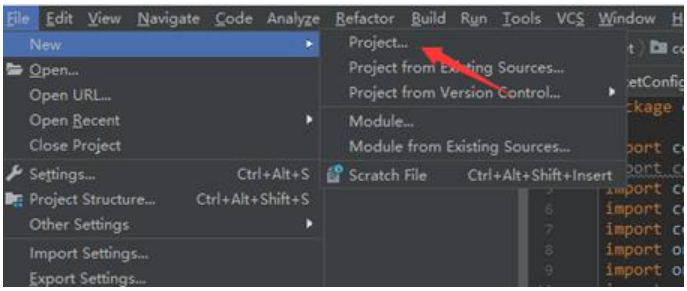
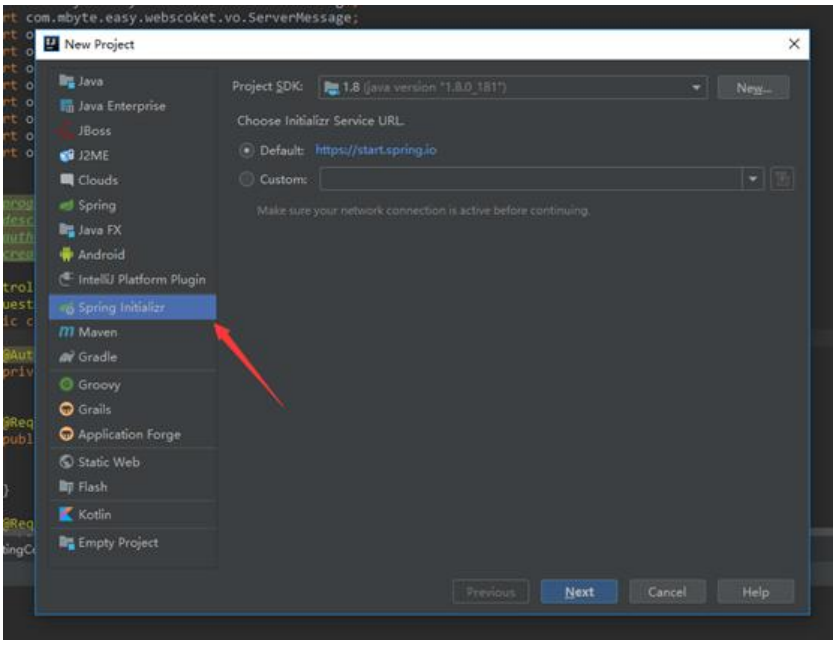
链滴

用 Spring boot 实现 web socket 应用

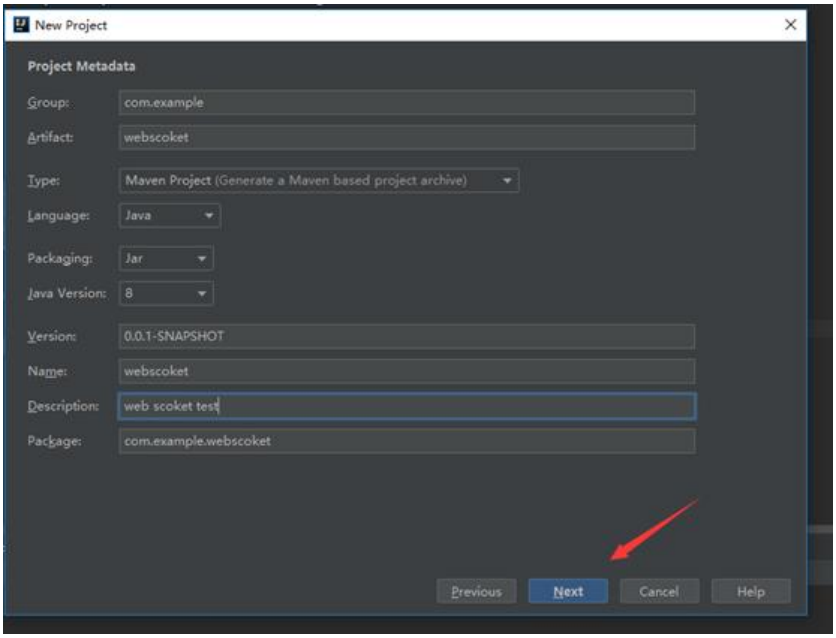
作者：byte2018
原文链接：https://ld246.com/article/1557219954726
来源网站：链滴

1、 首先通过idea创建一个Springboot项目，具体操作步骤如下：



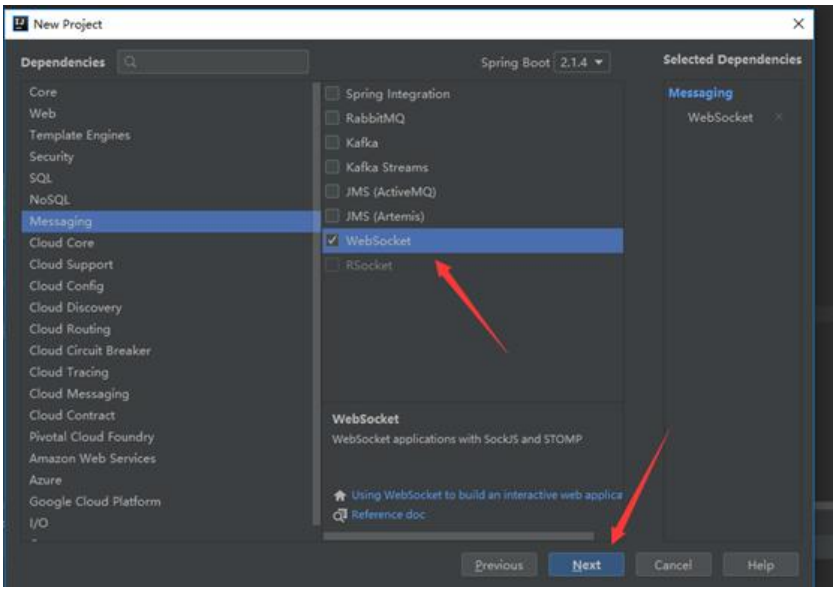


点击next

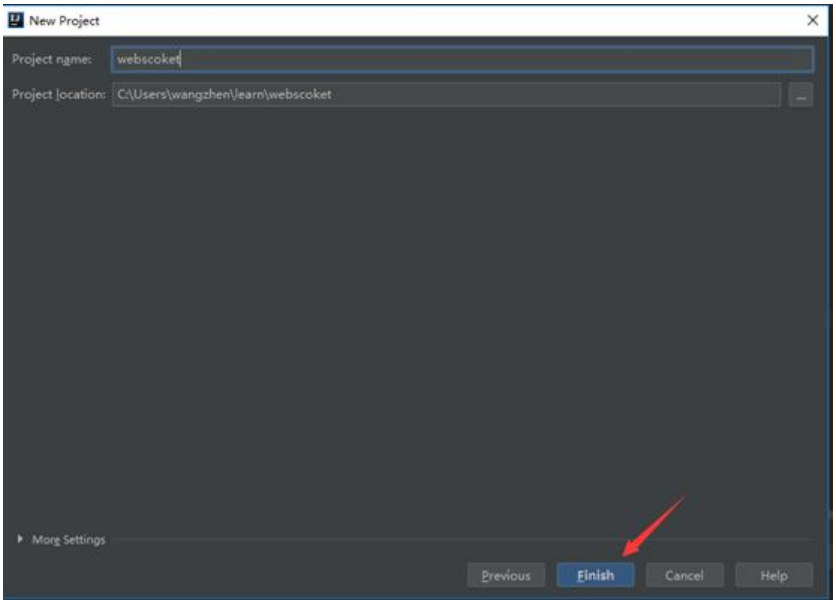


点击next

原文链接：用 Spring boot 实现 web socket 应用

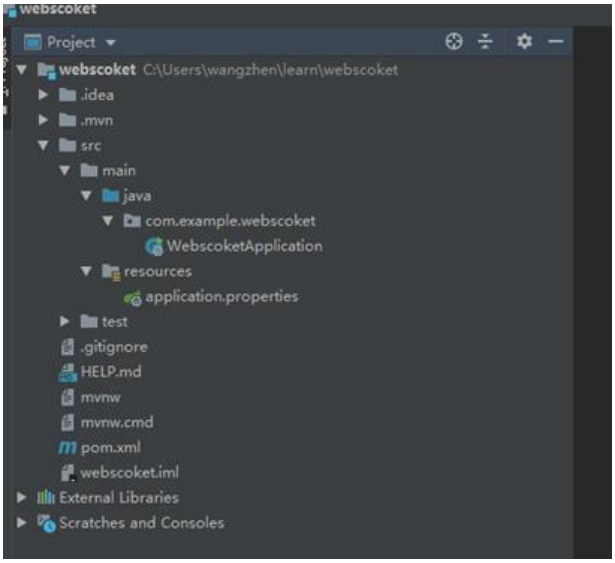


按照上图操作



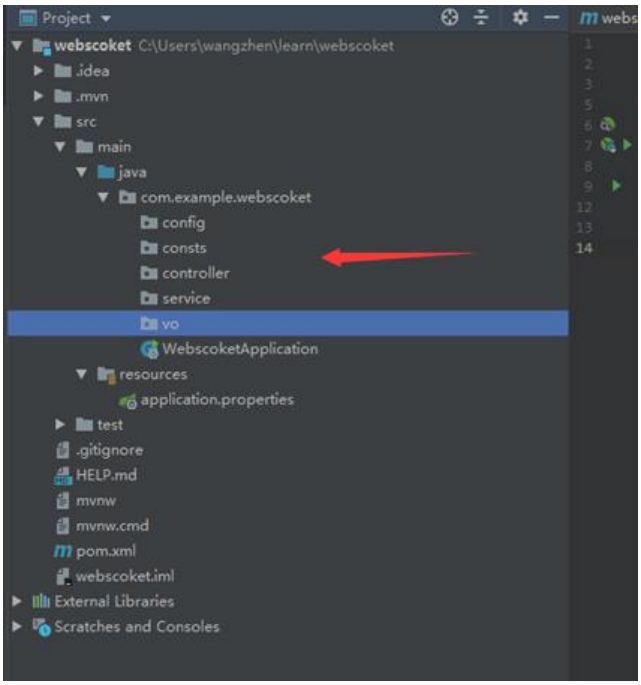
点击finish完成项目

2、 创建好的项目的目录结构



原文链接：用 Spring boot 实现 web socket 应用

3、 创建如下的目录结构

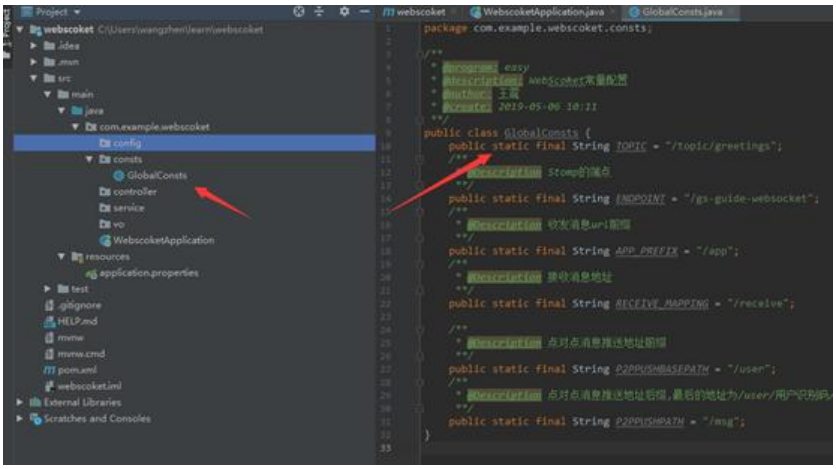


4、 先定义一个常量类备用，代码如下：

```
package com.example.webscoket.consts;


/**
 * @program: easy
 * @description: WebScoket常量配置
 * @author: 王震
 * @create: 2019-05-06 10:11
 **/
public class GlobalConsts {
    /**
     * @Description Stomp的端点
     **/
    public static final String ENDPOINT = "/gs-guide-websocket";
    /**
     * @Description 收发消息url前缀
     **/
    public static final String APP_PREFIX = "/app";
    /**
     * @Description 接收消息地址
     **/
    public static final String RECEIVE_MAPPING = "/receive";

    /**
     * @Description 点对点消息推送地址前缀
     **/
    public static final String P2PPUSHBASEPATH = "/user";
    /**
     * @Description 点对点消息推送地址后缀,最后的地址为/user/用户识别码/msg
     **/
```

```
    public static final String P2PPUSHPATH = "/msg";
}
```

代码结构如下：

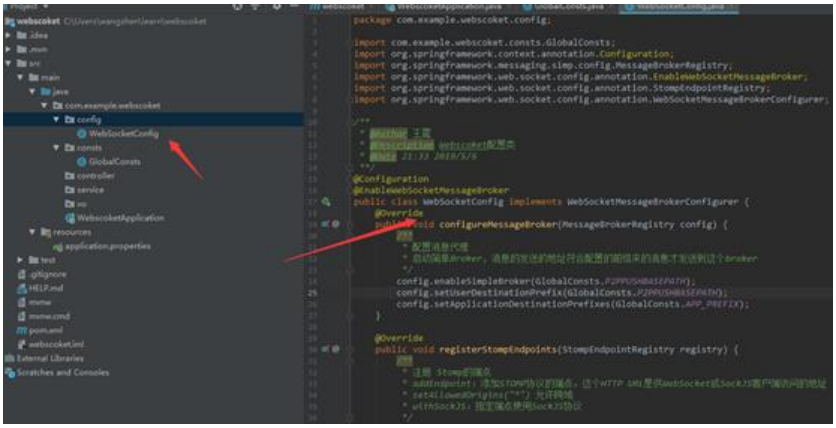


5、 创建配置类

```
package com.example.webscoket.config;


import com.example.webscoket.consts.GlobalConsts;
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigu
er;

/**
* @Author 王震
* @Description Webscoket配置类
* @Date 21:33 2019/5/6
**/
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
  @Override
  public void configureMessageBroker(MessageBrokerRegistry config) {
      /**
       * 配置消息代理
       * 启动简单Broker，消息的发送的地址符合配置的前缀来的消息才发送到这个broker
       */
      config.enableSimpleBroker(GlobalConsts.P2PPUSHBASEPATH);
      config.setUserDestinationPrefix(GlobalConsts.P2PPUSHBASEPATH);
      config.setApplicationDestinationPrefixes(GlobalConsts.APP_PREFIX);
  }

  @Override
  public void registerStompEndpoints(StompEndpointRegistry registry) {
      /**
       * 注册 Stomp的端点
```

```
     * addEndpoint：添加STOMP协议的端点。这个HTTP URL是供WebSocket或SockJS客户端访
的地址
     * setAllowedOrigins("*") 允许跨域
     * withSockJS：指定端点使用SockJS协议
     */
    registry.addEndpoint(GlobalConsts.ENDPOINT)
            .setAllowedOrigins("*")
            .withSockJS();
  }

}
```

程序结构如下：



6、 定义POJO类

接收客户端发过来的消息

```
package com.example.webscoket.vo;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * @program: easy
 * @description: 客户端发过来的消息
 * @author: 王震
 * @create: 2019-05-06 10:22
 **/
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ClientMessage {
    private int id;
    private String name;

}
```

服务端返回的消息

```
package com.example.webscoket.vo;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * @program: easy
 * @description: 服务端返回消息
 * @author: 王震
 * @create: 2019-05-06 10:25
 **/
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ServerMessage {
    private String content;
}
```

注意，项目采用了lombok和fastjson，需要在pom文件中加载相关依赖

```
    <dependency>

            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
<dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.54</version>
        </dependency>
```

7、 定义service接口和service实现类

```
package com.example.webscoket.service;

/**
 * @Author 王震
 * @Description 消息服务接口实现
 * @Date 21:46 2019/5/6
 **/
public interface SimpMessagingService {
    /**
     * @Author 王震
     * @Description 定义消息发送模板，点对点发送
     * @Date 21:48 2019/5/6
     * @Param [user, destination, payload]
     * @return void
     **/
    void sendP2P(String user, String destination, Object payload);
}


package com.example.webscoket.service.impl;
```

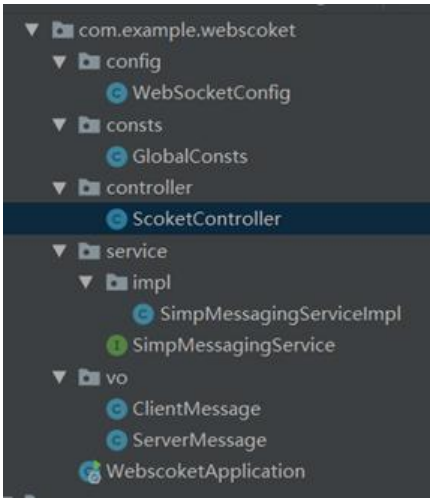
```
import com.example.webscoket.service.SimpMessagingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Service;

/**
 * @program: webscoket
 * @description: 消息服务接口实现
 * @author: 王震
 * @create: 2019-05-06 21:48
 **/
@Service
public class SimpMessagingServiceImpl implements SimpMessagingService {

    @Autowired
    private SimpMessagingTemplate template;
    /**
     * @Author 王震
     * @Description 定义消息发送模板，点对点发送
     * @Date 21:48 2019/5/6
     * @Param [user, destination, payload]
     * @return void
     **/
    @Override
    public void sendP2P(String user, String destination, Object payload) {
        template.convertAndSendToUser(user, destination, payload);
    }
}
```

8、 创建控制类，用于接收和响应消息

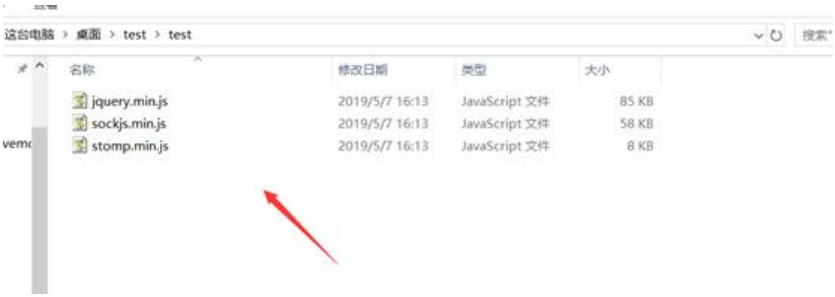
```
package com.example.webscoket.controller;


import com.alibaba.fastjson.JSON;
import com.example.webscoket.consts.GlobalConsts;

import com.example.webscoket.service.SimpMessagingService;
import com.example.webscoket.vo.ClientMessage;
import com.example.webscoket.vo.ServerMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.util.HtmlUtils;

/**
 * @program: easy
 * @description: webscoket测试controller
 * @author: 王震
 * @create: 2019-05-06 10:26
 **/
```

```
@Controller
public class ScoketController {

    @Autowired
    private SimpMessagingService simpMessagingService;

    /**
     * @Author 王震
     * @Description 接收和相应消息
     * @Date 16:42 2019/5/7
     * @Param [message]
     * @return void
     **/
    @MessageMapping(GlobalConsts.RECEIVE_MAPPING)
    public void scoket(ClientMessage message){
        simpMessagingService.sendP2P(message.getId() + "", GlobalConsts.P2PPUSHPATH, JSON
toJSON(new ServerMessage("Hello, " + HtmlUtils.htmlEscape(message.getName()) + "!")));
    }
}
```

9、 这样整个服务端的程序就搭建完成了，整体的代码结构如下图所示：



10、 现在编写测试程序，先下载相关的js库支持，如下图所示：



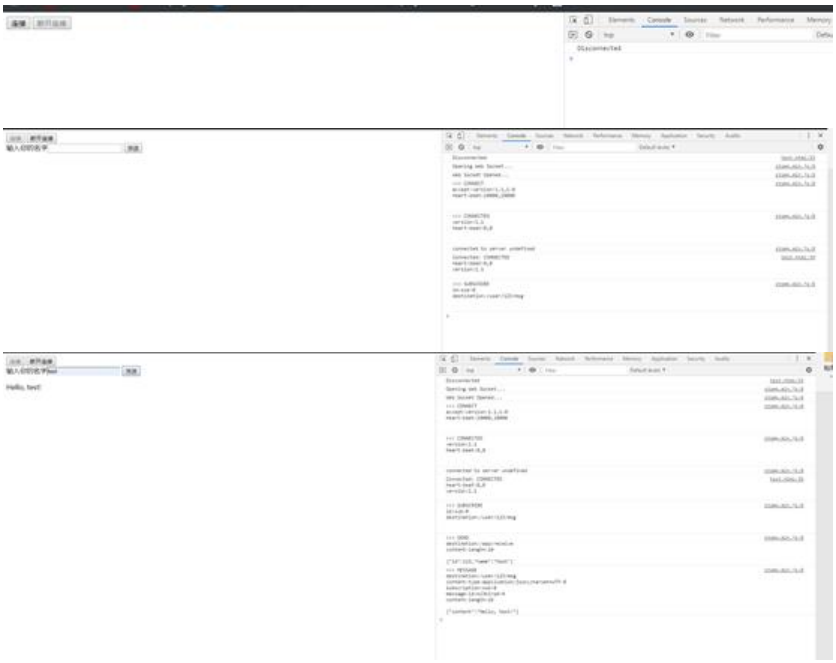
11、 编写测试页面，代码如下：

```
<!DOCTYPE html>

<html lang="en"><head><meta http-equiv="Content-Type" content="text/html; charset=U
F-8">
```

```
  <title>WebScoket测试</title>
  <script src="./test/jquery.min.js"></script>
  <script src="./test/sockjs.min.js"></script>
  <script src="./test/stomp.min.js"></script>
</head>
<body onload="disconnect()">
<div>
  <div>
     <button id="connect" onclick="connect();" disabled="">连接</button>
     <button id="disconnect" onclick="disconnect();">断开连接</button>
  </div>
  <div id="conversationDiv" style="visibility: visible;">
     <label>输入你的名字</label><input type="text" id="name">
     <button id="sendName" onclick="sendName();">发送</button>
     <p id="response"></p>
  </div>
</div>

<script type="text/javascript">
  var stompClient = null;

  function setConnected(connected) {
     document.getElementById('connect').disabled = connected;
     document.getElementById('disconnect').disabled = !connected;
     document.getElementById('conversationDiv').style.visibility = connected ? 'visible' : 'hidde
';
     $('#response').html();
  }

  function connect() {
     // websocket的连接地址，此值等于WebSocketMessageBrokerConfigurer中registry.addEnd
oint("/websocket-simple").withSockJS()配置的地址
     var socket = new SockJS('http://localhost:8080/gs-guide-websocket');
     stompClient = Stomp.over(socket);
     stompClient.connect({}, function(frame) {
        setConnected(true);
        console.log('Connected: ' + frame);
        // 客户端订阅消息的目的地址：此值BroadcastCtl中被@SendTo("/topic/getResponse")注
的里配置的值
        stompClient.subscribe('/user/' + 123 + '/msg', function(respnose){
           showResponse(JSON.parse(respnose.body).content);
        });
     });
  }


  function disconnect() {
     if (stompClient != null) {
        stompClient.disconnect();
     }
     setConnected(false);
     console.log("Disconnected");
  }
```

```
    function sendName() {
        var name = $('#name').val();
        // 客户端消息发送的目的：服务端使用BroadcastCtl中@MessageMapping("/receive")注解的
法来处理发送过来的消息
        stompClient.send("/app/receive", {}, JSON.stringify({ 'id': 123, 'name': name }));
    }

    function showResponse(message) {
        var response = $("#response");
        response.html(message + "\r\n" + response.html());
    }
</script>

</body></html>
```

12、 启动java程序，页面用浏览器打开，出现如下效果，证明搭建成功。



13、 项目下载

原文链接：用 Spring boot 实现 web socket 应用