# 基于WebSocket的web端IM即时通讯应用的开发

## 功能列表：

1、Web端的IM应用

2、支持上线、下线、实时在线提醒

3、单聊、群聊的建立

4、普通文字、表情、图片的传输（子定义富文本）

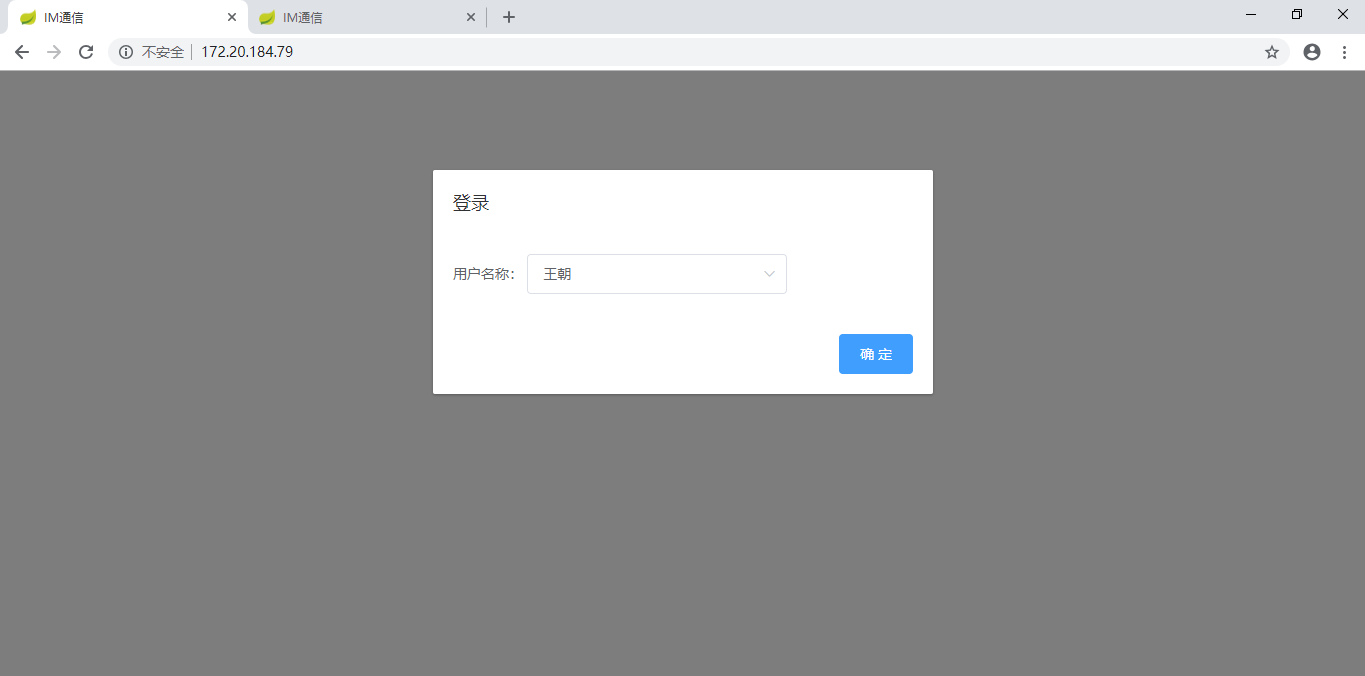
5、单人的顶级提醒，多对话的窗口的提醒

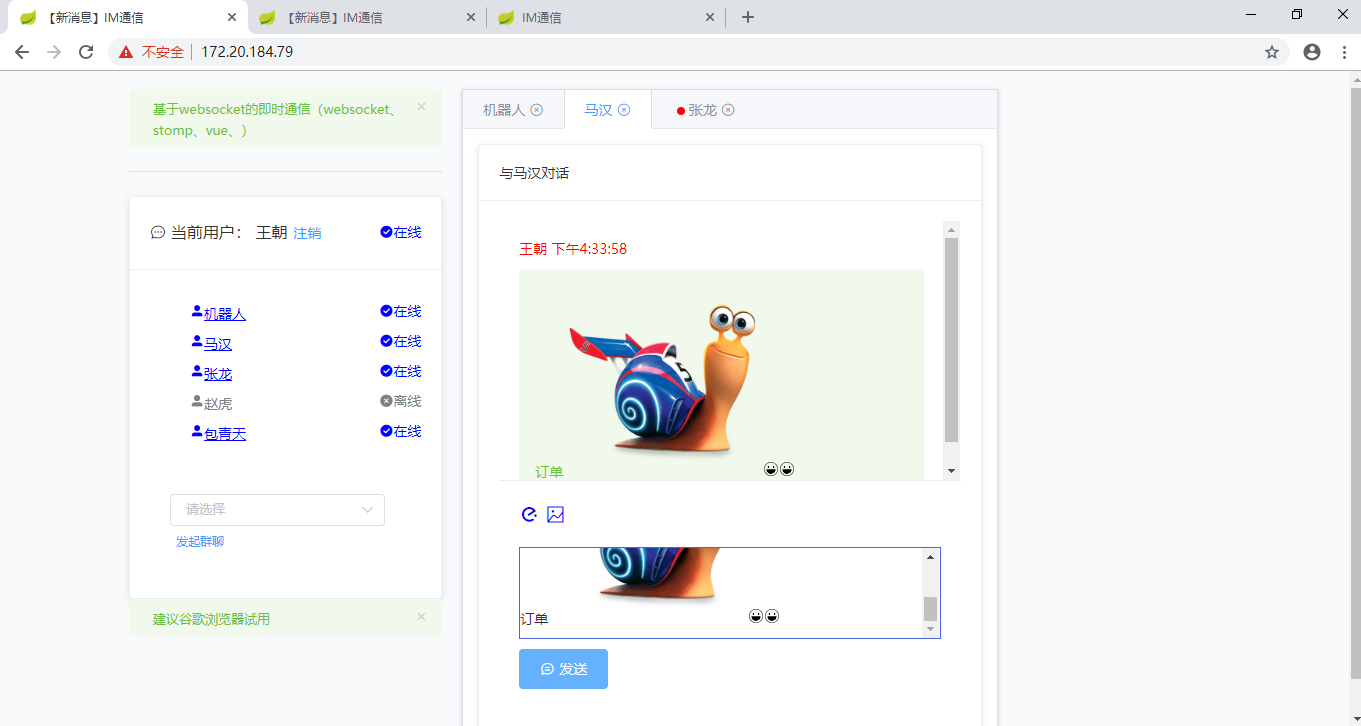
6、调用图灵机器人的自动回复演示

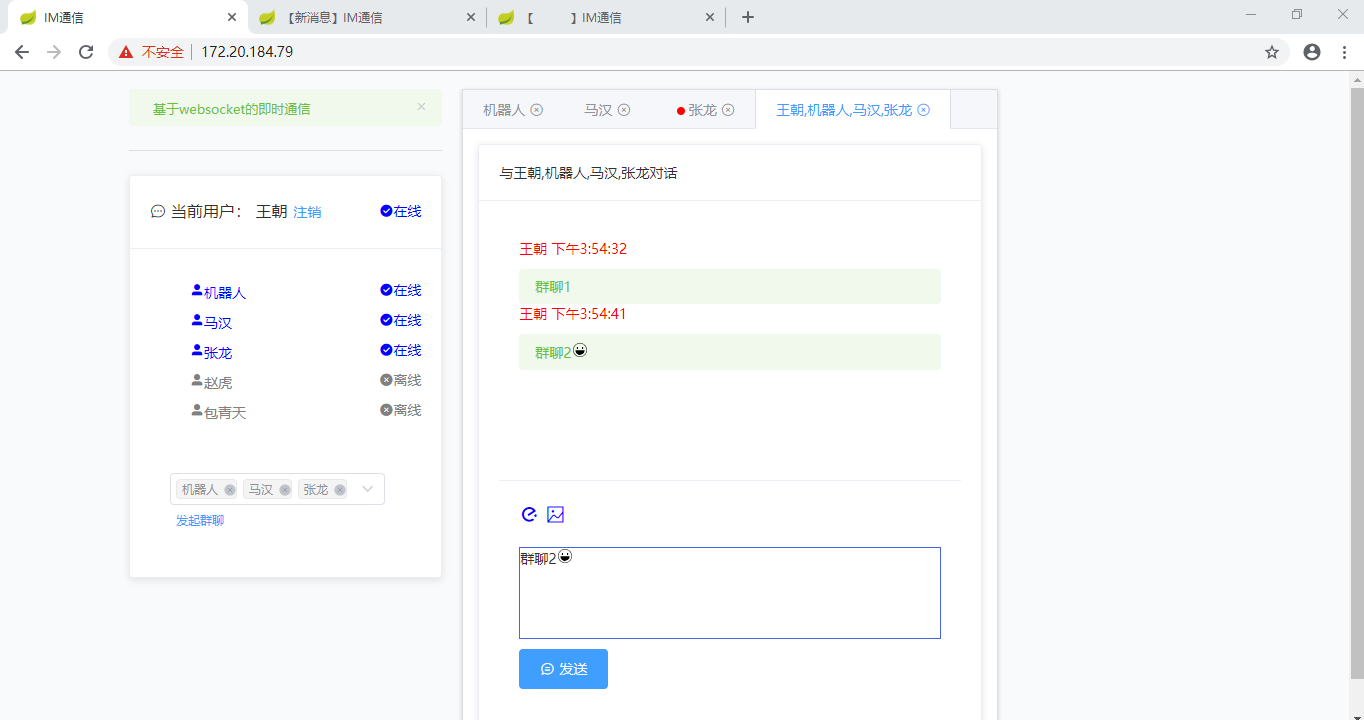
## 核心技术列表

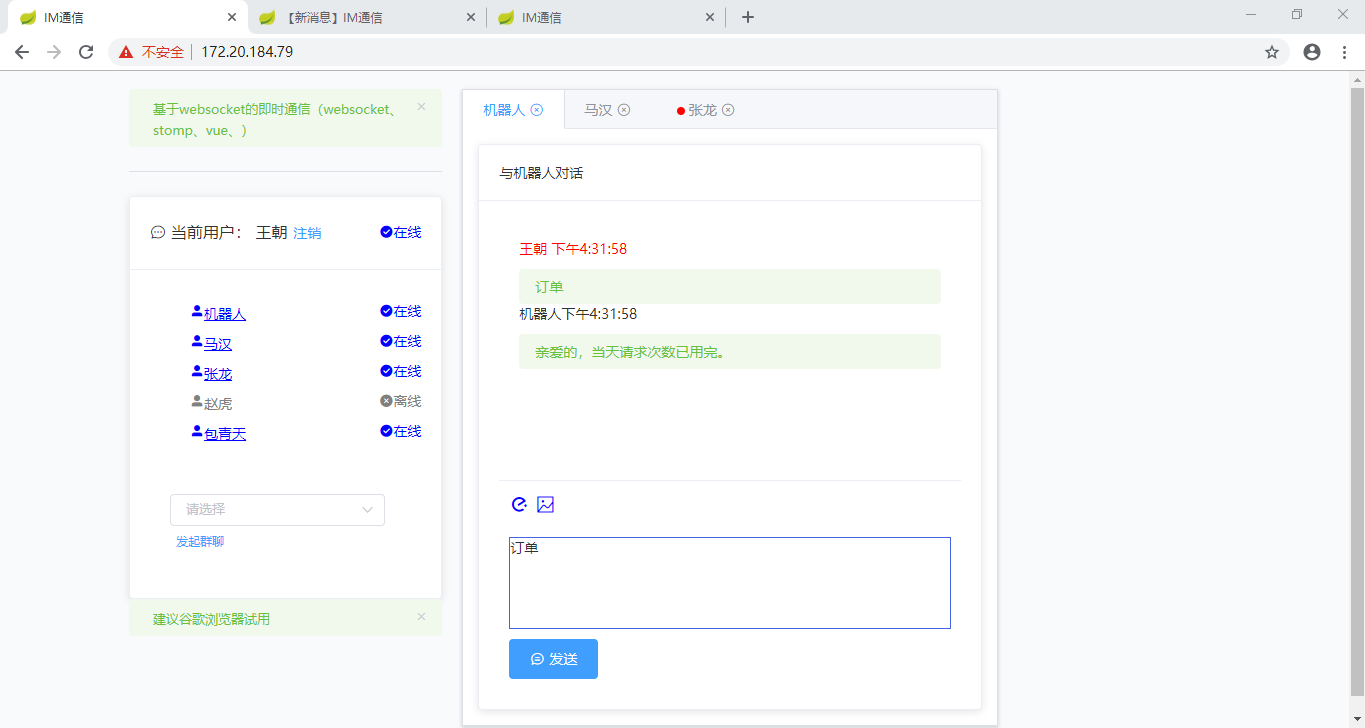
1. websocket、sockjs、stomp
2. 前端展示涉及的jquery、vue、elementUI、jquerybase64js
3. 后端springboot、jsoup、spring-security、spring-websocket

## 成果展示：









## 技术实现说明：

### Websocket部分

web端的IM应用，要想实现两个客户端的通信，必然要通过服务器进行信息的转发。例如A要和B通信，则应该是A先把信息发送给IM应用服务器，服务器根据A信息中携带的接收者将它再转发给B，同样B到A也是这种模式。

而要实现web端的实时通讯，websocket也是其中最好的方式，其他的协议如长轮询、短轮询、iframe数据、htmlfile等。

在实际开发中，我们通常使用的是一些别人写好的实时通讯的库，比如[socket.io](http://www.52im.net/forum.php?mod=viewthread&tid=190&ctid=15" \t "https://www.cnblogs.com/linsanshu/p/_blank)、[sockjs](https://github.com/sockjs/sockjs-client" \t "https://www.cnblogs.com/linsanshu/p/_blank)（我们本次使用了他，类似jquery，对其他即时通讯技术做了封装），他们的原理就是将上面（还有一些其他的如基于Flash的push）的一些技术进行了在客户端和服务端的封装，然后给开发者一个统一调用的接口。这个接口在支持websocket的环境下使用websocket，在不支持它的时候启用上面所讲的一些hack技术。

WebSocket是HTML5的一种新通信协议（ws协议），是一个消息架构，不强制使用任何特定的消息协议，它依赖于应用层解释消息的含义；与处在应用层的HTTP不同，WebSocket处在TCP上非常薄的一层，会将字节流转换为文本/二进制消息，因此，对于实际应用来说，WebSocket的通信形式层级过低，因此，可以在 WebSocket 之上使用 STOMP协议，来为浏览器 和 server间的 通信增加适当的消息语义。

STOMP(Simple Text-Orientated Messaging Protocol) 面向消息的简单文本协议。 同 HTTP 在 TCP 套接字上添加请求-响应模型层一样，STOMP 在 WebSocket 之上提供了一个基于帧的线路格式层，用来定义消息语义；

STOMP 源码http://cdn.bootcss.com/stomp.js/2.3.3/stomp.js，有兴趣的可以看一下能大致了解其原理和用法。

本例程序核心代码：

|  |
| --- |
| <!--TO 创建socket连接 并订阅相关频道--> **var** socket = **new** SockJS('/im-websocket'); stompClient = Stomp.over(socket); //设置stomp 控制台日志为不输出 stompClient.debug=**null**; stompClient.connect({}, **function** (frame) {  // 相当于连接 ws://localhost:8080/gs-guide-websocket/041/hk5tax0r/websocket hk5tax0r就是sessionid  console.log("正在连接",socket.\_transport.url);  //订阅通用私聊频道 群组也通过这里实现  stompClient.subscribe('/user/topic/private', **function** (greeting) {    }  );  //订阅用户上线下线的公共频道  stompClient.subscribe('/topic/userlist', **function** (greeting) {    }); },**function** errorCallBack (error) {  // 连接失败时（服务器响应 ERROR 帧）的回调方法   }); |

数据发送如下：

|  |
| --- |
| //第一个参数对应controller的 @MessageMapping注解 /app为后台定义的通用前缀 //第三个参数为内容字符串 stompClient.send("/app/private", {}, JSON.stringify(message));//发送服务器 |

对应服务端部分

|  |
| --- |
| #WebSocketConfig  @Configuration @EnableWebSocketMessageBroker public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {   public List<User> onlineUser=new ArrayList<User>();  @Autowired  private SimpMessagingTemplate template;  @Override  public void configureMessageBroker(MessageBrokerRegistry config) {  config.enableSimpleBroker("/topic");  config.setUserDestinationPrefix("/user");  config.setApplicationDestinationPrefixes("/app");  config.setCacheLimit(1048576);//大小 1M  }  @Override  public void registerStompEndpoints(StompEndpointRegistry registry) {  //注册的websocket接入点，前端链接的就是它  registry.addEndpoint("/im-websocket").withSockJS();  }   @Override  public void configureWebSocketTransport(final WebSocketTransportRegistration registration) {  //设置 文件缓冲 大小 1M  //如不设置文件稍微大一点就报错了  registration.setMessageSizeLimit(1048576);  registration.setSendBufferSizeLimit(1048576);  registration.addDecoratorFactory(new WebSocketHandlerDecoratorFactory() {  @Override  public WebSocketHandler decorate(final WebSocketHandler handler) {  return new WebSocketHandlerDecorator(handler) {  @Override  public void afterConnectionEstablished(final WebSocketSession session) throws Exception {  \*\*\*\*\*\*  }   @Override  public void afterConnectionClosed(WebSocketSession session, CloseStatus closeStatus)  throws Exception {  \*\*\*\*\*  }  };  }  });  super.configureWebSocketTransport(registration);  } } |
| #contoller  @MessageMapping("/private") public void privatechat(ImMessage message) throws Exception {  \*\*\*\*\*  template.convertAndSendToUser(message.getReceiver(),"/topic/private",message);  ////发送其订阅的频道  ///浏览器客户端，订阅了’/user/topic/private这条路径，  } |

其中@MessageMapping("bar") //@MessageMapping接收客户端消息

另外@SendTo("/topic/brocast") //@SendTo广播消息出去

@SendToUser("/topic/greetings")//发送对应人。

这两个注解可以用template.convertAndSendTo、template.convertAndSendToUser在代码实现。

### Spring-security部分

做了简单的登录验证，登录成功即在系统存有sessionid。结合上面的订阅，实际链接的后台地址会附加上sessionid，也就是httprequest中的登录授权信息即javax.security.Principal会被绑定到websocket的session中。即可以实现与指定登录人的双向链接。

### 页面展示部分

页面展示核心应用vue，vue不好实现的地方使用的是jquery。样式采用的elmentUI。

在开发这个im应用测试案例的时候。实现了一些前端效果。核心有用的列到下面，各位也许在后面的学习中能够用到。

1. vue兼容性的问题

因为本来不是webpack开发模式，属于直接引入js的普通HTML开发，如需要解决vue兼容性问题，可以引入

<https://cdn.bootcss.com/babel-polyfill/7.4.4/polyfill.min.js> 以解决。

1. vue用法（v-model、@click、v-html、v-for\v-if,v-bind）的应用，指令、过滤器、全局方法、watch的使用。其中指令用来实现div的默认焦点。全局方法用来代替过滤器，实现实时的消息内容base64解码。
2. 利用vue核心数据的双向绑定，不刷新显示更新。但数据设计为多层的json数组数据，当底层数据变化，vue不能自动检测到变化。需要进行手动检测。

代码-this.$forceUpdate();

1. 关于图片消息的用法

用图片上传按钮上，存在透明的fileinput文件，每次上传完，通过onchange方法，先检测文件大小、类型后，通过fileReader预览。这其中涉及富文本框焦点的问题、和base64转码的问题。另外当上传失败通过.val('')清空file，这样才能重新选择文件上传并成功触发onchange事件（值得了解，试验了半天才行）

|  |
| --- |
| **var** filec=$("#file"+index); **if** (filec&&filec.length>0) {  **var** fileList = filec[0].files;  **if**(!/image\/\w+/.test(fileList[0].type)) //判断获取的是否为图片文件  {  **this**.$message.error('请确保文件为图像文件');  //清空input 可以再次上传并触发onchange  filec.val('')  **return false**;  }  **if**(fileList[0].size>1048576){  **this**.$message.error('请确保图片不大于1M');  //清空input 可以再次上传并触发onchange  filec.val('')  **return false**;  }  fileReader.onload = **function** (e) {  // 获取文件的base64编码  **var** base64 = e.target.result  **var** image = **new** Image();  image.src = base64;  image.onload = **function**() {  //文件像素过大，调整为稍小的  **var** newW="";**var** newH="";  **if**(**this**.width>**this**.height&&**this**.width>200){  newW=200;  newH=200/**this**.width\***this**.height;  }  **if**(**this**.width<=**this**.height&&**this**.height>200){  newH=200;  newW=200/**this**.height\***this**.width;  }  **var** h = '<img src=' + base64 + ' width='+newW+' height='+newH+'>';  \_insertimg(h, index)//插入到富文本对应的位置  };  }  fileReader.readAsDataURL(fileList[0]); |

1. 关于富文本在指定焦点位置插入数据的问题，后续可以考虑baidueditor等成熟产品。

当前富文本主要利用了html5的属性contenteditable解决的

|  |
| --- |
| 具体可以查看\_insertimg方法 |

1. 在实现上述富文本的时候，类似插入表情、选择图片的时候，只要点击屏幕，则当前页面焦点即转移，影响实际插入的位置。所以需要设置这些按钮点击的时候屏蔽默认效果。

[一个是按钮的@click.prvent。另外可以通过](mailto:一个是按钮的@click.prvent。另外可以通过)下面的方法解决

|  |
| --- |
| document.addEventListener("mousedown", **function**(e){  **if**(e.target.id=="emoijT"){  e.preventDefault()  } }, **false**); |

1. 因为当前发送的消息是带html标签的富文本信息，为避免传输的问题，将内容进行base64转码，消息被接收后再转码回来。

|  |
| --- |
| **var** stompClient = **null**; //防止乱码 $.base64.utf8encode = **true**;  $.base64.btoa(thisMessage);//使用插件base64编码  //解码 $.base64.atob(c, **true**); |

1. 当前案例不仅实现了多对话窗口，隐藏的对话提醒。也实现了当前人的浏览器标题提醒。

|  |
| --- |
| **var** pageMessage = {  time: 0,  title: document.title,  timer: **null**,  // 显示新消息提示  show: **function** () {  **var** title = pageMessage.title.replace("【　　　】", "").replace("【新消息】", "");  // 定时器，设置消息切换频率闪烁效果就此产生  pageMessage.timer = setTimeout(**function** () {  pageMessage.time++;  pageMessage.show();  **if** (pageMessage.time % 2 == 0) {  document.title = "【新消息】" + title  }  **else** {  document.title = "【　　　】" + title  }  ;  }, 600);  **return** [pageMessage.timer, pageMessage.title];  },  // 取消新消息提示 v  clear: **function** () {  clearTimeout(pageMessage.timer);  document.title = pageMessage.title;  } }; |

1. 关于机器人自动对话，目前使用jsoup调用的远程接口，由其返回答案。虽然是免费接口，但是一天不能调用多次。

|  |
| --- |
| String url = "http://www.tuling123.com/openapi/api"; //请填写自己的key String userid="454995"; String post = "{\"key\": \"646d321c227045a69253fd07d8703840\",\"info\": \""+message.getContent()+"\",\"userid\":\""+userid+"\"}"; String body = Jsoup.*connect*(url).method(Connection.Method.*POST*)  .requestBody(post)  .header("Content-Type", "application/json; charset=utf-8")  .ignoreContentType(true).execute().body(); |