание смартфона и компьютера
3. Понятное отображение входящих и исходящих сообщений.

Для достижения кросс-платформенности будем реализовывать веб-клиент (в браузере). Это обеспечит не только единство (единообразие) программного кода для любой

платформы, но и значительно более удобную поддержку. Для будущего добавления новых функций или любого другого исправления кода не нужно заботиться о том, чтобы пользователи что-то сделали, чтобы их клиент обновился. Достаточно только нажатия кнопки браузера «обновить страницу».

Сейчас существуют 2 способа организации непрерывного соединения между клиентом и сервером:

1. AJAX - *Asynchronous Javascript and XML* — «асинхронный JavaScript и XML»

2. Web Sockets - протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

1. AJAX — подход к построению веб-приложений, использующий для запросов к серверу Javascript. Использование javascript обеспечивает обновления данных на странице без ее перезагрузки. Чаще всего это происходит с помощью отправки на сервер XMLHttpRequest (XHR) и отображения ответа сервера в каком-либо элементе страницы.

Данный подход достаточно удобен для использования при создании систем обмена сообщениями. Однако, в случае большого числа передаваемых сообщений нагрузка как на серверную часть приложения, так и на клиентскую сильно возрастает. Количество сообщений становится большим, когда сообщения отправляются не самим человеком, а клиентской частью приложения.

При подключении большого числа клиентов сервер должен за момент времени обрабатывать количество соединений, равное числу клиентов, умноженное на количество XHR в момент времени. Таким образом, для частого обновления данных на странице такая технология не подходит из-за большой нагрузки на сервер.

2. Web Sockets — достаточно «молодая» технология передачи данных. Это протокол передачи данных, реализуемый поверх протокола TCP. Для его использования между сервером и клиентом создается соединение с помощью javascript. Код запроса представлен на рисунке 1.

```
01. | var myWebSocket = new WebSocket("ws://www.example.com");
```

Рис. 1. Код на языке JavaScript, устанавливающий соединение по WebSocket

При выполнении браузером следующего кода на сервер посылается GET запрос с «просьбой» переключить протокол обмена данными с HTTP на Web Sockets (WS). Характерными заголовками для такого запроса будут:

Connection: Upgrade

Sec-WebSocket-Key: (случайно сгенерированная браузером строка)

Upgrade: websocket

Sec-WebSocket-Version: версия протокола

В ответ на такой запрос, если сервер имеет возможность устанавливать WS соединение, отдает ответ с заголовками следующего вида:

HTTP/1.1 101 WebSocket Protocol Handshake

Date:

Connection: Upgrade

Server: Kaazing Gateway

Upgrade: WebSocket

Access-Control-Allow-Origin: url сайта

Access-Control-Allow-Credentials: true

Sec-WebSocket-Accept: (случайно сгенерированная сервером строка)

Access-Control-Allow-Headers: content-type

После этого HTTP-соединение обрывается и заменяется на WebSocket-соединение поверх соединения по TCP/IP. Соединение WebSocket работает по тому же порту, что и HTTP (80) и HTTPS(443). В дальнейшем все сообщения между клиентом и сервером будут идти по уже установившемуся соединению.

Таким образом, если требуется написать клиент-серверное приложение, в котором будет нужно интенсивно использовать соединение и соединение должно быть постоянным, то лучшим выбором будет использование протокола Web Sockets.

Теперь разберем задачи, которые стоят при разработке сервера, который должен устанавливать соединение между браузером в смартфоне и браузером компьютера. В-первых нужно использовать удобный для пользователей механизм, с помощью которого можно было бы быстро идентифицировать смартфон и компьютер.

Самый удобный для пользователя вариант — при заходе из браузера компьютера «выдавать» ему короткий токен - сгенерированную случайным образом короткую строку. Такую строку будет удобно ввести при заходе из браузера смартфона. Со стороны

пользователя механизм соединения с компьютером будет выглядеть следующим образом:

1. Пользователь заходит на сайт, видит токен.
2. Заходит на тот же сайт с телефона, вводит токен
3. Соединение устанавливается и сообщения с мобильного устройства передаются на компьютер и наоборот.

Для удобства будущей поддержки и клиент и сервер напишем на языке JavaScript. Такой подход позволит в дальнейшем использовать данный сервер человеку, который знает только Javascript. На стороне клиента JavaScript-код встраивается в HTML-страницу, а в случае сервера используется веб-сервер node.js, позволяющий обрабатывать подключения с помощью языка JavaScript.

Определимся подробнее, как будет устанавливаться соединение. При заходе клиента на сайт на сервере потребуется определить, это смартфон или компьютер. При запросе страницы с сервера браузер отправляет некоторые данные об устройстве, операционной системе, названии и версии браузерного движка. Эти данные он отправляет в одном из заголовков HTTP-запроса **user-agent**. По этому заголовку мы будем идентифицировать вид пришедшего клиента.

Для соединения с помощью WebSockets будем использовать библиотеку SockJS, в состав которой входит минифицированный Javascript-файл для работы из браузера и модуль для Node.js.

Создадим обработчик GET-запросов. Код серверной части представлен на рисунке 2.

```
01. var http = require('http');
02. var server = http.createServer();
03. server.addListener('request', function(req, res) {
04.     if (req["method"] === "GET"){
05.         console.log("GET-Request!");
06.         res.write("200 OK");
07.         res.end();
08.     }
09. });
10. server.listen(9998, '0.0.0.0');
```

Рис. 2. Код в файле server.js

Теперь сервер в ответ на GET-запрос отдает строку «200 OK».

Создадим простую HTML-страницу, основу для клиентской части. Код страницы представлен на рисунке 3.

```

01. <!doctype html>
02. <html>
03. <head>
04. <style type="text/css">
05. .chat{
06.     height: 450px;
07.     width: 380px;
08.     position: relative;
09.     margin: 50px auto;
10. }
11. .toolbar{
12.     position: absolute;
13.     top: 0px;
14.     left: 0px;
15.     width: 100%;
16.     height: 50px;
17.     background: #515151;
18.     border-top-left-radius: 7px;
19.     border-top-right-radius: 7px;
20.     color: #fff;
21.     text-align: center;
22. }
23. .toolbar_text{
24.     padding-top: 15px;
25.     font-size: 20px;
26.     font-family: serif;
27. }
28. .messages{
29.     border: 1px solid #515151;
30.     border-radius: 7px;
31.     height: 400px;
32.     padding: 10px;
33.     padding-top: 50px;
34. }
35. }
36. .input{
37.     height: 45px;
38.     border: 1px solid #515151;
39.     border-radius: 7px;
40.     margin-top: 2px;
41.     padding: 5px;
42.     padding-left: 8px;
43. }
44. #inp{
45.     width: 90%;
46.     height: 35px;
47.     border: none;
48.     outline: none;
49.     font-size: 20px;
50. }
51. </style>
52. </head>
53. <body>
54. <div class="chat">
55.     <div class="toolbar">
56.         <div class="toolbar text">
57.             WebSockets Chat
58.         </div>
59.     </div>
60.     <div class="messages" id="messages">
61.     </div>
62.     <div class="input">
63.         <form id="form">
64.             <input type="text" name="message" id="inp">
65.         </form>
66.     </div>
67. </div>
68. </body>
69. </html>

```

Рис. 3. Код файла page.html.

И начнем «отдавать» ее с сервера в ответ на GET-запрос. Для этого изменим код сервера так, как показано на рисунке 4.

```

01. var http = require('http');
02. var fs = require('fs');
03.
04. var server = http.createServer();
05. server.addListener('request', function(req, res) {
06.     if (req["method"] === "GET"){
07.         console.log("GET-Request!");
08.         fs.readFile('page.html', function (err, data) {
09.             if (err){
10.                 console.log("file", "page.html", "not opened");
11.                 throw err;
12.             }
13.             res.write(data.toString());
14.             res.end();
15.         });
16.     }
17. });
18. server.listen(9998, '0.0.0.0');
19.

```

Рис. 4. Измененный код сервера, открывающий файл и отдающий его по GET-запросу

Логика подключения клиента к серверу будет следующая:

1. GET-запрос с клиента
2. С сервера отдается страница в зависимости от типа клиента
3. Клиент подключается к серверу по WebSockets
4. Сервер посылает компьютеру токен / ждет токен от мобильного устройства.

Создадим 2-е разные страницы для компьютера и мобильного устройства. Страница, отдаваемая на мобильные устройства будет подключаться по URL «/mobile», а компьютер по «/desktop». На сервере будут два обработчика подключений, у каждого своя логика работы. Добавим на страницу код, устанавливающий подключение так, как это показано на рисунке 5.

```

01. <script src="http://cdn.sockjs.org/sockjs-0.3.min.js"></script>
02. <script type="text/javascript">
03.     var sockjs_url = 'http://0.0.0.0:9998/desktop';
04.     var sockjs = new SockJS(sockjs_url);
05.     sockjs.onopen = function() {
06.         alert('socket has been opened', sockjs.protocol);
07.         sockjs.send('azaza');
08.     };
09.     sockjs.onmessage = function(event) {alert('message: ' + event.data)}
10.     sockjs.onclose = function() {alert('socket has been closed')};
11. </script>

```

Рис. 5. Клиентская часть установления соединения

Для установления соединения осталось добавить на сервер код для обработки соединений по префиксу «/desktop» и отправки пришедших сообщений обратно как на рисунке 6.

```
01. var sockjs = require('sockjs');
02.
03. var sockjs_desktop = sockjs.createServer();
04. sockjs_desktop.on('connection', function(conn) {
05.     conn.on('data', function(message) {
06.         conn.write(message);
07.         console.log(message);
08.     })
09.     conn.on('close', function (message) {
10.         console.log('connection closed');
11.     })
12. })
13. var server = http.createServer();
14.
15. ...
16.
17. sockjs_desktop.installHandlers(server, {prefix:'/desktop'});
18. server.listen(9998, '0.0.0.0');
```

Рис. 6. Обработка соединения на сервере

Теперь при установлении соединения страница выводит уведомляет об этом пользователя и отправляет на сервер сообщение «azaza». Сервер отправляет его обратно и записывает в консоль. При приходе сообщения с сервера страница выводит его в диалоговом окне. Проверим это, запустив сервер как на рисунке 7.

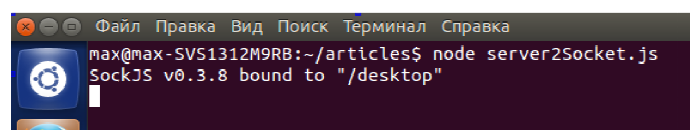


Рис. 7. Запуск сервера.

Тогда при подключении к 0.0.0.0:9998 мы увидим в консоли отправленное сообщение, которое потом увидим в диалоговом окне в браузере. Результат представлен на рисунках 8 и 9.

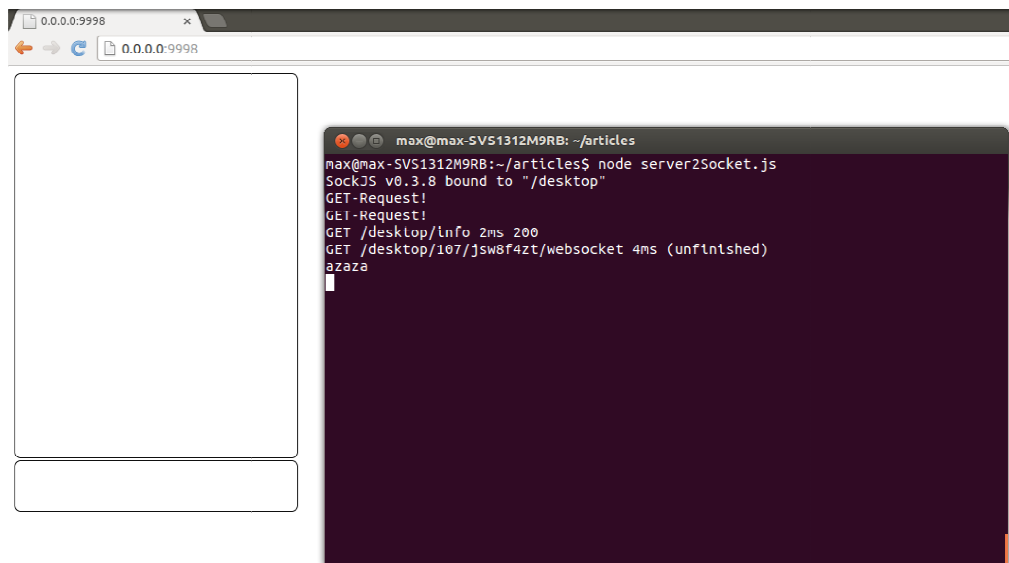


Рис. 8. Получение сообщения на сервере

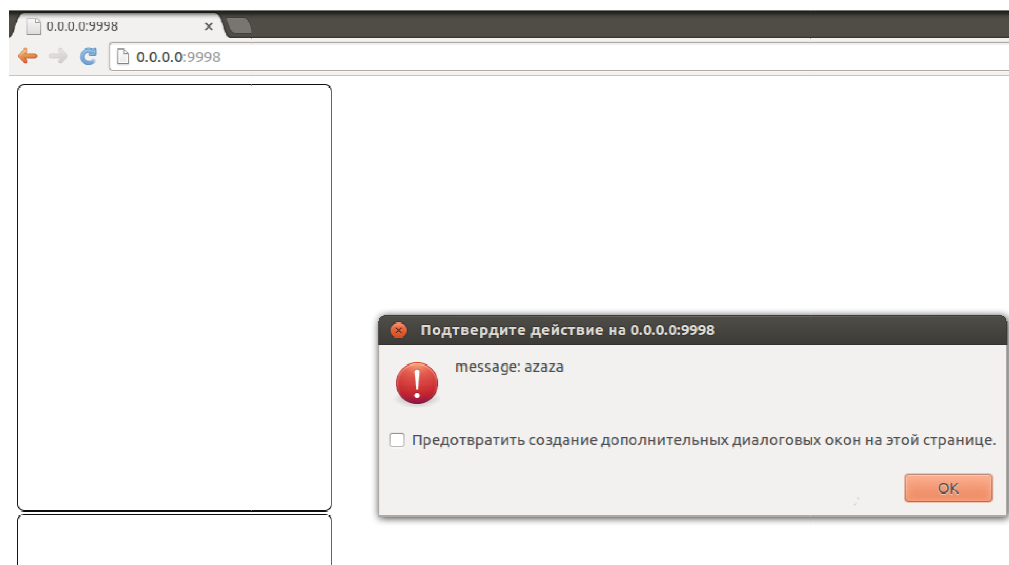


Рис. 9. Получение вернувшегося сообщения в браузере

Для мобильного устройства нам нужна другая страница. Скопируем и изменим страницу, отдаваемую компьютеру и будем ее отдавать мобильному устройству. Для определения мобильного устройства будем проверять, есть ли в user agent-е устройства название одной из популярных мобильных операционных систем. Единственное изменение,

которое надо внести в страницу — это изменение URL: мобильное устройство должно подключаться по адресу «/mobile». Для обработки такого подключения на сервере добавим обработчик на URL «/mobile» как показано на рисунке 10. Изменения в странице для мобильного устройства представлены на рисунке 11.

```
01. var http = require('http');
02. var fs = require('fs');
03. var sockjs = require('sockjs');
04.
05. var sockjs_desktop = sockjs.createServer();
06. sockjs_desktop.on('connection', function(conn) {
07.   conn.on('data', function(message) {
08.     conn.write(message);
09.     console.log(message);
10.   });
11.   conn.on('close', function (message) {
12.     console.log('connection closed');
13.   });
14. });
15. var sockjs_mobile = sockjs.createServer();
16. sockjs_mobile.on('connection', function(conn) {
17.   conn.on('data', function(message) {
18.     conn.write(message);
19.     console.log(message);
20.   });
21.   conn.on('close', function (message) {
22.     console.log('connection closed');
23.   });
24. });
25. var server = http.createServer();
26. server.addListener('request', function(req, res) {
27.   if (req["method"] === "GET"){
28.     if (req["headers"]["user-agent"].indexOf("iPhone") !== -1
29.       || req["headers"]["user-agent"].indexOf("Android") !== -1
30.       || req["headers"]["user-agent"].indexOf("OPR") !== -1
31.       || req["headers"]["user-agent"].indexOf("iPad") !== -1)
32.     {
33.       fs.readFile('mobile.html', function (err, data) {
34.         if (err){
35.           console.log("file", "mobile.html", "not opened");
36.           throw err;
37.         }
38.         res.write(data.toString());
39.         res.end();
40.       });
41.     } else {
42.       fs.readFile('page.html', function (err, data) {
43.         if (err){
44.           console.log("file", "page.html", "not opened");
45.           throw err;
46.         }
47.         res.write(data.toString());
48.         res.end();
49.       });
50.     }
51.   }
52. });
53.
54. sockjs_desktop.installHandlers(server, {prefix:'/desktop'});
55. sockjs_desktop.installHandlers(server, {prefix:'/mobile'});
56. server.listen(9998, '0.0.0.0');
```

Рис. 10. Обработка соединения и определение user agent-a

```
01. <script type="text/javascript">
02.   var sockjs_url = 'http://0.0.0.0:9998/mobile';
03.   var sockjs = new SockJS(sockjs_url);
04.   sockjs.onopen = function() {
05.     alert('socket has been opened', sockjs.protocol);
06.     sockjs.send('hello from mobile');
07.   };
08.   sockjs.onmessage = function(event) {alert('message: ' + event.data)}
09.   sockjs.onclose = function() {alert('socket has been closed')};
10. </script>
```

Рис. 11. Изменения в странице, отдаваемой на мобильные устройства

Теперь сервер идентифицирует тип клиента и отдает им разные страницы, с которых

происходит подключение по разным URL. На браузер, user agent которого отличается от мобильных по-прежнему отдается та же страница, только для наглядности она отправляет сообщение «hello from desktop». Результат подключения можно увидеть на рисунке 12. Для эмуляции мобильного устройства подставим user-agent Android в браузер так, как это представлено на рисунке 13 и получим «мобильную» страницу.

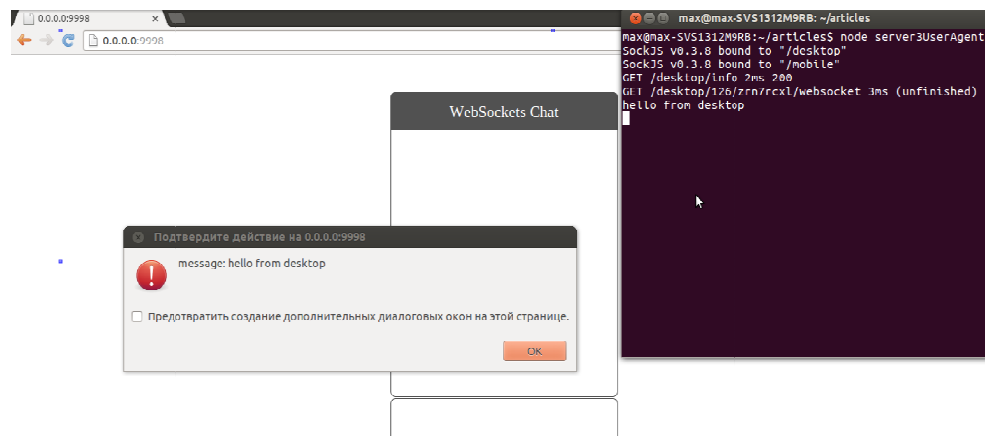


Рис. 12. Страница, получаемая обычным браузером

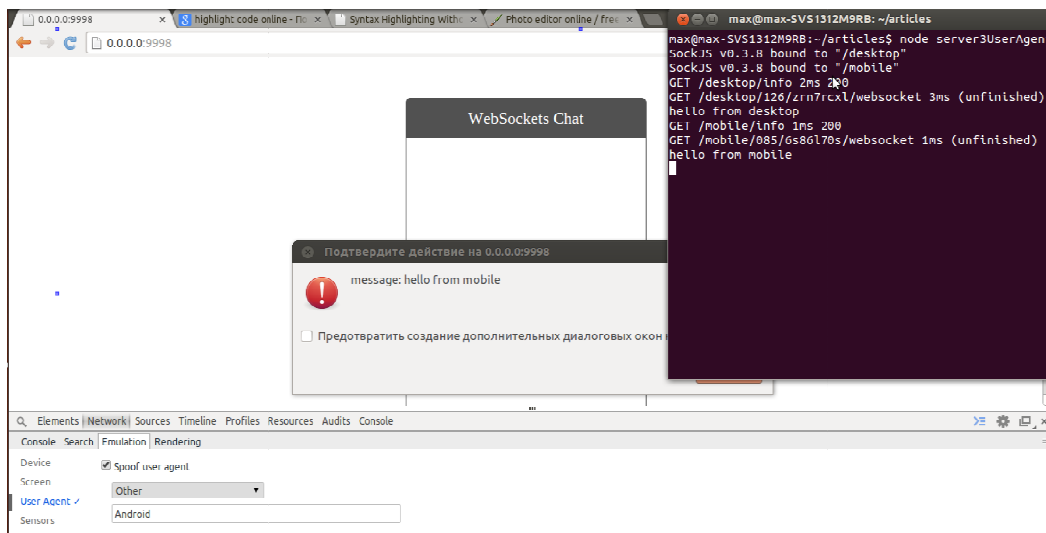


Рис. 13. Эмуляция мобильного устройства и получение «мобильной страницы»

Нам осталось только установить соединение между мобильным устройством и компьютером.

Реализуем функцию генерацию токена заданной длины, как на рисунке 14.

```
01. var randomlash = (function () {  
02.     var letters =  
03.         'AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz1234567890';  
04.     return function (len) {  
05.         var result = '';  
06.         for (var i=0; i < len; i++) {  
07.             result += letters[Math.floor(Math.random() * letters.length)];  
08.         }  
09.         return result;  
10.     }();  
11. }
```

Рис. 14. Функция генерации случайной строки заданной длины

Для связывания компьютера и мобильного устройства будем использовать следующую схему. При установлении websocket-соединения от компьютера отправим ему токен и в объект **tokenToConnection** занесем по ключу-токену объект соединения, код представлен на рисунке 15.

```
01. var tokenToConnection = {};  
02. var mobileToDesktop = {};  
03. var desktopToMobile = {};  
04. var sockjs_desktop = sockjs.createServer();  
05. sockjs_desktop.on('connection', function(conn) {  
06.     var token = randomHash(4);  
07.     tokenToConnection[token] = conn;  
08.     conn.write("your token: " + token);  
09.  
10.     conn.on('data', function(message) {  
11.         if (conn["session"]["session_id"] != undefined) {  
12.             var connId = conn["session"]["session_id"];  
13.             desktopToMobile[connId].write(message);  
14.         }  
15.     })  
16.     conn.on('close', function (message) {  
17.         console.log('connection closed');  
18.     })  
19. })
```

Рис. 15. Создание токена и занесение его в объект **tokenToConnection**

Обработаем полученный токен в обработчике как на рисунке 16.

```

01. var sockjs_mobile = sockjs.createServer();
02. sockjs_mobile.on('connection', function(conn) {
03.     conn.on('data', function(message) {
04.         if (conn["_session"]["session_id"] == undefined) {
05.             if (tokenToConnection[message] != undefined) {
06.                 var desktopConnection = tokenToConnection[message];
07.                 var desktopSessionId = randomHash(12);
08.                 desktopConnection["_session"]["session_id"] = desktopSessionId;
09.                 var mobileSessionId = randomHash(12);
10.                 conn["_session"]["session_id"] = mobileSessionId;
11.                 desktopToMobile[desktopSessionId] = conn;
12.                 mobileToDesktop[mobileSessionId] = desktopConnection;
13.                 conn.write('connection established');
14.                 desktopConnection.write('connection established')
15.             } else{
16.                 conn.write('bad token');
17.             }
18.         } else{
19.             var connectionId = conn["_session"]["session_id"];
20.             mobileToDesktop[connectionId].write(message);
21.         }
22.     })
23.     conn.on('close', function (message) {
24.         console.log('connection closed');
25.     })
26. })

```

Рис. 16. Обработка токена в обработчике мобильного подключения

Осталось выводить данные в удобном для пользователя формате. Для этого сообщения будем отображать в блоке с сообщениями, для дифференцирования сообщений, поступивших и отправленных добавим «имя собеседника». Для отправленных сообщений будем добавлять «**you:** », а для пришедших сообщений будем писать «**not you:** ». Для такой обработки сообщений изменим JavaScript-код на страницах так, как показано на рисунке 17.

```

01. var form = document.getElementById('form');
02. form.onsubmit = function(event){
03.     sockjs.send(document.getElementById('inp').value);
04.     print(document.getElementById('inp').value, true);
05.     document.getElementById('inp').value = '';
06.     event.preventDefault();
07. }
08. function print (argument, you) {
09.     if (you == true) {
10.         document.getElementById('messages').innerHTML += '<b>you: </b>' +
argument + '<br/>';
11.     }else{
12.         document.getElementById('messages').innerHTML += '<b>not you:
</b>' + argument + '<br/>';
13.     }
14. }
15. var sockjs_url = 'http://0.0.0.0:9998/desktop';
16. var sockjs = new SockJS(sockjs_url);
17. sockjs.onopen = function() {
18.     alert('socket has been opened', sockjs.protocol);
19. };
20. sockjs.onmessage = function(event) {
21.     print(event.data, false);
22. }
23. sockjs.onclose = function() {alert('socket has been closed');};

```

Рис. 17. Обработка отправленных и полученных сообщений на странице

Теперь проверим полученный результат. Подключение к серверу с обычного браузера и получение токена представлено на рисунке 18.

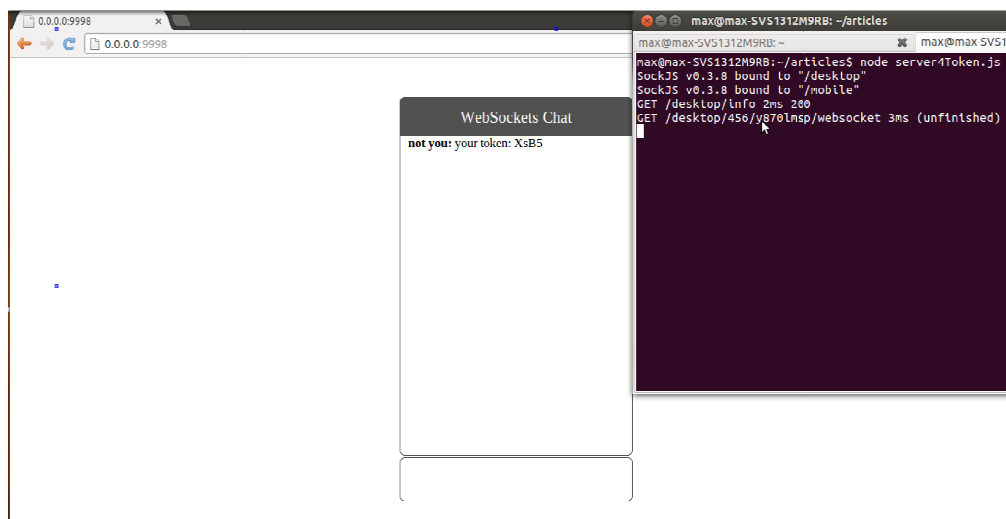


Рис. 18. Подключение к серверу из обычного браузера

Вид диалога с мобильного устройства и ввод токена представлен на рисунке 19.

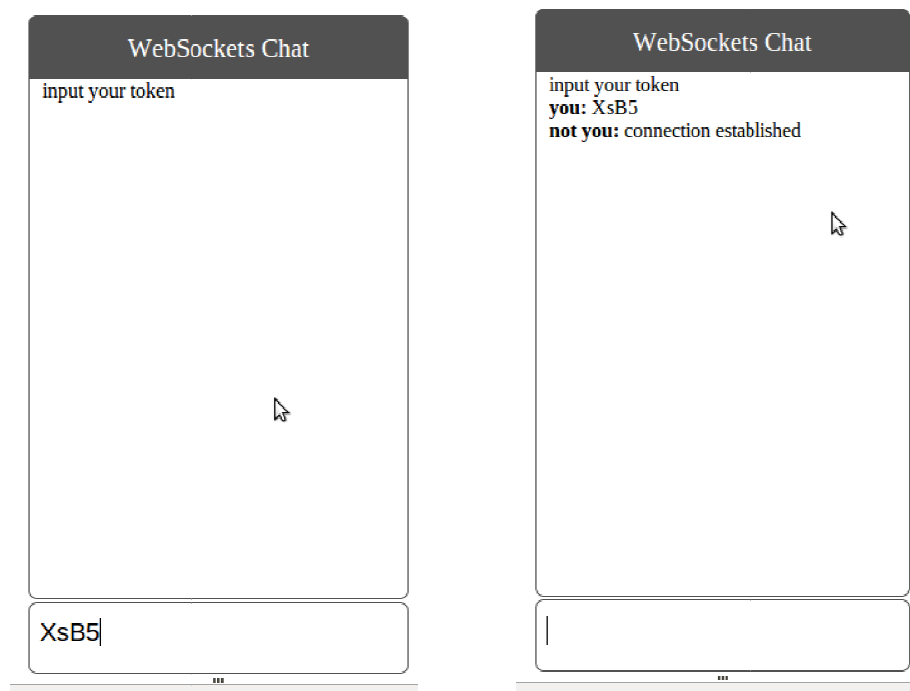


Рис. 19. Подключение с мобильного устройства и ввод токена

Финальный результат — диалог виден на рисунке 20.

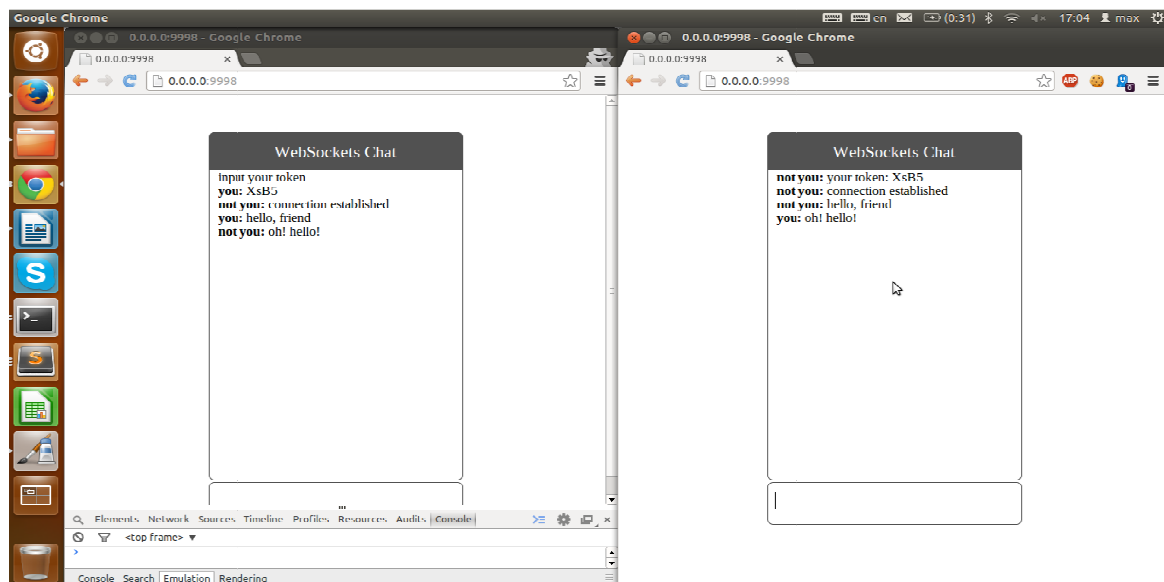


Рис. 20. Финальный результат

Выводы:

1. Исследование показало, что использование протокола WebSocket оптимальнее AJAX для передачи сообщений. Но только в том случае, когда соединений ощутимо меньше, чем количество передаваемых сообщений через одно соединение.
2. Был найден удобный для пользователя метод связывания мобильного устройства и компьютера. Этот метод подразумевает показ короткой строки на компьютере, которую надо будет ввести на мобильном устройстве.

Список литературы

1. AJAX. Available at: <http://ru.wikipedia.org/wiki/AJAX>, accessed 22.02.2014.
2. WebSocket. Available at: <http://ru.wikipedia.org/wiki/WebSocket>, accessed 23.02.2014.
3. About WebSocket: <http://www.websocket.org/aboutwebsocket.html>, accessed 23.02.2014.