[Websocket协议学习 2](#_Toc16855)

[1. websocket是什么 2](#_Toc18272)

[2. websocket的优点 2](#_Toc14133)

[3. 历史沿革 2](#_Toc31894)

[Ajax技术 2](#_Toc27457)

[Comet技术 2](#_Toc10886)

[Websocket 3](#_Toc13270)

[4. websocket逻辑 3](#_Toc18360)

[Websocket通信过程 3](#_Toc11533)

[Handshake版本 3](#_Toc18635)

[Opening Handshake 3](#_Toc17998)

[Data Framing 4](#_Toc32707)

[Closing Handshake 4](#_Toc4312)

[websocket的事件响应 4](#_Toc27369)

[5. 如何使用websocket 5](#_Toc9145)

[java 实现websocket的两种方式 5](#_Toc5999)

[简单说明 5](#_Toc18505)

[方式一：tomcat 5](#_Toc7803)

[服务器端代码 6](#_Toc31643)

[客户端js 6](#_Toc14149)

[方式二：spring 6](#_Toc29845)

[Springboot websocket 6](#_Toc28524)

# [Websocket协议学习](http://www.cnblogs.com/lizhenghn/p/5155933.html)

<https://www.cnblogs.com/lizhenghn/p/5155933.html>

## 1. websocket是什么

Websocket是html5提出的一个协议规范，参考rfc6455。

websocket约定了一个通信的规范，通过一个握手的机制，客户端（浏览器）和服务器（webserver）之间能建立一个类似tcp的连接，从而方便c－s之间的通信。在websocket出现之前，web交互一般是基于http协议的短连接或者长连接。

WebSocket是为解决客户端与服务端实时通信而产生的技术。websocket协议本质上是一个基于tcp的协议，是先通过HTTP/HTTPS协议发起一条特殊的http请求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。

注意：此时不再需要原HTTP协议的参与了。

## 2. websocket的优点

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。 浏览器和服务器只需要要做一个握手的动作，在建立连接之后，服务器可以主动传送数据给客户端，客户端也可以随时向服务器发送数据。 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；

因此从服务器角度来说，websocket有以下好处：

1. 节省每次请求的header；http的header一般有几十字节
2. Server Push；服务器可以主动传送数据给客户端

## 3. 历史沿革

### Ajax技术

### Comet技术

客户端与服务器端保持一个长连接，只有客户端需要的数据更新时，服务器才主动将数据推送给客户端。Comet的实现主要有两种方式：

1，基于Ajax的长轮询（long-polling）方式。有数据/超时时返回。

2，基于 Iframe 及 htmlfile 的流（http streaming）方式。Iframe是html标记，其标记的src属性会保持对指定服务器的长连接请求，服务器端则可以不停地返回数据

### Websocket

Websocket是一个全新的、独立的协议，基于TCP协议，与http协议兼容、却不会融入http协议，仅仅作为html5的一部分。

浏览器发起websocket请求后，和服务器进行全双工通讯。

## 4. websocket逻辑

### Websocket通信过程

Handshake（Opening Handshake）

建立连接，双向通信。

Handshake（Closing Handshake）

### Handshake版本

1. 基于flash的握手协议；用于IE和低版本浏览器，没有加密，可靠性很差。
2. 基于md5加密方式的握手协议；
3. 基于sha加密方式的握手协议；需要13以上的版本。server把客户端上报的key拼上GUID（ “258EAFA5-E914-47DA-95CA-C5AB0DC85B11″），进行SHA-1 hash计算，再把结果通过base64加密，最后返回给客户端。

### Opening Handshake

#### 通信报文

客户端发起连接Handshake请求

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

服务器端响应

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

Sec-WebSocket-Protocol: chat

#### 字段意义

**Upgrade：WebSocket**

表示这是一个特殊的 HTTP 请求，请求的目的就是要将客户端和服务器端的通讯协议从 HTTP 协议升级到 WebSocket 协议。

**Sec-WebSocket-Key**

浏览器base64加密的密钥，server收到后需要提取Sec-WebSocket-Key 信息，然后加密返回。

**Sec-WebSocket-Accept**

服务器返回给客户端的验证信息（就是Sec-WebSocket-Key）。如果加密算法错误，客户端在进行校检的时候会直接报错。如果握手成功，则客户端侧会出发onopen事件。

**Sec-WebSocket-Protocol**表示客户端请求提供的可供选择的子协议，及服务器端选中的支持的子协议，“Origin”服务器端用于区分未授权的websocket浏览器

**Sec-WebSocket-Version: 13**版本标识，表示这个是一个升级版本，现在的浏览器都是使用的这个版本。

**HTTP/1.1 101 Switching Protocols**

101为服务器返回的状态码，所有非101的状态码都表示handshake并未完成。

### Data Framing

Websocket协议通过序列化的数据帧传输数据。数据封包协议中定义了opcode、payload length、Payload data等字段。其中要求：

1. 客户端向服务器传输的数据帧必须进行掩码处理：服务器若接收到未经过掩码处理的数据帧，则必须主动关闭连接。
2. 服务器向客户端传输的数据帧一定不能进行掩码处理。客户端若接收到经过掩码处理的数据帧，则必须主动关闭连接。

针对上情况，发现错误的一方可向对方发送close帧（状态码是1002，表示协议错误），以关闭连接。

### Closing Handshake

主动关闭的一方向另一方发送一个关闭类型的数据包，对方收到此数据包之后，再回复一个相同类型的数据包，关闭完成。

关闭类型数据包遵守封包协议，Opcode为0x8，Payload data可以用于携带关闭原因或消息。

### websocket的事件响应

Opening Handshake---onopen 当接口打开时响应

Data Framing---onmessage 当收到信息时响应

Closing Handshake---onclose 当接口关闭时响应

任何程序语言的websocket api都至少要提供上面三个事件的api接口， 有的可能还提供的有onerror事件的处理机制。

websocket 在任何时候都会处于下面4种状态中的其中一种：

CONNECTING (0)：表示还没建立连接；

OPEN (1)： 已经建立连接，可以进行通讯；

CLOSING (2)：通过关闭握手，正在关闭连接；

CLOSED (3)：连接已经关闭或无法打开；

## 5. 如何使用websocket

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

示例：

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

//建立连接；WebSocket协议使用ws://开头，安全WebSocket协议使用wss://开头。

ws.onopen = function() //绑定响应函数

{

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!”); };

ws.send(“hello”); # 用于将消息发送到服务端

ws.recv($buffer); # 用于接收服务端的消息

# [java 实现websocket的两种方式](http://www.cnblogs.com/interdrp/p/7903736.html)

<https://www.cnblogs.com/interdrp/p/7903736.html>

## 简单说明

1.两种方式，一种使用tomcat的websocket实现，一种使用spring的websocket

2.tomcat的方式需要tomcat 7.x，JEE7的支持。

3.spring与websocket整合需要spring 4.x，并且使用了socketjs，对不支持websocket的浏览器可以模拟websocket使用

## 方式一：tomcat

使用这种方式无需别的任何配置，只需服务端一个处理类

### 服务器端代码

#### Tomcat7

1，继承抽象类WebSocketServlet。创建继承MessageInbound的内部类MsgInbound，填充onClose、onOpen、onBinaryMessage和onTextMessage等方法，完成各个事件的逻辑。onBinaryMessage是Binary方式下接收到客户端数据时被调用，onTextMessage是Text方式下接收到客户端数据时被调用。实现createWebSocketInbound()，返回MsgInbound实例。

2，配web.xml文件。

#### tomcat8

只需要写java类，通过注解实现。

@ServerEndpoint(“/websocket1”)；注解类？？？报不是servlet错误

@OnOpen,@OnMessage,@OnError,@OnClose；注解方法

从javax.websocket.Session获取信息。

### 客户端js

需判断当前浏览器是否支持WebSocket

if ('WebSocket' in window) {

websocket = new WebSocket("ws://" + document.location.host + "/WebChat/websocket/" + username + "/"+ \_img);

} else {

alert('当前浏览器 Not support websocket')

}

## 方式二：spring

TextWebSocketHandler

# Springboot websocket

<http://blog.csdn.net/xieliaowa9231/article/details/79424598>

1. 导包spring-boot-starter-web
2. 配置类

@Configuration

public class WebSocketConfig {

@Bean

public ServerEndpointExporter serverEndpointExporter(){

return new ServerEndpointExporter();

}

}

3，实现类，使用注解@ServiceEndpoint,@OnOpen,@OnMessage,@OnError,@OnClose。通过session获取信息。