|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий (ИТ) |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
| **ОТЧЕТ**  **ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4** | |
| **по дисциплине** |  |
| **«Архитектура клиент-серверных приложений»**  **Тема: Разработка клиент-серверного приложения с использованием**  **технологии WebSocket и Spring Framework** | |
| Выполнил студент группы ИКБО-20-22 | Шумахер М.Е. |
| Принял преподаватель кафедры ИиППО | Волков М.Ю. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | «04» сентября 2024 г. |  |
| «Зачтено» | «04» сентября 2024 г. |  |

Москва 2024

**Цель работы:** знакомство студентов с одной из базовых технологии реализации конечной точки для передачи и приема данных по сети - сокетом.

**Теоретическое введение:**

Websocket позволяет создать канал связи между клиентом и сервером. В частности, канал связи, который использует протокол WebSocket в качестве протокола связи. Протокол WebSocket совместим с протоколом HTTP, который также работает через TCP / IP. Однако он имеет исключительные улучшения, в части меньших накладных расходов, чем HTTP, и двунаправленную веб-связь. Таким образом, они в первую очередь предназначены для веб-приложений, которым требуется постоянное соединение с сервером. Связь WebSocket использоваться между любыми типами приложений, но чаще всего WebSocket используется для облегчения связи между серверным приложением и приложением клиентом на основе браузера.

Протокол WebSocket призван заменить существующие обходные механизмы HTTP и предоставить эффективный протокол для одновременной двунаправленной связи с малой задержкой между браузерами и серверами по одному TCP-соединению. HTTP изначально был разработан для передачи ресурсов типа "запрос-ответ" в распределенных гипермедийных системах, но не для одновременной двунаправленной связи. Для преодоления этих архитектурных ограничений используются несколько HTTP-механизмов (сгруппированных под неофициальным названием Comet), которые часто бывают сложными и неэффективными. В этой работе кратко описываются отношения между WebSocket и HTTP / 1.1. Поскольку HTTP не был разработан для поддержки сообщений, инициируемых сервером, для этого было разработано несколько механизмов, каждый из которых имеет свои преимущества и недостатки.

WebSocket — это протокол, который позволяет одновременную двунаправленную передачу текстовых и двоичных сообщений между клиентами (в основном браузерами) и серверами по одному TCP- соединению. WebSocket может обмениваться данными через TCP на порт 80 (схема «ws») или через TLS / TCP на порт 443 (схема «wss»).

Перед началом обмена сообщениями клиент и сервер согласовывают параметры устанавливаемого соединения. WebSocket повторно использует существующий механизм обновления HTTP с настраиваемыми заголовками Sec-WebSocket-\* для подключения. Подпротоколы WebSocket — это протоколы верхнего уровня, которые обеспечивают дополнительную функциональность для приложений (например, подпротокол STOMP обеспечивает модель обмена сообщениями публикация-подписка). Расширения WebSocket — это механизм для изменения фреймов сообщений, не затрагивая приложений. (например, расширение permessage-deflate сжимает данные полезной нагрузки с помощью алгоритма LZ77).

**Постановка задачи:** Используя информацию из данной практической работы, необходимо реализовать клиент- серверное приложение с использованием Websocket. Суть приложения заключается в следующем. При обращении клиентской части по адресу /webs необходимо выполнять обработку Websocket. В случае, получения в вебсокете данных, необходимо ответить их же содержимым.

**Программный код:**

WebSocketConfig.java

package main.prac4;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.socket.config.annotation.EnableWebSocket;  
import org.springframework.web.socket.config.annotation.WebSocketConfigurer;  
import org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;  
  
@Configuration  
@EnableWebSocket  
public class WebSocketConfig implements WebSocketConfigurer {  
  
 private final WebSocketMessageHandler messageHandler;  
  
 public WebSocketConfig(WebSocketMessageHandler messageHandler) {  
 this.messageHandler = messageHandler;  
 }  
  
 @Override  
 public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {  
 registry.addHandler(messageHandler, "/webs").setAllowedOrigins("\*");  
 }  
}

WebSocketMessageHandler.java

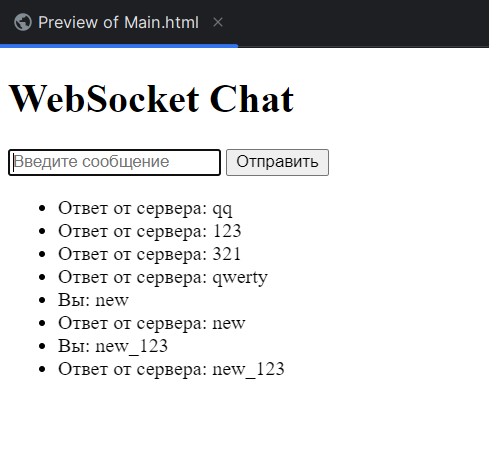
package main.prac4;  
import org.springframework.stereotype.Component;  
import org.springframework.web.socket.TextMessage;  
import org.springframework.web.socket.WebSocketSession;  
import org.springframework.web.socket.handler.TextWebSocketHandler;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.CopyOnWriteArrayList;  
  
@Component  
public class WebSocketMessageHandler extends TextWebSocketHandler {  
 private final List<String> messageHistory = new ArrayList<>();  
  
 private final List<WebSocketSession> sessions = new CopyOnWriteArrayList<>();  
  
 @Override  
 public void afterConnectionEstablished(WebSocketSession session) throws Exception {  
 sessions.add(session);  
  
 for (String message : messageHistory) {  
 session.sendMessage(new TextMessage(message));  
 }  
 }  
  
 @Override  
 public void handleTextMessage(WebSocketSession session, TextMessage message) throws Exception {  
 messageHistory.add(message.getPayload());  
  
 for (WebSocketSession activeSession : sessions) {  
 if (activeSession.isOpen()) {  
 activeSession.sendMessage(new TextMessage(message.getPayload()));  
 }  
 }  
 }  
  
 @Override  
 public void afterConnectionClosed(WebSocketSession session, org.springframework.web.socket.CloseStatus status) throws Exception {  
 sessions.remove(session);  
 }  
}

Main.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>WebSocket Chat</title>  
</head>  
<body>  
<h1>WebSocket Chat</h1>  
  
<form id="messageForm">  
 <input type="text" id="messageInput" placeholder="Введите сообщение" required>  
 <button type="submit">Отправить</button>  
</form>  
  
<ul id="messagesList"></ul>  
  
<script>  
 const socket = new WebSocket('ws://localhost:8080/webs');  
  
 const form = document.getElementById('messageForm');  
 const input = document.getElementById('messageInput');  
 const messagesList = document.getElementById('messagesList');  
  
 socket.onopen = function() {  
 console.log('WebSocket соединение установлено.');  
 };  
  
 socket.onmessage = function(event) {  
 const li = document.createElement('li');  
 li.textContent = 'Ответ от сервера: ' + event.data;  
 messagesList.appendChild(li);  
 };  
  
 socket.onerror = function(error) {  
 console.error('Ошибка WebSocket: ' + error.message);  
 };  
  
 socket.onclose = function() {  
 console.log('WebSocket соединение закрыто.');  
 };  
  
 form.addEventListener('submit', function(event) {  
 event.preventDefault();  
 const message = input.value;  
  
 if (message) {  
 socket.send(message);  
  
 const li = document.createElement('li');  
 li.textContent = 'Вы: ' + message;  
 messagesList.appendChild(li);  
  
 input.value = '';  
 }  
 });  
</script>  
</body>  
</html>

Вывод программы:





**Вывод**: в результате выполнения практической работы была освоена работа с одной из базовых технологии реализации конечной точки для передачи и приема данных по сети - сокетом.