FPT Education

FPT POLYTECHNIC

cdio™

Conceive Design Implement Operate

SPRING BOOT WEBSOCKET

LẬP TRÌNH JAVA 6

www.poly.edu.vn

AGENDA

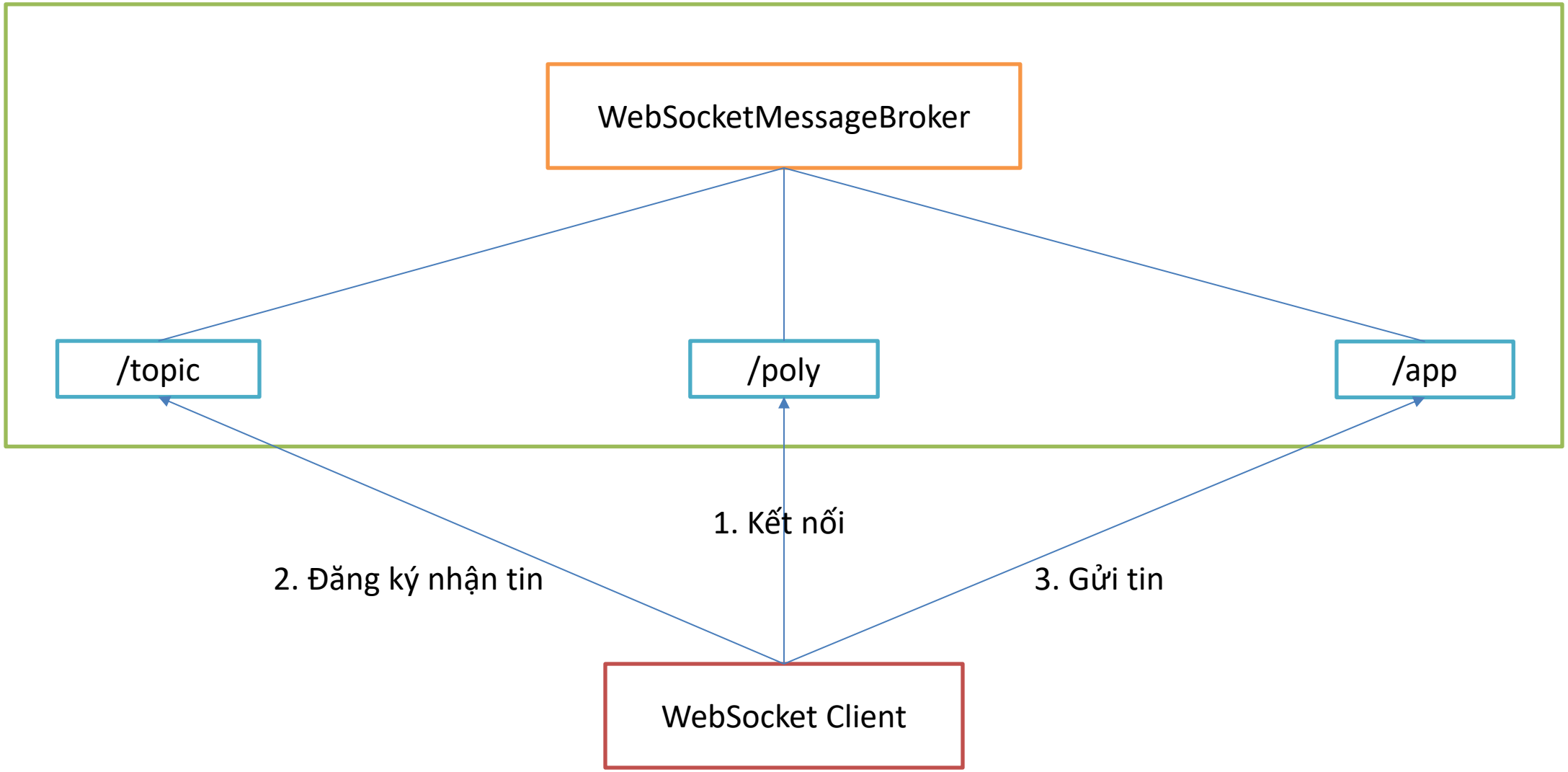
- Giới thiệu WebSocket
- Cấu hình WebSocketMessageBroker
- WebSocket Controller
- WebSocket client (StompJS)
- Xây dựng ứng dụng Chat trực tuyến
- WebSocketListener
- HandshakeInterceptor
- Chia sẻ dữ liệu giữa WebSocketController, WebSocketListener và HandshakeInterceptor

WebSocket

SPRING BOOT WEBSOCKET

- WebSocket là một giao thức truyền thông, nó giúp thành lập một kênh liên lạc 2 chiều giữa client và server.
- WebSocket không phải là giao thức request-response (Yêu cầu - Đáp ứng), nơi mà chỉ Client có mới thể gửi yêu cầu tới Server mà một khi kết nối với giao thức WebSocket được thiết lập, client & server có thể gửi dữ liệu tới cho nhau, cho tới khi kết nối ở tầng dưới là TCP được đóng lại.
- WebSocket về cơ bản rất giống với khái niệm TCP Socket, sự khác biệt là WebSocket được tạo ra để sử dụng cho các ứng dụng Web.

Mô hình ứng dụng WebSocket



WebSocketMessageBroker

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
        @Override
        public void registerStompEndpoints(StompEndpointRegistry registry) {
                registry.addEndpoint("/poly").setAllowedOrigins("*");
        }
        @Override
        public void configureMessageBroker(MessageBrokerRegistry registry) {
                registry.setApplicationDestinationPrefixes("/app");
                registry.enableSimpleBroker("/topic");
        }
}
```

WebSocketMessageBroker

- WebSocketMessageBroker là một chương trình trung gian, nó tiếp nhận các tin nhắn được gửi đến trước khi phân phát tới các địa chỉ cần thiết.
- Cấu hình WebSocketMessageBroker cần đưa ra 3 điểm cuối (endpoint) để client tương tác
  - /poly: điểm kết nối
    - Ví dụ: ws://localhost:8080/poly
  - /topic: phần đầu (prefix) của điểm để client đăng ký nhận tin nhắn
    - Ví dụ: /topic/simple-chat
  - /app: phần đầu (prefix) của điểm để client gửi tin nhắn đến
    - Ví dụ: /app/simple-chat

WebSocket Client

- StompJS là thư viện hỗ trợ lập trình websocket phía client.
  - <script src="https://cdn.jsdelivr.net/npm/@stomp/stompjs@7.0.0/bundles/stomp.umd.min.js"></script>
- Với StompJS
  - Tạo WebSocket client
  - Các phương thức tương tác
    - Mở kết nối
    - Đăng ký nhận và xử lý tin nhắn từ server
    - Gửi tin nhắn đến WebSocket Server
    - Đóng kết nối
  - Các phương thức xử lý sự kiện
    - onConnect
    - onWebSocketError
    - onStompError

StompJS API

- StompJS là thư viện JavaScript hỗ trợ lập trình web socket phía client.

```
<script src="https://cdn.jsdelivr.net/npm/@stomp/stompjs@7.0.0/bundles/stomp.umd.min.js"></script>
```

- StompJs API

Phương thức	Mô tả
StopmJs.Client({brokerURL})	Tạo WebSocket client
activate()	Mở kết nối đến WebSocket Server
deactivate()	Đóng kết nối
subscribe(topicUrl, callback)	Đăng ký nhận và xử lý tin nhắn phản hồi từ server
publish(appUrl, data)	Gửi tin nhắn đến server
onConnect(callback)	Được gọi khi kết nối thành công
onStompError(callback)	Được gọi khi WebSocket Server có sự cố
onWebSocketError(callback)	Được gọi khi WebSocket Client có sự cố

WebSocket Client

Tạo WebSocket client

```
const client = new StompJs.Client({
        brokerURL: 'ws://localhost:8080/poly',
})
```

Đăng ký nhận và xử lý tin nhắn từ server ngay sau khi kết nối thành công

```
client.onConnect = (response) => {
        client.subscribe('/topic/simple-chat', (payload) => {
                var msg = JSON.parse(payload.body);
                console.log(msg);
        })
}
```

Mở kết nối đến WebSocket Server

```
client.activate();
```

Gửi tin nhắn đến WebSocket Server

```
var msg = { name: 'A', text: 'B' };

client.publish({
        destination: '/app/simple-chat',
        body: JSON.stringify(msg)
})
```

WebSocket Controller

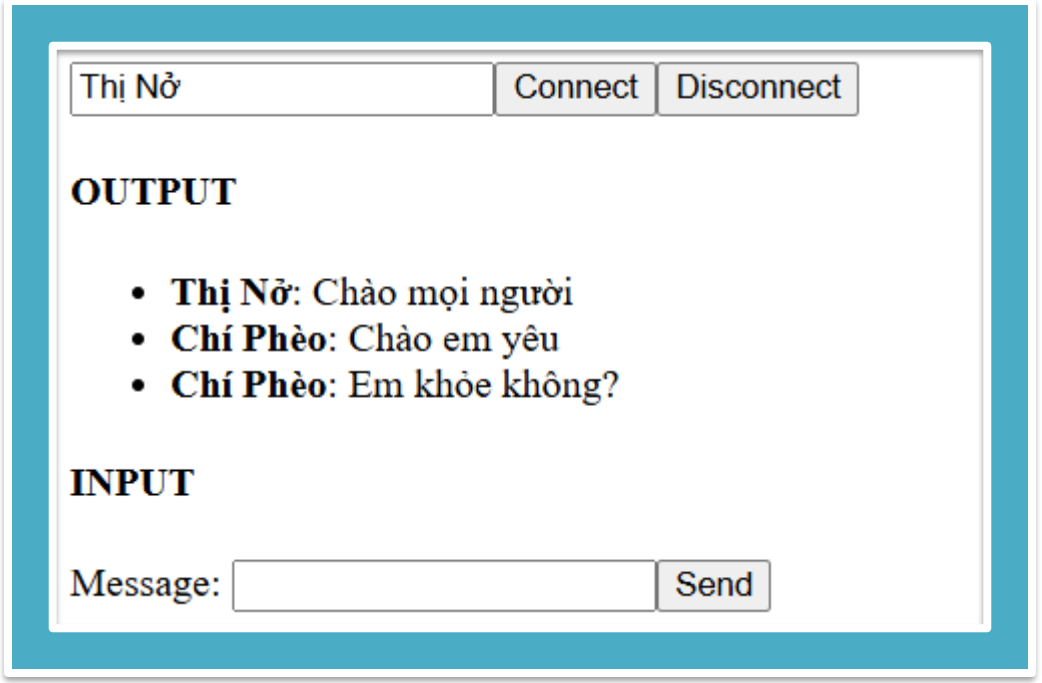
```
@Controller
public class SimpleChatController {
    @MessageMapping("/simple-chat ")
    @SendTo("/topic/simple-chat")
    public ChatMessage sendMessage(@Payload ChatMessage chatMessage) {
        return chatMessage;
    }
}
```

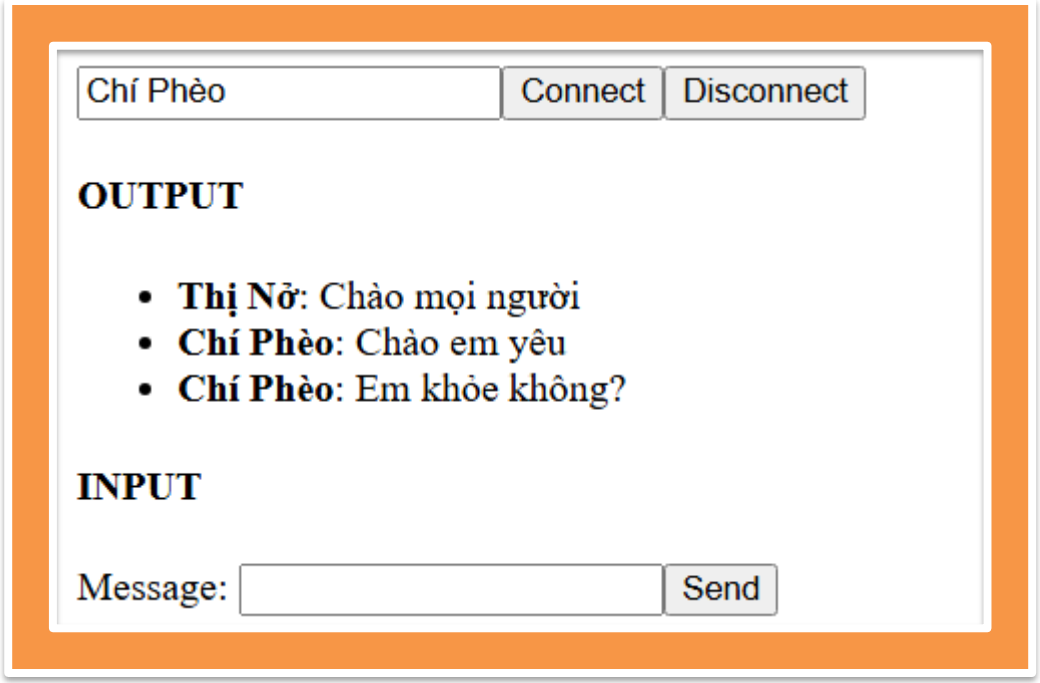
- WebSocket Controller chứa các phương thức điều khiển tiếp nhận và xử lý gửi tin nhắn được gửi từ WebSockerMessageBroker được map bắt đầu bởi /app sau đó chuyển kết quả trả về đến /topic
- @MessageMapping("/simple-chat")
  - MessageBroker sẽ gọi phương thức này
- @SendTo("/topic/simple-chat")
  - Chuyển kết quả trả về của phương thức điều khiển đến /topic/simple-chat
- @Payload
  - Yêu cầu chuyển đổi json của phần body của payload và chuyển thành đối tượng java

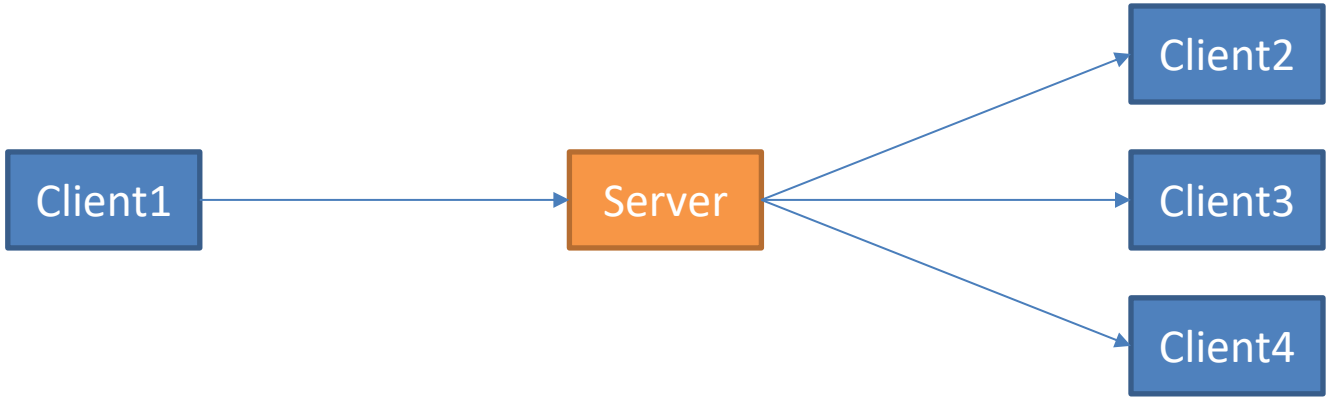
Simple Chat

GIỚI THIỆU CHAT ONLINE

- Chat là ứng dụng cho phép trao đổi trực tiếp qua mạng internet







CÔNG VIỆC CẦN THỰC HIỆN

- Khai báo thư viện cần thiết (dependency)
- Cấu hình WebSocketMessageBroker
- Tạo ChatMessage mô tả cấu trúc dữ liệu tin nhắn
- Tạo Message Controller
- Tạo WebSocket Client

DEPENDENCY

- Thư viện cần thiết hỗ trợ lập trình WebSocket trong Spring Boot

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
```

Cấu hình Web Socket Message Broker

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
        @Override
        public void registerStompEndpoints(StompEndpointRegistry registry) {
                registry.addEndpoint("/poly").setAllowedOrigins("*");
        }
        @Override
        public void configureMessageBroker(MessageBrokerRegistry registry) {
                registry.setApplicationDestinationPrefixes("/app");
                registry.enableSimpleBroker("/topic");
        }
}
```

TẠO CHATMESSAGE

```java
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Data
public class ChatMessage {
    @Default
    private MessageType type = MessageType.CHAT;
    private String content;
    private String sender;
    public enum MessageType {
        CHAT, JOIN, LEAVE
    }
}
```

Java

⇔

```
type: 'CHAT',
sender: 'Chí phèo'
content: 'Anh yêu em'
```

JSON

TẠO MESSAGE CONTROLLER

```
@Controller
public class SimpleChatController {
    @MessageMapping("/simple-chat ")
    @SendTo("/topic/simple-chat")
    public ChatMessage sendMessage(@Payload ChatMessage chatMessage) {
        return chatMessage;
    }
}
```

TẠO WEBSOCKET CLIENT

```
<script setup>
    import {ref} from 'vue'
    const form = ref({sender: "", content: "", type: 'CHAT'}) // Buộc vào form
    const list = ref([]) // danh sách tin nhắn
    const ctrl = {
        connect() {},
        disconnect() {},
        send() {}
    }
</script>
```

Tạo SimpleChat.vue có giao diện và tổ chức mã JavaScript như hình

```
<template>
    <input v-model="form.sender" />
    <button @click="ctrl.connect()">Connect</button>
    <button @click="ctrl.disconnect()">Disconnect</button>
    <h4>OUTPUT</h4>
    <li v-for="(e, i) in list" :key="i"><b>{{ e.sender }}</b>: {{ e.content }}</li>
    <h4>INPUT</h4>
    Message: <input v-model="form.content" />
    <button @click="ctrl.send()">Send</button>
</template>
```

Tạo WebSocket Client

- Nhúng thư viên StompJs sau vào file index.html
  - <script src="https://cdn.jsdelivr.net/npm/@stomp/stompjs@7.0.0/bundles/stomp.umd.min.js"></script>
- Bổ sung đoạn mã sau vào cuối thẻ <script>

```
// Tạo WebSocket Client
const client = new StompJs.Client({
        brokerURL: 'ws://localhost:8080/poly'
})
// Được gọi khi kết nối thành công
client.onConnect = (frame) => {
        console.log("client.onConnect()", frame)
        // Đăng ký nhận tin nhắn từ @/topic/simple-chat
        client.subscribe('/topic/simple-chat', (response) => {
                var message = JSON.parse(response.body);
                list.value.push(message);
        });
};
```

```
// Được gọi khi kết nối lỗi
client.onWebSocketError = (error) => {
        alert("onWebSocketError()")
        console.log("onWebSocketError()", error);
}
// Được gọi khi server lỗi
client.onStompError = (error) => {
        alert("onStompError()")
        console.log("onStompError()", error);
}
```

Tạo WebSocket Client

- Hoàn thiện mã cho các phương thức connect(), disconnect() và send() của ctrl như sau và import <SimpleChat.vue> vào App.vue và chạy

```
connect() {
        client.activate();
},
disconnect() {
        client.deactivate();
},
send() {
        var message = form.value;
        client.publish({
                destination: "/app/simple-chat",
                body: JSON.stringify(message)
        });
        form.value.content = "";
}
```

WebSocketListener

- WebSocket Server cung cấp các listener giúp ứng dụng lắng nghe và xử lý 1 số sự kiện trong quá trình giao tiếp với client.
- Các sự kiện có thể nghe trong listener
  - SessionConnectEvent
    - Yêu cầu kết nối của client kết nối với server
  - SessionConnectedEvent
    - Sau khi hoàn thành kết nối với client
  - SessionDisconnectEvent
    - Yêu cầu của client ngắt kết nối với server
  - SessionSubscribeEvent
    - Yêu cầu đăng ký nhận tin nhắn của client
  - SessionUnsubscribeEvent
    - Yêu cầu hủy kết nối nhận tin nhắn của client

WebSocketListener

```
@Component
public class WebSocketListener {
    @Autowired
    private SimpMessageSendingOperations smso;
    @EventListener
    public void disconnected(SessionDisconnectEvent event) {
        ChatMessage msg = new ChatMessage();
        msg.setType(ChatMessage.MessageType.LEAVE);
        msg.setSender("poly");
        smso.convertAndSend("/topic/simple-chat", msg);
    }
}
```

Bean SimpMessageSendingOperations được cung cấp sẵn bởi WebSocket Server cho phép gửi tin nhắn đến /topic

- Đoạn mã cài đặt để lắng nghe sự kiện ngắt kết nối từ client và gửi thông báo LEAVE đến những client còn lại.

HandshakeInterceotor

- WebSocket hình thành bởi sự kết hợp giữa Web và Socket (HTTP và TCP). Khi một client kết nối đến server thì sẽ có sự bắt tay giữa 2 giao thức này.
- WebSocket cung cấp HandshakeInterceotor để xử lý các sự kiện xảy ra trước và sau khi bắt tay.
- Thông thường chúng ta truy xuất các tài nguyên của web (HTTP) để chia sẽ với WebSocket.

HANDSHAKEINTERCEOTOR

```
@Component
public class WebSocketInterceptor implements HandshakeInterceptor {
	@Override
	public boolean beforeHandshake(ServerHttpRequest request, ServerHttpResponse response,
			WebSocketHandler wsHandler, Map<String, Object> attributes) throws Exception {
		System.out.println("beforeHandshake()");
		return true;
	}
	@Override
	public void afterHandshake(ServerHttpRequest request, ServerHttpResponse response,
			WebSocketHandler wsHandler, Exception exception) {
		System.out.println("afterHandshake()");

	}
}
```

HANDSHAKEINTERCEPTOR

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
    @Autowired
    AppHandshakeInterceptor interceptor;
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/poly").setAllowedOrigins("*").addInterceptors(interceptor);
    }
    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.setApplicationDestinationPrefixes("/app");
        registry.enableSimpleBroker("/topic");
    }
}
```

Gắn kết Interceptor với điểm cuối kết nối

HANDSHAKEINTERCEOTOR

```
@Override
public boolean beforeHandshake(ServerHttpRequest request, ServerHttpResponse response,
        WebSocketHandler wsHandler, Map<String, Object> attributes) throws Exception {
    ServletServerHttpRequest servletRequest = (ServletServerHttpRequest) request;
    HttpSession session = servletRequest.getServletRequest().getSession();
    attributes.put("sessionId", session.getId());
    return true;
}
```

- Cài đặt mã cho beforeHandshake() như trên để chia sẻ HttpSession ID với WebSocket Controller hoặc @EventListener bằng cách bổ sung thuộc tính vào attributes.

HANDSHAKEINTERCEOTOR

```
@Controller
public class SimpleChatController {
    @MessageMapping("/simple-chat") @SendTo("/topic/simple-chat")
    public ChatMessage sendMessage(@Payload ChatMessage chatMessage,
                SimpMessageHeaderAccessor headerAccessor) {
                var sessionId = (String)headerAccessor.getSessionAttributes().get("sessionId");
                headerAccessor.getSessionAttributes().put("username", "Chí Phèo");
        return chatMessage;
    }
}
```

- Trong Controller có thể truy xuất các thuộc tính đã chia sẽ như trên. Hoặc có thể bổ sung thêm các thuộc tính mới vào.

HANDSHAKEINTERCEOTOR

```
@EventListener
public void disconnected(SessionDisconnectEvent event) {
        StompHeaderAccessor headerAccessor = StompHeaderAccessor.wrap(event.getMessage());
        String username = (String) headerAccessor.getSessionAttributes().get("username");
        if (username != null) {
                ChatMessage msg = new ChatMessage();
                msg.setType(ChatMessage.MessageType.LEAVE);
                msg.setSender(username);
                smso.convertAndSend("/topic/simple-chat", msg);
        }
}
```

- Trong @EventListener có thể truy xuất các attributes đã được chia sẻ.

SUMMARY

- ☑ Giới thiệu WebSocket
- ☑ Cấu hình WebSocketMessageBroker
- ☑ WebSocket Controller
- ☑ WebSocket client (StompJS)
- ☑ Xây dựng ứng dụng Chat trực tuyến
- ☑ WebSocketListener
- ☑ HandshakeInterceptor
- ☑ Chia sẻ dữ liệu giữa WebSocketController, WebSocketListener và HandshakeInterceptor


TIN HỌC VĂN PHÒNG QUỐC TẾ THEO CHUẨN ICDL
Syllabus 5 - Office 2007
LUẬT KINH TẾ
LUẬT KINH TẾ
KỸ THUẬT SOẠN THẢO VÀ TRÌNH BÀY VĂN BẢN
TIN HỌC VĂN PHÒNG QUỐC TẾ THEO CHUẨN ICDL
Syllabus 5 - Office 2007
Cảm ơn