

Architecture of Enterprise Applications 3

WebSocket

Haopeng Chen

***RE**liable, **IN**telligent and **SC**alable Systems Group (**REINS**)*

Shanghai Jiao Tong University

Shanghai, China

<http://reins.se.sjtu.edu.cn/~chenhp>

e-mail: chen-hp@sjtu.edu.cn

- Contents

- Introduction to WebSocket
- WebSocket Programming Model
- WebSocket Samples

- Objectives

- 能够根据系统需求，分析前后端之间适用于异步通信机制的业务场景，并设计并实现基于WebSocket的实现方案

- **WebSocket** is an application protocol that provides **full-duplex** communications between two peers over the TCP protocol.
 - In the traditional request-response model used in HTTP, the client requests resources and the server provides responses.
 - The exchange is always initiated by the client; the server cannot send any data without the client requesting it first.
 - The **WebSocket** protocol provides a full-duplex communication channel between the client and the server.
 - Combined with other client technologies, such as **JavaScript** and **HTML5**, **WebSocket** enables web applications to deliver a richer user experience.

- In a WebSocket application, the server publishes a WebSocket **endpoint** and the client uses the endpoint's URI to connect to the server.
 - The WebSocket protocol is symmetrical after the connection has been established:
 - The client and the server can send messages to each other at any time while the connection is open, and they can close the connection at any time.
 - Clients usually connect only to one server, and servers accept connections from multiple clients.
- The WebSocket protocol has two parts:
 - **handshake** and **data transfer**.

- The client initiates the **handshake** by sending a request to a WebSocket endpoint using its URI.
 - The handshake is compatible with existing HTTP-based infrastructure:
 - web servers interpret it as an HTTP connection upgrade request.
 - An example handshake from a **client** looks like this:

```
GET /path/to/websocket/endpoint HTTP/1.1
```

```
Host: localhost
```

```
Upgrade: websocket
```

```
Connection: Upgrade
```

```
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
```

```
Origin: http://localhost
```

```
Sec-WebSocket-Version: 13
```

- The client initiates the **handshake** by sending a request to a WebSocket endpoint using its URI.
 - The handshake is compatible with existing HTTP-based infrastructure:
 - web servers interpret it as an HTTP connection upgrade request.
 - An example handshake from the **server** in response to the client looks like this:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=

- The server applies a known operation to
 - the value of the **Sec-WebSocket-Key** header to generate the value of the **Sec-WebSocket-Accept** header.
- The client applies the same operation to
 - the value of the **Sec-WebSocket-Key** header, and the connection is established successfully if the result matches the value received from the server.
- The client and the server can send messages to each other after a successful handshake.

- WebSocket endpoints are represented by URIs that have the following form:
 - `ws://host:port/path?query`
 - `wss://host:port/path?query`
- The `ws` scheme represents an unencrypted WebSocket connection, and
- the `wss` scheme represents an encrypted connection.
- The `port` component is optional;
 - the default port number is 80 for unencrypted connections and
 - 443 for encrypted connections.
- The `path` component indicates the location of an endpoint within a server.
- The `query` component is optional.

- The Java API for WebSocket consists of the following packages:
 - The `javax.websocket.server` package contains annotations, classes, and interfaces to create and configure server endpoints.
 - The `javax.websocket` package contains annotations, classes, interfaces, and exceptions that are common to client and server endpoints.
- WebSocket endpoints are instances of the `javax.websocket.Endpoint` class.
 - The Java API for WebSocket enables you to create two kinds of endpoints: programmatic endpoints and annotated endpoints.
 - To create a **programmatic endpoint**, you extend the `Endpoint` class and override its lifecycle methods.
 - To create an **annotated endpoint**, you decorate a Java class and some of its methods with the annotations provided by the packages above.
 - After you have created an endpoint, you deploy it to an specific URI in the application so remote clients can connect to it.

- The process for creating and deploying a WebSocket endpoint:
 1. Create an endpoint class.
 2. Implement the lifecycle methods of the endpoint.
 3. Add your business logic to the endpoint.
 4. Deploy the endpoint inside a web application.
- The process is slightly different for **programmatic endpoints** and **annotated endpoints**

- **EchoEndpoint**

```
public class EchoEndpoint extends Endpoint {
    @Override
    public void onOpen(final Session session,
                      EndpointConfig config)
    {
        session.addMessageHandler(
            new MessageHandler.Whole<String>() {
                @Override
                public void onMessage(String msg) {
                    try {
                        session.getBasicRemote().sendText(msg);
                    } catch (IOException e) { ... }
                }
            });
    }
}
```

- To deploy this programmatic endpoint, use the following code in your Java EE application:
`ServerEndpointConfig.Builder.create(EchoEndpoint.class, "/echo").build();`
- When you deploy your application, the endpoint is available at `ws://<host>:<port>/<application>/echo`;
 - for example, `ws://localhost:8080/echoapp/echo`.

- EchoEndpoint

```
@ServerEndpoint("/echo")
public class EchoEndpoint {
    @OnMessage
    public void onMessage(Session session, String msg) {
        try {
            session.getBasicRemote().sendText(msg);
        } catch (IOException e) { ... }
    }
}
```

Annotation	Event	Example
OnOpen	Connection opened.	<code>@OnOpen public void open(Session session, EndpointConfig conf) { }</code>
OnMessage	Message received.	<code>@OnMessage public void message (Session session, String msg) { }</code>
OnError	Connection error.	<code>@OnError public void error(Session session, Throwable error) { }</code>
OnClose	Connection closed.	<code>@OnClose public void close(Session session, CloseReason reason) { }</code>

- Send messages

```
@ServerEndpoint("/echoall")
public class EchoAllEndpoint {
    @OnMessage
    public void onMessage(Session session, String msg)
    {
        try {
            for (Session sess : session.getOpenSessions()) {
                if (sess.isOpen())
                    sess.getBasicRemote().sendText(msg);
            }
        } catch (IOException e) { ... }
    }
}
```

- Receive messages

```
@ServerEndpoint("/receive")
public class ReceiveEndpoint {
    @OnMessage
    public void textMessage(Session session, String msg)
    { System.out.println("Text message: " + msg); }

    @OnMessage
    public void binaryMessage(Session session, ByteBuffer msg)
    { System.out.println("Binary message: " + msg.toString()); }

    @OnMessage
    public void pongMessage(Session session, PongMessage msg)
    {
        System.out.println("Pong message: " +
            msg.getApplicationData().toString());
    }
}
```


- ETFEndPoint.java

```
@ServerEndpoint("/dukeetf")
public class ETFEndpoint {
    private static final Logger logger =
        Logger.getLogger("ETFEndpoint");
    static Queue<Session> queue = new ConcurrentLinkedQueue<>();

    public static void send(double price, int volume) {
        String msg = String.format("%.2f, %d", price, volume);
        try {
            for (Session session : queue) {
                session.getBasicRemote().sendText(msg);
                logger.log(Level.INFO, "Sent: {0}", msg);
            }
        } catch (IOException e) {
            logger.log(Level.INFO, e.toString());
        }
    }
}
```

- ETFEndPoint.java

```
@OnOpen
public void openConnection(Session session) {
    queue.add(session);
    logger.log(Level.INFO, "Connection opened.");
}
@OnClose
public void closedConnection(Session session) {
    queue.remove(session);
    logger.log(Level.INFO, "Connection closed.");
}
@OnError
public void error(Session session, Throwable t) {
    queue.remove(session);
    logger.log(Level.INFO, t.toString());
    logger.log(Level.INFO, "Connection error.");
}
}
```

- ETFListener.java

```
@WebListener
```

```
public class ETFListener implements ServletContextListener {
```

```
    private Timer timer = null;
```

```
    public void contextInitialized(ServletContextEvent event) {
```

```
        timer = new Timer(true);
```

```
        event.getServletContext().log("The Timer is started");
```

```
        timer.schedule(new ReportBean(event.getServletContext()),  
                        0, 1000);
```

```
        event.getServletContext().log("The task is added");
```

```
    }
```

```
}
```

- ReportBean.java

```
public class ReportBean extends TimerTask {  
  
    private ServletContext context = null;  
    private Random random = new Random();  
    private double price = 100.0;  
    private int volume = 300000;  
  
    public ReportBean(ServletContext context)  
    { this.context = context; }  
  
    public void run() {  
        context.log("Task started");  
        price += 1.0*(random.nextInt(100)-50)/100.0;  
        volume += random.nextInt(5000) - 2500;  
        ETFEndpoint.send(price, volume);  
        context.log("Task ended");  
    }  
}
```

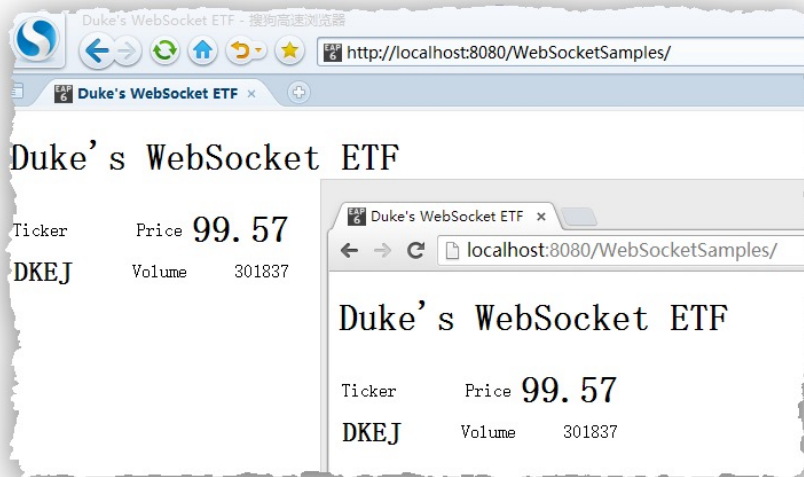
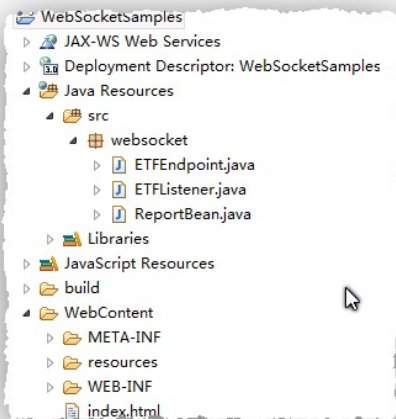
- Index.html

```
<html>
<head>
  <title>Duke's WebSocket ETF</title>
  <script type="text/javascript">
    var wsocket;
    function connect() {
      wsocket = new WebSocket
        ("ws://localhost:8080/WebSocketSamples/dukeetf");
      wsocket.onmessage = onMessage;
    }
    function onMessage(evt) {
      var arraypv = evt.data.split(",");
      document.getElementById("price").innerHTML = arraypv[0];
      document.getElementById("volume").innerHTML = arraypv[1];
    }
    window.addEventListener("load", connect, false);
  </script>
</head>
```

- Index.html

```
<body>
  <h1>Duke's WebSocket ETF</h1>
  <table>
    <tr>
      <td width="100">Ticker</td>
      <td align="center">Price</td>
      <td id="price"
        style="font-size:24pt;font-weight:bold;">--.--</td>
    </tr>
    <tr>
      <td style="font-size:18pt;font-weight:bold;"
        width="100">DKEJ</td>
      <td align="center">Volume</td>
      <td id="volume" align="right">--</td>
    </tr>
  </table>
</body>
</html>
```

An example



- The Java API for WebSocket provides
 - support for converting between WebSocket messages and custom Java types using encoders and decoders.
 - An **encoder** takes a Java object and produces a representation that can be transmitted as a WebSocket message;
 - for example, encoders typically produce JSON, XML, or binary representations.
 - A **decoder** performs the reverse function: it reads a WebSocket message and creates a Java object.
 - This mechanism simplifies WebSocket applications, because it decouples the business logic from the serialization and deserialization of objects.

- Implement one of the following interfaces:
 - `Encoder.Text<T>` for text messages
 - `Encoder.Binary<T>` for binary messages

```
public class MessageATextEncoder implements Encoder.Text<MessageA> {  
    @Override  
    public void init(EndpointConfig ec) { }  
  
    @Override  
    public void destroy() { }  
  
    @Override  
    public String encode(MessageA msgA) throws EncodeException {  
        // Access msgA's properties and convert to JSON text...  
        return msgAJsonString;  
    }  
}
```

- Then, add the encoders parameter to the `ServerEndpoint` annotation as follows:

```
@ServerEndpoint(  
    value = "/myendpoint",  
    encoders = {  
        MessageATextEncoder.class,  
        MessageBTextEncoder.class } )  
public class EncEndpoint { ... }
```

- Now you can send `MessageA` and `MessageB` objects as `WebSocket` messages using the `sendObject` method as follows:

```
MessageA msgA = new MessageA(...);  
MessageB msgB = new MessageB(...);  
session.getBasicRemote.sendObject(msgA);  
session.getBasicRemote.sendObject(msgB);
```

- Implement one of the following interfaces:
 - `Decoder.Text<T>` for text messages
 - `Decoder.Binary<T>` for binary messages

```
public class MessageTextDecoder implements Decoder.Text<Message> {  
    @Override  
    public void init(EndpointConfig ec) { }  
    @Override  
    public void destroy() { }  
    @Override  
    public Message decode(String string) throws DecodeException {  
        // Read message...  
        if ( /* message is an A message */ ) return new MessageA(...);  
        else if ( /* message is a B message */ ) return new MessageB(...);  
    }  
    @Override  
    public boolean willDecode(String string) {  
        // Determine if the message can be converted into either a  
        // MessageA object or a MessageB object...  
        return canDecode;  
    }  
}
```

- Then, add the decoders parameter to the `ServerEndpoint` annotation as follows:

```
@ServerEndpoint(  
    value = "/myendpoint",  
    encoders = {  
        MessageATextEncoder.class,  
        MessageBTextEncoder.class }  
    decoders = { MessageTextDecoder.class }  
)  
public class EncEndpoint { ... }
```

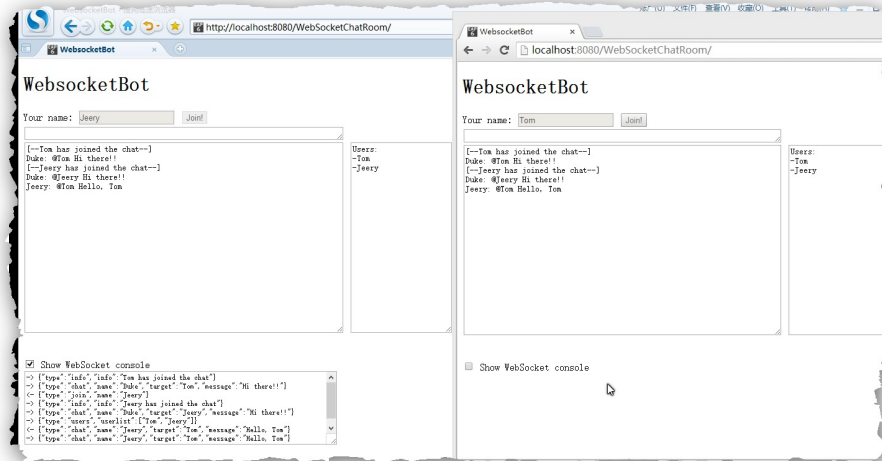
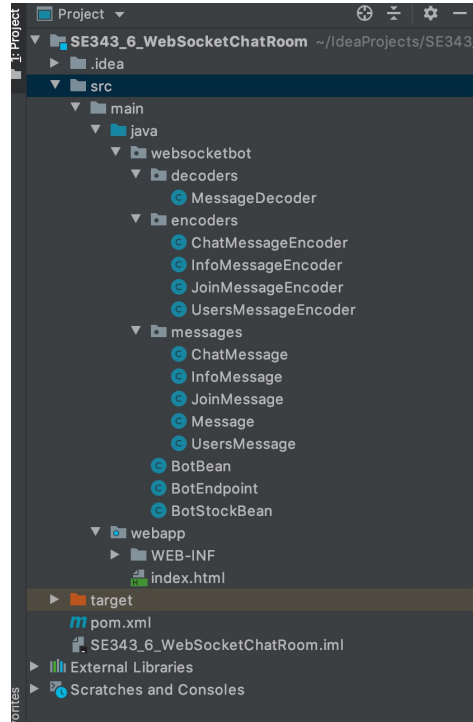
- Now define a method in the endpoint class that receives `MessageA` and `MessageB` objects as follows:

```
@OnMessage public void message(Session session, Message msg) {  
    if (msg instanceof MessageA) {  
        // We received a MessageA object...  
    }  
    else if (msg instanceof MessageB) {  
        // We received a MessageB object... }  
}
```

- To designate a method that handles errors in an annotated WebSocket endpoint, decorate it with `@OnError`:

```
@ServerEndpoint("/testendpoint")
public class TestEndpoint {
    ...
    @OnError
    public void error(Session session, Throwable t)
    {
        t.printStackTrace();
        ...
    }
}
```

An example - Chatroom



- index.html

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>WebsocketBot</title>
  <script type="text/javascript">
    var wsocket;    // Websocket connection
    var userName;   // User's name
    var textarea;   // Chat area
    var wsconsole;  // Websocket console area
    var userlist;   // User list area

    function connect() {
      textarea = document.getElementById("textarea");
      wsconsole = document.getElementById("wsconsole");
      userlist = document.getElementById("userlist");
      wsocket = new WebSocket(
        "ws://localhost:8080/WebSocketChatRoom/websocketbot");
      wsocket.onmessage = onMessage;
      document.getElementById("name").focus();
      document.getElementById("consolediv").style.visibility = 'hidden';
    }
  </script>
</head>
<body>
  <div id="name">Name: <input type="text" value="User Name" />
  <div id="chat">
    <div id="consolediv">
      <div id="wsconsole">
        <div id="userlist">
          <div id="textarea">
            <div id="button">
              <div id="button">
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

- index.html

```
function onMessage(evt) {
    var line = "";
    var msg = JSON.parse(evt.data);
    if (msg.type === "chat") {
        line = msg.name + ": ";
        if (msg.target.length > 0)
            line += "@" + msg.target + " ";
        line += msg.message + "\n";
        textarea.value += "" + line;
    } else if (msg.type === "info") {
        line = "[--" + msg.info + "--]\n";
        textarea.value += "" + line;
    } else if (msg.type === "users") {
        line = "Users:\n";
        for (var i=0; i < msg.userlist.length; i++)
            line += "-" + msg.userlist[i] + "\n";
        userlist.value = line;
    }
    textarea.scrollTop = 999999;
    wsconsole.value += "-> " + evt.data + "\n";
    wsconsole.scrollTop = 999999;
}
```


- index.html

```
function sendJoin() {  
    var input = document.getElementById("input");  
    var name = document.getElementById("name");  
    var join = document.getElementById("join");  
    var jsonstr;  
    if (name.value.length > 0) {  
        var joinMsg = {};  
        joinMsg.type = "join";  
        joinMsg.name = name.value;  
        jsonstr = JSON.stringify(joinMsg);  
        wsocket.send(jsonstr);  
        name.disabled = true;  
        join.disabled = true;  
        input.disabled = false;  
        userName = name.value;  
        wsconsole.value += "<- " + jsonstr + "\n";  
        wsconsole.scrollTop = 999999;  
    }  
}
```

- index.html

```
function sendMessage(evt) {  
    var input = document.getElementById("input");  
    var jsonstr;  
    var msgstr;  
    if (evt.keyCode === 13 && input.value.length > 0) {  
        var chatMsg = {};  
        chatMsg.type = "chat";  
        chatMsg.name = userName;  
        msgstr = input.value;  
        chatMsg.target = getTarget(msgstr.replace(/,/g, ""));  
        chatMsg.message = cleanTarget(msgstr);  
        chatMsg.message =  
            chatMsg.message.replace(/(\r\n|\n|\r)/gm, "");  
        jsonstr = JSON.stringify(chatMsg);  
        wsocket.send(jsonstr);  
        input.value = "";  
        wsconsole.value += "<- " + jsonstr + "\n";  
        wsconsole.scrollTop = 999999;  
    }  
}
```

- index.html

```
function checkJoin(evt) {
    var name = document.getElementById("name");
    var input = document.getElementById("input");
    if (evt.keyCode === 13 && name.value.length > 0) {
        sendJoin();
        input.focus();
    }
}

function getTarget(str) {
    var arr = str.split(" ");
    var target = "";
    for (var i=0; i<arr.length; i++) {
        if (arr[i].charAt(0) === '@') {
            target = arr[i].substring(1,arr[i].length);
            target = target.replace(/(\r\n|\n|\r)/gm,"");
        }
    }
    return target;
}
```

- index.html

```
function cleanTarget(str) {
    var arr = str.split(" ");
    var cleanstr = "";
    for (var i=0; i<arr.length; i++) {
        if (arr[i].charAt(0) !== '@')
            cleanstr += arr[i] + " ";
    }
    return cleanstr.substring(0,cleanstr.length-1);
}
function showHideConsole() {
    var chkbox = document.getElementById("showhideconsole");
    var consolediv = document.getElementById("consolediv");
    if (chkbox.checked)
        consolediv.style.visibility = 'visible';
    else
        consolediv.style.visibility = 'hidden';
}
window.addEventListener("load", connect, false);
</script>
</head>
```

- index.html

```
<body>
  <h1>WebsocketBot</h1>
  Your name: <input id="name" type="text" size="20" maxlength="20"
              onkeyup="checkJoin(event);"/>
  <input type="submit" id="join" value="Join!"
          onclick="sendJoin();"/><br/>
  <textarea id="input" cols="70" rows="1" disabled="true"
            onkeyup="sendMessage(event);"></textarea><br/>
  <textarea id="textarea" cols="70" rows="20"
            readonly="true"></textarea>
  <textarea id="userlist" cols="20" rows="20"
            readonly="true"></textarea>
  <br/><br/><br/>
  <input id="showhideconsole" type="checkbox"
        onclick="showHideConsole();"/>
  Show WebSocket console<br/>
  <div id="consolediv"><textarea id="wsconsole" cols="80" rows="8"
                                readonly="true" style="font-size:8pt;"></textarea></div>
</body>
</html>
```

- BotEndPoint.java

```
@ServerEndpoint(  
    value = "/websocketbot",  
    decoders = { MessageDecoder.class },  
    encoders = { JoinMessageEncoder.class, ChatMessageEncoder.class,  
                 InfoMessageEncoder.class, UsersMessageEncoder.class }  
)  
public class BotEndpoint {  
    private static final Logger logger = Logger.getLogger("BotEndpoint");  
  
    @OnOpen  
    public void openConnection(Session session) {  
        logger.log(Level.INFO, "Connection opened.");  
    }  
}
```

- BotEndPoint.java

```
@OnMessage
public void message(final Session session, Message msg) {
    if (msg instanceof JoinMessage) {
        JoinMessage jmsg = (JoinMessage) msg;
        session.getUserProperties().put("name", jmsg.getName());
        session.getUserProperties().put("active", true);
        logger.log(Level.INFO, "Received: {0}", jmsg.toString());
        sendAll(session, new InfoMessage(jmsg.getName() + " has
            joined the chat"));
        sendAll(session, new ChatMessage("Duke", jmsg.getName(),
            "Hi there!!"));
        sendAll(session, new UsersMessage(this.getUserList(session)));
    } else if (msg instanceof ChatMessage) {
        final ChatMessage cmsg = (ChatMessage) msg;
        logger.log(Level.INFO, "Received: {0}", cmsg.toString());
        sendAll(session, cmsg);
    }
}
```

- BotEndPoint.java

```
@OnClose
public void closedConnection(Session session) {
    session.getUserProperties().put("active", false);
    if (session.getUserProperties().containsKey("name")) {
        String name =
            session.getUserProperties().get("name").toString();
        sendAll(session, new InfoMessage(name +
            " has left the chat"));
        sendAll(session, new UsersMessage(this.getUserList(session)));
    }
    logger.log(Level.INFO, "Connection closed.");
}

@OnError
public void error(Session session, Throwable t) {
    logger.log(Level.INFO, "Connection error ({0})", t.toString());
}
```


- BotEndPoint.java

```
public synchronized void sendAll(Session session, Object msg) {
    try {
        for (Session s : session.getOpenSessions()) {
            if (s.isOpen()) {
                s.getBasicRemote().sendObject(msg);
                logger.log(Level.INFO, "Sent: {0}", msg.toString());
            }
        }
    } catch (IOException | EncodeException e) {
        logger.log(Level.INFO, e.toString());
    }
}

public List<String> getUserList(Session session) {
    List<String> users = new ArrayList<>();
    for (Session s : session.getOpenSessions()) {
        if (s.isOpen() && (boolean) s.getUserProperties().get("active"))
            users.add(s.getUserProperties().get("name").toString());
    }
    return users;
}
```

- Message.java

```
public class Message {}
```

- ChatMessage.java

```
public class ChatMessage extends Message {
```

```
    private String name;
```

```
    private String target;
```

```
    private String message;
```

```
    public ChatMessage(String name, String target, String message) {
```

```
        this.name = name;
```

```
        this.target = target;
```

```
        this.message = message;
```

```
    }
```

```
    .....
```

```
    public String getMessage() { return message; }
```

```
    public void setMessage(String message) { this.message = message; }
```

```
    .....
```

```
}
```

- UserMessage.java

```
public class Message {}  
public class UsersMessage extends Message {  
    private List<String> userlist;  
  
    public UsersMessage(List<String> userlist) {  
        this.userlist = userlist;  
    }  
  
    public List<String> getUserList() { return userlist; }  
    .....  
}
```

- JoinMessage.java

```
public class JoinMessage extends Message {  
    private String name;  
    public JoinMessage(String name) { this.name = name; }  
    public String getName() { return name; }  
    .....  
}
```

- InfoMessage.java

```
public class InfoMessage extends Message {  
  
    private String info;  
  
    public InfoMessage(String info) {  
        this.info = info;  
    }  
  
    public String getInfo() {  
        return info;  
    }  
  
    /* For logging purposes */  
    @Override  
    public String toString() {  
        return "[InfoMessage] " + info;  
    }  
}
```

- ChatMessageEncoder.java

```
public class ChatMessageEncoder implements Encoder.Text<ChatMessage> {

    @Override
    public void init(EndpointConfig ec) { }

    @Override
    public void destroy() { }

    @Override
    public String encode(ChatMessage chatMessage) throws EncodeException
    {
        StringWriter swriter = new StringWriter();
        try (JsonGenerator jsonGen = Json.createGenerator(swriter)) {
            jsonGen.writeStartObject()
                .write("type", "chat")
                .write("name", chatMessage.getName())
                .write("target", chatMessage.getTarget())
                .write("message", chatMessage.getMessage())
                .writeEnd();
        }
        return swriter.toString();
    }
}
```

- JoinMessageEncoder.java

```
public class JoinMessageEncoder implements Encoder.Text<JoinMessage> {  
    @Override  
    public void init(EndpointConfig ec) { }  
  
    @Override  
    public void destroy() { }  
  
    @Override  
    public String encode(JoinMessage joinMessage) throws EncodeException  
    {  
        StringWriter swriter = new StringWriter();  
        try (JsonGenerator jsonGen = Json.createGenerator(swriter)) {  
            jsonGen.writeStartObject()  
                .write("type", "join")  
                .write("name", joinMessage.getName())  
                .writeEnd();  
        }  
        return swriter.toString();  
    }  
}
```

- InfoMessageEncoder.java

```
public class InfoMessageEncoder implements Encoder.Text<InfoMessage> {  
    @Override  
    public void init(EndpointConfig ec) { }  
  
    @Override  
    public void destroy() { }  
  
    @Override  
    public String encode(InfoMessage joinMessage) throws EncodeException  
    {  
        StringWriter swriter = new StringWriter();  
        try (JsonGenerator jsonGen = Json.createGenerator(swriter)) {  
            jsonGen.writeStartObject()  
                .write("type", "info")  
                .write("info", joinMessage.getInfo())  
                .writeEnd();  
        }  
        return swriter.toString();  
    }  
}
```

- UsersMessageEncoder.java

```
public class UsersMessageEncoder implements Encoder.Text<UsersMessage> {  
    @Override  
    public void init(EndpointConfig ec) { }  
  
    @Override  
    public void destroy() { }  
  
    @Override  
    public String encode(UsersMessage usersMessage) throws  
        EncodeException {  
        StringWriter swriter = new StringWriter();  
        try (JsonGenerator jsonGen = Json.createGenerator(swriter)) {  
            jsonGen.writeStartObject()  
                .write("type", "users")  
                .writeStartArray("userlist");  
            for (String user : usersMessage.getUserList())  
                jsonGen.write(user);  
            jsonGen.writeEnd().writeEnd();  
        }  
        return swriter.toString();  
    }  
}
```


- MessageDecoder.java

```
public class MessageDecoder implements Decoder.Text<Message> {
    private Map<String,String> messageMap;

    @Override
    public void init(EndpointConfig ec) { }

    @Override
    public void destroy() { }

    /* Create a new Message object if the message can be decoded */
    @Override
    public Message decode(String string) throws DecodeException {
        Message msg = null;
        if (willDecode(string)) {
            switch (messageMap.get("type")) {
                case "join":
                    msg = new JoinMessage(messageMap.get("name"));
                    break;
                case "chat":
                    msg = new ChatMessage(messageMap.get("name"),
                                           messageMap.get("target"),
                                           messageMap.get("message"));
            }
        } else {
            throw new DecodeException(string, "[Message] Can't decode.");
        }
        return msg;
    }
}
```

- MessageDecoder.java

```
@Override
public boolean willDecode(String string) {
    boolean decodes = false;
    messageMap = new HashMap<>();
    JsonParser parser = Json.createParser(new StringReader(string));
    while (parser.hasNext()) {
        if (parser.next() == JsonParser.Event.KEY_NAME) {
            String key = parser.getString();
            parser.next();
            String value = parser.getString();
            messageMap.put(key, value);
        }
    }

    Set keys = messageMap.keySet();
    if (keys.contains("type")) {
        switch (messageMap.get("type")) {
            case "join":
                if (keys.contains("name"))
                    decodes = true;
                break;
            case "chat":
                String[] chatMsgKeys = {"name", "target", "message"};
                if (keys.containsAll(Arrays.asList(chatMsgKeys)))
                    decodes = true;
                break;
        }
    }
    return decodes;
}
```

- BotEndpoint.java

```
@ServerEndpoint(
    value = "/websocketbot",
    decoders = { MessageDecoder.class },
    encoders = { JoinMessageEncoder.class, ChatMessageEncoder.class,
                 InfoMessageEncoder.class, UsersMessageEncoder.class }
)
public class BotEndpoint {

    @OnOpen
    public void openConnection(Session session) {
        logger.log(Level.INFO, "Connection opened.");
    }

    @OnMessage
    public void message(final Session session, Message msg) {
        logger.log(Level.INFO, "Received: {0}", msg.toString());

        if (msg instanceof JoinMessage) {
            JoinMessage jmsg = (JoinMessage) msg;
            session.getUserProperties().put("name", jmsg.getName());
            session.getUserProperties().put("active", true);
            logger.log(Level.INFO, "Received: {0}", jmsg.toString());
            sendAll(session, new InfoMessage(jmsg.getName() + " has joined the chat"));
            sendAll(session, new ChatMessage("Duke", jmsg.getName(), "Hi there!!"));
            sendAll(session, new UsersMessage(this.getUserList(session)));
        } else if (msg instanceof ChatMessage) {
            final ChatMessage cmsg = (ChatMessage) msg;
            logger.log(Level.INFO, "Received: {0}", cmsg.toString());
            sendAll(session, cmsg);
        }
    }
}
```

- BotEndpoint.java

```
@OnClose
public void closedConnection(Session session) {
    session.getUserProperties().put("active", false);
    if (session.getUserProperties().containsKey("name")) {
        String name = session.getUserProperties().get("name").toString();
        sendAll(session, new InfoMessage(name + " has left the chat"));
        sendAll(session, new UsersMessage(this.getUserList(session)));
    }
    logger.log(Level.INFO, "Connection closed.");
}

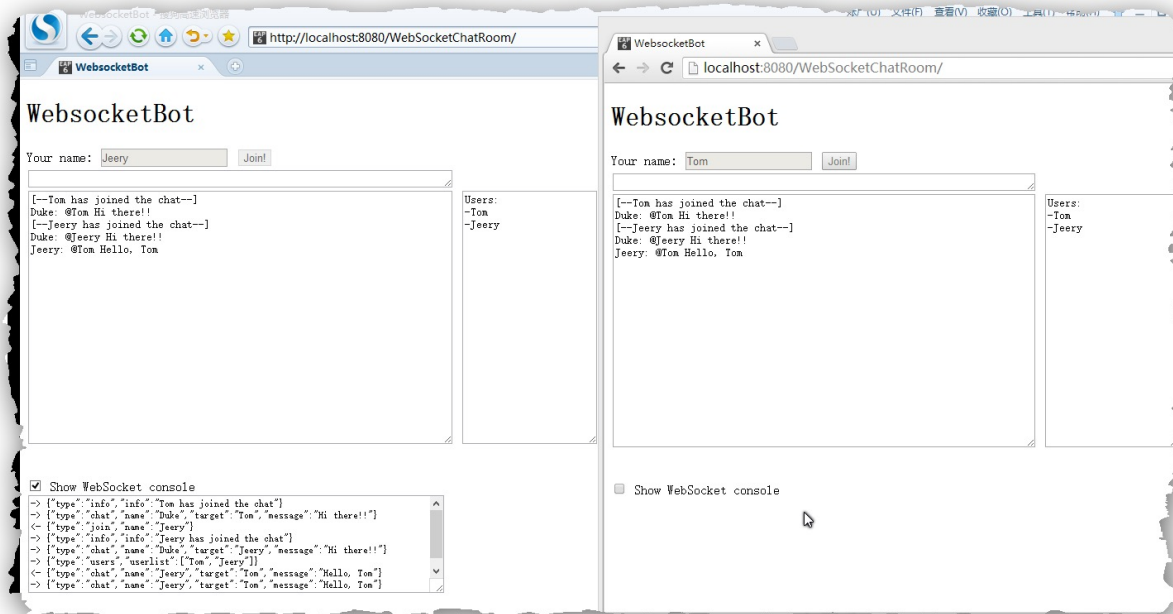
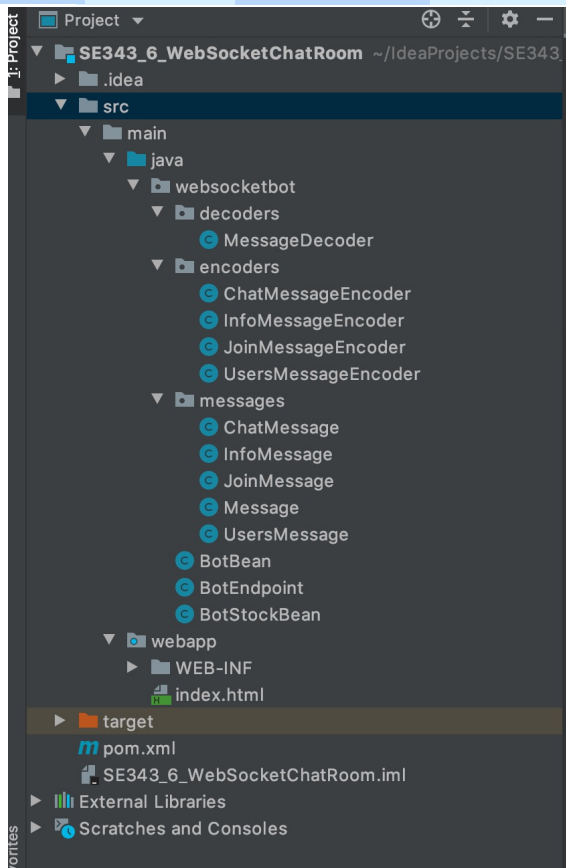
@OnError
public void error(Session session, Throwable t) {
    logger.log(Level.INFO, "Connection error ({0})", t.toString());
}

public synchronized void sendAll(Session session, Object msg) {
    try {
        for (Session s : session.getOpenSessions()) {
            if (s.isOpen()) {
                s.getBasicRemote().sendObject(msg);
                logger.log(Level.INFO, "Sent: {0}", msg.toString());
            }
        }
    } catch (IOException | EncodeException e) {
        logger.log(Level.INFO, e.toString());
    }
}
```

- BotEndpoint.java

```
public List<String> getUserList(Session session) {  
    List<String> users = new ArrayList<>();  
    for (Session s : session.getOpenSessions()) {  
        if (s.isOpen() && (boolean) s.getUserProperties().get("active"))  
            users.add(s.getUserProperties().get("name").toString());  
    }  
    return users;  
}  
}
```

An example - Chatroom



- 请你在大二开发的E-Book系统的基础上，完成下列任务：
 1. 编写一个WebSocket聊天室，用于用户在线聊天，具体功能参考上课案例。
 - 你应该将上述功能集成到你的E-Book系统中，如果你无法将上述2项任务集成到你的E-Book系统中，可以单独建立工程实现，但是会适当扣分。
 - 请将你编写的相关代码整体压缩后上传，请勿压缩整个工程提交。
- 评分标准：
 1. 聊天室代码运行正常，且能够集成到E-Book中 (3分)

- The Java EE 8 Tutorial
 - <https://javaee.github.io/tutorial/toc.html>
- Java API for WebSocket
 - <https://javaee.github.io/tutorial/websocket.html#GKJIQ5>



Thank You!