文章编号：2096-1472(2016)-10-30-03

使用Websocket和Servlet实现服务器定点推送

钱宇虹
（武汉职业技术学院计算机技术与软件工程学院，湖北 武汉 430074）

摘　要：针对Java Web实时系统中的消息推送这一关键问题，本文基于Tomcat8和WebSocket标准，提出并且实现了一个服务端定点推送模型，能够向指定的Web注册用户进行消息推送。该模型具有非广播的特点，即在没有浏览器请求的情况下通过Servlet主动将消息推送至某个已注册的Web用户。该模型已经成功地应用于呼叫中心系统中将电话分机状态和弹屏信息推送给Web坐席，该模型也可以用于即时消息、游戏应用、实时证券报价、股票系统等Web实时系统，具有广泛的使用价值。

关键词：Tomcat8；WebSocket；定点推送
中图分类号：TP311　　**文献标识码**：A

The Implementation of Server Accurate Push by Using Websocket and Servlet

QIAN Yuhong
(*Wuhan Polytechnic,School of Computer Technology and Software Engineering,Wuhan* 430074,*China*)

Abstract: Based on Tomcat 8 and Web Socket Standards,the paper proposes and implements a model of server accurate push,which can push messages to specified web registered users.The push in this model is non-broadcast,actively sending message to a registered web user through Servlet without browser requests.This model has been successfully applied in the Call Center system,pushing extension status and popup screen information to web Call Center agents.The model can also be applied in instant messaging,gaming,real-time stock quotation,and other real-time web systems,with extensive practical value.

Keywords: Tomcat8;WebSocket;server accurate push

1 引言(Introduction)

众所周知，HTTP协议是基于请求-响应模式的无状态的单向协议，即每次都要由客户端(如浏览器)主动向服务端发出“请求”，服务端做出处理后将结果作为“响应”返回给客户端。“无状态”指的是每次的请求-响应都必须经历建立连接和释放连接的过程，前一次连接和后一次连接相互之间没有关系。“单向”指的是客户端是主动方，服务端是被动方，服务端不能主动向客户端发送数据。HTTP协议的这一特点对传统的Web应用，像电子商务网站、搜索引擎等等非常合适，但是不能满足日益增强的实时性需求的Web应用，这些应用要求在无需客户端发出请求的前提下，服务端能实时地将信息主动推送到客户端，如基于Web的聊天系统、即时消息、游戏应用、实时证券报价和股票系统等。

过去，针对实时性较强的应用，开发人员使用轮询和Comet技术这些折中方案。轮询就是客户端按照预先设置的时间间隔向服务端发送请求，周期性地查询是否有数据更新。早期的轮询是通过在HTML头部插入META元信息来实现网页的自动刷新，后来人们使用AJAX轮询。AJAX轮询带来的最大好处就是在不刷新整个页面的前提下可以实现网页的局部更新，既提高了做事的效率又增强了用户体验，但是AJAX还是受限于请求-响应模式，由于无法预期什么时候推送消息，造成很多无效的请求。而Comet是使用一种新的技术去做轮询得到的效果，这种技术虽然可以实现双向通信，但是依然需要发出请求，而且Comet普遍采用长连接，这会导致大量消耗服务器带宽和资源。Comet技术本质上还是没有摆脱HTTP请求-响应模式，不能算是真正意义上的双向通信，并且开发复杂度也较高。

随着HTML5推出WebSocket，为真正解决服务器推送提供了技术保障。该规范旨在浏览器中实现和服务器端的双向通信，用以拓展浏览器上的应用类型，如实时监控系统[1]、校园通知系统[2]。WebSocket规范实际上由两部分组成，一部分是浏览器中的WebSocket API，由W3C制订；一部分是WebSocket协议，由IETF制订。它引入WebSocket接口，在WEB上通过一个单一的套接字(Socket)定义了一个全双工的通信通道，与通过维持两个连接来模拟全双工通信的轮询和长轮询方案相比，WebSocket大幅降低了不必要的网络流量和延迟[3]。WebSocket是为解决WEB客户端与服务端实时通信而产生的技术，协议设计的一个重要原则就是能和现有的Web方式和睦共处，其本质是首先通过HTTP/HTTPS协议进行握手后创建一个用于交换数据的TCP连接，此后WEB客户端与服务端

就此TCP连接进行实时通信[4]。

2 Tomcat8对WebSocket的支持(Tomcat8 support for WebSocket)

WebSocket在浏览器端的实现遵循标准的HTML5，这个和服务器采用Tomcat或者Jetty是无关的，标准化的形式是WebSocket JavaScript API[5]，主流的浏览器上都得到很好的支持，所以通过JavaScript书写Web客户端的代码既标准也较为简单。值得大家注意的是服务器端的实现在形成统一标准之前，各个实现都有自己的一套API，所以使用WebSocket开发服务器端存在一定的风险。

Tomcat7.0.27是Tomcat支持WebSocket的第一个版本，其关键是提供了org.apache.catalina.websocket.WebSocketServlet接口，在Tomcat7.0.27中需要一个Servlet来处理WebSocket请求，这个Servlet要继承自WebSocketServlet这个类，然后实现createWebSocketInbound方法，该方法返回StreamInbound对象。这个接口是非标准的WebSocket实现，到Tomcat8中，WebSocketServlet和StreamInbound这两个类都过期了，所以基于Tomcat7.0.27来实现服务器推送的技术在这里不再赘述[6]。

在Java社区技术的进步常常经历这样的阶段，就是不同的供应商和开发人员编写类库实现某种技术，随着时间的推移当该技术成熟时它就被标准化，使得开发者能够在不同的实现之间相互操作，避免冒锁定特定供应商的风险。JSR356就是把WebSocket的Java API进行标准化的结果，它已经成为JavaEE 7标准的一部分，这意味着所有JavaEE 7兼容的应用服务器都遵守JSR356的WebSocket协议标准。

Tomcat8是真正支持JSR356标准的，它自带的WebSocket API包是在Tomcat8安装目录下的lib目录中的websocket-api.jar[7]，我们需要此jar包设置到项目的classpath中。在Tomcat8中使用WebSocket的代码如下：

(1)Web客户端(JavaScript代码，与服务器类型无关)

```
<script type="text/javascript">
var websocket=new WebSocket('ws://'+window.location.host+'/test/websocket/mywebsocket');
websocket.onopen=function(event){...}  //连接成功建立的回调方法
websocket.onmessage=function(event){...}  //接收到消息的回调方法
websocket.onclose=function(){...}  //连接关闭的回调方法
websocket.onerror=function(){...}  //连接发生错误的回调方法
function send(message){websocket.send(message);}  //发送消息
</script>
```

(2)服务端Java代码[8]

WebSocket服务端处理代码的主要功能包括：a.接收客户端的请求；b.向客户端发送数据。

```
@ServerEndpoint("/websocket/mywebsocket")
public class MyWebSocket {
    private Session session;  /*与某客户端的连接会话，通过它给客户端发送数据*/
    @OnOpen  //连接建立成功调用的方法
    public void onOpen(Session session){this.session=session;}
    @OnClose  //连接关闭调用的方法
public void onClose(){  }
    @OnMessage  //收到客户端消息后调用的方法
public void onMessage(String message,Session session){
    sendMessage(socketStatus );//向客户端返回消息}
  //这个方法没有用注解，是根据自己需要添加的方法。
    public void sendMessage(String message)throws IOException{
    this.session.getBasicRemote().sendText(message);}
}
```

上述代码有一个注解@ServerEndpoint,其作用是将目前的类定义成一个websocket服务器端，注解的值"/websocket/mywebsocket"指定一个URI，客户端通过这个URI连接到WebSocket服务器端，正是因为它的存在，所以我们无需在web.xml中配置。

(2)Java客户端代码

WebSocket客户端有两种形式：一种是在浏览器端用javascript语言实现的Web客户端，同时WebSocket Java API提供了使用java语言实现客户端的途径。代码如下所示：

```
@ClientEndpoint  //注解将目前的类定义成一个websocket客户端
public class WebSocketClient{
    protected WebSocketContainer container;
    protected Session userSession=null;
    public WebSocketClient(){
        container=ContainerProvider.getWebSocketContainer();
    }
   //与服务端建立连接
    public void Connect(String sServer) {
```

```
            Try{userSession=container.
connectToServer(this, new URI(sServer));
            } catch (Exception e){...}
        }
        //发送消息
        public void sendMessage(String sMsg)throws
IOException{
            userSession.getBasicRemote().
sendText(sMsg);}
        @OnOpen
        public void onOpen(Session session){...}
        @OnClose
        public void onClose(Session session,CloseReason
closeReason){...}
        @OnMessage
        public void onMessage(Session session,String
msg){...}
    }
```

3 定点推送要解决的难题(The key problem of accurate push)

Websocket协议是一个独立的TCP协议，它和HTTP之间的唯一关系是Websocket连接建立前的握手是通过HTTP服务器解释完成的。这就意味着当通过Tomcat8.0实现的WebSocket的Web应用时，WebSocket Session和Servlet HttpSession间没有直接的关系，因此试图通过Servlet直接访问WebSocket session没有理论上的基础。尽管在Tomcat8.0同时实现了Servlet Container和WebSocket服务端，但是要通过Servlet直接访问Websocket服务端对象实例则是不可能的。这是由WebSocket的标准决定的(参考标准[9]Chapter1.7)因此需要找到在Servlet中通过Websocket推送消息的方法。

4 定点推送模型设计(Accurate push model design)

定点推送要实现的目标是能够向指定的Web注册用户进行消息推送，其特点一是非广播的，特点二是在没有浏览器请求的情况下通过Servlet主动将消息推送至某个已注册的Web用户。为了实现这个目标，设计了如图1所示的定点推送模型。

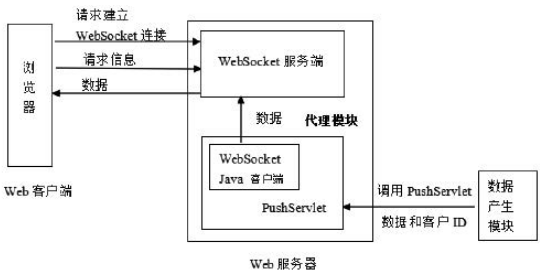

图1 定点推送模型

Fig.1 Server accurate push model

4.1 代理模块

经过上面的分析已知通过Servlet直接访问Websocket服务端对象实例是不可能的，为此我们的方法是建立一个代理模块，即建立一个专门和WebSocket服务端通信的PushServlet，该PushServlet拥有一个WebSocket Java客户端，在PushServlet doGet/doPost方法中调用WebSocket Java客户端实例将消息发送到WebSocket服务端，最后由WebSocket服务端将消息推送到Web客户端。为了使得WebSocket服务端能够将消息精准推送到指定Web客户端，前提是管理好每个客户端对应的连接会话。

4.2 会话管理，即管理每个客户端对应的连接会话

会话管理要解决的主要问题是绑定WebSocket Session和Web注册用户。为此需要定义一个数据结构进行会话管理，使用 Hashtable<String,Session>保存每个客户的clientId对应的Session，这是通过连接建立后由Web客户端将自己的clientId发送给WebSocket服务对象实现的，为此要设计一套命令格式，使得服务端能够区分客户端发来的数据是clientId还是其他消息。服务端Java代码需要做以下改进。

```
    /*保存每个clientId对应的Session对象 */
     static Hashtable<String,Session> connections=
new Hashtable<String,Session>();
        @OnOpen
    public void onOpen(Session session){this.
session=session;
      sessionId=this.session.getId();
      }
       @OnMessage    //收到客户端消息后调用的方法
    public void onMessage(String message,Session session)
{
    //如果是"绑定"请求,将clientId与session的对应关系保
存到connections
    //如果是普通消息则做消息处理
                    }
```

4.3 数据的推送过程

当数据产生模块产生了消息，它首先调用PushServlet将消息通过Servlet WebSocket Java客户端送至WebSocket服务端，在消息中带上目标浏览器的clientId。接着WebSocket服务端收到推送消息后，通过已建立的clientId和WebSocket Session的绑定，找到被推送的浏览器对应的Session，最后通过绑定的目标Session发送消息到最终定点推送注册用户所在的浏览器。

5 结论(Conclusion)

消息推送一直是Java Web实时系统要解决的关键问题，

过去人们采取的一些技术，如AJAX轮询和Comet技术仍然受制于HTTP的请求—响应模式，直到HTML5推出WebSocket规范，建立了在浏览器中实现和服务器端的双向通信机制，为真正解决服务器推送提供了技术保障。本文基于Tomcat8和JSR356标准，设计并实现了一个服务端定点推送模型，目标是能够向指定的Web注册用户进行消息推送，其特点一是非广播的，特点二是在没有浏览器请求的情况下通过Servlet主动将消息推送至某个已注册的Web用户。该模型已经成功地应用于呼叫中心系统中电话分机状态的推送和弹屏信息的推送，该模型也可以用于即时消息、游戏应用、实时证券报价、股票系统等Web实时系统，具有广泛的使用价值。

参考文献(References)

[1] Lijing Zhang,Xiaoxiao Shen.Research and Development of Real-time Monitoring System Based on WebSocket Technology[J].Mechatronic Sciences,Electric Engineering and Computer(MEC),Proceedings 2013 International Conference,2013:1955-1958.

[2] Danish,et al.PushNotify:Push Server Application [J].PROCEEDINGS OF THE 2013 3RD IEEE INTERNATIONAL ADVANCE COMPUTING CONFERENCE(IACC)Book Series:IEEE International Advance Computing Conference,2013:377-382.

[3] Slodziak,et al.Performance Analysis of Web Systems Based on XMLHttpRequest,Server-Sent Events and WebSocket[J].INFORMATION SYSTEMS ARCHITECTURE AND TECHNOLOGY,ISAT 2015,PT II Book Series:Advances in Intelligent Systems and Computing,2016,430:71-83.

[4] WebSocket协议RFC6455.The WebSocket Protocol[S].https://tools.ietf.org/html/rfc6455.

[5] W3C The WebSocket API[S].https://www.w3.org/TR/2011/WD-websockets-20110419.

[6] Apache Tomcat 7 WebSocket How-To[EB/OL].https://tomcat.apache.org/tomcat-7.0-doc/web-socket-howto.html.

[7] Apache Tomcat 8 WebSocket How-To[EB/OL].https://tomcat.apache.org/tomcat-8.0-doc/web-socket-howto.html

[8] tomcat Websocket开发人员的测试源代码[EB/OL].http://svn.apache.org/viewvc/tomcat/trunk/test/org/apache/tomcat/websocket.

[9] JSR 356:Java API for WebSocket[S].https://jcp.org/en/jsr/detail?id=356.

作者简介：

钱宇虹(1967-)，女，硕士，副教授.研究领域：软件开发与应用，软件工程.

(上接第39页)

有效解决输液监控系统功耗高、成本高、稳定性差三个技术难点。云端服务器保证了信息录入快捷且用户安全性得到了保障。整个系统提高了医护人员的工作效率，减轻了医护人员的负担。

参考文献(References)

[1] Sutherland A M,Edgar D,Duncan P.International Infusion in Practice—From Cultural Awareness to Cultural Intelligence[J].Hypertension,2015,3(3):60-85.

[2] 徐光宪,郭琳,陆伟.智能输液监控系统的设计与实现[J].激光杂志,2014(9):119-121.

[3] Breland B D.Continuous Quality Improvement Using Intelligent Infusion Pump Data Analysis[J].American Journal of Health-System Pharmacy,2010,67(17):1446-1455.

[4] 姜远海.临床医学工程技术[M].北京:科学出版社,2009.

[5] 陈章进,姚真平,张建峰.基于ZigBee技术的医疗输液监护系统设计[J].计算机测量与控制,2015,23(3):797-800.

[6] 陈晓.基于ZigBee的医疗输液监控系统的研究[D].河北工业大学,2014.

[7] Sullivan M.Improving Patient Safety with Intelligent Infusion Devices[J].American Journal of Health-System Pharmacy:AJHP:Official Journal of the American Society of Health-System Pharmacists,2010,67(17):1410-1415.

[8] 原羿,苏鸿根.基于ZigBee技术的无线网络应用研究[J].计算机应用与软件,2004,21(6):89-91.

[9] 杨艳,程荣龙,胡伟全.基于Zigbee的无线智能输液通信模型设计[J].微计算机信息,2012(10):342-343.

[10] Wei-Ping Z,Ming-Xin L,Huan C.Using MongoDB to Implement Textbook Management System Instead of MySQL[C].Communication Software and Networks(ICCSN),2011 IEEE 3rd International Conference on.IEEE,2011:303-305.

作者简介：

刘启航(1996-)，男，本科生.研究领域：通信工程.

周晓庄(1996-)，男，本科生.研究领域：计算机应用.

李 超(1975-)，男，博士，副教授.研究领域：信息安全，嵌入式.