



## LẬP TRÌNH WEBSOCKET

---

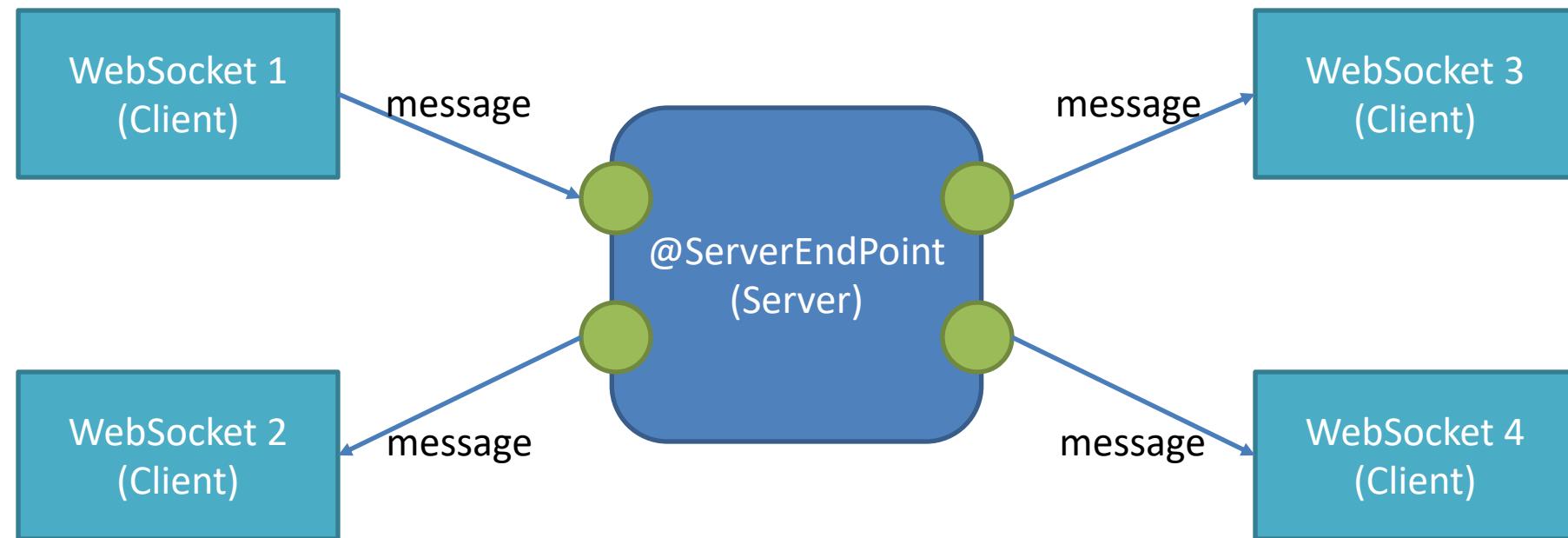
## LẬP TRÌNH JAVA #4 (P8.1)

- ❑ Giới thiệu ứng dụng thời gian thực
- ❑ Giới thiệu WebSocket
- ❑ Xây dựng ứng dụng Chat
  - ❖ ChatClient với WebSocket
  - ❖ ChatServer với @ServerEndpoint



- ❑ Ứng dụng thời gian thực (realtime) là ứng dụng phục vụ cho nhiều người sử dụng đồng thời. Khi dữ liệu thay đổi bởi một người sử dụng nào đó ngay lập tức được cập nhật cho những người đang sử dụng hiện tại.
- ❑ Một số ứng dụng thường gặp
  - ❖ Chat
  - ❖ Live Stream
  - ❖ Google Sheet
  - ❖ Cảnh báo tức thì
  - ❖ ...

- WebSocket là công nghệ truyền thông hai chiều giữa client và server theo thời gian thực cho các ứng dụng web.



## ❑ Sự kiện

- ❖ WebSocket.**onopen**(response)
  - Sự kiện này xảy ra khi Server chấp nhận kết nối
- ❖ WebSocket.**onmessage**(response)
  - Sự kiện này xảy ra khi Client nhận dữ liệu từ Server.
- ❖ WebSocket.**onerror**(response)
  - Sự kiện này xảy ra khi có bất kỳ lỗi nào trong giao tiếp.
- ❖ WebSocket.**onclose**(response)
  - Sự kiện này xảy ra khi kết nối được đóng.

## ❑ Phương thức

- ❖ WebSocket.**send**(text)
- ❖ WebSocket.**close**()

```
// Tạo WebSocket kết nối đến server
var websocket = new WebSocket('server-endpoint-url');
// Được gọi khi kết nối thành công
websocket.onopen = function(response) {...}
// Được gọi khi nhận được message từ server
websocket.onmessage = function(response) {...}
// Được gọi khi nhận được lỗi từ server
websocket.onerror = function(response) {...}
// Được gọi khi nhận đóng kết nối
websocket.onclose = function(response) {...}
```

*Đối số response là đối tượng JS phản hồi từ Server.*

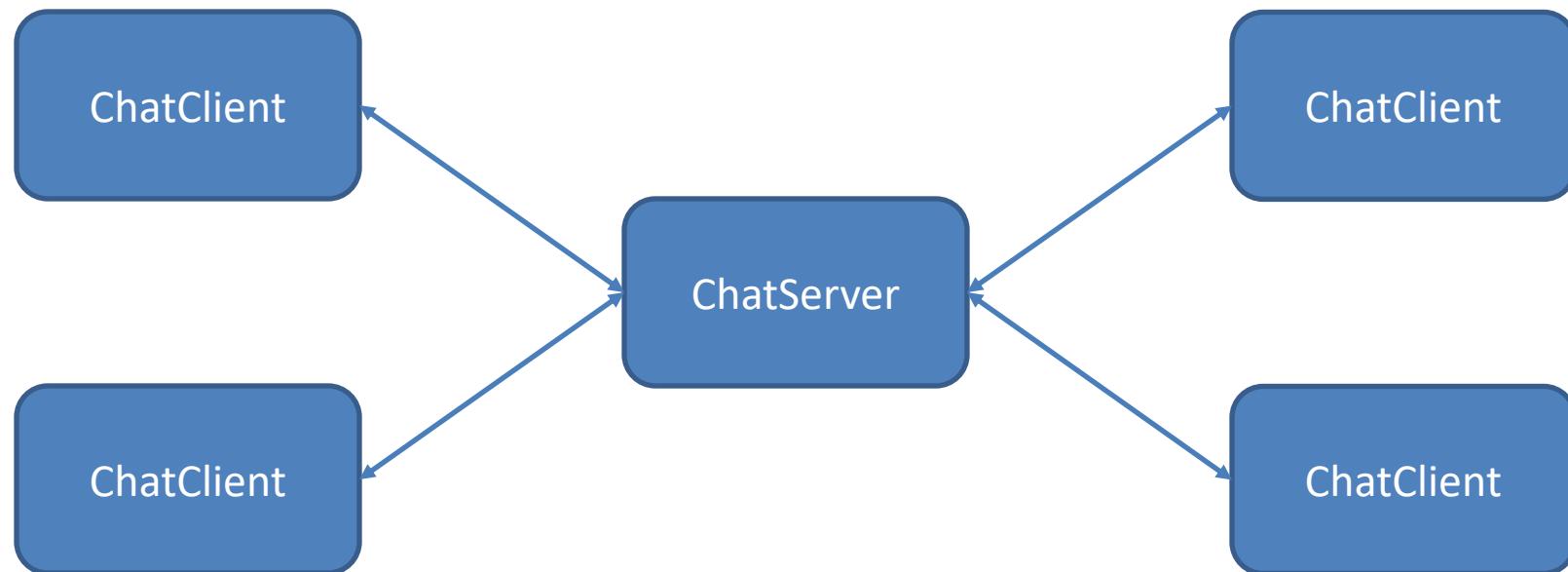
- ❑ **@ServerEndpoint** được sử dụng để chú thích cho class tạo server phục vụ kết nối các WebSocket
- ❑ Các annotation mức phương thức
  - ❖ **@OnOpen**
    - Được gọi khi chấp nhận kết nối từ WebSocket
  - ❖ **@OnMessage**
    - Được gọi khi nhận được message từ WebSocket
  - ❖ **@OnClose**
    - Được gọi khi một WebSocket đóng
  - ❖ **@OnError**
    - Được gọi khi có lỗi xảy ra trong quá trình giao tiếp với client

```
@ServerEndpoint("server-endpoint-path")
public class MyServerEndpoint {
    @OnOpen
    public void onOpen(Session session) {...}
    @OnMessage
    public void onMessage(String message, Session session) {...}
    @OnClose
    public void onClose(Session session) {...}
    @OnError
    public void onError(Session session, Throwable throwable) {...}
}
```

*Session là đối tượng được tạo ra để giao tiếp với mỗi WebSocket khi kết nối thành công*

## ❑ Yêu cầu của ứng dụng

- ❖ Gửi tin nhắn chat bằng text cho tất cả mọi chatter đang kết nối
- ❖ Thông báo vào/ra



6: joined the chat!

6: Xin chào mình là Tèo đây

Send

```
<html>
<head>
    <meta charset="UTF-8">
    <title>Simple Chat - Websockets</title>
    <script src="js/text-chat.js"></script>
</head>
<body onload="init()">
    <div id="messages" style="height: 200px; overflow: auto;"></div>
    <hr>
    <input id="message">
    <button onclick="send()">Send</button>
</body>
</html>
```

```
var websocket = null;  
function init() {  
    ...  
}  
function send() {  
    var input = document.getElementById("message");  
    websocket.send(input.value);  
    input.value = " ";  
}
```

```
function init() {  
    websocket = new WebSocket('ws://localhost:8080/websocket/text/chat');  
    websocket.onopen = function(resp) {}  
    websocket.onmessage = function(resp) {  
        var output = document.getElementById('messages');  
        output.innerHTML = '${ output.innerHTML }<p>${resp.data}</p>';  
    }  
    websocket.onerror = function(resp) {  
        alert('An error occurred, closing application');  
    }  
    websocket.onclose = function(resp) {  
        alert(resp.reason || 'Goodbye');  
    }  
}
```

```
@ServerEndpoint("/text/chat")
public class TextChatServerEndpoint {
    static Map<String, Session> sessions = new HashMap<>();
    void broadcast(String message) {...}
    @OnOpen
    public void onOpen(Session session) {...}
    @OnClose
    public void onClose(Session session) {...}
    @OnMessage
    public void onMessage(String message, Session session) {...}
    @OnError
    public void onError(Session session, Throwable throwable) {...}
}
```

```
static Map<String, Session> sessions = new HashMap<>();
void broadcast(String message) {
    sessions.forEach((id, session) -> {
        try {
            session.getBasicRemote()
                .sendText(id + ":" + message.trim());
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
}
```

Map nắm giữ tất cả các Session kết nối với WebSocket đang hoạt động  
Gửi tin nhắn cho tất cả mọi Session hiện có trong Map

@OnOpen

```
public void onOpen(Session session) {  
    sessions.put(session.getId(), session);  
    this.broadcast(" joined the chat!");  
}
```

@OnClose

```
public void onClose(Session session) {  
    this.broadcast("left the chat!");  
    sessions.remove(session.getId());  
}
```

*Thêm session vào Map khi chấp nhận kết nối từ WebSocket  
Xóa session khỏi Map khi kết nối đóng lại*

@OnMessage

```
public void onMessage(String message, Session session) {  
    this.broadcast(message);  
}
```

@OnError

```
public void onError(Session session, Throwable throwable) {  
    try {  
        if (session.isOpen()) {  
            session.close();  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Nhận và chuyển tiếp tin nhắn đến tất cả chatter  
Đóng session khi có lỗi xảy ra



# DEMOSTATION

---

- ❑ Giới thiệu ứng dụng thời gian thực
- ❑ Giới thiệu WebSocket
- ❑ Xây dựng ứng dụng Chat
  - ❖ ChatClient với WebSocket
  - ❖ ChatServer với @ServerEndpoint





Cảm ơn