xNPE Lv2
2018年09月19日 阅读 2432

关注

WebSocket的故事（五）—— Springboot中，实现网页聊天室之自定义消息代理

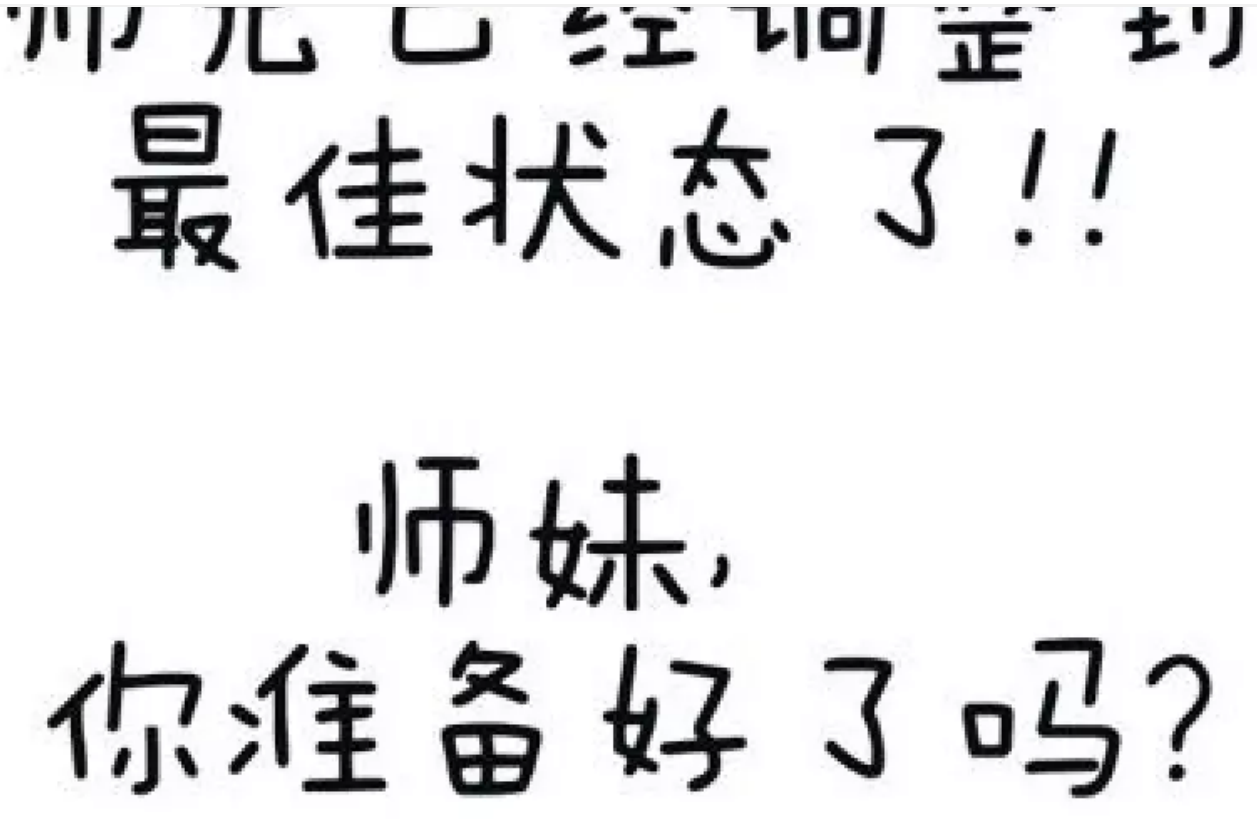
概述

WebSocket的故事系列计划分五大篇六章，旨在由浅入深的介绍WebSocket以及在Springboot中如何快速构建和使用WebSocket提供的能力。本系列计划包含如下几篇文章：

第一篇，什么是WebSocket以及它的用途
第二篇，Spring中如何利用STOMP快速构建WebSocket广播式消息模式
第三篇，Springboot中，如何利用WebSocket和STOMP快速构建点对点的消息模式(1)
第四篇，Springboot中，如何利用WebSocket和STOMP快速构建点对点的消息模式(2)
第五篇，Springboot中，实现网页聊天室之自定义WebSocket消息代理
第六篇，Springboot中，实现更灵活的WebSocket

本篇的主线

本篇将通过一个接近真实的网页聊天室Demo，来详细讲述如何利用WebSocket来实现一些具体的产品功能。本篇将只采用WebSocket本身，不再使用STOMP等这些封装。亲自动手实现消息的接收、处理、发送以及WebSocket的会话管理。**这也是本系列的最重要的一篇，不管你们激不激动，反正我是激动了**。下面我们就开始。

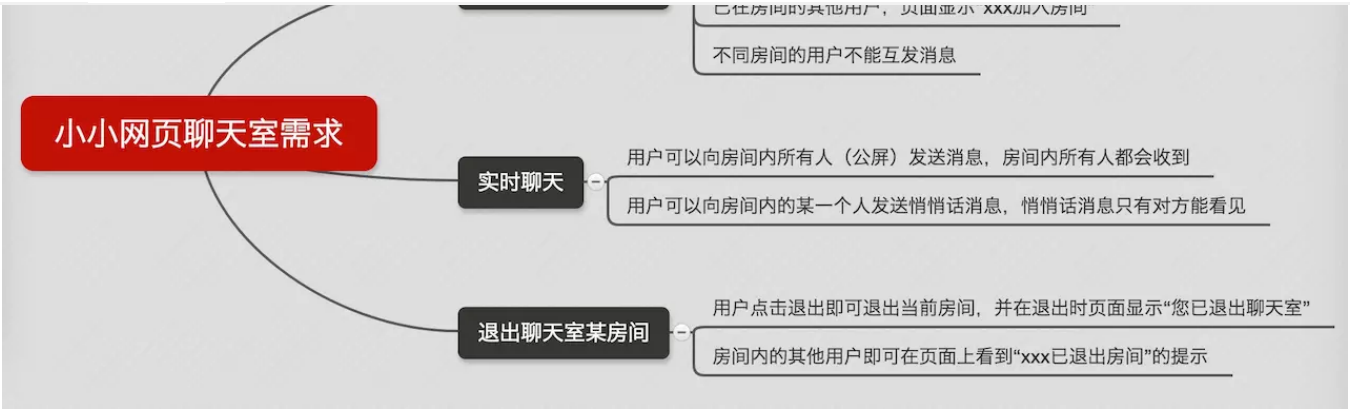


本篇适合的读者

想了解如何在Springboot上自定义实现更为复杂的WebSocket产品逻辑的同学以及各路有志青年。

小小网页聊天室的需求

为了能够目标明确的表达本文中所要讲述的技术要点，我设计了一个小小聊天室产品，先列出需求，这样大家在看后面的实现时能够知其所以然。



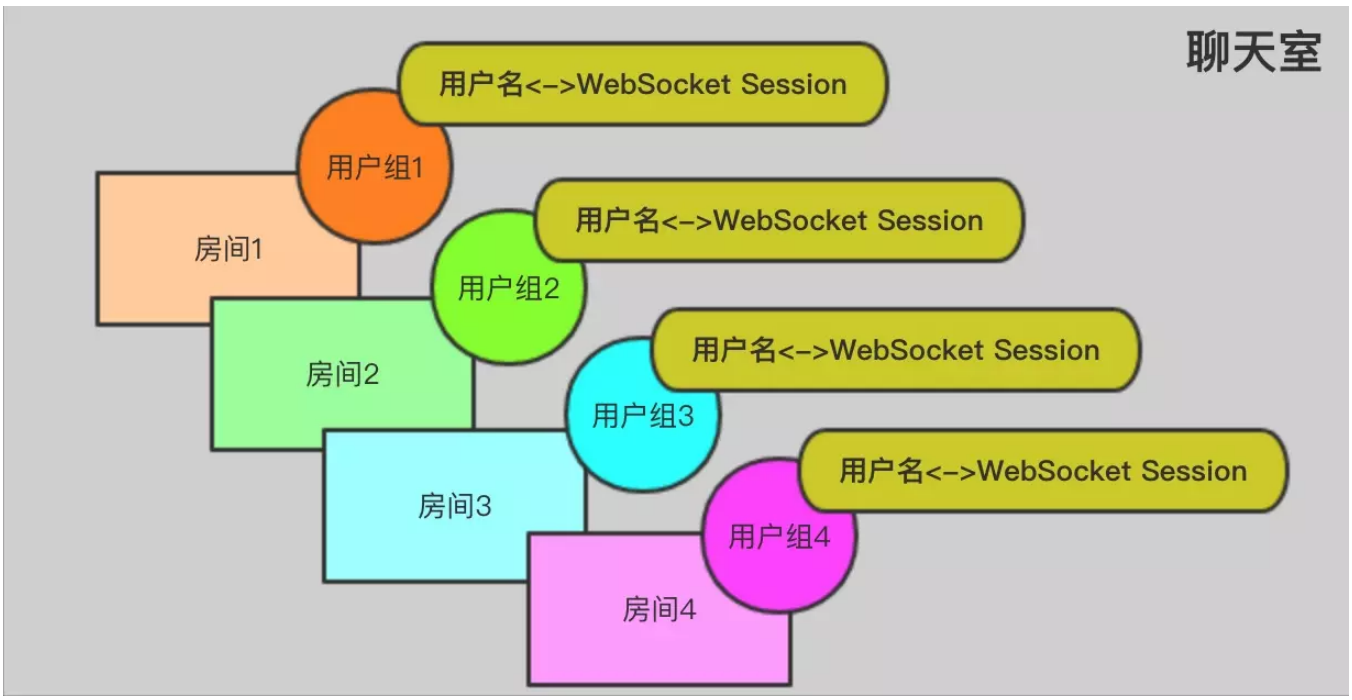
以上就是我们本篇要实现的需求。简单说，就是：

用户可加入，退出某房间，加入后可向房间内所有人发送消息，也可向某个人发送悄悄话消息。

需求分析和设计

设计用户存储

很容易想到我们设计的主体就是用户、会话和房间，那么在用户管理上，我们就可以用下面这个图来表示他们之间的关系：

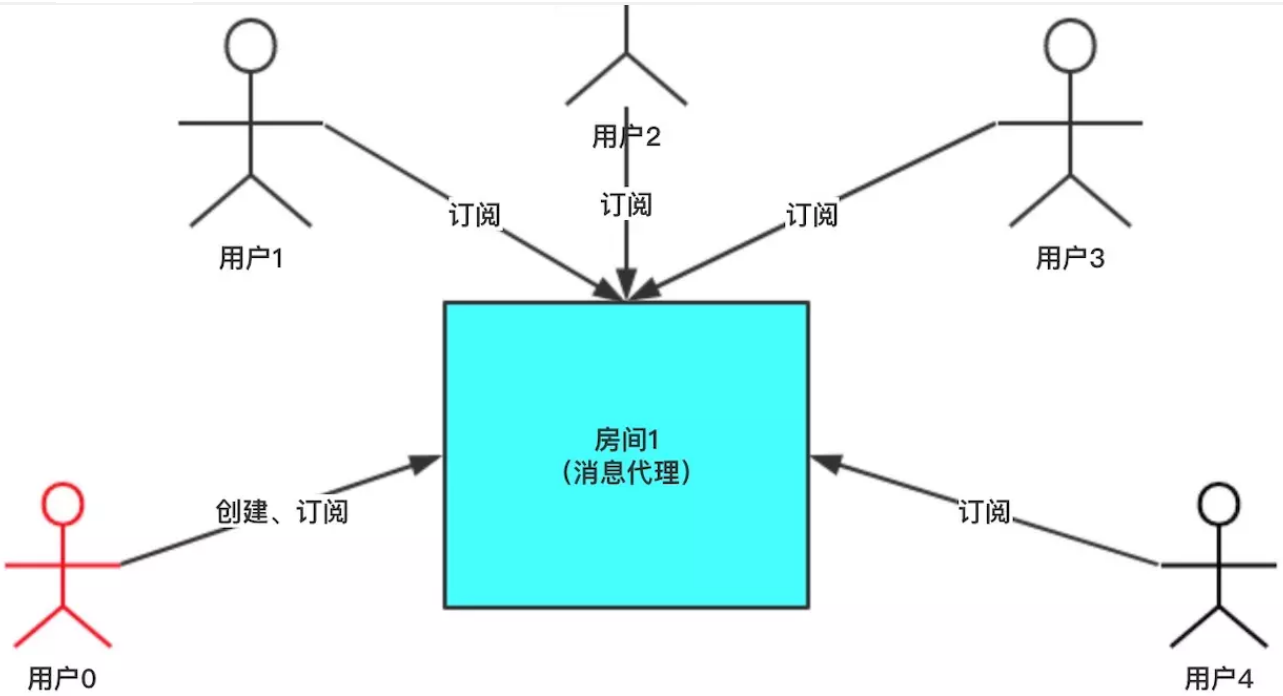


这样我们就可以用一个简单的Map来存储 房间<->用户组 这样的映射关系，在用户组内我们再使用一个Map来存储 用户名<->会话Session 这样的映射关系（假设没有重名）。这样，我们就解决了房间和用户组、用户和会话，这些关系的存储和维护。

有兄弟看到这说了，“你讲这么半天了，跟之前几篇讲的什么STOMP，什么消息代理，有毛线的关系？”大兄弟你先消消气，我们学STOMP，学消息代理，学点对点消息，重要的是学思想，你说对不？下面我们就用上了。

当用户加入到某房间之后，房间里有任何风吹草动，即有人加入、退出或者发公屏消息，都会“通知”给该用户。到此，我们就可以将创建房间理解成“**创建消息代理**”，将用户加入房间，看成是对房间这个“**消息代理**”的一个“**订阅**”，将用户退出房间，看成是对房间这个“**消息代理**”的一个“**解除订阅**”。

那么，第一个加入房间的人，我们定义为“**创建房间**”，即创建了一个消息代理。为了好理解，上图：



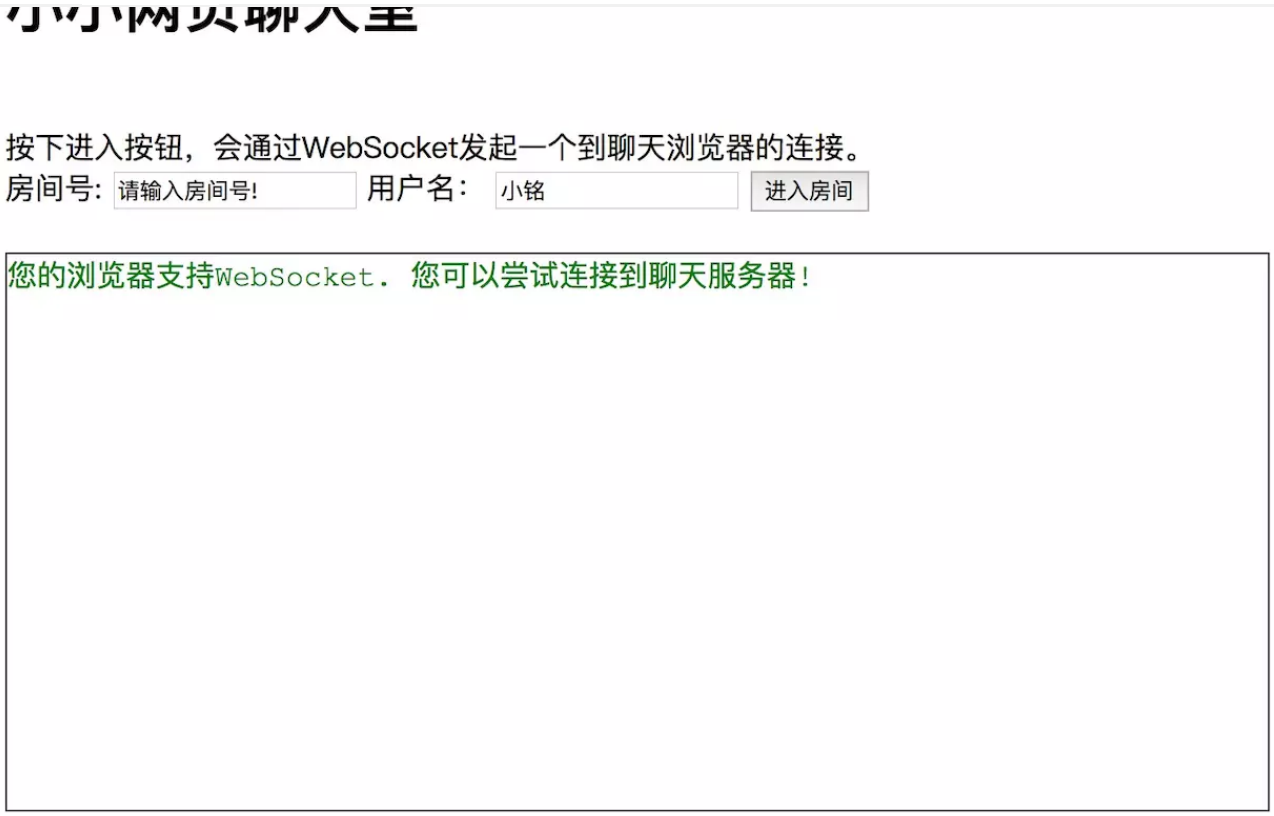
其中红色的小人表示第一个加入房间的用户，即创建房间的人。当某用户发送消息时，如果选择将消息发送给聊天室的所有人，即相当于在房间里发送了一个广播，所有订阅这个房间的用户，都会收到这个广播消息；如果选择发送悄悄话，则只将消息发送给特定用户名的用户，即点对点消息。

总结一下我们要实现的要点：

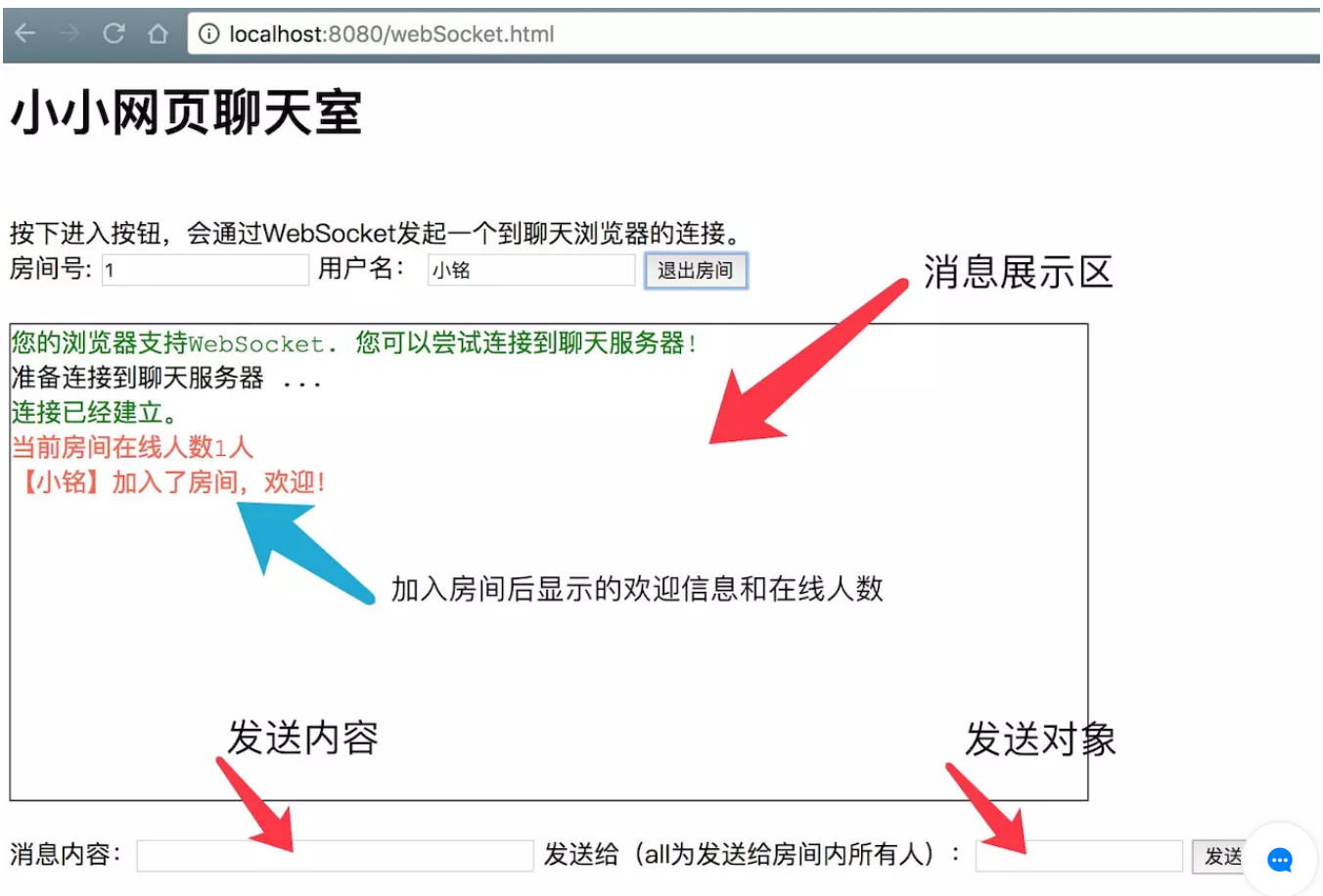
- **用户存储，即用户，房间，会话之间的关系和对象访问方式。**
- **动态创建消息代理（房间），并实现用户对房间的绑定（订阅）。**
- **单独发送给某个用户消息的能力。**

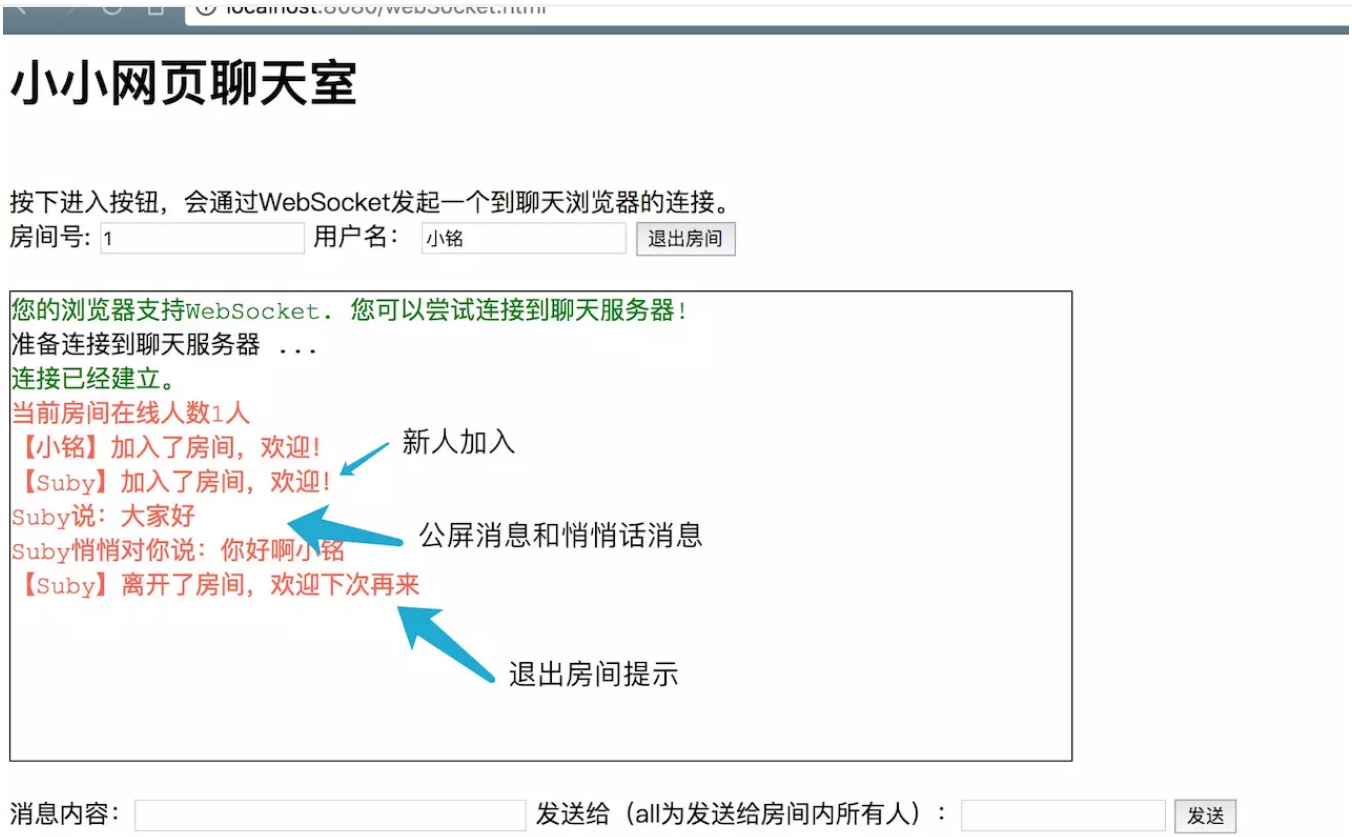
大体设计就到此为止，还有一些细节，我们先来看一下演示效果，再来看通过代码来讲解实现。

聊天室效果展示



用浏览器打开客户端页面后，展示输入框和加入按钮。输入**房间号1**和用户名**小铭**，**点击进入房间**。





当有其他人加入或退出房间时，展示通知信息。可以发送公屏消息和私聊消息。

下面就让我们看一下这些主要功能如何来实现吧。

代码实现

按照我们上述的设计，我会着重介绍重点部分的代码设计和技术要点。

服务端实现

1. 配置WebSocket

```java
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {
    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        registry.addHandler(new MyHandler(), "/webSocket/{INFO}").setAllowedOrigins("*")
                .addInterceptors(new WebSocketInterceptor());
```

要点解析：

- 注册 `WebSocketHandler` （ `MyHandler` ），这是用来处理WebSocket建立以及消息处理的类，后面会详细介绍。
- 注册 `WebSocketInterceptor` 拦截器，此拦截器用来在客户端向服务端发起初次连接时，记录客户端拦截信息。
- 注册WebSocket地址，并附带了 `{INFO}` 参数，用来注册的时候携带用户信息。

以上都会在后续代码中详细介绍。

2. 实现握手拦截器

```java
public class WebSocketInterceptor implements HandshakeInterceptor {
    @Override
    public boolean beforeHandshake(ServerHttpRequest serverHttpRequest, ServerHttpResponse
        if (serverHttpRequest instanceof ServletServerHttpRequest) {
            String INFO = serverHttpRequest.getURI().getPath().split("INFO=")[1];
            if (INFO != null && INFO.length() > 0) {
                JSONObject jsonObject = new JSONObject(INFO);
                String command = jsonObject.getString("command");
                if (command != null && MessageKey.ENTER_COMMAND.equals(command)) {
                    System.out.println("当前session的ID="+ jsonObject.getString("name"));
                    ServletServerHttpRequest request = (ServletServerHttpRequest) serverHtt
                    HttpSession session = request.getServletRequest().getSession();
                    map.put(MessageKey.KEY_WEBSOCKET_USERNAME, jsonObject.getString("name")
                    map.put(MessageKey.KEY_ROOM_ID, jsonObject.getString("roomId"));
                }
            }
        }
        return true;
    }
}
```

要点解析：

- `HandshakeInterceptor` 用来拦截客户端第一次连接服务端时的请求，即客户端连接 `/webSocket/{INFO}` 时，我们可以获取到对应 `INFO` 的信息。
- 实现 `beforeHandshake` 方法，进行用户信息保存，这里我们将用户名和房间号保存到 `Sessi` 上。

```
public class MyHandler implements WebSocketHandler {

    //用来保存用户、房间、会话三者。使用双层Map实现对应关系。
    private static final Map<String, Map<String, WebSocketSession>> sUserMap = new HashMap<

    //用户加入房间后，会调用此方法，我们在这个节点，向其他用户发送有用户加入的通知消息。
    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws Exception {
        System.out.println("成功建立连接");
        String INFO = session.getUri().getPath().split("INFO=")[1];
        System.out.println(INFO);
        if (INFO != null && INFO.length() > 0) {
            JSONObject jsonObject = new JSONObject(INFO);
            String command = jsonObject.getString("command");
            String roomId = jsonObject.getString("roomId");
            if (command != null && MessageKey.ENTER_COMMAND.equals(command)) {
                Map<String, WebSocketSession> mapSession = sUserMap.get(roomId);
                if (mapSession == null) {
                    mapSession = new HashMap<>(3);
                    sUserMap.put(roomId, mapSession);
                }
                mapSession.put(jsonObject.getString("name"), session);
                session.sendMessage(new TextMessage("当前房间在线人数" + mapSession.size() + "
                System.out.println(session);
            }
        }
        System.out.println("当前在线人数：" + sUserMap.size());
    }

    //消息处理方法
    @Override
    public void handleMessage(WebSocketSession webSocketSession, WebSocketMessage<?> webSoc
        try {
            JSONObject jsonobject = new JSONObject(webSocketMessage.getPayload().toString()
            Message message = new Message(jsonobject.toString());
            System.out.println(jsonobject.toString());
            System.out.println(message + ":来自" + webSocketSession.getAttributes().get(Mess
            if (message.getName() != null && message.getCommand() != null) {
                switch (message.getCommand()) {
                        //有新人加入房间信息
                    case MessageKey.ENTER_COMMAND:
                        sendMessageToRoomUsers(message.getRoomId(), new TextMessage("【" + 
                        break;
                        //聊天信息
                    case MessageKey.MESSAGE_COMMAND:
                        if (message.getName().equals("all")) {
```

```
                    } else {
                        sendMessageToUser(message.getRoomId(), message.getName(), new T
                                "悄悄对你说：" + message.getInfo()));
                    }
                    break;
                    //有人离开房间信息
                case MessageKey.LEAVE_COMMAND:
                    sendMessageToRoomUsers(message.getRoomId(), new TextMessage("【" + 
                    break;
                    default:
                        break;
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * 发送信息给指定用户
 */
public boolean sendMessageToUser(String roomId, String name, TextMessage message) {
    if (roomId == null || name == null) return false;
    if (sUserMap.get(roomId) == null) return false;
    WebSocketSession session = sUserMap.get(roomId).get(name);
    if (!session.isOpen()) return false;
    try {
        session.sendMessage(message);
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

/**
 * 广播信息给某房间内的所有用户
 */
public boolean sendMessageToRoomUsers(String roomId, TextMessage message) {
    if (roomId == null) return false;
    if (sUserMap.get(roomId) == null) return false;
    boolean allSendSuccess = true;
    Collection<WebSocketSession> sessions = sUserMap.get(roomId).values();
    for (WebSocketSession session : sessions) {
        try {
            if (session.isOpen()) {
```

```
                e.printStackTrace();
                allSendSuccess = false;
            }
        }

        return allSendSuccess;
    }

    //退出房间时的处理
    @Override
    public void afterConnectionClosed(WebSocketSession webSocketSession, CloseStatus closeS
        System.out.println("连接已关闭：" + closeStatus);
        Map<String, WebSocketSession> map = sUserMap.get(getRoomIdFromSession(webSocketSess
        if (map != null) {
            map.remove(getNameFromSession(webSocketSession));
        }
    }
}
```

要点解析：

- 使用 sUserMap 这个静态变量来保存用户信息。对应我们上述的关系图。
- 实现 afterConnectionEstablished 方法，当用户进入房间成功后，保存用户信息到 Map ，并调用 sendMessageToRoomUsers 广播新人加入信息。
- 实现 handleMessage 方法，处理用户加入，离开和发送信息三类消息。
- 实现 afterConnectionClosed 方法，用来处理当用户离开房间后的信息销毁工作。从 Map 中清除该用户。
- 实现 sendMessageToUser 、 sendMessageToRoomUsers 两个向客户端发送消息的方法。直接通过 Session 即可发送结构化数据到客户端。 sendMessageToUser 实现了点对点消息的发送， sendMessageToRoomUsers 实现了广播消息的发送。

客户端实现

客户端我们就使用HTML5为我们提供的WebSocket JS接口即可。

```html
<html>
    <script type="text/javascript">
        function ToggleConnectionClicked() {
            if (SocketCreated && (ws.readyState == 0 || ws.readyState == 1)) {
                lockOn("离开聊天室...");
                SocketCreated = false;
```

```
                ws.send(msg);
                ws.close();
            } else if(document.getElementById("roomId").value == "请输入房间号!") {
                Log("请输入房间号！");
            } else {
                lockOn("进入聊天室...");
                Log("准备连接到聊天服务器 ...");
                groom = document.getElementById("roomId").value;
                gname = document.getElementById("txtName").value;
                try {
                    if ("WebSocket" in window) {
                        ws = new WebSocket(
                            'ws://localhost:8080/webSocket/INFO={"command":"enter","name":"
                    }
                    else if("MozWebSocket" in window) {
                        ws = new MozWebSocket(
                            'ws://localhost:8080/webSocket/INFO={"command":"enter","name":"
                    }
                    SocketCreated = true;
                    isUserloggedout = false;
                } catch (ex) {
                    Log(ex, "ERROR");
                    return;
                }
                document.getElementById("ToggleConnection").innerHTML = "断开";
                ws.onopen = WSonOpen;
                ws.onmessage = WSonMessage;
                ws.onclose = WSonClose;
                ws.onerror = WSonError;
            }
        };


        function WSonOpen() {
            lockOff();
            Log("连接已经建立。", "OK");
            $("#SendDataContainer").show();
            var msg = JSON.stringify({'command':'enter', 'roomId':groom , 'name': "all",
                'info': gname + "加入聊天室"})
            ws.send(msg);
        };
    </html>
```

要点解析：

`INFO` 后加入了用户个人信息，服务端收到后，即可根据这个信息标记此会话。

- 连接建立后，发送给房间内其他人一条加入信息。通过 `ws.send()` 方法实现。

至此代码部分就介绍完了，过多的代码就不再堆叠了，更详细的代码，请参见后面的Github地址。

本篇总结

通过一个相对完整的网页聊天室例子，介绍了我们自己使用WebSocket时的几个细节：

- 服务端想在建立连接，即握手阶段搞事情，实现 `HandshakeInterceptor` 。
- 服务端想在建立连接之后和处理客户端发来的消息，实现 `WebSocketHandler` 。
- 服务端通过 `WebSocketSession` 即可向客户端发送消息，通过用户和 `Session` 的绑定，实现对应关系。

想加深理解的同学，还是要深入到代码中仔细体会。限于篇幅，而且在文章中加入大量代码本身也不容易读下去。所以大家还是要实际对着代码理解比较好。

本篇涉及到的代码

完整代码实现-小小网页聊天室

欢迎持续关注原创，喜欢的别忘了收藏关注，码字实在太累，你们的鼓励就是我坚持的动力！

小铭出品，必属精品

欢迎关注xNPE技术论坛，更多原创干货每日推送。

> **关注下面的标签，发现更多相似文章**

JavaScript 后端 产品 WebSocket

xNPE Lv2
非典型程序员 @ 自由开发者
获得点赞 452 · 获得阅读 23,325

关注

> 安装掘金浏览器插件

评论

输入评论...

相关推荐

专栏 · 幻魂 · 刚刚 · React.js / JavaScript

应战Vue3 setup，Concent携手React出招了！

专栏 · 叁公子KCN · 8分钟前 · 后端 / Python

图像搜索：给你爬的美女图建一个搜索引擎

专栏 · LinDaiDai_霖呆呆 · 5小时前 · JavaScript

JavaScript进阶-内存空间详解(双十一过后的一更)

4

专栏 · 江五渣 · 12小时前 · Go / 后端

聊聊 Go 语言中的字符表示与字符串遍历

7 7

专栏 · 故事胶片 · 3天前 · JavaScript / 前端

前端Vue中常用rules校验规则

506 39

专栏 · 徐小夕_Lab实验室 · 1天前 · JavaScript / 前端

《前端实战总结》之使用pace.js为你的网站添加加载进度条

97 6

热 · 专栏 · 神三元 · 13天前 · JavaScript

(建议精读)原生JS灵魂之问(中)，检验自己是否真的熟悉JavaScript?

1314 151

1242 76

专栏 · LIJING0906 · 9小时前 · JavaScript

浏览器进程、JS事件循环机制、宏任务和微任务

4

专栏 · 技术漫谈 · 2天前 · JavaScript / 前端

精讲 JavaScript 的 "switch" 语句

19 10