利用websocket和java完成的消息推送功能，服务器用的是tomcat7.0，一些东西是自己琢磨的，也不知道恰不恰当，不恰当处，还请各位见谅，并指出。

程序简单来说，就是客户A可以发送消息给客户B，但有很多可以扩展的地方，

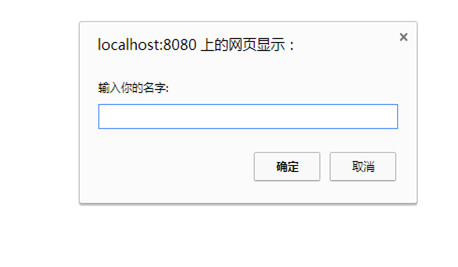
比如

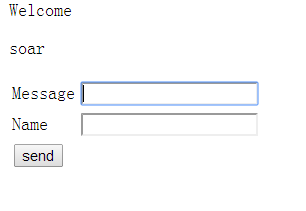
1.如果加入数据库后，A发消息时客户B未上线，服务端将会把消息存在数据库中，等客户B上线后，在将消息取出发送给客户B

2.服务端也可发送消息到任意客户端上。

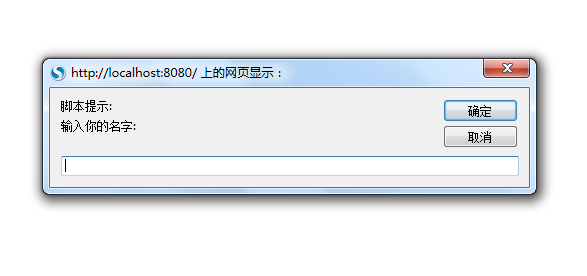
程序的运行效果截图如下（在chrome，搜狗，firefox下测试通过）：代码将在最后给出

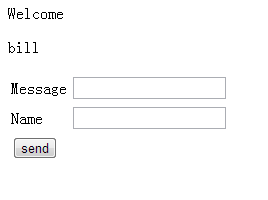
首先我们打开一个浏览器，显示输入您的名字，这里我输入soar

[](http://www.alixixi.com/web/UploadPic/2014-3/2014324165452452.png)

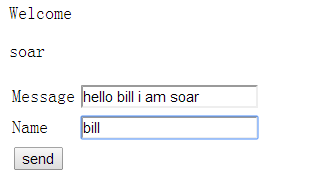
[](http://www.alixixi.com/web/UploadPic/2014-3/2014324165453254.png)

在打开第二个浏览器，这里我输入bill

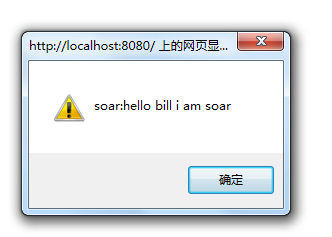
[](http://www.alixixi.com/web/UploadPic/2014-3/2014324165454506.png)

[](http://www.alixixi.com/web/UploadPic/2014-3/2014324165455488.png)

这是如果我发送hello bill i am soar给bill，点击send

[](http://www.alixixi.com/web/UploadPic/2014-3/2014324165455193.png)

在另一个浏览器上就可以看到

[](http://www.alixixi.com/web/UploadPic/2014-3/2014324165455361.png)

**Websocket**

**1.websocket是什么？**

WebSocket是为解决客户端与服务端实时通信而产生的技术。其本质是先通过HTTP/HTTPS协议进行握手后创建一个用于交换数据的TCP连接，

此后服务端与客户端通过此TCP连接进行实时通信。

**2.websocket的优点**

以前我们实现推送技术，用的都是轮询，在特点的时间间隔有浏览器自动发出请求，将服务器的消息主动的拉回来，在这种情况下，我们需要不断的向服务器 发送请求，然而HTTP request 的header是非常长的，里面包含的数据可能只是一个很小的值，这样会占用很多的带宽和服务器资源。会占用大量的带宽和服务器资源。

WebSocket API最伟大之处在于服务器和客户端可以在给定的时间范围内的任意时刻，相互推送信息。在建立连接之后，服务器可以主动传送数据给客户端。

此外，服务器与客户端之间交换的标头信息很小。

WebSocket并不限于以Ajax(或XHR)方式通信，因为Ajax技术需要客户端发起请求，而WebSocket服务器和客户端可以彼此相互推送信息；

关于ajax,comet,websocket的详细介绍，和websocket报文的介绍，大家可以参看[http://www.shaoqun.com/a/54588.aspx](http://www.shaoqun.com/a/54588.aspx" \o "http://www.shaoqun.com/a/54588.aspx)  网页设计]Ajax、Comet与Websocket，

我如果以后有时间，也会写出来的

**3.如何使用websocket**

**客户端**

在支持WebSocket的浏览器中，在创建socket之后。可以通过onopen，onmessage，onclose即onerror四个事件实现对socket进行响应

一个简单是示例

var ws = new WebSocket(“ws://localhost:8080”);

ws.onopen = function()

{

  console.log(“open”);

  ws.send(“hello”);

};

ws.onmessage = function(evt)

{

  console.log(evt.data)

};

ws.onclose = function(evt)

{

  console.log(“WebSocketClosed!”);

};

ws.onerror = function(evt)

{

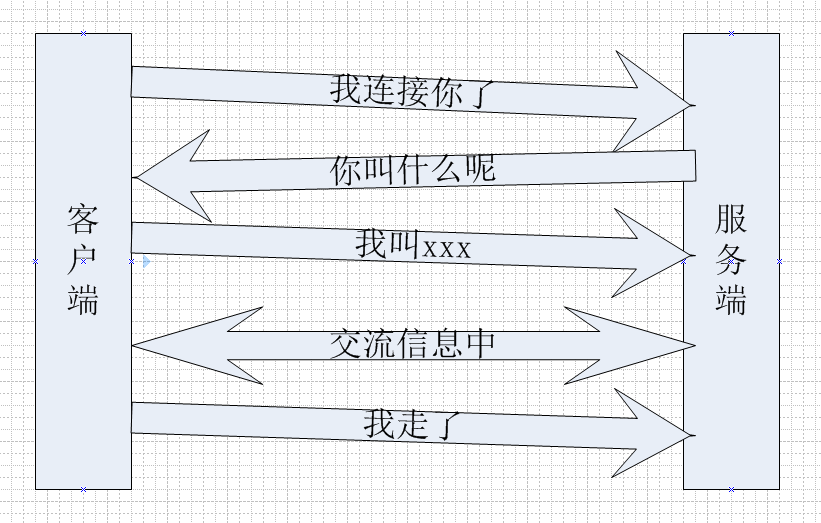
  console.log(“WebSocketError!”);

};

1.var ws = new WebSocket(“ws://localhost:8080”);  
  
申请一个WebSocket对象，参数是需要连接的服务器端的地址，同http协议使用http://开头一样，WebSocket协议的URL使用ws://开头，另外安全的WebSocket协议使用wss://开头。  
  
ws.send(“hello”);  
  
用于叫消息发送到服务端  
  
   
  
2.ws.onopen = function() { console.log(“open”)};  
  
当websocket创建成功时，即会触发onopen事件  
  
   
  
3.ws.onmessage = function(evt) { console.log(evt.data) };  
  
当客户端收到服务端发来的消息时，会触发onmessage事件，参数evt.data中包含server传输过来的数据  
  
   
  
4.ws.onclose = function(evt) { console.log(“WebSocketClosed!”); };  
  
当客户端收到服务端发送的关闭连接的请求时，触发onclose事件  
  
   
  
5.ws.onerror = function(evt) { console.log(“WebSocketError!”); };  
  
如果出现连接，处理，接收，发送数据失败的时候就会触发onerror事件  
  
我们可以看出所有的操作都是采用事件的方式触发的，这样就不会阻塞UI，使得UI有更快的响应时间，得到更好的用户体验。  
  
   
  
服务端  
  
现在有很多的服务器软件支持websocket，比如node.js,jetty,tomcat等  
  
这里我使用的是tomat 7.0和eclipse4.2  
  
在tomcat下使用websocket首先需要导入相关的jar  
  
tomcat7提供的与WebSocket相关的类均位于包org.apache.catalina.websocket之中(包org.apache.catalina.websocket的实现包含于文件catalina.jar之中  
  
   
  
这里我们把tomcat的全部导入就行了  
  
在build path->configure build path->librarise->add library->server runtime->apache tomcat v7.0  
  
image  
  
   
  
   
  
同时需要import以下包  
  
import org.apache.catalina.websocket.MessageInbound;  
import org.apache.catalina.websocket.StreamInbound;  
import org.apache.catalina.websocket.WsOutbound;  
import org.apache.catalina.websocket.WebSocketServlet;  
  
   
  
我们需要两个类  
  
第一个用于处理websocket请求  
  
第二个用于处理每一次具体的WebSocket任务  
  
   
  
第一个类  
  
  
public class SocketServer extends WebSocketServlet {  
    private static final long serialVersionUID = 1L;  
    //……  
    @Override  
    protected StreamInbound createWebSocketInbound(String arg0,  
            HttpServletRequest arg1) {  
        // TODO Auto-generated method stub  
        return new ChatWebSocket(users);  
    }  
}  
  
  
  
这个Servlet继承自WebSocketServlet，实现createWebSocketInbound方法。该方法返回第二个类的实例。  
  
   
  
第二个类  
  
  
public class ChatWebSocket extends MessageInbound {  
  
        @Override  
        protected void onTextMessage(CharBuffer message) throws IOException {  
  
        }  
  
        @Override  
        protected void onOpen(WsOutbound outbound) {  
              
        }  
  
        @Override  
        protected void onClose(int status) {  
              
  
        }  
  
        @Override  
        protected void onBinaryMessage(ByteBuffer arg0) throws IOException {  
  
        }  
//其余略  
  
    }  
  
  
  
   
  
protected void onTextMessage(CharBuffer message) throws IOException { }  
  
文本消息响应  
  
protected void onBinaryMessage(ByteBuffer arg0) throws IOException { }  
  
二进制消息响应  
  
protected void onOpen(WsOutbound outbound) { }  
  
建立连接的触发的事件  
  
protected void onClose(int status) { }  
  
关闭连接时触发的事件  
  
   
  
   
  
   
  
   
  
4.程序代码  
  
html部分  
  
  
<!DOCTYPE HTML>  
<html>  
<head>  
<meta charset="utf-8">  
<script type="text/javascript" src="js/jquery.js"></script>  
<script type="text/javascript" src="js/socket.js"></script>  
<title>无标题文档</title>  
</head>  
<script language="javascript">  
  
</script>  
<body>  
<table>  
  <tr>  
    <td>Message</td>  
    <td><input type="text" id="message"></td>  
  </tr>  
  <tr>  
    <td>Name</td>  
    <td><input type="text" id="othername"></td>  
  </tr>  
  <tr>  
    <td><input id="sendbutton" type="button" value="send" onClick="click"  disabled="true">  
      </input></td>  
  </tr>  
</table>  
<script>  
  
</script>  
</body>  
</html>  
  
  
  
   
  
js部分(关于jquery部分不进行讲解)  
  
  
var username = window.prompt("输入你的名字:");  
  
document.write("Welcome<p id=\"username\">"+username+"</p>");  
  
if (!window.WebSocket && window.MozWebSocket)  
    window.WebSocket=window.MozWebSocket;  
if (!window.WebSocket)  
    alert("No Support ");  
var ws;  
  
$(document).ready(function(){  
      
     $("#sendbutton").attr("disabled", false);  
     $("#sendbutton").click(sendMessage);  
  
    startWebSocket();  
})  
  
function sendMessage()  
{  
    var othername=$("#othername").val();  
    var msg="MSG\t"+username+"\_"+othername+"\_"+$("#message").val();  
    send(msg);  
}  
function send(data)  
{  
    console.log("Send:"+data);  
    ws.send(data);  
}  
function startWebSocket()  
{      
    ws = new WebSocket("ws://" + location.host + "/WebSocket/SocketServer");  
    ws.onopen = function(){  
        console.log("success open");  
        $("#sendbutton").attr("disabled", false);  
    };  
     ws.onmessage = function(event)  
     {  
         console.log("RECEIVE:"+event.data);  
         handleData(event.data);   
     };  
      ws.onclose = function(event) {   
    console.log("Client notified socket has closed",event);   
  };   
    
}  
  
function handleData(data)  
{  
    var vals=data.split("\t");  
    var msgType=vals[0];  
    switch(msgType)  
    {  
    case "NAME":  
        var msg=vals[1];  
        var mes="NAME"+"\t"+msg+"\_"+ username;  
        send(mes);  
        break;  
    case "MSG":  
        var val2s=vals[1].split("\_");  
        var from=val2s[0];  
        var message=val2s[2];  
        alert(from+":"+message);  
        break;  
    default:  
        break;  
              
    }  
}  
  
  
  
java部分  
  
  
import java.io.IOException;  
import java.nio.ByteBuffer;  
import java.nio.CharBuffer;  
  
import javax.servlet.http.HttpServletRequest;  
import java.util.Set;  
  
  
import java.util.concurrent.CopyOnWriteArraySet;  
  
import org.apache.catalina.websocket.MessageInbound;  
import org.apache.catalina.websocket.StreamInbound;  
import org.apache.catalina.websocket.WsOutbound;  
import org.apache.catalina.websocket.WebSocketServlet;  
  
  
  
public class SocketServer extends WebSocketServlet {  
    private static final long serialVersionUID = 1L;  
    public final Set<ChatWebSocket> users = new CopyOnWriteArraySet<ChatWebSocket>();  
  
    public static int USERNUMBER = 1;  
    @Override  
    protected StreamInbound createWebSocketInbound(String arg0,  
            HttpServletRequest arg1) {  
        // TODO Auto-generated method stub  
        return new ChatWebSocket(users);  
    }  
    public class ChatWebSocket extends MessageInbound {  
  
        private String username;  
        private Set<ChatWebSocket> users = new CopyOnWriteArraySet<ChatWebSocket>();;  
  
        public ChatWebSocket() {  
  
        }  
  
        public ChatWebSocket(Set<ChatWebSocket> users) {  
            this.users = users;  
        }  
  
        @Override  
        protected void onTextMessage(CharBuffer message) throws IOException {  
            // 这里处理的是文本数据  
        }  
  
        public void onMessage(String data) {  
            String[] val1 = data.split("\\t");  
            if(val1[0].equals("NAME"))  
            {  
                String[] val2=val1[1].split("\_");  
                for(ChatWebSocket user:users){  
                    if (user.username.equals(val2[0])){  
                        user.username=val2[1];  
                    }  
                }  
            }  
            else if(val1[0].equals("MSG"))  
            {  
                String[] val2=val1[1].split("\_");  
                for(ChatWebSocket user:users){  
                    if (user.username.equals(val2[1])){  
                        try {  
                            CharBuffer temp=CharBuffer.wrap(data);  
                            user.getWsOutbound().writeTextMessage(temp);  
                        } catch (IOException e) {  
                            // TODO Auto-generated catch block  
                            e.printStackTrace();  
                        }  
                    }  
                }          
            }  
            else  
            {  
                System.out.println("ERROR");  
            }  
  
        }  
  
        @Override  
        protected void onOpen(WsOutbound outbound) {  
            // this.connection=connection;  
            this.username = "#" + String.valueOf(USERNUMBER);  
            USERNUMBER++;  
            try {  
                String message = "NAME" + "\t" + this.username;  
                CharBuffer buffer = CharBuffer.wrap(message);  
                this.getWsOutbound().writeTextMessage(buffer);  
            } catch (IOException e) {  
                // TODO Auto-generated catch block  
                e.printStackTrace();  
            }  
            users.add(this);  
        }  
  
        @Override  
        protected void onClose(int status) {  
            users.remove(this);  
  
        }  
  
        @Override  
        protected void onBinaryMessage(ByteBuffer arg0) throws IOException {  
  
        }  
  
    }  
      
}  
   
  
解释  
  
这里我的想法是  
  
1 每个用户在访问的时候首先需要输入自己的名字，接着向服务端发送连接请求  
  
2 服务端在接受到客户端的连接请求后，会new ChatWebSocket(users);用于处理这个请求，并把它加入在线的用户列表中，由于这个时候，服务端尚不知道客户的名字。它会给这个用户假定一个名字，#1，然后服务端会发送"NAME" + "\t" +“#1”给客户端，你叫什么？  
  
3 客户端收到这个消息会知道，服务器在问自己叫什么名字，于是客户端会发送"NAME"+"\t"+“#1”+"\_"+ 自己的名字到服务端，（我叫xxx）  
  
4 服务端收到这个消息后根据#1在当前在线的用户列表中进行查找，将#1替换为客户的名字，这样服务端就知道了这个客户的名字了

5 当客户离开时，服务端会触发onClose事件，服务端会把当前用户从在线列表中移除

用图画出来类似这样（画的不好，—\_—!!）

[](http://www.alixixi.com/web/UploadPic/2014-3/2014324165456490.png)

代码  
  
js  
  
ws = new WebSocket("ws://" + location.host + "/WebSocket/SocketServer");  
  
   
  
连接服务端  
  
   
  
java  
  
protected StreamInbound createWebSocketInbound(String arg0,  
            HttpServletRequest arg1) {  
        // TODO Auto-generated method stub  
        return new ChatWebSocket(users);  
    }  
  
创建一个chatwebsocket用于处理这个请求，触发该chatwebsocket对象的onOpen事件  
  
  
@Override  
    protected void onOpen(WsOutbound outbound) {  
        // this.connection=connection;  
        this.username = "#" + String.valueOf(USERNUMBER);  
        USERNUMBER++;  
        try {  
            String message = "NAME" + "\t" + this.username;  
            CharBuffer buffer = CharBuffer.wrap(message);  
            this.getWsOutbound().writeTextMessage(buffer);  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
        users.add(this);  
    }  
  
  
  
为这个客户假定一个姓名，并发送NAME+“\t”+假定的姓名  给该客户端，同时将该客户端加入当前连接的客户列表中  
  
   
  
js  
  
  
function handleData(data)  
{  
    var vals=data.split("\t");  
    var msgType=vals[0];  
    switch(msgType)  
    {  
    case "NAME":  
        var msg=vals[1];  
        var mes="NAME"+"\t"+msg+"\_"+ username;  
        send(mes);  
        break;  
   //………              
    }  
}  
  
  
  
接受并处理服务端发来到的消息，发现是服务端问自己叫什么名字，于是发送”NAME"+"\t"+假定的名字+"\_"+ 真正的名字 给服务端  
  
   
  
   
  
java  
  
  
public void onMessage(String data) {  
        String[] val1 = data.split("\\t");  
        if(val1[0].equals("NAME"))  
        {  
            String[] val2=val1[1].split("\_");  
            for(ChatWebSocket user:users){  
                if (user.username.equals(val2[0])){  
                    user.username=val2[1];  
                }  
            }  
        }  
  
//………  
}  
  
  
  
处理并接受客户端发来的消息，发现是客户端回复自己叫什么名字，于是在根据先前假定的名字在当前连接的客户列表中进行查找，将假名变成真名  
  
   
  
js  
  
function sendMessage()  
{  
    var othername=$("#othername").val();  
    var msg="MSG\t"+username+"\_"+othername+"\_"+$("#message").val();  
    send(msg);  
}  
  
客户对另一个人发起对话，消息格式为：“MSG”+自己的名字+\_+对方的名字+\_+消息  
  
   
  
   
  
java  
  
  
public void onMessage(String data) {  
       ///…………  
        else if(val1[0].equals("MSG"))  
        {  
            String[] val2=val1[1].split("\_");  
            for(ChatWebSocket user:users){  
                if (user.username.equals(val2[1])){  
                    try {  
                        CharBuffer temp=CharBuffer.wrap(data);  
                        user.getWsOutbound().writeTextMessage(temp);  
                    } catch (IOException e) {  
                        // TODO Auto-generated catch block  
                        e.printStackTrace();  
                    }  
                }  
            }          
        }  
       ///…………  
  
    }  
  
  
  
发现是客户发送的消息，根据对方的姓名，在当前连接的客户列表中查找，并将消息发给他  
  
   
  
   
  
js  
  
  
function handleData(data)  
{  
    var vals=data.split("\t");  
    var msgType=vals[0];  
    switch(msgType)  
    {  
    ///…  
  
    case "MSG":  
        var val2s=vals[1].split("\_");  
        var from=val2s[0];  
        var message=val2s[2];  
        alert(from+":"+message);  
        break;  
    default:  
        break;  
              
    }  
}  
  
  
  
发现是另一个客户发来的消息,通过alert显示出来  
  
   
  
java  
  
@Override  
        protected void onClose(int status) {  
            users.remove(this);  
  
        }  
  
发现客户离开了，将客户从连接的客户列表中移除  
  
可以改进的地方  
  
1.若客户端A发送消息给B时，B不在线，可将消息存入数据库中，当发现B上线时，从数据库中取出，发送给B  
  
2 服务端发送你叫什么时，可加入超时机制，若客户端一定时间内没有回复自己叫什么，则可将该客户从在线列表中删掉