# 基本socket（短连接）

## Sender（发送）

|  |
| --- |
| package com.hengbao.sptsm.service.socket.impl;  import java.io.DataOutputStream;  import java.io.IOException;  import java.io.InputStream;  import java.net.Socket;  import com.hengbao.sptsm.service.socket.ISocketSender;  public class SocketSenderDefault implements ISocketSender{  @Override  public String doSend(String ip, int port, String msg) throws Exception {  Socket socket = null;  DataOutputStream out = null;  InputStream input = null;  String ret = "";  try {  socket = new Socket(ip, port); // 创建一个流套接字并将其连接到指定主机上的指定端口号  input = socket.getInputStream(); // 读取服务器端数据  out = new DataOutputStream(socket.getOutputStream()); // 向服务器端发送数据  out.write(msg.getBytes());  socket.shutdownOutput();  StringBuilder sb = new StringBuilder();  byte[] result = new byte[1024];  int len = 0;  while ((len = input.read(result)) != -1) {  sb.append(new String(result, 0, len));  }  ret = sb.toString();  } catch (Exception e) {  throw e;  } finally {  if (socket != null) {  try {  socket.close();  } catch (IOException e) {  socket = null;  throw e;  }  }  if (out != null) {  try {  out.close();  } catch (IOException e) {  throw e;  }  }  if (input != null) {  try {  input.close();  } catch (IOException e) {  input = null;  throw e;  }  }  }  return ret;  }  } |

## Receiver（接收）

|  |
| --- |
| package com.hengbao.sptsm.service.socket.impl;  import java.io.DataOutputStream;  import java.io.IOException;  import java.io.InputStream;  import java.net.ServerSocket;  import java.net.Socket;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;  import javax.annotation.PostConstruct;  import javax.annotation.PreDestroy;  import org.apache.log4j.Logger;  import org.springframework.beans.factory.annotation.Autowired;  import com.hengbao.sptsm.service.msg.DispatcherService;  import com.hengbao.sptsm.service.socket.SocketService;  import com.hengbao.sptsm.util.ConfigUtil;  public class SocketServiceDefault implements SocketService {  public static Logger log = Logger.getLogger(SocketServiceDefault.class);  @Autowired  private DispatcherService dispatcherService;  private ServerSocket serverSocket = null;  private ExecutorService exec = Executors.newCachedThreadPool();  @Override  public void recieveSocket() {  try {  serverSocket = new ServerSocket(Integer.valueOf(ConfigUtil.getProperties("serverSocket")));  while (true) {  Socket client = serverSocket.accept(); // 一旦有堵塞, 则表示服务器与客户端获得了连接  exec.execute(new HandlerThread(client)); // 处理这次连接  }  } catch (Exception e) {  log.error("serverSocket服务异常:", e);  }  }  private class HandlerThread implements Runnable {  private Socket socket;  public HandlerThread(Socket client) {  socket = client;  }  public void run() {  DataOutputStream out = null;  InputStream input = null;  String resMsg = "";  String reqMsg = "";  try {  input = socket.getInputStream(); // 读取客户端数据    StringBuilder sb = new StringBuilder();  byte[] result = new byte[1024];  int len = 0;  while((len=input.read(result))!=-1){  sb.append(new String(result,0,len));  }  reqMsg = sb.toString();  // 处理客户端数据  resMsg = dispatcherService.dispatchRequest(reqMsg);  // 向客户端回复信息  out = new DataOutputStream(socket.getOutputStream());  out.write(resMsg.getBytes());  socket.shutdownOutput();  } catch (Exception e) {  log.error("sockerService处理报文时异常", e);  } finally {  if (socket != null) {  try {  socket.close();  } catch (Exception e) {  socket = null;  log.error("sockerService处理报文时 finally 异常", e);  }  }  if (out != null) {  try {  out.close();  } catch (Exception e) {  log.error("sockerService处理报文时 finally 异常", e);  }  }  if (input != null) {  try {  input.close();  } catch (Exception e) {  log.error("sockerService处理报文时 finally 异常", e);  }  }  }  }  }  @PreDestroy  public void destory() {  if (serverSocket != null) {  try {  serverSocket.close();  } catch (IOException e) {  log.error(e);  }  }  if (exec != null) {  try {  exec.shutdown();  } catch (Exception e) {  log.error(e);  }  }  }  @PostConstruct  public void init() {  new Thread() {  @Override  public void run() {  recieveSocket();  }  }.start();  }  } |

# mina（一种socket框架）

## Sender（发送）

|  |
| --- |
| package com.hengbao.sptsm.service.socket.impl;  import java.net.InetSocketAddress;  import java.util.concurrent.TimeUnit;  import org.apache.log4j.Logger;  import org.apache.mina.core.future.ReadFuture;  import org.apache.mina.core.session.IoSession;  import org.apache.mina.filter.codec.ProtocolCodecFilter;  import org.apache.mina.filter.codec.serialization.ObjectSerializationCodecFactory;  import org.apache.mina.transport.socket.nio.NioSocketConnector;  import com.hengbao.sptsm.service.socket.ISocketSender;  public class SocketSenderLs implements ISocketSender {  public static Logger log = Logger.getLogger(SocketSenderLs.class);    NioSocketConnector connector = null;  IoSession session = null;    public SocketSenderLs(){  connector = new NioSocketConnector();  connector.getFilterChain().addLast("codec", new ProtocolCodecFilter(new ObjectSerializationCodecFactory()));  connector.getSessionConfig().setUseReadOperation(true);  }    @Override  public String doSend(String ip, int port, String msg) throws Exception {  if(this.session==null || this.session.isBothIdle() || this.session.isClosing()){  session = connector.connect(new InetSocketAddress(ip, port)) .awaitUninterruptibly().getSession();  }  log.info("#### sptsm the session in doSend method is : "+session);  String ret = "";  session.write(msg).awaitUninterruptibly();  ReadFuture readFuture = session.read();  if (readFuture.awaitUninterruptibly(20, TimeUnit.SECONDS)) {  ret = (String)readFuture.getMessage();  } else {  throw new Exception("SocketSenderLs send message failed ! time out , 20 seconds has passed after request message send ! ");  }  return ret;  }  } |

## Receiver（接收）

|  |
| --- |
| package com.hengbao.sptsm.service.socket.impl;  import java.net.InetSocketAddress;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;  import javax.annotation.PostConstruct;  import javax.annotation.PreDestroy;  import org.apache.log4j.Logger;  import org.apache.mina.core.service.IoHandlerAdapter;  import org.apache.mina.core.session.IdleStatus;  import org.apache.mina.core.session.IoSession;  import org.apache.mina.filter.codec.ProtocolCodecFilter;  import org.apache.mina.filter.codec.serialization.ObjectSerializationCodecFactory;  import org.apache.mina.filter.executor.ExecutorFilter;  import org.apache.mina.transport.socket.nio.NioSocketAcceptor;  import org.springframework.beans.factory.annotation.Autowired;  import com.hengbao.sptsm.service.msg.DispatcherService;  import com.hengbao.sptsm.service.socket.SocketService;  import com.hengbao.sptsm.util.ConfigUtil;  public class SocketServiceLs implements SocketService{  public static Logger log = Logger.getLogger(SocketServiceLs.class);    @Autowired  private DispatcherService dispatcherService;  private ExecutorService threadPool = Executors.newCachedThreadPool();// 建立线程池  private NioSocketAcceptor acceptor = null;    @Override  public void recieveSocket() {  try {  acceptor = new NioSocketAcceptor();  acceptor.getFilterChain().addLast("codec", new ProtocolCodecFilter(new ObjectSerializationCodecFactory()));  acceptor.getFilterChain().addLast("exector", new ExecutorFilter(threadPool));  acceptor.setHandler(new Handler());  acceptor.bind(new InetSocketAddress((Integer.valueOf(ConfigUtil.getProperties("serverSocket")))));    log.info("#### sptsm SocketServiceLs startup ...... ");  } catch (Exception e) {  log.error("#### sptsm SocketServiceLs",e);  }  }  /\*\*  \* 接收请求  \*  \* @param ip  \* IP地址  \* @return 返回内容  \*/  private class Handler extends IoHandlerAdapter {    public void messageReceived(IoSession session, Object message)  throws Exception {  String reqMsg = (String)message;  try {  log.info("#### sptsm the session in messageReceived method is : "+session);  // 处理客户端数据  String resMsg = dispatcherService.dispatchRequest(reqMsg);  session.write(resMsg);  } catch (Exception e) {  log.error("#### sptsm SocketServiceLs receive error : ", e);  }  }    public void exceptionCaught(IoSession session, Throwable cause)  throws Exception {  log.error("SocketServiceLs.Handler.exception:", cause);  }  public void messageSent(IoSession session, Object message)  throws Exception {  log.debug("SocketServiceLs.Handler.mseeageSent");  }  public void sessionClosed(IoSession session) throws Exception {  session.close(true);  log.debug("SocketServiceLs.Handler.sessionClosed");  }  public void sessionCreated(IoSession session) throws Exception {  log.debug("SocketServiceLs.Handler.sessionCreated");  }  public void sessionIdle(IoSession session, IdleStatus status)  throws Exception {  session.close(true);  log.debug("SocketServiceLs.Handler.sessionIdle");  }  public void sessionOpened(IoSession session) throws Exception {  log.debug("SocketServiceLs.Handler.sessionOpened");  }  }    @PreDestroy  public void destory() {  if (threadPool != null) {  try {  threadPool.shutdown();  log.info("#### sptsm SocketServiceLs threadPool shutdown ");  } catch (Exception e) {  log.error(e);  }  }  if (acceptor != null) {  try {  acceptor.dispose();  log.info("#### sptsm SocketServiceLs acceptor dispose ");  } catch (Exception e) {  log.error(e);  }  }  }  @PostConstruct  public void init() {  new Thread() {  @Override  public void run() {  recieveSocket();  }  }.start();  }    } |