# 为什么要使用WebSocket

在HTTP协议中，所有的请求都是由客户端发起的，服务端对客户端的请求进行响应，服务端无法主动向客户端推送消息，但是在一些即时应用中，又不可避免地需要服务端主动向客户端推送消息。为此，WebSocket因此而生。

# WebSocket简介

WebSocket是一种在单个TCP连接上进行全双工通信的协议，已被W3C定位标准；

使用WebSocket可以使得客户端和服务器之间的数据交换变得更加简单，它允许服务端主动向客户端推送数据；

在WebSocket协议中，浏览器和服务器只需要完成一次握手，两者之间就可以直接创建持久性的连接，并进行双向数据传输；

WebSocket使用了HTTP/1.1的协议升级特性，一个WebSocket请求首先使用非正常的HTTP请求已特定的模式访问一个URL，这个URL有两种模式：ws和wss，分别对应HTTP协议中的HTTP和HTTPS，在请求头中有一个ConnectionUpgrader字段，表示客户端想要对协议进行升级，另外还有一个Upgrade：websocket字段，表示客户端想要将请求协议升级为WebSocket协议，这两个字段共同告诉服务器要将连接升级为WebSocket这样一种全双工协议，如果服务器同意协议升级，那么在握手完成之后，文本消息或者其他二进制消息就可以同时在两个方向上进行发送，而不需要关闭和重建连接。此时的客户端和服务端关系是对等的，它们可以互相向对方主动发送消息。

WebSocket特点如下：

1. WebSocket使用时需要先创建连接，这使得WebSocket称为一种有状态的协议，在之后的通信过程中可以省略部分状态信息（例如身份认证等）
2. WebSocket连接在80（ws）端口或者443（wws）端口上创建，与HTTP使用的端口相同，这样，基本上所有的防火墙连接都不会阻止WebSocket连接；
3. WebSocket使用HTTP协议进行握手，因此它可以自然而然地集成到网络浏览器和HTTP服务器中，而不需要使用额外的成本；
4. 心跳消息（ping和pong）将被反复的发送，进而保持WebSocket连接一直处于活跃状态；
5. WebSocket关闭时将发送一个特殊的关闭消息；
6. WebSocket支持跨域，可以避免Ajax的限制；
7. HTTP规范要求浏览器将并发连接数限制为每个主机名两个连接，但是当我们使用WebSocket的时候，当握手成功，该限制将不复存在，因为此时的连接不再是HTTP连接了；
8. WebSocket协议支持协议扩展，用户可以扩展协议，实现部分自定义的子协议；
9. 更好的二进制支持以及更好的压缩效果；

# WebSocket使用场景

1. 在线股票网站；
2. 即时聊天；
3. 多人在线游戏；
4. 系统性能实时监控；

当然，不限于此；

# Spring Boot整合WebSocket

## 实现消息群发

1. 添加依赖



spring-boot-starter-websocket依赖时WebSocket相关依赖，其他的都是前端库，使用jar包的形式对这些前端库进行统一管理，使用webjar添加到项目中的前端库，在Spring Boot项目中，默认添加了静态资源过滤，因此可以直接使用；

1. 配置WebSocket

Spring框架提供了基于WebSocket的STOMP支持，STOMP是一个简单的可互操作的协议，通常被用于通过中间件服务器在客户端之间进行异步消息传递。WebSocket配置如下：



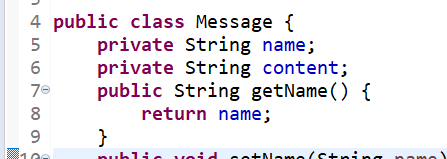
该自定义配置类使用@EnableWebSocketMessageBroker注解开启WebSocket消息代理；

config.enableSimpleBroker(“/topic”)表示设置消息代理的前缀，即如果消息的前缀是”/topic”，就会将消息转发给消息代理（broker），再由消息代理将消息广播给当前连接的客户端；

config.setApplicationDestinationPrefixes(“/app”)表示配置一个或多个前缀，通过这些前缀过滤出需要被注解方法处理的消息，例如，前缀为”/app”的destination可以通过@MessageMapping注解的请求方法处理，而其他destination，例如”/topic”将被直接脚背broker（消息代理）处理；

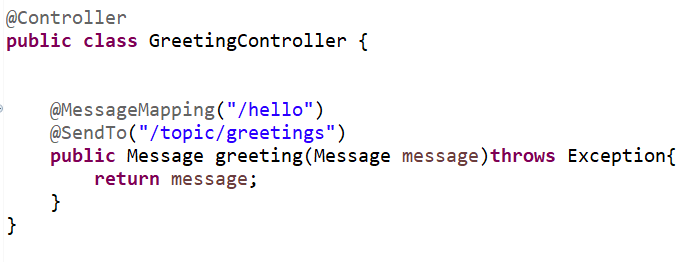
registry.addEndpoint(“/chat”).withSockJS( )则表示定义一个前缀为“/chat”的endPoint，并开启sockjs支持，sockjs可以解决浏览器对WebSocket的兼容性问题，客户端将通过这里配置的URL来建立WebSocket连接；

1. 创建一个实体类，表示消息



name表示消息名称，content表示消息内容

1. 定义Controller用来实现消息的处理

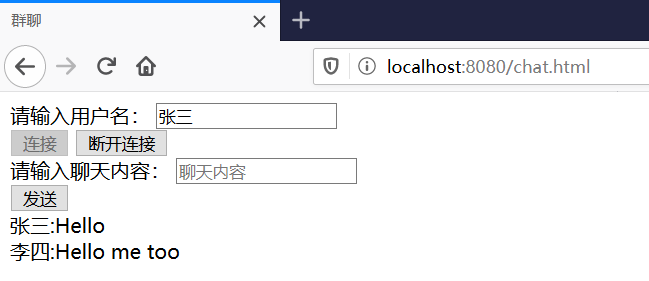


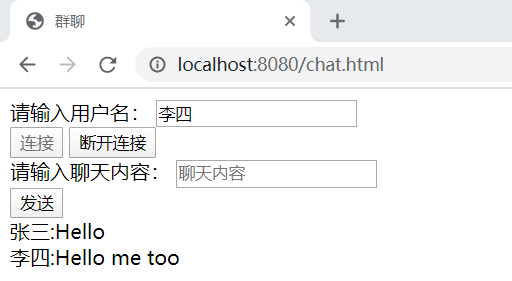
1. 在resources/static目录下构建聊天视图



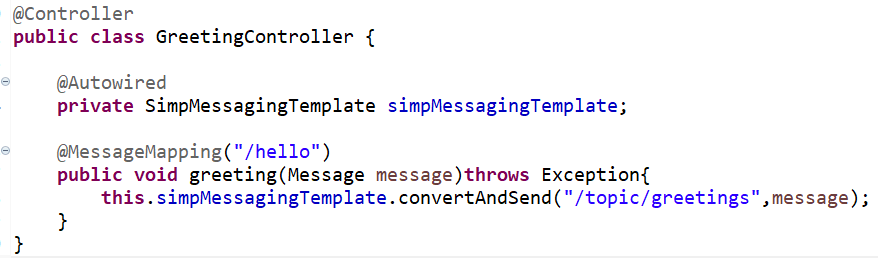
第6~8行引入了外部的js库，这些前面以及通过添加依赖的方式添加进来了，第9行引入的是一个自定义的JS（试着分析自定义JS文件（即app.js）中的代码）。

1. 启动项目，打开两个浏览器，分别访问chat.html，进行测试。如下：





1. 在上面的消息发送中使用的是@SendTo注解，该注解将方法处理过的消息转发到broker（消息代理），再由broker进行消息广播。除了使用@SendTo注解外，Spring还提供了SimpMessagingTemplate类来让开发者更加灵活地发送消息，如使用SimpMessagingTemplate可以替代@SendTo注解，如下：

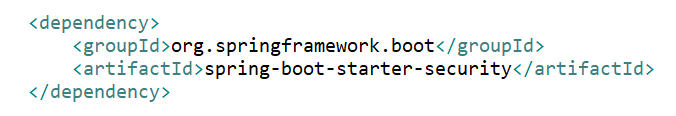


在Spring Boot中，SimpMessgingTemplate已经配置好，开发者直接注入使用即可

## 消息点对点发送

WebSocket也可以实现单聊，即消息点对点发送。

1. 既然是点对点发送，就应该有用户的概念，因此，在上一节的基础上，加入Spring Security依赖，如下：

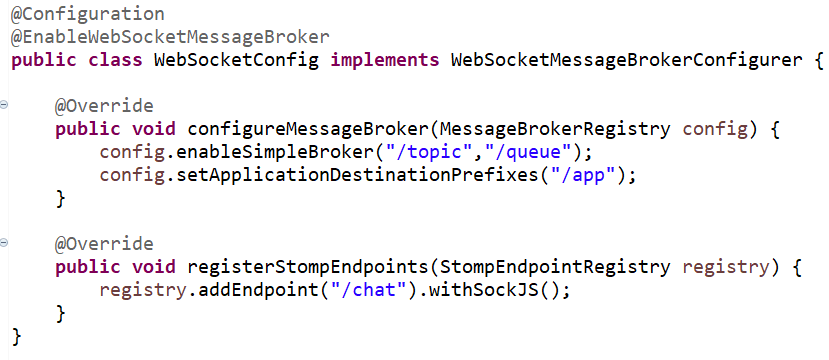


1. 对Spring Security进行配置，添加两个用户，同时配置所有地址都认证后才能访问，如下：

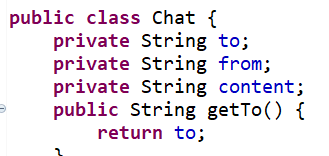


在这里配置了两个用户，admin和user，以及这两个用户是什么角色；对于其他不明白的地方，请参考第10章；

1. 编写WebSocket配置类，如下：



1. 编写实体类，如下：



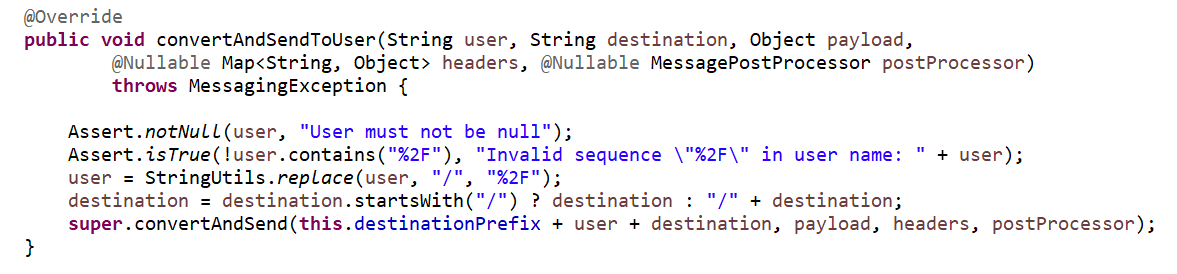
to表示消息发送的目标用户，from表示谁发送的消息，content表示消息内容；

1. 编写Controller，如下：



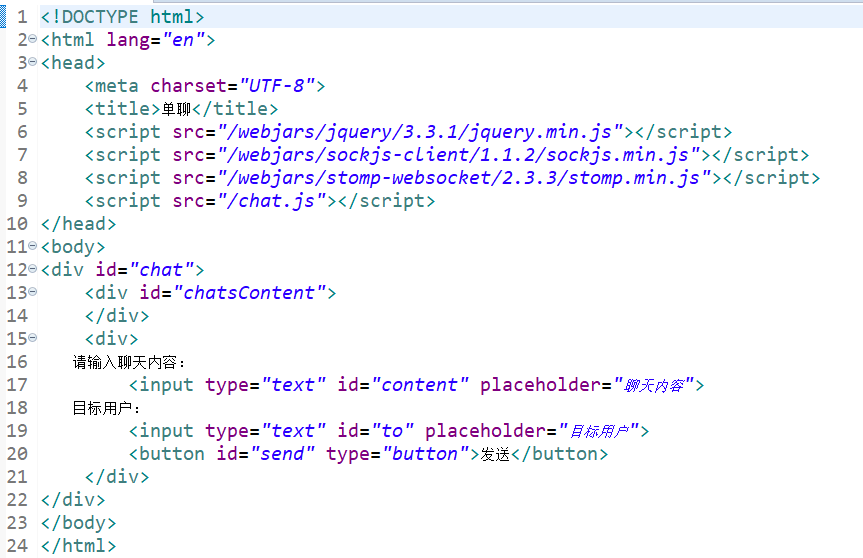
群发消息依然使用@SendTo注解实现，在这里的点对点消息使用SimpMessageTemplate来实现；

chat这个请求处理方法上使用了@MessageMapping(“/chat”)注解表示来自“/app/chat”路径的消息将被chat方法处理。chat方法的第一个参数Principal可以用来获取当前登录用户的信息，第二个参数则是客户端发送来的消息；在chat方法中，首先获取当前用户的用户名，设置给chat对象的from属性，再将消息发送出去，发送的目标用户就是chat对象的to属性值，发送消息使用的方法是convertAndSendToUser，调用该方法时会调用convertAndSend方法，并对消息路径做了处理，部分源码如下：



这里的destinationPrefix的默认值是“/user”，也就是说消息的最终发送路径是“/user/用户名/queue/chat”。

1. 创建在线聊天页面，如下：



引入的chat.js是一个自定义的javascript文件，其作用请自行分析；

1. 启动项目，打开两个浏览器分别访问onlinechat.html，首先会跳到登录页，使用先前配置的两个用户admin/123和liu/123登录，登录成功后，就可以在线聊天了。如下：

