cmsen-common

Java Project development common package

SocketClient

package com.cmsen.common.socket

```
// 单元测试用例
public class SocketClientTest {
    private SocketClient client;

    public static void main(String[] args) {
        SocketClientTest socketClientTest = new SocketClientTest();
        socketClientTest.create();
        // 模拟发送一个消息 - 使用定时发送
        new Timer().schedule(new TimerTask() {
            @Override
            public void run() {
                socketClientTest.API();
            }
        }, 3000);
    }

    private void create() {
        // 创建一个客户端
        client = new SocketClient();
        // 设置连接服务端IP, 默认127.0.0.1
        client.setServerIp("127.0.0.1");
        // 设置连接服务端端口, 默认5209
        client.setServerPort(2000);
            // 设置阻塞模式, 默认非阻塞模式
            // client.setBlockingMode(true);
        // 设置持久的连接
        client.setKeepAlive(true);
            // 设置重置等待时间, 默认3000ms
            // client.setResetWaitTime(3000);
            // 设置发送缓冲区字节大小, 默认10个字节
            // client.setSendBufferSize(10);
            // 设置接收缓冲区字节大小, 默认8个字节
            // client.setReceiveBufferSize(8);
            // 设置消息队列大小, 默认100个
            // client.setMessageQueueSize(100);
        // 添加消息事件监听, 见SocketMessageTest类
        client.setSocketMessage(SocketMessageTest.class);
        // 打开客户端
        client.open();
    }

    // 模拟接口调用发送消息
    private void API() {
        // 消息队列 按照先进先出原则推送
        client.setMessage(SocketMessageTest.MESSAGES_INT)
                // 立即发送
                .send();
    }
}


// --------- SocketMessageTest.class
public class SocketMessageTest implements SocketMessage {
    /**
     * 监听端口
     */
    public static int PORT = 2000;
```

```
        /**
         * 监听主机
         */
        public static String HOST = "127.0.0.1";

        /**
         * 消息正文
         * 2521234253 = [00 FC 00 01 00 02 00 03 00 FD]
         */
        public static int[] MESSAGES_INT = {252, 1, 2, 3, 253};


        public SocketMessageTest() {
        }

        // 心跳检测间隔, 默认0-关闭心跳检测
        @Override
        public long heartRate() {
            // 0-关闭心跳检测
            return 0;
        }

        // 心跳检测数据, 默认0xff
        @Override
        public int heartRateData() {
            return 0xff;
        }

        // 心跳检测错误, 默认false
        @Override
        public boolean heartRateError(Exception e) {
            // true=启动自动重连
            return true;
        }

        // 完成连接时
        @Override
        public void finishConnect(Socket socket) {
            // 可完成些其他方法在完成连接时调用
        }

        // 连接错误, 默认false, true=连接失败自动重连
        @Override
        public boolean connectError(Exception e) {
            // true=启动自动重连
            return true;
        }

        // 消息发送时
        @Override
        public void send(ByteBuffer buffer) throws Exception {
            System.err.print(String.format("Send message: [ length: %d, hex: %s]", buffer.limit(),
ByteUtil.toHex(buffer.array())));
        }

        // 消息发送时错误打印, 默认false, true=发送失败自动重连
        @Override
        public boolean sendError(Exception e) {
            System.err.println("Sending message error: " + e.getMessage());
            return true;
        }

        // 消息接收时
        @Override
        public void receive(ByteBuffer buffer) {
            System.err.print(String.format("Received message: [ length: %d, hex: %s]", buffer.limit(),
ByteUtil.toHex(buffer.array())));
        }
```

```
    // 消息接收时错误打印，默认false，true=尝试重新接收
    @Override
    public boolean receiveError(Exception e) {
        System.err.println("Receiving message error: " + e.getMessage());
        return false;
    }
}
```