<http://www.cnblogs.com/xdp-gacl/p/5193279.html>

# [孤傲苍狼](http://www.cnblogs.com/xdp-gacl/)

**只为成功找方法，不为失败找借口！**

## [Java后端WebSocket的Tomcat实现](http://www.cnblogs.com/xdp-gacl/p/5193279.html)

## 一.WebSocket简单介绍

　　随着互联网的发展，传统的HTTP协议已经很难满足Web应用日益复杂的需求了。近年来，随着HTML5的诞生，WebSocket协议被提出，它实现了浏览器与服务器的全双工通信，扩展了浏览器与服务端的通信功能，使服务端也能主动向客户端发送数据。

　　我们知道，传统的HTTP协议是无状态的，每次请求（request）都要由客户端（如 浏览器）主动发起，服务端进行处理后返回response结果，而服务端很难主动向客户端发送数据；这种客户端是主动方，服务端是被动方的传统Web模式 对于信息变化不频繁的Web应用来说造成的麻烦较小，而对于涉及实时信息的Web应用却带来了很大的不便，如带有即时通信、实时数据、订阅推送等功能的应 用。在WebSocket规范提出之前，开发人员若要实现这些实时性较强的功能，经常会使用折衷的解决方法：**轮询（polling）**和**Comet**技术。其实后者本质上也是一种轮询，只不过有所改进。

**轮询**是最原始的实现实时Web应用的解决方案。轮询技术要求客户端以设定的时间间隔周期性地向服务端发送请求，频繁地查询是否有新的数据改动。明显地，这种方法会导致过多不必要的请求，浪费流量和服务器资源。

**Comet技术**又可以分为**长轮询**和**流技术**。**长轮询**改进了上述的轮询技术，减小了无用的请求。它会为某些数据设定过期时间，当数据过期后才会向服务端发送请求；这种机制适合数据的改动不是特别频繁的情况。**流技术**通常是指客户端使用一个隐藏的窗口与服务端建立一个HTTP长连接，服务端会不断更新连接状态以保持HTTP长连接存活；这样的话，服务端就可以通过这条长连接主动将数据发送给客户端；流技术在大并发环境下，可能会考验到服务端的性能。

　　这两种技术都是基于请求-应答模式，都不算是真正意义上的实时技术；它们的每一次请求、应答，都浪费了一定流量在相同的头部信息上，并且开发复杂度也较大。

　　伴随着HTML5推出的WebSocket，真正实现了Web的实时通信，使B/S模式具备了C/S模式的实时通信能力。WebSocket的工作流程是这 样的：浏览器通过JavaScript向服务端发出建立WebSocket连接的请求，在WebSocket连接建立成功后，客户端和服务端就可以通过 TCP连接传输数据。因为WebSocket连接本质上是TCP连接，不需要每次传输都带上重复的头部数据，所以它的数据传输量比轮询和Comet技术小 了很多。本文不详细地介绍WebSocket规范，主要介绍下WebSocket在Java Web中的实现。

　　JavaEE 7中出了JSR-356:Java API for WebSocket规范。不少Web容器，如Tomcat,Nginx,Jetty等都支持WebSocket。Tomcat从7.0.27开始支持 WebSocket，从7.0.47开始支持JSR-356，下面的Demo代码也是需要部署在**Tomcat7.0.47**以上的版本才能运行。

## 二.WebSocket示例

### 2.1.新建JavaWeb测试项目

　　在pom.xml中添加Jar包依赖

1 <dependency>

2 <groupId>javax</groupId>

3 <artifactId>javaee-api</artifactId>

4 <version>7.0</version>

5 <scope>provided</scope>

6 </dependency>

客户端（Web主页）代码：

[复制代码](javascript:void(0);)

1 <%@ page language="java" pageEncoding="UTF-8" %>

2 <!DOCTYPE html>

3 <html>

4 <head>

5 <title>Java后端WebSocket的Tomcat实现</title>

6 </head>

7 <body>

8 Welcome<br/><input id="text" type="text"/>

9 <button onclick="send()">发送消息</button>

10 <hr/>

11 <button onclick="closeWebSocket()">关闭WebSocket连接</button>

12 <hr/>

13 <div id="message"></div>

14 </body>

15

16 <script type="text/javascript">

17 var websocket = null;

18 //判断当前浏览器是否支持WebSocket

19 if ('WebSocket' in window) {

20 websocket = new WebSocket("ws://localhost:8080/websocket");

21 }

22 else {

23 alert('当前浏览器 Not support websocket')

24 }

25

26 //连接发生错误的回调方法

27 websocket.onerror = function () {

28 setMessageInnerHTML("WebSocket连接发生错误");

29 };

30

31 //连接成功建立的回调方法

32 websocket.onopen = function () {

33 setMessageInnerHTML("WebSocket连接成功");

34 }

35

36 //接收到消息的回调方法

37 websocket.onmessage = function (event) {

38 setMessageInnerHTML(event.data);

39 }

40

41 //连接关闭的回调方法

42 websocket.onclose = function () {

43 setMessageInnerHTML("WebSocket连接关闭");

44 }

45

46 //监听窗口关闭事件，当窗口关闭时，主动去关闭websocket连接，防止连接还没断开就关闭窗口，server端会抛异常。

47 window.onbeforeunload = function () {

48 closeWebSocket();

49 }

50

51 //将消息显示在网页上

52 function setMessageInnerHTML(innerHTML) {

53 document.getElementById('message').innerHTML += innerHTML + '<br/>';

54 }

55

56 //关闭WebSocket连接

57 function closeWebSocket() {

58 websocket.close();

59 }

60

61 //发送消息

62 function send() {

63 var message = document.getElementById('text').value;

64 websocket.send(message);

65 }

66 </script>

67 </html>

[复制代码](javascript:void(0);)

Java Web后端代码

[复制代码](javascript:void(0);)

1 package me.gacl.websocket;

2

3 import java.io.IOException;

4 import java.util.concurrent.CopyOnWriteArraySet;

5

6 import javax.websocket.\*;

7 import javax.websocket.server.ServerEndpoint;

8

9 /\*\*

10 \* @ServerEndpoint 注解是一个类层次的注解，它的功能主要是将目前的类定义成一个websocket服务器端,

11 \* 注解的值将被用于监听用户连接的终端访问URL地址,客户端可以通过这个URL来连接到WebSocket服务器端

12 \*/

13 @ServerEndpoint("/websocket")

14 public class WebSocketTest {

15 //静态变量，用来记录当前在线连接数。应该把它设计成线程安全的。

16 private static int onlineCount = 0;

17

18 //concurrent包的线程安全Set，用来存放每个客户端对应的MyWebSocket对象。若要实现服务端与单一客户端通信的话，可以使用Map来存放，其中Key可以为用户标识

19 private static CopyOnWriteArraySet<WebSocketTest> webSocketSet = new CopyOnWriteArraySet<WebSocketTest>();

20

21 //与某个客户端的连接会话，需要通过它来给客户端发送数据

22 private Session session;

23

24 /\*\*

25 \* 连接建立成功调用的方法

26 \* @param session 可选的参数。session为与某个客户端的连接会话，需要通过它来给客户端发送数据

27 \*/

28 @OnOpen

29 public void onOpen(Session session){

30 this.session = session;

31 webSocketSet.add(this); //加入set中

32 addOnlineCount(); //在线数加1

33 System.out.println("有新连接加入！当前在线人数为" + getOnlineCount());

34 }

35

36 /\*\*

37 \* 连接关闭调用的方法

38 \*/

39 @OnClose

40 public void onClose(){

41 webSocketSet.remove(this); //从set中删除

42 subOnlineCount(); //在线数减1

43 System.out.println("有一连接关闭！当前在线人数为" + getOnlineCount());

44 }

45

46 /\*\*

47 \* 收到客户端消息后调用的方法

48 \* @param message 客户端发送过来的消息

49 \* @param session 可选的参数

50 \*/

51 @OnMessage

52 public void onMessage(String message, Session session) {

53 System.out.println("来自客户端的消息:" + message);

54 //群发消息

55 for(WebSocketTest item: webSocketSet){

56 try {

57 item.sendMessage(message);

58 } catch (IOException e) {

59 e.printStackTrace();

60 continue;

61 }

62 }

63 }

64

65 /\*\*

66 \* 发生错误时调用

67 \* @param session

68 \* @param error

69 \*/

70 @OnError

71 public void onError(Session session, Throwable error){

72 System.out.println("发生错误");

73 error.printStackTrace();

74 }

75

76 /\*\*

77 \* 这个方法与上面几个方法不一样。没有用注解，是根据自己需要添加的方法。

78 \* @param message

79 \* @throws IOException

80 \*/

81 public void sendMessage(String message) throws IOException{

82 this.session.getBasicRemote().sendText(message);

83 //this.session.getAsyncRemote().sendText(message);

84 }

85

86 public static synchronized int getOnlineCount() {

87 return onlineCount;

88 }

89

90 public static synchronized void addOnlineCount() {

91 WebSocketTest.onlineCount++;

92 }

93

94 public static synchronized void subOnlineCount() {

95 WebSocketTest.onlineCount--;

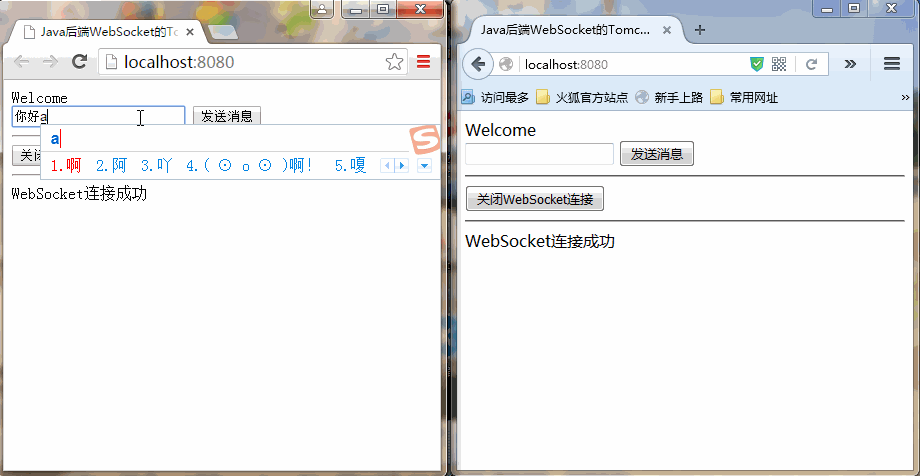
96 }

97 }

[复制代码](javascript:void(0);)

### 1.2.运行效果

　　同时打开Google浏览器和火狐浏览器进行多客户端模拟测试,运行效果如下:



**该Demo在Jdk1.7+Tomcat7.0.65下环境测试过,**[示例项目代码下载](http://pan.baidu.com/s/1hrx1GFy)

　　本篇博客的大部分内容转载自http://blog.chenzuhuang.com/archive/28.html,然后在此基础上进行完善,在此对作者表示感谢.