**测试环境部署情况：**

服务器IP地址：192.168.243.18

端口：9032

前端地址：<http://192.168.243.18:9032/index.html>

接口地址：<http://192.168.243.18:9032/hello?name=nihao>

**浏览器支持**：所有的最新浏览器支持最新WebSocket规范([RFC 6455](https://link.jianshu.com?t=https:/tools.ietf.org/html/rfc6455)) ，从[维基百科](https://link.jianshu.com?t=https:/zh.wikipedia.org/wiki/WebSocket)上介绍浏览器对WebSocket的支持如下表所示：

| **浏览器** | **Chrome** | **Edge** | **Firfox** | **IE** | **Opera** | **Safari** |
| --- | --- | --- | --- | --- | --- | --- |
| 最低版本 | 16 | 支持 | 11.0 | 10 | 12.10 | 6.0 |

**java WebSocket实现**

Oracle 发布的 java 的 WebSocket 的规范是 [JSR356规范](https://link.jianshu.com?t=http:/www.oracle.com/technetwork/articles/java/jsr356-1937161.html) ,Tomcat从7.0.27开始支持WebSocket，从7.0.47开始支持JSR-356。

JDK需要JDK7或以上。

实现 后台

**1. 添加pom文件依赖**

pom.xml中增加如下jar包如下：

 <!-- SpringWebSocket依赖 -->

       <dependency>

           <groupId>org.springframework.boot</groupId>

           <artifactId>spring-boot-starter-websocket</artifactId>

       </dependency>

**2.java方式配置websocket stomp**

新建WebSocket 的配置类

WebSocketConfig.java

WebSocketConfig.java

package com.us.example.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.messaging.simp.config.MessageBrokerRegistry;

import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;

import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;

import org.springframework.web.socket.config.annotation.StompEndpointRegistry;

@Configuration

@EnableWebSocketMessageBroker

//通过EnableWebSocketMessageBroker 开启使用STOMP协议来传输基于代理(message broker)的消息,此时浏览器支持使用@MessageMapping 就像支持@RequestMapping一样。

public class WebSocketConfig extends WebSocketMessageBrokerConfigurer{

   @Override

   public void registerStompEndpoints(StompEndpointRegistry registry) { //endPoint 注册协议节点,并映射指定的URl

       registry.addEndpoint("/bd-util-websocket").withSockJS();//添加一个/endpointWisely，客户端就可以通过这个端点来进行连接；withSockJS作用是添加SockJS支持。

  }

   @Override

   public void configureMessageBroker(MessageBrokerRegistry config) {//配置消息代理(message broker)

       config.enableSimpleBroker("/topic"); //定义了两个客户端订阅地址的前缀信息，也就是客户端接收服务端发送消息的前缀信息

       config.setApplicationDestinationPrefixes("/app");//定义了服务端接收地址的前缀，也即客户端给服务端发消息的地址前缀

       // 订阅Broker名称

       //registry.enableSimpleBroker("/queue", "/topic");

       // 全局使用的消息前缀（客户端订阅路径上会体现出来）

       //registry.setApplicationDestinationPrefixes("/app");

       // 点对点使用的订阅前缀（客户端订阅路径上会体现出来），不设置的话，默认也是/user/

       // registry.setUserDestinationPrefix("/user/");

  }

}

代码详解：

1. EnableWebSocketMessageBroker 注解表明： 这个配置类不仅配置了 WebSocket，还配置了基于代理的 STOMP 消息；
2. 它复写了 registerStompEndpoints() 方法：添加一个服务端点，来接收客户端的连接。将 “/bd-util-websocket” 路径注册为 STOMP 端点。这个路径与之前发送和接收消息的目的路径有所不同， 这是一个端点，客户端在订阅或发布消息到目的地址前，要连接该端点，即用户发送请求 ：url=’/127.0.0.1:8080/chat’ 与 STOMP server 进行连接，之后再转发到订阅url；
3. 它复写了 configureMessageBroker() 方法：配置了一个 简单的消息代理，通俗一点讲就是设置消息连接请求的各种规范信息。

**4.书写控制层**

import com.us.example.bean.Message;

import com.us.example.bean.Response;

import org.springframework.messaging.handler.annotation.MessageMapping;

import org.springframework.messaging.handler.annotation.SendTo;

import org.springframework.stereotype.Controller;

@Controller

public class WebSocketController {

   @RequestMapping(value = "/ws")

   public String ws(){

       return  "ws";

  }

   @MessageMapping("/welcome")//浏览器发送请求通过@messageMapping 映射/welcome 这个地址。

   @SendTo("/topic/getResponse")//服务器端有消息时,会订阅@SendTo 中的路径的浏览器发送消息。

   public Response say(Message message) throws Exception {

       Thread.sleep(1000);

       return new ServerMessage("Welcome, " + message.getName() + "!");

  }

   @SubscribeMapping("/subscribeTest")

   public ServerMessage sub() {

       logger.info("XXX用户订阅了我。。。");

       return new ServerMessage("感谢你订阅了我。。。");

  }

}

代码详解：

@MessageMapping 标识客户端发来消息的请求地址，前面我们全局配置中制定了服务端接收的地址以“/app”开头，所以客户端发送消息的请求连接是：/app/welcome

接收客户端发送的消息，当客户端发送消息的目的地为/app/welcome时，交给该注解所在的方法处理消息，其中/app是在

registry.setApplicationDestinationPrefixes("/app");

一步配置的客户端请求前缀

若没有添加@SendTo注解且该方法有返回值，则返回的目的地地址为/topic/welcome，经过消息代理，客户端需要订阅了这个主题才能收到返回消息

@SubscribeMapping(“/subscribeTest”)

接收客户端发送的订阅，当客户端订阅的目的地为/app/subscribeTest时，交给该注解所在的方法处理订阅，其中/app为客户端请求前缀

若没有添加@SendTo注解且该方法有返回值，则返回的目的地地址为/app/subscribeTest，不经过消息代理，客户端需要订阅了这个主题才能收到返回消息

@SendTo(“/topic/subscribeTest”)

修改返回消息的目的地地址为/topic/subscribeTest，经过消息代理，客户端需要订阅了这个主题才能收到返回消息

**5.书写客户端**

在html页面上引入相关的js文件：

<script src="http://cdn.bootcss.com/stomp.js/2.3.3/stomp.min.js"></script>

<script src="https://cdn.bootcss.com/sockjs-client/1.1.4/sockjs.min.js"></script>

<script type="text/javascript">

   // 建立连接对象（还未发起连接）

   var socket = new SockJS("http://localhost:9032/bd-util-websocket");

   // 获取 STOMP 子协议的客户端对象

   var stompClient = Stomp.over(socket);

在js文件中添加：

1、STOMP客户端创建：

var socket = new SockJS('/bd-util-websocket');  
stompClient = Stomp.over(socket);

2、连接到服务端：

stompClient.connect({}, function (frame) {   
 console.log('Connected: ' + frame);   
});

3、断开连接：

stompClient.disconnect();

4、订阅和接收消息：

stompClient.subscribe('/topic/greetings', function (json) {  
 console.log(JSON.parse(json.body).respData);  
});

5、发送消息：

stompClient.send("/app/hello", {}, JSON.stringify({'name': $("#name").val()}));

**使用Template发送数据**

上面是使用@sendTo 注解方式，若想在service层发送消息感觉不太方便，

pom.xml中新增引入：

         <dependency>

           <groupId>org.springframework</groupId>

           <artifactId>spring-messaging</artifactId>

           <version>4.2.3.RELEASE</version>

       </dependency>

**上面我们介绍了如何在controller中发送数据给clients，spring还提供了一个叫做SimpMessagingTemplate的模板，以便我们在应用的任何地方发送数据给Broker**

@Controller

public class GreetingController {

   @Autowired

   private SimpMessagingTemplate template;

   @RequestMapping(path="/greetings", method=POST)

   public void greet(String greeting) {

       String text = "[" + getTimestamp() + "]:" + greeting;

      template.convertAndSend("/topic/greetings", text);

  }

}

@Service

public class GreetingServiceImpl {

   @Autowired

   private SimpMessagingTemplate template;

   public void greet(String greeting) {

       String text = "[" + getTimestamp() + "]:" + greeting;

       template.convertAndSend("/topic/greetings", text);

  }

}