Computing, Performance and Communication systems (2016) 1: 28-32
Clausius Scientific Press, Canada

Design and Implementation of a High Performance Event-Driven WebSocket Server

Wenbin Cao[1,a], Xinming Tan[1,b], Bei Liu[1,c], Chuanwen Liu[1,d]

[1]Computer Science Technology, Wuhan University of Technology,Wuhan, Hubei 430063, China
[a]397108590@qq.com, [b]tanxm@whut.edu.cnl, [c]liubei1203@163.com, [d]chwliu@whut.edu.cn

Keywords: WebSocket protocol, Node.js, High concurrent, Event-driven, Stability.

Abstract: In recent years, instant messaging has been more and more widely used on the Internet. The Pull model adopted in the traditional servers can not well meet the requirements of real-time information acquisition and high concurrent using accesses in the practical applications. In order to deal with the aforementioned problem, using the Push model in the real-time message transmission has become a research hotspot. Firstly, based on the open source projects of Node.js, Redis and RabbitMQ, a WebSocket server which can provide real-time message push service to a large number of different users' subscription requests is designed and implemented. Secondly, the function modules and implementation details of each layer are analyzed in detail. At last, experimental results show that the feasibility of WebSocket server.

1. Introduction

With the high-speed development of Internet Web 2.0 technology, the Web application in the browser has become one of the main ways for users to access Internet information. Due to the depth application of the Web technology in different areas and the constant appearance of the massive data under the high concurrent connection requests in information age, a higher demand is needed in Web applications for real-time data transmission [1], mainly reflected in two aspects, one is the real-time data transfer, the other is the massive data brought by high concurrency. The traditional instant massage mostly adopt the "pull" technology. However, instead of improving the real-time data transmission, the "pull" technology restrict the performance of real-time Web applications. Compared with the traditional methods, the "push" , based on the WebSocket protocol, is the way that the server will automatically send the message to the browser user. This communication instead of a single TCP socket, through the first request to establish a full-duplex Communication and real-time response Socket connection, which can reduce the number of browser-server interaction, thereby reducing the burden of network communication, and making the communication efficiency.

2. Related Work

WebSocket technology is used for the messages real-time transmission In Web realtime communication, social subscription, real-time monitoring and other scenes. [2] WebSocket connection, replacing the Http connection,is used to measure the one-way transmission delay of the real-time wind sensor data. [3] proposes remote monitoring and control method for printing and packaging machinery based on WebSocket, effectively improving the production process digitization in the printing and packaging industry. [4] WebSocket is used in the cloud monitoring system,and the experimental results show that the average delay time of WebSocket monitoring system is usually lower than the polling, FlashSocket and Socket solutions. [5] adopts WebSocket push technology and SVG technology to improve the real-time stability of the monitoring system.

In general: 1) The above-mentioned literatures only refer to the advantage of WebSocket technology in the real-time data pushing, for may bring , but do not explain how to deal with the massive real-time data broght by the high concurrent requests. 2) The above-mentioned literatures only introduce the WebSocket technology on the server side to realize the function of the data real-time pushing under a certain scene, but do not make WebSocket technology as the core to design and implement a WebSocket server.

3. Design of WebSocket Server

The entire server uses a three-tier architecture, namely the user service layer, the middle layer and the data services layer. User service layer is responsible for receiving user access to real-time data connection request, and the user information to the middle layer. After receiving the user information from the user service layer and processing the real-time data from the data services layer, the middle layer pushes the real-time data through the user service layer to the user. The data service layer will call API provided by the data center to achieve the real-time data. The specific layered architecture is shown in Figure 1.

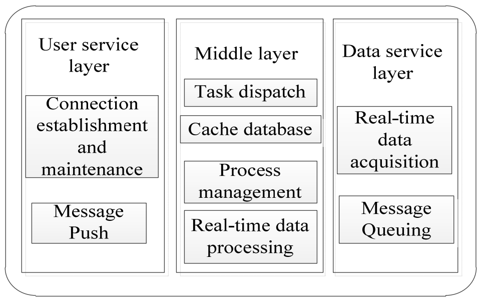


Fig. 1 Layered architecture of Websocket server.

4. The Implementation of WebSocket Server

4.1 The User Requests a Connection

WebSocket server user request processing includes that the user requests the connection and the user sends the message body formulate. The process that client requests WebSocket server to establish the connection, is the handshake protocol in the opening phase, and you can choose Socket.IO to achieve this.

4.2 Real-time Data Acquisition and Data Processing

Real-time data acquisition refers to the that WebSocket server obtains real-time updated data from the data center. The data center provides the interface getRealTimeData (int appID, List ids), where appID is the application number and ids is the set of data id that the user requests to subscribe. WebSocket server call the interface in every constant time interval, to ensure the real-time data.

The data processing mainly refers to the conversion of the data type. The WebSocket protocol can only deal with text data messages and binary data messages [6]. After WebSocket obtains the real-time updated data from the data center, it needs to process different types of real-time data and convert them into binary. Node.js provides a global constructor，Buffer, which can manipulate binary data to achieve the conversion of string, int, Json and other formats.

4.3 Message Queue

RabbitMQ is an open source implementation of the AMQP (Advanced Message Queuing) protocol. As a veteran message middleware, RabbitMQ has not only the perfect functions, but also many optional plug-ins, providing a rich API [7].

The amqp module in Node.js provides clients in the Node.js environment for all message queuing servers that follow the AMQP protocol. Specific implementation steps including : 1) installing RabbitMQ message server to do the environment variables and permissions settings. 2) opening http: // localhost: 15672 to check if you have access to the RabbitMQ management console. 3) Install the node-amqp module that Node.js needs to interact with RabbitMQ. 4) installing connection through the amqp.create Connection function to achieve the queue generation and queue initialization.

4.4 Cache Database

When receiving the message body from the client, the WebSocket server can obtain the connection information between the client and it. The WebSocket server can get {"appID" + "msg"} information through analyzing the message body. The client may send multiple message bodies, WebSocket server will split the message body. With the {"appid" + "msg"} as the key and the connections subscribing the key as the value, these are stored in Redis. The analytical process of the message body and the new data generated in Redis are as shown in Figure 2.

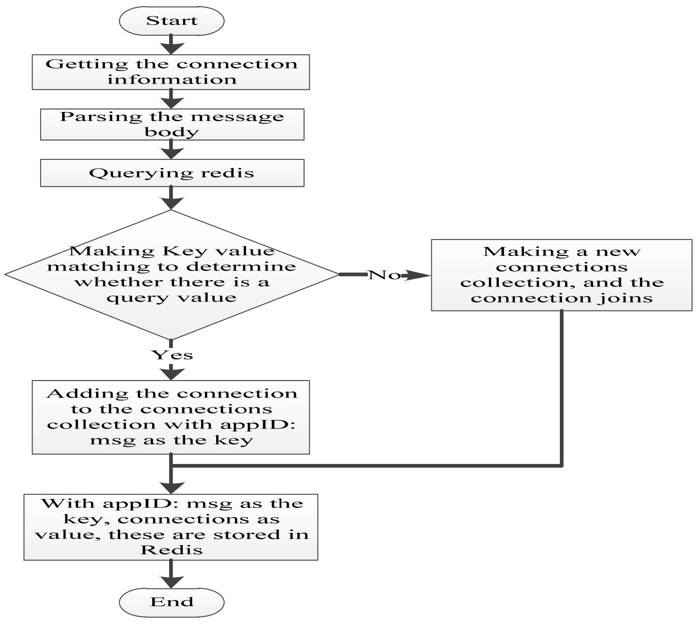

Start
Getting the connection information
Parsing the message body
Querying redis
Making Key value matching to determine whether there is a query value
No
Making a new connections collection, and the connection joins
Yes
Adding the connection to the connections collection with appID: msg as the key
With appID: msg as the key, connections as value, these are stored in Redis
End

Fig.2 The storage flow chart in Redis

4.5 Message Push

WebSocket server provides two message push interface, respectively, they support the text type and binary type of data push, the specific description is as shown in Table 1.

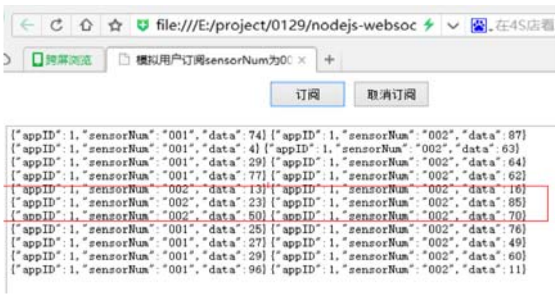
Table 1 The data push interface of WebSocket Server

Name of the interface	Description
sendTest(data,callback)	This interface is called when the data type is the text type
sendBinary(data,callback)	This interface is called when the data type is the binary type

5. Experiment and Results

5.1 Function Testing

A WebSocket connection request is sent to the WebSocket server as soon as the page is opened. After receiving the user request and successfully building the connection, the WebSocket server will timely push the data information to the client browser according to the user subscription message. As shown in Figure 3 that the subscribe button on the page emulates the user sending a WebSocket subscription request, and the cancel button on the page emulates the user sending a WebSocket unsubscribe request.


file:///E:/project/0129/nodejs-websoc
在4S店看
跨屏浏览
模拟用户订阅sensorNum为0C
订阅
取消订阅

```
{"appID":1,"sensorNum":"001","data":74}{"appID":1,"sensorNum":"002","data":87}
{"appID":1,"sensorNum":"001","data":4}{"appID":1,"sensorNum":"002","data":63}
{"appID":1,"sensorNum":"001","data":29}{"appID":1,"sensorNum":"002","data":64}
{"appID":1,"sensorNum":"001","data":77}{"appID":1,"sensorNum":"002","data":62}
{"appID":1,"sensorNum":"002","data":13}{"appID":1,"sensorNum":"002","data":16}
{"appID":1,"sensorNum":"002","data":23}{"appID":1,"sensorNum":"002","data":85}
{"appID":1,"sensorNum":"002","data":50}{"appID":1,"sensorNum":"002","data":70}
{"appID":1,"sensorNum":"001","data":25}{"appID":1,"sensorNum":"002","data":76}
{"appID":1,"sensorNum":"001","data":27}{"appID":1,"sensorNum":"002","data":49}
{"appID":1,"sensorNum":"001","data":29}{"appID":1,"sensorNum":"002","data":60}
{"appID":1,"sensorNum":"001","data":96}{"appID":1,"sensorNum":"002","data":11}
```

Fig.3 Unsubscribe and resubscribe

The content in the red frame in Figure 3 indicates that the browser client will no longer receive data with a sensorNum of 002 after canceling the subscription, and data with sensorNum of 001 will continue to be received. The experiment also shows that WebSocket server can meet the basic functional requirements.

5.2 Performance Testing

Table 2 shows that when the number of concurrent connections is less than 750, the error rate is 0, and when the number of connections reaches 1500, the error rate of 1.83% or so, which means that the Websocket server can meet the real-time data push requirements under the multi-application multi-tenant mode.

Table 2 System performance testing

The number of concurrent connection	Average response time/ms	Error rate/%
500	1047	0
750	1852	0
900	2173	0.25
1200	2763	0.41
1500	3242	1.83

6. Conclusions

This paper reviews the technical solutions in the development of real-time communication, summarizes the characteristics of WebSocket. After analyzing the application of WebSocket technology in server side, there is no designed a WebSocket server to solve the problem of high concurrency and stability. Finally, the WebSocket server is tested, and the results show that the WebSocket server achieves the intended design goals in terms of functionality and performance. In the next stage of the study, the log management module will be added to the WebSocket server, which can playback the running state of the server.

Acknowledgements

This work is partially supported by Hubei Natural Science Foundation of China(2014CFA050).

References

[1] Lu Chen, Feng Xiang-yang, Su Hou-qin. Study and Implementation of Html5 WebSocket Handshake Protocol [J]. Computer Applications and Software,2015, 32(1):128-131.

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

[3] Cai Jin-da, Jiang Zhen-fei. Remote Monitoring and Control Method for Printing and Packaging Machinery Based on WebSocket [J]. Packaging Engineering,2013(15):87-90.

[4] Ma K, Zhang W. Introducing browser-based high-frequency cloud monitoring system using WebSocket Proxy[J]. International Journal of Grid & Utility Computing, 2015, 6(1).

[5] Jin Feng, Zhang Yue, Yong Peng. Design and Implementation of Monitoring System in B/S Mode Base on Two Servers [J]. Computer Simulation,2014, 31(2):201-205.

[6] Melnikov A. The WebSocket Protocol[J]. 2011.

[7] Videla A, Williams J J W. RabbitMQ in action : distributed messaging for everyone[J]. 2012.