7.4 .Netty 初认识 --websocket 服务端开发示例

作者：289306290
原文链接：https://ld246.com/article/1583405039163
来源网站：链滴

示例效果图:

文本框内输入文字，点击发送消息，服务端会收到内容，加上固定后缀话术输出到最下面。

WebSocketServer.java 内容如下:

```
package club.wujingjian.com.wujingjian.netty.websocket;

import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.Channel;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelPipeline;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioServerSocketChannel;
import io.netty.handler.codec.http.HttpObjectAggregator;
import io.netty.handler.codec.http.HttpServerCodec;
import io.netty.handler.stream.ChunkedWriteHandler;

public class WebSocketServer {
    public void run(int port) throws  Exception {
        EventLoopGroup bossGroup = new NioEventLoopGroup();
        EventLoopGroup workerGroup = new NioEventLoopGroup();
        try {
            ServerBootstrap b = new ServerBootstrap();
            b.group(bossGroup, workerGroup)
                    .channel(NioServerSocketChannel.class)
                    .childHandler(new ChannelInitializer<SocketChannel>() {

                        @Override
```

```
                protected void initChannel(SocketChannel socketChannel) throws Exception {
                    ChannelPipeline pipeline = socketChannel.pipeline();
                    //首先添加HttpServerCodec，将请求和应答消息编码或者解码为HTTP消息；
                    pipeline.addLast("http-codes", new HttpServerCodec())
                            //增加HttpObjectAggregator，它的目的是将HTTP消息的多个部分组合成
条完整的HTTP消息；
                            .addLast("aggregator", new HttpObjectAggregator(65536))
                            //添加ChunkedWriteHandler，来向客户端发送HTML5文件，
                            //它主要用于支持浏览器和服务端进行WebSocket通信；
                            .addLast("http-chunked", new ChunkedWriteHandler())
                            //增加WebSocket服务端handler。
                            .addLast("handler", new WebSocketServerHandler());
                }
            });
        Channel ch = b.bind(port).sync().channel();
        System.out.println("Web socket server started at port: " + port);
        System.out.println("Open your browser and navigate to http://localhost:" + port + "/");
        ch.closeFuture().sync();
    } finally {
        bossGroup.shutdownGracefully();
        workerGroup.shutdownGracefully();
    }
}

public static void main(String[] args) throws Exception {
    int port= 8080;
    new WebSocketServer().run(port);
}
}
```

WebSocketServerHandler.java

```
package club.wujingjian.com.wujingjian.netty.websocket;

import io.netty.buffer.ByteBuf;
import io.netty.buffer.Unpooled;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelFutureListener;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.SimpleChannelInboundHandler;
import io.netty.handler.codec.http.*;
import io.netty.handler.codec.http.websocketx.*;
import io.netty.util.CharsetUtil;

import java.util.Date;

import static io.netty.handler.codec.http.HttpHeaders.Names.CONTENT_LENGTH;

public class WebSocketServerHandler extends SimpleChannelInboundHandler<Object> {

    private WebSocketServerHandshaker handshaker;

    @Override
```

```
    protected void messageReceived(ChannelHandlerContext ctx, Object msg) throws Exceptio
{
        //传统的HTTP接入
        //第一次握手请求消息由HTTP协议承载，所以它是一个HTTP消息，执行handleHttpRequest
法来处理WebSocket握手请求。
        if (msg instanceof FullHttpRequest) {
            handleHttpRequest(ctx, (FullHttpRequest) msg);
        }
        //WebSocket接入
        // 客户端通过文本框提交请求消息给服务端，WebSocketServerHandler接收到的是已经解码
的WebSocketFrame消息。
        if (msg instanceof WebSocketFrame) {
            handleWebSocketFrame(ctx, (WebSocketFrame) msg);
        }
    }

    @Override
    public void channelReadComplete(ChannelHandlerContext ctx) throws Exception {
        ctx.flush();
    }

    private void handleHttpRequest(ChannelHandlerContext ctx, FullHttpRequest req) throws E
ception {
        //如果HTTP解码失败,返回HTTP异常
        // 首先对握手请求消息进行判断，如果消息头中没有包含Upgrade字段或者它的值不是websoc
et，则返回HTTP 400响应。
        if (!req.getDecoderResult().isSuccess()
                || (!"websocket".equals(req.headers().get("Upgrade")))) {
            sendHttpResponse(ctx, req, new DefaultFullHttpResponse(HttpVersion.HTTP_1_1, Htt
ResponseStatus.BAD_REQUEST));
            return;
        }

        //构造握手响应返回,本机测试
        // 握手请求简单校验通过之后，开始构造握手工厂，
        // 创建握手处理类WebSocketServerHandshaker，
        WebSocketServerHandshakerFactory wsFactory = new WebSocketServerHandshakerFact
ry(
                "ws://localhost:8080/websocket", null, false);
        handshaker = wsFactory.newHandshaker(req);
        if (handshaker == null) {
            WebSocketServerHandshakerFactory.sendUnsupportedWebSocketVersionResponse(ct
.channel());
        } else {
            // 通过它构造握手响应消息返回给客户端，
            // 同时将WebSocket相关的编码和解码类动态添加到ChannelPipeline中，用于WebSocket
息的编解码，(哪里添加到pipeline了???)
            // 添加WebSocketEncoder和WebSocketDecoder之后，服务端就可以自动对WebSocket
息进行编解码了
            handshaker.handshake(ctx.channel(), req);
        }
    }

    private void handleWebSocketFrame(ChannelHandlerContext ctx, WebSocketFrame frame)
```

```
        //判断是否是关闭链路的命令
        // 首先需要对控制帧进行判断，如果是关闭链路的控制消息
        // 就调用WebSocketServerHandshaker的close方法关闭WebSocket连接；
        if (frame instanceof CloseWebSocketFrame) {
            handshaker.close(ctx.channel(), (CloseWebSocketFrame) frame.retain());
            return;
        }
        //判断是否是ping消息
        // 如果是维持链路的Ping消息，则构造Pong消息返回。
        if (frame instanceof PingWebSocketFrame) {
            ctx.channel().write(new PongWebSocketFrame(frame.content().retain()));
            return;
        }
        //本例程仅支持文本消息,不支持二进制消息
        // WebSocket通信双方使用的都是文本消息，所以对请求消息的类型进行判断，不是文本的抛
异常。
        if (!(frame instanceof TextWebSocketFrame)) {
            throw new UnsupportedOperationException(String.format("%s frame type not suppor
ed",
                    frame.getClass().getName()));
        }
        //返回应答消息
        // 从TextWebSocketFrame中获取请求消息字符串，
        String request = ((TextWebSocketFrame) frame).text();
        // 对它处理后通过构造新的TextWebSocketFrame消息返回给客户端，
        // 由于握手应答时动态增加了TextWebSocketFrame的编码类，所以，可以直接发送TextWeb
ocketFrame对象。
        System.out.println(String.format("%s received %s", ctx.channel(), request));
        ctx.channel().write(new TextWebSocketFrame(request + ",欢迎使用Netty WebSocket服务
现在时刻:" + new Date().toString()));
    }

    private static void sendHttpResponse(ChannelHandlerContext ctx, FullHttpRequest req, Ful
HttpResponse res) {
        //返回应答给客户端
        if (res.getStatus().code() != 200) {
            ByteBuf buf = Unpooled.copiedBuffer(res.getStatus().toString(), CharsetUtil.UTF_8);
            res.content().writeBytes(buf);
            buf.release();
            setContentLength(res, res.content().readableBytes());
        }
        //如果是非Keep-alive,关闭连接
        ChannelFuture f = ctx.channel().writeAndFlush(res);
        if (!isKeepAlive(req) || res.getStatus().code() != 200) {
            f.addListener(ChannelFutureListener.CLOSE);
        }
    }

    private static void setContentLength(HttpResponse response,long length){
        response.headers().set(CONTENT_LENGTH, length);
    }

    private static boolean isKeepAlive(FullHttpRequest fullHttpRequest) {
```

```
        if(HttpHeaders.isKeepAlive(fullHttpRequest)){
            return true;
        }
        return false;
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Excepti
n {
        cause.printStackTrace();;
        ctx.close();
    }
}
```

页面websocket.html如下:

```
<html>
<head>
    <meta charset="utf-8">
    Netty WebSocket 时间服务器
</head>
<br>
<body>
<br>
<script type="text/javascript">
    var socket;

    if(window.WebSocket) {
        socket = new WebSocket("ws://localhost:8080/websocket");
        socket.onmessage = function(event){
            var ta = document.getElementById('responseText');
            ta.value = "";
            ta.value = event.data;
        };
        socket.onopen = function(event){
            var ta = document.getElementById('responseText');
            ta.value = "打开websocket服务正常,浏览器支持";
        };
        socket.onclose = function(event){
            var ta = document.getElementById('responseText');
            ta.value = "";
            ta.value = "websocket关闭";
        };
    }else{
        alert("抱歉,你的浏览器不支持websocket协议");
    }

    function send(message){
        if(!window.WebSocket){
            return;
        }
        if(socket.readyState==WebSocket.OPEN){
            socket.send(message);
```

```
        }else {
            alert("websocket连接没有建立成功")
        }
    }
</script>
<form>
    <input type="text" name="message" value="Netty WebSocket"/>
    <br><br>
    <input type="button" value="发送消息" onclick="send(this.form.message.value)">
    <hr color="blue"/>
    <h3>服务器返回的应答消息</h3>
    <textarea id="responseText" style="width: 500px;height: 300px;"></textarea>
</form>
</body>
</html>
```