Research and Implementation of Campus Information Push System Based on WebSocket

Yangyang Hu
School of Computer Science
Nanjing University of Posts and Telecommunications
Nanjing, China
15062203608@163.com

Weiqing Cheng*
1. Key Laboratory of Computer Network and Information Integration, Southeast University
2. School of Computer Science, Nanjing University of Posts and Telecommunications
Nanjing, China
chengweiq@njupt.edu.cn

***Abstract*—WebSocket is a TCP-based protocol, providing full-duplex communication channels over a single TCP connection. In recent years, more and more developers choose to use it to develop applications. This paper discusses the traditional real-time Web communication schemes, and analyzes the advantages of WebSocket. Based on the deep study of WebSocket, a real-time campus information push solution based on WebSocket and node.js is proposed. This paper expatiates the overall structure of the scheme, designs and realizes the functions of each module, and the functional testing results show that the scheme is feasible.**

Keywords-WebSocket; Campus information push; node.js; real-time Web communication

I. INTRODUCTION

Historically, creating web applications that need bidirectional communication between a client and a server (e.g., instant messaging and gaming applications) has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls [RFC6202].

At present, there are a considerable number of real time Web communication schemes in the market, which mostly have some disadvantages. First, the server is forced to use a number of different underlying TCP connections for each client: one for sending information to the client and a new one for each incoming message. Second, the wire protocol has a high overhead, with each client-to-server message having an HTTP header. Third, the client-side script is forced to maintain a mapping from the outgoing connections to the incoming connection to track replies.

A simpler solution would be to use a single TCP connection for traffic in both directions. This is what the WebSocket Protocol provides[1]. Combined with the WebSocket API [WSAPI], it provides an alternative to traditional communication from a web page to a remote server. In this paper, a real-time solution to campus information push system based on WebSocket is proposed to effectively avoid the shortcomings of traditional real-time Web communication schemes[2].

II. WEBSOCKET TECHNOLOGY

A. Traditional real - time Web communication schemes

Before WebSocket technology appeared, in order to achieve real-time Web application, the following methods were adopted: traditional polling, Ajax polling, long polling and HTTP streaming. Although these methods are widely used in a variety of applications, they are not perfect. There are some problems, such as resource consumption, complicated implementation process, and bad real time. Here are some specific analyses.

Traditional polling: In this case, a client sends a request to a server at a fixed time interval to keep information synchronized between the client and the server. There is a problem. Most of the time, we only need to update a part of the page, rather than the entire page, so that some duplicate data are repeatedly transferred between the client and the server, which results in waste of network resources.

Ajax polling: Ajax allows for Web pages or Web applications, to change content dynamically without the need to reload the entire page. Compared with traditional polling, it reduces the burden of the network and optimizes the user experience. Because of the asynchronous nature of Ajax, that each chunk of data is sent or received by the client occurs in a connection established specifically for that event. Thus, for every action, the client must poll the server, instead of listening, which incurs significant overhead. This overhead leads to higher latency with Ajax than what can be achieved with websockets.

Long polling: The client sends the request to the server, the server does not return immediately if no data is updated. If there is no updated data, the connection will remain for a period of time until the data is updated or the time expires. After receiving the server response, the client often immediately issues another server request. Compared with Ajax polling, it has no performance improvement in nature. And the server needs to do some configuration, which can damage the overall performance of the server at high concurrency.

HTTP streaming (HTTP server push): There is no need to send requests frequently. The entire process needs only one HTTP connection request. The web server keeps the connection open and periodically sends data to the client. It can run in most browsers except IE. The compatibility is not very good.

B. Websocket advantage

Today, the demand for real-time information is starting to get higher. There are instant messaging systems, train ticketing systems, stock trading systems and so on. It requires the client can receive the latest data once the server data is updated.

HTML5 proposes a WebSocket protocol, which enables full duplex communication between server and client side [3]. Compared with the traditional real-time

978-1-5386-1829-5/17/$31.00©2017 IEEE

web communication scheme, the WebSocket protocol saves a lot of network bandwidth resources and server resources, and the real-time performance is greatly improved. The WebSocket protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. The protocol contains two parts: handshake and data transfer.

The opening handshake is intended to be compatible with HTTP-based server-side software and intermediaries, so that a single port can be used by both HTTP clients talking to that server and WebSocket clients talking to that server. To this end, the WebSocket client's handshake is an HTTP Upgrade request [4]. The steps for establishing a WebSocket connection are as follows.

1) The WebSocket client sends an HTTP Upgrade request

The WebSocket client initiates a handshake to the server. Two WebSocket URIs are defined:

ws-URI = "ws:" "//" host [":" port] path ["?" query]
wss-URI = "wss:" "//" host [":" port] path ["?" query]

The port component is optional; the default for "ws" is port 80, while the default for "wss" is port 443. Here's an example of a client-side handshake message [4]:

GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13

The relevant fields are explained as follows:

Upgrade: Upgrade is the header field in HTTP1.1 that defines the conversion protocol. It says that if the server supports it, the client wants to switch to the WebSocket protocol.

Connection: HTTP1.1 specifies that Upgrade can only be used in "direct connection", so the HTTP1.1 message with the Upgrade header must contain the Connection header. Upgrade indicates that the request is a protocol upgrade request.

Sec-WebSocket-Key: A 24-bit random character sequence generated by the client for the handshake authentication process;

Sec-WebSocket-Version: 13 indicates the Version of the WebSocket protocol supported by the client;

2) The server reads the client's opening handshake and sends the server's opening handshake

After the server receives the request, it determines that the request is a WebSocket request based on its header information. Then take out the sec-websocket-Key field information, generate a new character sequence according to an algorithm, and write it in the Sec-WebSocket-Accept header field. A standard WebSocket server's response is as belows.

HTTP/1.1 101 Switching Protocols
Upgrade: websocket.
Connection: Upgrade.
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat.

The value of the Sec-WebSocket-Accept header field is constructed by concatenating the Sec-WebSocket-Key, with the string "258EAFA5-E914-47DA-95CA-C5AB0DC85B11", taking the SHA-1 hash of this concatenated value to obtain a 20-byte value and base64-encoding this 20-byte hash [4].

The other header information indicates that the server has received a request to upgrade the client from the HTTP protocol to the WebSocket protocol.

3) The client validates the server's response

After receiving the server's response, the client must validate the response and verify that the server has received the protocol upgrade request and returned the handshake reply message, and then computes the Sec-WebSocket-Accept on the client according to the same algorithm as the server used to generate the Sec-WebSocket-Accept. Then the result is compared with the content of Sec-WebSocket-Accept returned by the server. If the match information is valid, the connection is established. Otherwise, the connection establishment fails.

To establish a WebSocket connection[5], the client needs to send an HTTP request to the server. The protocol is upgraded to the WebSocket protocol. The server recognizes the request type according to the HTTP header. It will be upgraded to a WebSocket connection if requested by the WebSocket, and both parties will start full-duplex communication, which means the client and the server can exchange the data at any time until the client or server side closes the connection. Fig. 1 shows the WebSocket communication model.

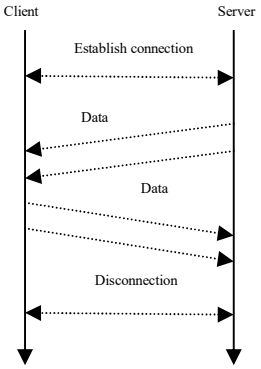

Fig. 1. WebSocket communication model

WebSocket has more communication capability than traditional communication schemes, which really realizes real-time data communication[6]. Once the WebSocket connection is established, the server and the client perform a two-way flow of data, which enhances the server's push capability. Compared with the HTTP protocol, its head information is more concise and reduces the transmission of redundant data. With high user scale and high real-time requirement, it has more advantages to reduce network load than traditional real-time communication.

III. RESEARCH AND IMPLEMENTATION OF WEBSOCKET

In short, Node.js is a JavaScript runtime environment built on Chrome's V8 JavaScript engine that compiles JavaScript source code to native machine code instead of interpreting it in real time. Node.js uses an event-driven,

non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world[7].

A. WebSocket event

The WebSocket API is purely event-driven, which retrieves the connection state and updates the state by listening event while processing the data. The client is no longer polling the server as traditionally. Messages and events arrive asynchronously after the server sends data. WebSocket objects can listen to events through callback functions, and the overall follows the asynchronous programming model. WebSocket objects have four types of events:

Open: After the server responds to the WebSocket connection request, the open event is triggered. The response callback function is named onopen. Once triggered, WebSocket clients and servers can start sending and receiving data.

Message: After receiving the message, the message event is triggered and the response callback function onmessage is called. The WebSocket messaging mechanism can not only handle text messages, but also handle binary data, which is divided into Blob and ArrayBuffer two types, and the data type needs to be determined in advance.

Error: Once failed, the error event is triggered, the corresponding function onerror is called, and the connection is closed.

Close: Closing the connection triggers the close event, which corresponds to the onclose. Once the WebSocket connection is closed, the server and client will not send any further data and or will discard any further data received.

B. WebSocket method

The WebSocket object has two methods: Send () and Close ():

Send (): The server and the client can use send () to send messages after establishing a two-way connection. Use the send () to send data while the connection is open, and throw the exception when the connection is closed.

Close (): The connection is closed by the close(). It means data can no longer be sent after calling the close().

C. WebSocket property

The WebSocket object has three properties: readyState, bufferedAmount, and protocol.

ReadyState: ReadyState is connection state. WebSocket object needs readyState to get the current connection status. Table I shows the different connection states represented by readyState .

BufferedAmount: The browser caches the application data, and the bufferedAmount gets the size of the data that has not been transferred to the queue.

Protocol: The server uses the protocol parameter to know that the protocol used by the client is the WebSocket protocol.

When the client connects to the server, the core code is as follows:

TABLE I. CONNECTION STATES

Attributes	Value	Status
WebSocket.CONNECTING	0	The connection is in progress, but has not been established..
WebSocket.OPEN	1	The connection has been established and messages can be sent.
WebSocket.CLOSING	2	The connection is closing the handshake.
WebSocket.CLOSED	3	The connection has been closed or can not be opened.

```
var ws = new WebSocket("ws:// server address:8080");
(1)
ws.onopen =
function(){                                          (2)
        alert("opened");
        ws.send("I'm client");
};
ws.onmessage =
function(evt){                                       (3)
        alert(evt.data);
};
ws.onclose =
function(){                                          (4)
        alert("closed");
};
ws.onerror =
function(err){                                       (5)
        alert("error:"+err);
}
```

(1) creates a WebSocket object, and the parameter is the address of server. The WebSocket protocol uses the ws:// at the beginning, like HTTP protocol using http:// at the beginning. (2)~(5) are the processing function of messages registered by the WebSocket object. The WebSocket object supports four event monitoring functions, which are onopen, onmessage, onclose, and onerror. When the connection of Browser and WebSocket Server is successful, the onopen function is triggered; If the connection fails, or sending or receiving data fails, browser triggers the onerror function; When the browser receives the data sent by WebSocket server, it will trigger the onmessage function, the parameter "evt" contains data transferred by the server; When the browser receives the shutdown connection request sent by the WebSocket server side, the onclose function is triggered.

IV. CAMPUS MESSAGE PUSH SYSTEM BASED ON WEBSOCKET

This paper designs a campus message push system based on Websocket. The main scenarios are as belows: The campus secretary pushes relevant campus notices to each teacher's client, and the teacher client displays the latest campus notifications in real time and can also display historical records. Historical records can be searched according to keywords, which can support multiple keyword searches. The campus notification can be stored in files.

The system uses node.js to build the server, the campus secretary client and the teacher client use JavaScript, HTML and CSS technology, and keyword search uses the Jquery class library[8]. The whole development uses WebStorm as development tool, which is a popular development software that supports node.js development.

A. Overall architecture design

The overall architecture is shown in Fig. 2, which has two parts.

1) First, the campus secretary enters the system to input a message. Then, the message is sent to the system server and is stored in the server's file. Finally, the server pushes the message in the file to all teacher clients, as shown in Fig. 3.

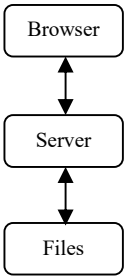

Fig. 2. The overall architecture

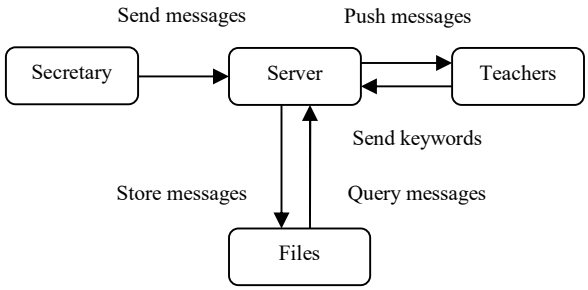

Fig. 3. Messgae flow graph

2) First, the teacher enters the system to input keywords for query. Then, the keywords arrive at the system server, the server queries the files. Finally, the server returns the corresponding message to the teacher client.

B. Module design and implementation

The system mainly consists of user management module, messages sending module, messages storing module, messages pushing module and messages querying module. The main functional modules are shown in Fig 4:

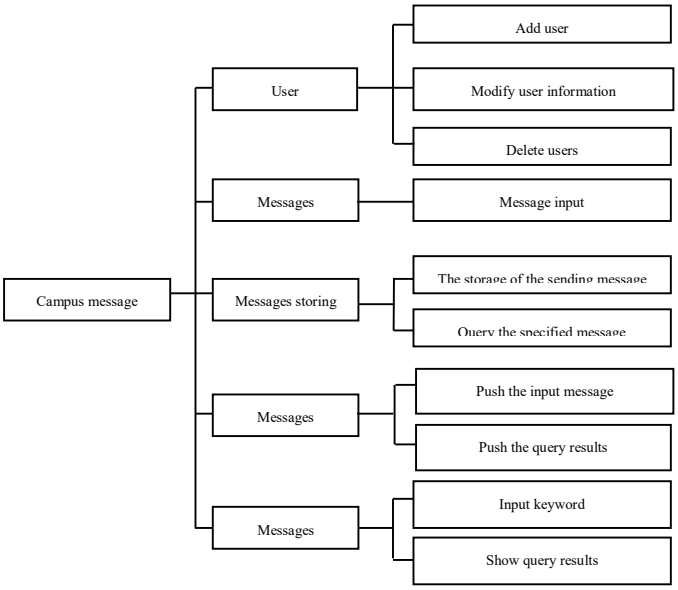

Fig 4. Functional modules

1) Message sending module

The message sending module is shown in Fig. 5, which is responsible for sending the message entered by the secretary to the server. When ws.readyState===Websocket.OPEN, campus secretary client sends messages. The page for the secretary to send messages is as shown in Fig. 6, and the page for the teacher to receive messages is as shown in Fig. 7.

2) User management module

After landing system, the administrator can set the various users of the system: add a user into the system, modify the login user's username and password, and delete a user.

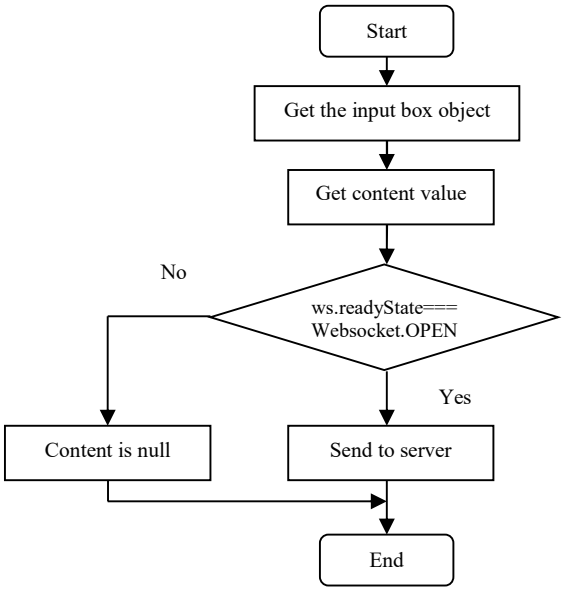

Fig. 4. Message sending module

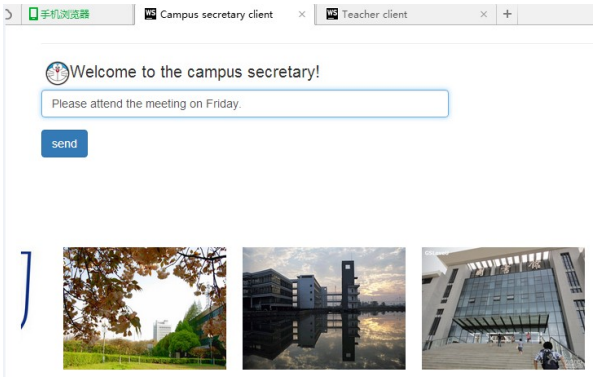

Fig. 5. The page for the secretary to send messages

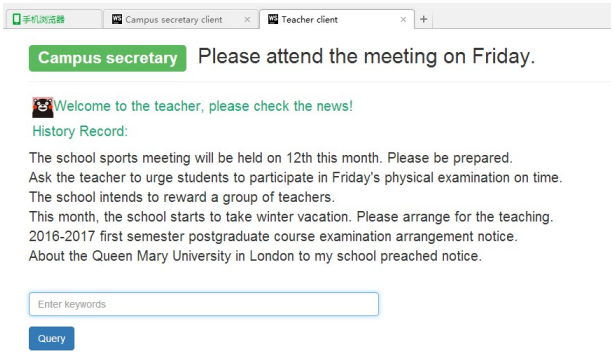

Fig. 6. The page for the teacher to receive messages

3) Messages storing module

The message storing module is responsible for storing messages which the client sent to the server. Node.js has "fs" module that handles the read-write of files. The server gets the "fs" module, and uses fs.appenFile () to save the message to files. The fs.readFile () is responsible for reading messages from files. Of course, if you ne to store a lot of information and use it for a long time, it's convenient and safe to use database storage.

4) Messages pushing module

The module means the server pushes messages to clients. First, get the node.js "ws" module. "ws" module is an easy to use, ultra-fast Websocket implementation, which can be used to quickly build Websocket server. The Websocketserver is built as follows:

```
Var WebSocketServer = require('ws').Server,
wss = new WebSocketServer({port:8080});
wss.on('connection',function(ws){
console.log('client connected');
ws.on('message',function (message){
    console.log(message);
    });
});
```

And then get node.js "node-uuid" module, which assigns a unique identifier to each client. Combined with the "ws" module and "node-uuid" module, it traverses all the clients, and broadcast all clients. It is necessary to determine whether clientSocket.readyState === WebSocket.OPEN. Only in the open state, it can push the message. In addition, it is also necessary to determine if it is a "secretary", because messages are only pushed to "teachers". The push message module is shown in Fig. 8.

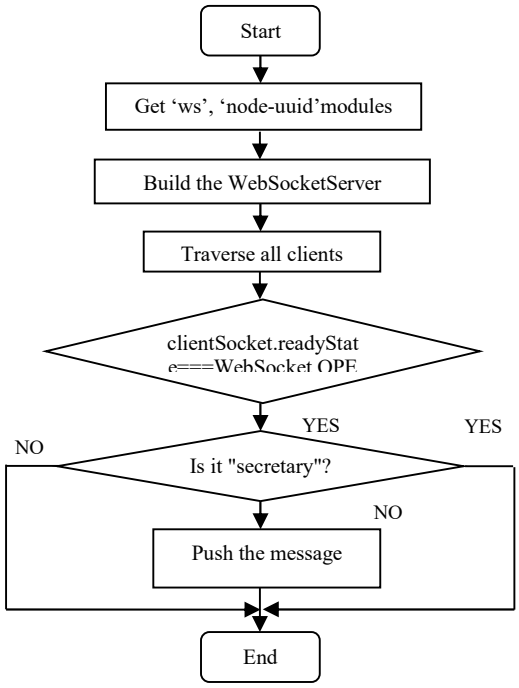

Fig. 7. Message pushing module

5) Messages querying module

The message querying module supports teachers to query required messages. The module uses regular expressions to filter messages, and then returns the matching messages. First, special characters are escaped. Then use RegExpObject.Test () to determine if there is a match, and if so, return the matched messages. The query results are shown in Fig. 9. The core code of the regular expression is as follows:

```
var tags = /[^<>]+|<(\/?)([A-Za-z]+)([^<>]*)>/g;
var a = v_html.match(tags),test =0;
$.each(a,function(i,c){
      if(!/<(?:.|\s)*?>/.test(c)){
        $.each(arr,function(index,con){
          if(con ==””){return;}
          var reg = new RegExp($.regTrim(con),”g”);
              if(reg.test(c)){
                c = c.replace(reg,” ♂”+con+”♀”);
                test = 1;
              }
        })
        …
      }
})
```

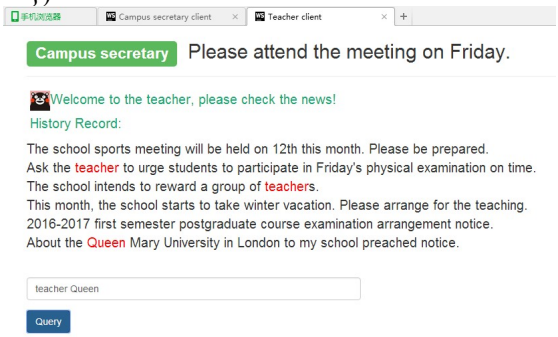

Fig.9. The page for message querying

V. CONCLUSION

This paper analyzes the advantages of WebSocket technology, and details the core principles of WebSocket technology. On the basis of deep research on WebSocket technology, this paper proposes a technical scheme of campus information push system based on WebSocket technology, to pursue a more convenient and efficient real-time Web application implementation scheme. Next we will enrich messages storing module and explore the scheme of real-time picture pushing.

ACKNOWLEDGMENT

This work was supported by the Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, under Grants No. K93-9-2014-04B, and in part by the National Natural Science Foundation of China under Grants No. 61170322.

REFERENCES

[1] Hanson L. Inside Websockets[C]// Applicative. ACM, 2016.

[2] Zhang L, Shen X. Research and development of real-time monitoring system based on WebSocket technology[C]// International Conference on Mechatronic Sciences, Electric Engineering and Computer. IEEE, 2013:1955-1958.

[3] Skvorc D, Horvat M, Srbljic S. Performance evaluation of Websocket protocol for implementation of full-duplex web streams[J]. Proceedings of International Convention on Information & Communication Technology Electronics & Microelectronics Mipro Opatija, 2014:1003-1008.

[4] [RFC6455] , I.Fette, A.Melnikov,"The Websocket Protocol" , December 2011.

[5] Anusas-Amornkul T, Silawong C. The study of compression algorithms for WebSocket protocol[C]// International Conference on Electrical Engineering/electronics, Computer, Telecommunications and Information Technology. IEEE, 2014:1-6.

[6] Pimentel V, Nickerson B G. Communicating and Displaying Real-Time Data with WebSocket[J]. IEEE Internet Computing, 2012, 16(4):45-53.

[7] https://nodejs.org/en/

[8] Li J, Peng C. ¡Query-based Ajax general interactive architecture[C]// IEEE, International Conference on Software Engineering and Service Science. IEEE, 2012:304-306.