package com.b\_noble.n\_life\_final;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.lang.management.ManagementFactory;

import org.apache.commons.lang.StringUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.b\_noble.n\_life\_final.conf.ApplicationConfig;

import com.b\_noble.n\_life\_final.server.OperationLogBootstrap;

import com.b\_noble.n\_life\_final.server.RedisBootstrap;

/\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：程序启动入口；

\* 作者：zhangfan

\* 时间：2016-04-13

\* 版本：V0.0.1

\*/

public class NLifeRun {

private static final Logger LOG = LoggerFactory.getLogger(NLifeRun.class);

public static final Object forWait = new Object();

public static void main(String[] args) throws InterruptedException, IOException {

// 获取jvm名

String vm = ManagementFactory.getRuntimeMXBean().getName();

if (StringUtils.isBlank(vm)) {

LOG.error("系统不能获取到进程id，服务端启动失败");

return;

}

FileOutputStream out = null;

try {

// 将jvm进程id保存到pid文件

File pid = new File("pid");

if (pid.exists()) {

pid.mkdir();

}

out = new FileOutputStream(pid);

out.write(vm.split("@")[0].getBytes());

out.flush();

} finally {

if (out != null) {

out.close();

}

}

LOG.info("主程序开始启动 --"+vm);

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(ApplicationConfig.class);

context.start();

LOG.info("redis开始启动--"+vm);

RedisBootstrap rb = new RedisBootstrap();

rb.run();

LOG.info("MQ开始启动--"+vm);

OperationLogBootstrap olb = new OperationLogBootstrap();

olb.run();

synchronized (forWait) {

forWait.wait();

}

context.close();

}

}

Service.java

/\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：程序启动Service；

\* 作者：zhangfan

\* 时间：2016-04-13

\* 版本：V0.0.1

\*/

package com.b\_noble.n\_life\_final;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.Lifecycle;

import org.springframework.stereotype.Component;

import com.b\_noble.n\_life\_final.server.MyBootstrap;

@Component

public class Service implements Lifecycle {

private static final Logger LOG = LoggerFactory.getLogger(Service.class);

private final static ExecutorService threadPool = Executors.newCachedThreadPool();

private boolean started = false;

private Future future;

@Autowired

private MyBootstrap myBootstrap;

public void start() {

LOG.info("serverRun start");

started = true;

future = threadPool.submit(myBootstrap);

int sleepCount = 5;

while (!myBootstrap.isRun()) {

if (sleepCount < 0) {

future.cancel(true);

return;

}

try {

sleepCount--;

Thread.sleep(1000);

} catch (InterruptedException e) {

LOG.info("InterruptedException:", e);

e.printStackTrace();

}

}

}

public void stop() {

System.out.println("serverRun stop");

started = false;

threadPool.shutdown();

if (future != null) {

future.cancel(true);

}

}

public boolean isRunning() {

System.out.println("serverRun isRunning");

return started;

}

public boolean isInterface(Class clazz, Class infClazz) {

if (clazz == Object.class) {

return false;

}

Class[] ifaces = clazz.getInterfaces();

boolean trueType = false;

for (Class<?> iface : ifaces) {

if (iface.equals(infClazz)) {

trueType = true;

break;

}

}

if (trueType) {

return true;

}

clazz = clazz.getSuperclass();

return isInterface(clazz, infClazz);

}

}

MyBootstrap.java

package com.b\_noble.n\_life\_final.server;

import javax.annotation.Resource;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

import io.netty.bootstrap.ServerBootstrap;

import io.netty.buffer.PooledByteBufAllocator;

import io.netty.channel.Channel;

import io.netty.channel.ChannelFuture;

import io.netty.channel.ChannelInitializer;

import io.netty.channel.ChannelOption;

import io.netty.channel.EventLoopGroup;

import io.netty.channel.socket.SocketChannel;

import io.netty.channel.socket.nio.NioServerSocketChannel;

import io.netty.handler.logging.LogLevel;

import io.netty.handler.logging.LoggingHandler;

/\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：服务端启动类

\* 作者：zhangfan

\* 时间：2016-04-13

\* 版本：V0.0.1

\*/

@Component

public class MyBootstrap implements Runnable {

private static final Logger LOG = LoggerFactory.getLogger(MyBootstrap.class);

private volatile boolean run = false;

/\*\*

\* 逻辑处理线程

\*/

//public static ProcessRunnable processRunnable = new ProcessRunnable();

@Autowired

private ServerBootstrap serverBootstrap;

@Resource(name = "ServerChannelInit")

private ChannelInitializer<SocketChannel> serverChannelInit;

@Resource(name = "boss")

private EventLoopGroup boss;

@Resource(name = "worker")

private EventLoopGroup worker;

/\*\*

\* 本地tcp服务端口

\*/

@Value("${service.port}")

private String servicePort;

public void run() {

LOG.info("设置serverbootstrap");

// 设置工作线程池

serverBootstrap.group(boss, worker);

serverBootstrap.channel(NioServerSocketChannel.class);

serverBootstrap.handler(new LoggingHandler(LogLevel.INFO));

serverBootstrap.childHandler(serverChannelInit);

serverBootstrap.childOption(ChannelOption.WRITE\_BUFFER\_HIGH\_WATER\_MARK, 32 \* 1024);

serverBootstrap.childOption(ChannelOption.WRITE\_BUFFER\_LOW\_WATER\_MARK, 8 \* 1024);

serverBootstrap.childOption(ChannelOption.ALLOCATOR, PooledByteBufAllocator.DEFAULT);

serverBootstrap.childOption(ChannelOption.SO\_KEEPALIVE, true);

// 直接发包

serverBootstrap.childOption(ChannelOption.TCP\_NODELAY, true);

Channel ch;

try {

ChannelFuture bindf = serverBootstrap.bind(Integer.valueOf(servicePort));

ChannelFuture bsync = bindf.await();

ch = bsync.channel();

LOG.info("listen port:{}", servicePort);

run = true;

LOG.info("等待TCP结束...");

ch.closeFuture().sync();

} catch (InterruptedException e) {

LOG.info("interrupted", e);

} finally {

boss.shutdownGracefully();

worker.shutdownGracefully();

}

}

public boolean isRun() {

return run;

}

}

OperationLogBootstrap.java

package com.b\_noble.n\_life\_final.server;

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.DeliveryMode;

import javax.jms.Destination;

import javax.jms.MessageProducer;

import javax.jms.Session;

import org.apache.activemq.ActiveMQConnection;

import org.apache.activemq.ActiveMQConnectionFactory;

import com.b\_noble.n\_life\_final.utils.Global;

/\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：操作日志MQ启动类

\* 作者：zhangfan

\* 时间：2016-04-13

\* 版本：V0.0.1

\*/

public class OperationLogBootstrap implements Runnable {

public void run() {

// ConnectionFactory ：连接工厂，JMS 用它创建连接

ConnectionFactory connectionFactory;

// Connection ：JMS 客户端到JMS Provider 的连接

Connection connection = null;

// Session： 一个发送或接收消息的线程

Session session;

// Destination ：消息的目的地;消息发送给谁.

Destination destination;

// MessageProducer：消息发送者

MessageProducer producer;

// TextMessage message;

// 构造ConnectionFactory实例对象，此处采用ActiveMq的实现jar

connectionFactory = new ActiveMQConnectionFactory(ActiveMQConnection.DEFAULT\_USER,

ActiveMQConnection.DEFAULT\_PASSWORD, "tcp://192.168.1.140:61616");

try {

// 构造从工厂得到连接对象

connection = connectionFactory.createConnection();

// 启动

connection.start();

// 获取操作连接

session = connection.createSession(Boolean.TRUE, Session.AUTO\_ACKNOWLEDGE);

// 获取session注意参数值xingbo.xu-queue是一个服务器的queue，须在在ActiveMq的console配置

destination = session.createQueue("ztzfQueue");

// 得到消息生成者【发送者】

producer = session.createProducer(destination);

// 设置不持久化，此处学习，实际根据项目决定

producer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);

//设置全局值

Global.session = session;

Global.producer = producer;

} catch (Exception e) {

e.printStackTrace();

}

}

}

ServerChannelInit.java

package com.b\_noble.n\_life\_final.handler;

import java.nio.charset.Charset;

import org.springframework.stereotype.Component;

import io.netty.channel.ChannelInitializer;

import io.netty.channel.ChannelPipeline;

import io.netty.channel.socket.SocketChannel;

import io.netty.handler.codec.LengthFieldBasedFrameDecoder;

import io.netty.handler.codec.string.StringDecoder;

import io.netty.handler.codec.string.StringEncoder;

import io.netty.handler.logging.LogLevel;

import io.netty.handler.logging.LoggingHandler;

import io.netty.handler.timeout.IdleStateHandler;

import io.netty.util.CharsetUtil;

/\*\*

\* @author zhangfan

\* @date 2016年4月13日

\*/

@Component("ServerChannelInit")

public class ServerChannelInit extends ChannelInitializer<SocketChannel>{

@Override

protected void initChannel(SocketChannel ch) throws Exception {

ChannelPipeline pipeline = ch.pipeline();

pipeline.addLast(new LoggingHandler(LogLevel.INFO));

//pipeline.addLast(new IdleStateHandler(400, 0, 0));

pipeline.addLast("TransforDecoder", new TransforDecoder());

pipeline.addLast("TransforEecoder", new TransforEncoder());

//pipeline.addLast("encoder", new StringEncoder(Charset.forName("GBK")));

pipeline.addLast("serverInboundHandler", new ServerInboundHandler());

}

}

/\*\*

\*

\*/

package com.b\_noble.n\_life\_final.handler;

import java.util.Random;

import java.util.concurrent.ScheduledFuture;

import java.util.concurrent.TimeUnit;

import org.apache.commons.lang.StringUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.b\_noble.n\_life\_final.common.SysConst;

import com.b\_noble.n\_life\_final.conn.Connection;

import com.b\_noble.n\_life\_final.conn.ConnectionManager;

import com.b\_noble.n\_life\_final.event.EventEnum;

import com.b\_noble.n\_life\_final.event.NettyMessage;

import com.b\_noble.n\_life\_final.server.MyBootstrap;

import com.b\_noble.n\_life\_final.service.ProcessRunnable;

import com.b\_noble.n\_life\_final.utils.CommUtils;

import com.b\_noble.n\_life\_final.utils.DateHelp;

import com.b\_noble.n\_life\_final.utils.NetUtils;

import io.netty.channel.ChannelHandler.Sharable;

import io.netty.channel.ChannelHandlerAdapter;

import io.netty.channel.ChannelHandlerContext;

import io.netty.handler.timeout.IdleState;

import io.netty.handler.timeout.IdleStateEvent;

import io.netty.util.Attribute;

import io.netty.util.ReferenceCountUtil;

/\*\*

\* 服务端接收到客户端事件的handler

\* @author zhangfan

\* @date 2016年4月13日

\*/

@Sharable

public class ServerInboundHandler extends ChannelHandlerAdapter {

private static final Logger LOG = LoggerFactory.getLogger(ServerInboundHandler.class);

// 用来生成随机key

private static final char[] dic = new char[] {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',

'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'};

private static final int LENGTH = 49;

/\*\*

\* 随机获取key

\*/

private String randomKey(int length) {

Random random = new Random(System.currentTimeMillis());

StringBuilder sb = new StringBuilder("");

int index = 0;

for (int i = 0; i < length; i++) {

index = random.nextInt(dic.length);

sb.append(dic[index]);

}

return sb.toString();

}

/\*\*

\* 如果5秒没有验证通过，关闭连接

\* @description DelayClose

\* @author zhangfan

\* @date 2016年4月14日

\*/

class DelayClose implements Runnable {

private ChannelHandlerContext ctx;

public DelayClose(ChannelHandlerContext ctx) {

this.ctx = ctx;

}

public void run() {

System.out.println("验证超时----------------------");

Attribute<ScheduledFuture<?>> futureAttr = ctx.channel().attr(SysConst.DELAY\_KEY);

if(futureAttr.get() != null){

futureAttr.remove();

}

if (!ctx.channel().isRegistered()) {

LOG.info("{}检查2秒内有没有通过验证，连接已经被unRegistered", ctx.channel().toString());

return;

}

String mac = CommUtils.getMacFromAttr(ctx);

if (StringUtils.isBlank(mac)) {

// 没有验证通过，关闭连接

LOG.info("{} at 2秒内没有通过验证，关闭连接", ctx.channel().toString());

ctx.pipeline().close();

}

}

}

@Override

public void channelRegistered(ChannelHandlerContext ctx) throws Exception {

// 新的Connection

Connection conn = ConnectionManager.getInstance().getNewConnection(ctx);

Attribute<Connection> connAttr = ctx.channel().attr(SysConst.CONN\_KEY);

connAttr.set(conn);

LOG.info("新的客户端发出链接请求");

// 2秒没有验证通过，关闭连接

DelayClose delayClose = new DelayClose(ctx);

ScheduledFuture<?> future = ctx.executor().schedule(delayClose, 2, TimeUnit.SECONDS);

Attribute<ScheduledFuture<?>> futureAttr = ctx.channel().attr(SysConst.DELAY\_KEY);

if(futureAttr.get() != null){

futureAttr.get().cancel(false);

}

futureAttr.set(future);

}

@Override

public void channelUnregistered(ChannelHandlerContext ctx) throws Exception {

Attribute<Connection> attr = ctx.channel().attr(SysConst.CONN\_KEY);

Connection conn = attr.get();

String mac = conn.getMac();

String clienttype = conn.getClienttype();

if (StringUtils.isNotBlank(conn.getMac())) {

LOG.info("remove a connetion:{}, from connectionmanager", conn.getMac());

ConnectionManager.getInstance().removeConn(conn.getMac());

}

Attribute<ScheduledFuture<?>> futureAttr = ctx.channel().attr(SysConst.DELAY\_KEY);

if(futureAttr.get() != null){

LOG.info("remove future");

futureAttr.get().cancel(false);

futureAttr.remove();

}

LOG.info("remove a connection======{}========clienttype:"+clienttype,mac);

attr.remove();

ctx.channel().attr(SysConst.MAC\_KEY).remove();

ctx.channel().attr(SysConst.SECURE\_KEY).remove();

ctx.channel().attr(SysConst.WIFI\_VER\_KEY).remove();

ctx.channel().attr(SysConst.MCU\_VER\_KEY).remove();

}

@Override

public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {

try {

if (msg instanceof NettyMessage) {

if (((NettyMessage)msg).getType()!=EventEnum.TYPE) {

String mac = CommUtils.getMacFromAttr(ctx);

// handshake不需要验证

if ((((NettyMessage) msg).getType() != EventEnum.LOGIN && ((NettyMessage) msg).getType() != EventEnum.LOGIN\_APP) && StringUtils.isBlank(mac)) {

LOG.info("连接未验证");

ctx.pipeline().close();

return;

}

ProcessRunnable.pushUpMsg((NettyMessage) msg);

}else{

ConnectionManager.getInstance().getConn(CommUtils.getMacFromAttr(ctx)).setClienttype(new String(((NettyMessage) msg).getData()));;

}

} else {

LOG.error("error Object in channelRead:{}", msg.toString());

}

} finally {

// 如果是NettyMessage，不是ReferenceCounted，将不会release

ReferenceCountUtil.release(msg);

}

}

@Override

public void channelReadComplete(ChannelHandlerContext ctx) throws Exception {

ctx.flush();

}

@Override

public void userEventTriggered(ChannelHandlerContext ctx, Object evt) throws Exception {

if (evt instanceof IdleStateEvent) {

IdleStateEvent e = (IdleStateEvent) evt;

if (e.state() == IdleState.READER\_IDLE) {

// 空闲时间过久，关闭连接

LOG.info("空闲超过心跳时间，断开连接.mac:{}", CommUtils.getMacFromAttr(ctx));

ctx.pipeline().close();

}

}

}

@Override

public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception {

LOG.error("ServerInboundHanlder异常", cause);

//ConnectionManager.getInstance().removeConn(ConnectionManager.getInstance().getNewConnection(ctx));

//ctx.pipeline().close();

}

}

package com.b\_noble.n\_life\_final.handler;

import java.util.List;

import org.json.JSONObject;

import com.b\_noble.n\_life\_final.event.EventEnum;

import com.b\_noble.n\_life\_final.event.NettyMessage;

import com.b\_noble.n\_life\_final.utils.Test16Binary;

import io.netty.buffer.ByteBuf;

import io.netty.channel.ChannelHandlerContext;

import io.netty.handler.codec.ByteToMessageDecoder;

/\*\*

\* NettyMessage解码类

\* @author zhangfan

\* @date 2016年4月13日

\*/

public class TransforDecoder extends ByteToMessageDecoder {

@Override

protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) throws Exception {

if (in.readableBytes() < 4) {

ctx.pipeline().close();

return;

}

byte header = in.readByte();

int totalLength = in.readShort();

byte[] sn = new byte[8];

in.readBytes(sn);

int seq = in.readShort();

byte controlFlag = in.readByte();

byte cmd = in.readByte();

//byte state = in.readByte();

byte[] req = new byte[totalLength - 25];

in.readBytes(req);

//tbd + crc

in.readBytes(10);

NettyMessage event = new NettyMessage();

event.setCtx(ctx);

event.setType(EventEnum.valuesOf(cmd));

event.setData(req);

event.setSn(new String(sn));

event.setTotalLength(totalLength);

event.setCmd(cmd);

out.add(event);

}

}

/\*\*

\*

\*/

package com.b\_noble.n\_life\_final.handler;

import com.b\_noble.n\_life\_final.event.NettyMessage;

import io.netty.buffer.ByteBuf;

import io.netty.channel.ChannelHandlerContext;

import io.netty.handler.codec.MessageToByteEncoder;

/\*\*

\* nettyMessage编码类

\* @author zhangfan

\* @date 2016年4月13日

\*/

public class TransforEncoder extends MessageToByteEncoder<NettyMessage> {

@Override

protected void encode(ChannelHandlerContext ctx, NettyMessage msg, ByteBuf out) throws Exception {

System.out.println("编码类-------------------");

//header

out.writeByte(0XFE);

//totalLength

out.writeShort(msg.getTotalLength());

//sn

out.writeBytes(msg.getSn().getBytes());

//seqNumber

out.writeShort(1);

//controlFlag

out.writeByte(0xEF);

//command

out.writeByte(msg.getCmd());

//data

out.writeBytes(msg.getData());

//TBD

for (int i = 0; i < 8; i++) {

out.writeByte(0x00);

}

//checkcode

out.writeByte(0x01);

//CRC8

out.writeByte(0x00);

}

}

package com.b\_noble.n\_life\_final.exception;

/\*\*

\* 非法数据异常类

\* @author zhangfan

\*

\*/

public class IllegalDataException extends Exception {

private static final long serialVersionUID = -9222480292500296268L;

public IllegalDataException() {

super();

}

public IllegalDataException(String message, Throwable cause,

boolean enableSuppression, boolean writableStackTrace) {

super(message, cause, enableSuppression, writableStackTrace);

}

public IllegalDataException(String message, Throwable cause) {

super(message, cause);

}

public IllegalDataException(String message) {

super(message);

}

public IllegalDataException(Throwable cause) {

super(cause);

}

}

package com.b\_noble.n\_life\_final.dao;

import javax.jms.JMSException;

import javax.jms.MessageProducer;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.b\_noble.n\_life\_final.server.OperationLogBootstrap;

import com.b\_noble.n\_life\_final.utils.Global;

import com.b\_noble.n\_life\_final.utils.Test16Binary;

/\*\*

\* 使用activemq存储操作日志

\* @author zhangfan

\*

\*/

public class OperationLogDao {

private static final Logger LOG = LoggerFactory.getLogger(OperationLogDao.class);

/\*\*

\* 向mq中写数据

\* @param log

\*/

public void send(String log){

System.out.println("mq接收到数据"+Test16Binary.bytes2hex03(log.getBytes()));

Session session = Global.session;

MessageProducer producer = Global.producer;

TextMessage message;

try {

message = session.createTextMessage(log);

producer.send(message);

session.commit();

} catch (JMSException e) {

// TODO Auto-generated catch block

e.printStackTrace();

LOG.debug(log+"存入mq异常！！！！");

LOG.info("MQ重新启动--");

OperationLogBootstrap olb = new OperationLogBootstrap();

olb.run();

}

}

}

package com.b\_noble.n\_life\_final.conn;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

import io.netty.channel.ChannelHandlerContext;

/\*\*

\* 客户端链接

\* @author zhangfan

\* @date 2016年4月13日

\*/

public class Connection {

/\*\*

\* 链接id,链接的唯一标示

\*/

private final long id;

/\*\*

\* 链接的ChannelHandlerContext

\*/

private ChannelHandlerContext ctx;

private volatile String mac;

private boolean isSetMac = false;

private Lock lock = new ReentrantLock();

/\*\*

\* 附带属性

\*/

private ConcurrentHashMap<String, Object> attr = new ConcurrentHashMap<String, Object>();

/\*\*

\* 客户端类型

\*/

private String clienttype;

public Object getAttr(String key) {

return attr.get(key);

}

public Object putAttr(String key, Object value) {

return attr.put(key, value);

}

public Object removeAttr(String key) {

return attr.remove(key);

}

public Connection(long id, ChannelHandlerContext ctx) {

this.id = id;

this.ctx = ctx;

}

public long getId() {

return id;

}

public ChannelHandlerContext getCtx() {

return ctx;

}

public String getClienttype() {

return clienttype;

}

public void setClienttype(String clienttype) {

this.clienttype = clienttype;

}

/\*\*

\* 线程安全,没有验验证通过前，返回null

\*/

public String getMac() {

try {

lock.lock();

return mac;

} finally {

lock.unlock();

}

}

/\*\*

\* 设置mac地址，只可以设置一次，线程安全

\*/

public boolean setMac(String mac) {

try {

lock.lock();

if (isSetMac) {

return false;

}

this.mac = mac;

isSetMac = true;

return true;

} finally {

lock.unlock();

}

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + (int) (id ^ (id >>> 32));

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null) {

return false;

}

if (getClass() != obj.getClass()) {

return false;

}

Connection other = (Connection) obj;

if (id != other.id) {

return false;

}

return true;

}

@Override

public String toString() {

return getClass().getName() + "@" + Integer.toHexString(super.hashCode()) + "->Connection [id=" + id

+ ", ctx=" + ctx + "]";

}

}

package com.b\_noble.n\_life\_final.conn;

import java.util.ArrayList;

import java.util.Enumeration;

import java.util.List;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.atomic.AtomicLong;

import io.netty.channel.ChannelHandlerContext;

/\*\*

\* 链接池管理类

\* @author zhangfan

\* @date 2016年4月13日

\*/

public class ConnectionManager {

private static final ConnectionManager INSTANCE = new ConnectionManager();

public static ConnectionManager getInstance() {

return INSTANCE;

}

private ConnectionManager() {

}

private AtomicLong atomicLong = new AtomicLong();

private ConcurrentHashMap<String, Connection> conns = new ConcurrentHashMap<String, Connection>();

public Connection getNewConnection(ChannelHandlerContext ctx) {

Connection conn = new Connection(atomicLong.incrementAndGet(), ctx);

return conn;

}

public Enumeration<String> keys() {

return conns.keys();

}

public List<String> listAllConn(String cilenttype) {

Enumeration<String> keys = conns.keys();

List<String> connLst = new ArrayList<String>();

while (keys.hasMoreElements()) {

Connection c = conns.get(keys.nextElement());

if (c != null) {

if (cilenttype!=null && c.getClienttype().equals(cilenttype)) {

connLst.add(c.getMac());

}else if(cilenttype==null){

connLst.add(c.getMac());

}

}

}

return connLst;

}

public Connection addToConns(String key, Connection conn) {

Connection oldConn = conns.put(key, conn);

if (oldConn == null) {

// 新的key

return null;

}

ChannelHandlerContext ctx = oldConn.getCtx();

if (ctx == null) {

// oldConn无效

return oldConn;

}

if (oldConn.equals(conn)) {

// 同key同value

return null;

}

// 被替换的conn必须关闭

ctx.pipeline().close();

return oldConn;

}

public Connection getConn(String key) {

return conns.get(key);

}

public Connection removeConn(String key) {

return conns.remove(key);

}

public Connection removeConn(Connection conn) {

Enumeration<String> keys = conns.keys();

while (keys.hasMoreElements()) {

String key = keys.nextElement();

Connection v = conns.get(key);

if (conn.equals(v)) {

return conns.remove(key);

}

}

return null;

}

public void closeAllConn() {

Enumeration<String> keys = conns.keys();

while (keys.hasMoreElements()) {

String key = keys.nextElement();

Connection v = conns.get(key);

v.getCtx().close();

}

}

}

**package** com.b\_noble.n\_life\_final.event;

/\*\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：控制类型枚举类

\* 作者：zhangfan

\* 时间：2016-08-02

\* 版本：V0.0.1

\*/

public enum ControlTypeEnum {

/\*\*

\* 获取当前所有设备

\*/

GET\_DEVICE(0x86),

/\*\*

\* 发送心跳包

\*/

SEND\_HEARTBEAT(0X88),

/\*\*

\* 设置指定设备开关状态

\*/

SET\_DEVICE\_STATE(0x8E),

/\*\*

\* 设置指定设备的亮度

\*/

SET\_DEVICE\_LEVEL(0x90),

/\*\*

\* 设置指定设备颜色

\*/

SET\_DEVICE\_COLOR(0x92),

/\*\*

\* 获取指定设备的开关状态

\*/

GET\_DEVICE\_STATE(0x8F),

/\*\*

\* 获取指定设备的亮度

\*/

GET\_DEVICE\_LEVEL(0X91),

/\*\*

\* 获取指定设备的色调

\*/

GET\_DEVICE\_HUE(0X93),

/\*\*

\* 获取指定设备的饱和度

\*/

GET\_DEVICE\_SATURATION(0X88),

/\*\*

\* 获取指定设备色温

\*/

GET\_DEVICE\_COLORTEMPERATURE(0x95),

/\*\*

\* 绑定设备

\*/

BIND\_DEVICE(0X89),

/\*\*

\* 获取场景成员

\*/

GET\_SENCE\_MEMBER(0XAF),

/\*\*

\* 删除场景成员

\*/

DELETE\_SENCE\_MEMBER(0XAC),

/\*\*

\* 修改场景名称

\*/

UPDATE\_SENCE\_NAME(0XB0),

/\*\*

\* 获取组

\*/

GET\_GROUP(0X8E),

/\*\*

\* 添加指定设备到组

\*/

ADD\_DEVICE\_GROUP(0X8F),

/\*\*

\* 获取场景

\*/

GET\_SENCE(0XAD),

/\*\*

\* 添加场景

\*/

ADD\_SENCE(0XAA),

/\*\*

\* 添加设备到场景

\*/

ADD\_DEVICE\_SENCE(0xAB),

/\*\*

\* 调用场景

\*/

RECALL\_SENCE(0XB7),

/\*\*

\* 删除场景

\*/

DELETE\_SENCE(0XB1),

/\*\*

\* 更改指定设备名

\*/

UPDATE\_DEVICE\_NAME(0X8D),

/\*\*

\* 删除指定设备

\*/

DELETE\_DEVICE(0X8C),

/\*\*

\* 取消绑定

\*/

UNBIND\_DEVICE(0X96),

/\*\*

\* 从组中删除指定设备

\*/

DELTEE\_DEVICE\_GROUP(0X97),

/\*\*

\* 获取组成员名单

\*/

GET\_GROUP\_MEMBER(0X98),

/\*\*

\* 获取定时任务

\*/

GET\_TIMERTASK(0X99),

/\*\*

\* 添加定时任务

\*/

ADD\_TIMERTASK(0X9A),

/\*\*

\* 获取指定场景的定时任务

\*/

GET\_TASK\_SCENE(0XBD),

/\*\*

\* 获取指定设备的联动任务

\*/

GET\_TASK\_UID(0XBE),

/\*\*

\* 删除定时任务

\*/

DELETE\_TIMERTASK(0X9B),

/\*\*

\* 修改任务

\*/

UPDATE\_TASK(0XBB),

/\*\*

\* 获取网关信息

\*/

GET\_GATEWAY\_INFO(0X80),

/\*\*

\* 设置传感器上报时间

\*/

SET\_SENSOR\_TIME(0X9E),

/\*\*

\* 允许入网

\*/

ALLOW\_NET(0X82),

/\*\*

\* 打开/关闭在线查询

\*/

UPDATE\_NETQUERY\_STATE(0XA0),

/\*\*

\* 复位网关

\*/

RESET\_GATEWAY(0XA1),

/\*\*

\* 0xA2 获取 RSSI

\*/

GET\_RSSI(0XA2),

/\*\*

\* 增加任务

\*/

ADD\_TASK(0XB9),

/\*\*

\* 删除任务

\*/

DELETE\_TASK(0XBA),

/\*\*

\* 查看任务详情

\*/

GET\_TASK\_INFO(0XBC),

/\*\*

\* 获取所有任务

\*/

GET\_ALL\_TASK(0XB8),

/\*\*

\* 红外相关

\*/

IR\_RELEVANT(0XA7),

/\*\*

\* 设置设备色温

\*/

SET\_COLOR\_TEMPERATURE(0X96),

/\*\*

\* 获取设备色温

\*/

GET\_COLOR\_TEMPERATURE(0XA9),

/\*\*

\* 对网关中一字符串信息操作

\*/

OPER\_GATEWAY\_STRING(0XAB),

/\*\*

\* cie相关指令

\*/

CIE\_RELEVANT(0XAC),

/\*\*

\* 网关和 ZigBee 信息相关

\*/

GATEWAY\_ZIGBEE\_RELEVANT(0XAE),

/\*\*

\* 修改组名

\*/

UPDATE\_GROUP\_NAME(0XAF),

/\*\*

\* 添加设备定时任务

\*/

ADD\_TIMING\_TASK(0XC1),

/\*\*

\* 删除设备定时任务

\*/

DELETE\_TIMING\_TASK(0XC2),

/\*\*

\* 更新设备定时任务

\*/

UPDATE\_TIMING\_TASK(0XC3),

/\*\*

\* 获取设备定时任务

\*/

GET\_TIMING\_TASK(0xC4),

/\*\*

\* 登录验证指令

\*/

LOGIN\_VALODATE(0xDF);

private int v;

private ControlTypeEnum(int v) {

this.v = v;

}

public int getVal() {

return v;

}

public static ControlTypeEnum valuesOf(int e) {

ControlTypeEnum[] vs = ControlTypeEnum.values();

if (vs == null || vs.length == 0) {

return null;

}

for (ControlTypeEnum event : vs) {

if (event.getVal() == e) {

return event;

}

}

return null;

}

}

**package** com.b\_noble.n\_life\_final.event;

/\*\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：响应类型枚举类

\* 作者：zhangfan

\* 时间：2016-08-02

\* 版本：V0.0.1

\*/

public enum ResponseTypeEnum {

/\*\*

\* 获取当前所有设备响应

\*/

RESPONSE\_GET\_DEVICE(0x26),

/\*\*

\* 设置设备开关状态响应类型

\*/

RESPONSE\_SET\_DEVICE\_STATE(0x2E),

/\*\*

\* 删除指定设备响应类型

\*/

RESPONSE\_DELETE\_DEVICE(0x2C),

/\*\*

\* 更新设备名称响应类型

\*/

RESPONSE\_UPDATE\_DEVICENAME(0x2D),

/\*\*

\* 获取设备的开关状态响应类型

\*/

RESPONSE\_GET\_DEVICE\_STATE(0x2F),

/\*\*

\* 心跳响应类型

\*/

RESPONSE\_SEND\_HEARTBEAT(0x28),

/\*\*

\* 获取设备亮度相应类型

\*/

RESPONSE\_GET\_DEVICE\_LEVEL(0x31),

/\*\*

\* 设置设备亮度响应类型

\*/

RESPONSE\_SET\_DEVICE\_LEVEL(0x30),

/\*\*

\* 获取设备色调与饱和度响应类型

\*/

RESPONSE\_GET\_DEVICE\_HUE(0x33),

/\*\*

\* 设置设备色调响应类型

\*/

RESPONSE\_SET\_DEVICE\_HUE(0x32),

/\*\*

\* 获取设备色温响应类型

\*/

RESPONSE\_GET\_DEVICE\_COLORTEMPERATURE(0x35),

/\*\*

\* 设置设备色温响应类型

\*/

RESPONSE\_SET\_COLOR\_TEMPERATURE(0x36),

/\*\*

\* 获取组响应类型

\*/

RESPONSE\_GET\_GROUP(0x0C),

/\*\*

\* 添加指定设备到组响应类型

\*/

REPOSNE\_ADD\_DEVICE\_GROUP(0x0B),

/\*\*

\* 从组中删除指定设备相应类型

\*/

REPONSE\_DELETE\_DEVICE\_GROUP(0x0F),

/\*\*

\* 获取组成员名单响应类型

\*/

REPONSE\_GET\_GROUP\_MEMBER(0X10),

/\*\*

\* 获取场景响应类型

\*/

RESPONSE\_GET\_SENCE(0x4D),

/\*\*

\* 添加/修改场景响应类型

\*/

RESPONSE\_ADD\_SENCE(0x4A),

/\*\*

\* 添加设备到场景

\*/

REPOSNE\_ADD\_DEVICE\_SENCE(0x4B),

/\*\*

\* 获取场景成员详细信息响应类型

\*/

RESPONSE\_GET\_SENCE\_DETAILED(0x4F),

/\*\*

\* 删除场景中的指定成员响应类型

\*/

RESPONSE\_DELETE\_SENCE\_MEMBER(0x4C),

/\*\*

\* 更新场景名称响应类型

\*/

RESPONSE\_UPDATE\_SCENENAME(0x50),

/\*\*

\* 调用场景返回

\*/

RESPONSE\_RECALL\_SCENE(0X4E),

/\*\*

\* 删除场景返回

\*/

RESPONSE\_DELETE\_SCENE(0X51),

/\*\*

\* 获取定时任务响应类型

\*/

RESPONSE\_GET\_TIMERTASK(0x11),

/\*\*

\* 获取所有任务

\*/

REPOSNE\_GET\_ALL\_TASK(0x58),

/\*\*

\* 添加定时任务响应类型

\*/

RESPONSE\_ADD\_TASK(0x59),

/\*\*

\* 获取任务详细

\*/

REPOSNE\_GET\_TASKINFO(0x5C),

/\*\*

\* 获取指定场景定时任务

\*/

RESPONSE\_TASK\_SCENE(0x5D),

/\*\*

\* 获取指定设备的联动任务

\*/

RESPONSE\_TASK\_UID(0x5E),

/\*\*

\* 删除定时任务响应类型

\*/

RESPONSE\_DELETE\_TIMERTASK(0x13),

/\*\*

\* 更新任务响应类型

\*/

RESPONSE\_UPDATE\_TASK(0x5B),

/\*\*

\* 获取网关信息响应类型

\*/

RESPONSE\_GET\_GATE\_INFO(0x20),

/\*\*

\* 获取 RSSI响应类型

\*/

RESPONSE\_GET\_RSSI(0x16),

/\*\*

\* 获取所有任务响应类型

\*/

RESPONSE\_GET\_ALL\_TASK(0x25),

/\*\*

\* 删除任务响应类型

\*/

RESPONSE\_DELETE\_TASK(0x5A),

/\*\*

\* 查看指定任务详情响应类型

\*/

RESPONSE\_GET\_TASKDETAILED(0x24),

/\*\*

\* 获取指定设备的色温响应类型

\*/

RESPONSE\_GET\_TEMPERATURE(0x27),

/\*\*

\* 对网关中一字符串信息操作响应类型

\*/

RESPONSE\_GATEWAY\_STRING(0x28),

/\*\*

\* 组改名响应类型

\*/

RESPONSE\_UPDATE\_GROUPNAME(0xAF),

/\*\*

\* 节点主动上报响应类型

\*/

REPSONSE\_NODE\_REPORT(0xA1),

/\*\*

\* 添加指定设备定时任务

\*/

RESPONSE\_ADD\_TIMING\_TASK(0x61),

/\*\*

\* 删除指定设备定时任务

\*/

RESPONSE\_DELETE\_TIMING\_TASK(0x62),

/\*\*

\* 更新设备定时任务

\*/

RESPONSE\_UPDATE\_TIMING\_TASK(0x63),

/\*\*

\* 获取设备定时任务

\*/

RESPONSE\_GET\_TIMING\_TASK(0x64);

private int v;

private ResponseTypeEnum(int v) {

this.v = v;

}

public int getVal() {

return v;

}

public static ControlTypeEnum valuesOf(int e) {

ControlTypeEnum[] vs = ControlTypeEnum.values();

if (vs == null || vs.length == 0) {

return null;

}

for (ControlTypeEnum event : vs) {

if (event.getVal() == e) {

return event;

}

}

return null;

}

}

package com.b\_noble.n\_life;

import java.util.List;

import java.util.Map;

import com.b\_noble.n\_life.info.DeviceInfo;

import com.b\_noble.n\_life.info.GateWayInfo;

import com.b\_noble.n\_life.info.SenceInfo;

import com.b\_noble.n\_life.info.TaskDeviceAction;

import com.b\_noble.n\_life.info.TaskInfo;

import com.b\_noble.n\_life.info.TaskTimerAction;

import com.b\_noble.n\_life.info.TimingInfo;

import android.content.Context;

/\*\*

\* @company：深圳市中泰智丰物联网科技有限公司

\* @project：智能家居

\* @describe：app调用接口

\* @author：zhangfan @date：2016-08-04 @version：V0.0.1

\*/

public interface Serial {

public void setmContext(Context mContext);

/\*\*

\* 连接到远程的网关

\*

\* @param userName

\* 用户名

\* @param passWd

\* 密码

\* @return 1，连接成功且已绑定网关,-1为账号或密码错误,0为未绑定网关,-2与网关建立链接失败，-3鉴权系统链接超时

\*/

public int connectRemoteZll(String userName, String passWd);

/\*\*

\* 向服务端发送登录验证指令

\*/

public void sendLoginCmd();

/\*\*

\* 通过ip、snid、port来连接到本地的网关

\*

\* @param ip

\* @param snid

\* @param port

\* @return -1 连接失败，-3连接超时，>0连接成功

\*/

public int connectLANZllByIp(String ip, String snid, int port);

/\*\*

\* 获取当前找到所有的网关ip,结果在newGate\_CallBack中回调返回

\*

\*/

public Map<String, String> getGatewayIps();

/\*\*

\* 查看网关信息

\*/

public void getGateWayInfo();

/\*\*

\* 复位网关

\*

\* @return

\*/

public int resetGateway();

/\*\*

\* 获取当前连接的所有设备 可以在newDevice\_CallBack（）这个回调中，来获得所有设备

\*/

public DeviceInfo[] getDevices();

/\*\*

\* 更改设备或组名称,传入字符数组

\*

\* @param dInfo

\* 要更改的设备（必须有device UId 和deviceName）

\* @param dName

\* 新的设备名 执行该方法后结果在ChangeDeviceName\_callBack(byte[] uid,String

\* deviceName) 中返回

\*/

public int ChangeDeviceName(DeviceInfo dInfo, String dName);

/\*\*

\* 删除当前设备

\*

\* @param info

\* 要删除的设备（必须有device Uid）

\*/

public int deleteDevice(DeviceInfo info);

/\*\*

\* 改变设备状态：1 开/ 0 关

\*

\* @param info

\* 要改变状态的设备（必须要有Uid）

\* @param state

\* 要改变的状态 执行此方法后在setDeviceState\_CallBack() 中返回状态

\*/

public int setDeviceState(DeviceInfo info, int state);

/\*\*

\* 获得设备状态，结果在getDeviceState\_CallBack这个回调中会有返回

\*

\* @param info

\* 要获取的设备（必须要有Uid）

\*/

public int getDeviceState(DeviceInfo info);

/\*\*

\* 改变设备值，（亮度）

\*

\* @param info

\* 要改变设备值的设备（必须要有Uid）

\* @param value

\* 要改变的值

\*/

public int setDeviceLevel(DeviceInfo info, byte value);

/\*\*

\* 改变设备值，（亮度）

\*

\* @param info

\* 要改变设备值的设备（必须要有Uid）

\* @param value

\* 要改变的值

\* @param transitionTime

\* 延时时间（单位100ms）

\* @return

\*/

public int setDeviceLevel(DeviceInfo info, byte value, short transitionTime);

/\*\*

\* 获得设备值（输出级别， 如亮度） ， 结果在getDeviceLevel\_CallBack这个回调中有返回

\*

\* @param info

\* 要获取设备值的设备（必须要有Uid）

\*/

public int getDeviceLevel(DeviceInfo info);

/\*\*

\* 改变设备色调、饱和度

\*

\* @param info

\* 要改变色调、饱和度的设备

\* @param hue

\* 色调

\* @param sat

\* 饱和度

\*/

public int setDeviceHueSat(DeviceInfo info, byte hue, byte sat);

/\*\*

\* 改变设备色调、饱和度

\*

\* @param info

\* 要改变色调、饱和度的设备

\* @param hue

\* 色调

\* @param sat

\* 饱和度

\* @param transitionTime

\* 延时时间（单位100ms）

\*

\* @return

\*/

public int setDeviceHueSat(DeviceInfo info, byte hue, byte sat, short transitionTime);

/\*\*

\* 改变设备色温

\*

\* @param info

\* 要改变色温的设备

\* @param value

\* 色温值

\*/

public int SetColorTemperature(DeviceInfo info, int value);

/\*\*

\*

\* 改变设备色温

\*

\* @param info

\* 要改变色温的设备

\* @param value

\* 色温值

\* @param transitionTime

\* 延时时间（单位100ms）

\* @return

\*/

public int SetColorTemperature(DeviceInfo info, int value, short transitionTime);

/\*\*

\* 获取设备色调与饱和度，结果在 getDeviceHueAndSat\_CallBack这个回调中

\*

\* @param info

\* 要获取设备色调的设备 （必须要有Uid）

\*/

public int getDeviceHueAndSat(DeviceInfo info);

/\*\*

\* 获取设备色温值，结果在 getColorTemperature\_CallBack这个回调中

\*

\* @param info

\* 要获取设备色温的设备（必须要有Uid）

\*/

public int getColorTemperature(DeviceInfo info);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* about sence \*/

/\*\*

\* 获取当前所有场景

\*

\* @return 返回NULL，其结果在newSence\_CallBack（） 中返回

\*/

public SenceInfo[] getSences();

/\*\*

\* 创建新场景

\*

\* @param senceName

\* 场景名

\* @param uid

\* 设备uid

\* @param delaytime

\* 延迟时间为0-60s

\*

\* 添加完成后新场景在addSence\_callBack(byte sceneId,String senceName)中返回

\*/

public int addSence(String senceName, List<byte[]> uid, int delaytime);

/\*\*

\* 将指定的设备动作添加到指定的场景中，若场景不存在，则创建新场景

\*

\* @param senceId

\* 场景id

\* @param uids

\* 要添加的设备列表

\*/

public int addDeviceToSence(int senceId, List<byte[]> uids, int delaytime);

/\*\*

\* 打开指定的场景

\*

\* @param senceId

\* 要打开的场景的ID

\* @param sceneName

\* 要打开的场景的name

\*/

public int recallScene(int sceneId, String sceneName);

/\*\*

\* 获取指定场景的详细信息， 结果在getSenceMember\_CallBack()这个回调中返回

\*

\* @param senceId

\* 场景Id

\*/

public int getSenceMember(int sceneId);

/\*\*

\* 删除场景中指定设备成员

\*

\* @param senceName

\* 场景名

\* @param uIds

\* 设备uId列表

\*/

public int deleteSenceMember(int sceneId, String sceneName, List<byte[]> uIds);

/\*\*

\* 删除指定场景

\*

\* @param senceName

\*/

public int deleteSence(int senceId, String sceneName);

/\*\*

\* 修改场景名称

\*

\* @param senceId

\* 场景Id

\* @param sName

\* 新的场景名

\*/

public int ChangeSceneName(short sceneId, String newSceneName);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* about task \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* 获取当前所有任务

\*

\* @return 获取的任务在newTask\_CallBack这个回调函数中

\*/

public void getTasks();

/\*\*

\* 简述 根据场景id查询该场景的所有定时任务

\*

\* @param sceneId

\* 场景id

\* @author zhangfan

\* @date 2016-07-26

\* @return 此方法在getTimerTaskBySceneId\_CallBack这个回调方法中返回

\*/

public void getTimerTaskBySceneId(int sceneId);

/\*\*

\* 简述 根据设备uid查询该设备的所有联动任务

\*

\* @param uid

\* 设备uid shortAddress+endPoint

\* @author zhangfan

\* @date 2016-07-27

\* @return 此方法在getSensorTaskByUid\_CallBack这个回调方法中返回

\*/

public void getSensorTaskByUid(byte[] uid);

/\*\*

\* 添加定时任务

\*

\* @param taskInfo

\* @return

\*/

public int addTimerTask(String taskName, TaskTimerAction timerAction, int sceneId);

/\*\*

\* 添加设备触发任务

\*

\* @param taskName

\* @param isAlarm

\* 是否需要报警

\* @return

\*/

public int addDeviceTask(String taskName, TaskDeviceAction deviceAction, short sceneId, byte isAlarm);

/\*\*

\* 添加设备报警任务

\*

\* @param taskName

\* @param deviceAction

\* @param SenceName

\* @param isAlarm

\* @return

\*/

public int addDeviceAlarmTask(String taskName, TaskDeviceAction deviceAction);

/\*\*

\* 修改定时任务

\*

\* @param taskInfo

\* @return

\*/

public int updateTimerTask(int taksId, String taskName, TaskTimerAction timerAction, short sceneId);

/\*\*

\* 修改设备触发任务

\*

\* @param taskName

\* @param isAlarm

\* 是否需要报警

\* @return

\*/

public int updateDeviceTask(int taskId, String taskName, TaskDeviceAction deviceAction, short sceneId,

byte isAlarm);

/\*\*

\* 修改设备报警任务

\*

\* @param taskName

\* @param deviceAction

\* @param SenceName

\* @param isAlarm

\* @return

\*/

public int updateDeviceAlarmTask(int taskId, String taskName, TaskDeviceAction deviceAction);

/\*\*

\* 删除指定任务

\*

\* @param taskID

\* 指定的任务id

\* @return

\*/

public int deleteTask(int taskId);

/\*\*

\* 查看指定任务的详细信息,结果在getTimerTaskDetails\_CallBack/getDeviceTaskDetails\_CallBack

\*

\* @param taskId

\* 指定的任务id

\* @return

\*/

public int getTaskInfo(int taskId);

/\*\*

\* 添加设备定时任务

\*

\* @param timingName

\* 任务名称

\* @param uid

\* 设备唯一标示

\* @param taskTimerAction

\* 定时信息

\* @param switchState

\* 开关状态

\* @param brightness

\* 灯光亮度

\* @param hue

\* 饱和度

\* @param saturation

\* 色调

\* @param colorTemperature

\* 色温

\* @param enable

\* @return

\*/

public int addDeviceTimingTask(String timingName, byte[] uid, TaskTimerAction taskTimerAction, byte switchState,

byte brightness, byte hue, byte saturation, byte colorTemperature, byte enable);

/\*\*

\* 删除设备定时任务

\*

\* @param timingID

\* @return

\*/

public int deleteDeviceTimingTask(byte timingID);

/\*\*

\* 更新设备定时任务

\*

\* @param timingID

\* 任务id

\* @param timingName

\* 任务名称

\* @param uid

\* 设备唯一标示

\* @param taskTimerAction

\* 定时信息

\* @param switchState

\* 开关状态

\* @param brightness

\* 灯光亮度

\* @param hue

\* 饱和度

\* @param saturation

\* 色调

\* @param colorTemperature

\* 色温

\* @param enable

\* @return

\*/

public int updateDeviceTimingTask(byte timingID, String timingName, byte[] uid, TaskTimerAction taskTimerAction,

byte switchState, byte brightness, byte hue, byte saturation, byte colorTemperature, byte enable);

/\*\*

\* 获取指定设备的所有定时任务，结果在getDeviceTimingTask\_CallBack中返回

\*

\* @param uid

\* 设备唯一标示

\*/

public void getDeviceTimingTask(byte[] uid);

/\*\*

\* 向网关发送心跳信息 响应心跳在sendHeartbeat\_CallBack中返回

\*/

public void sendHeartbeat();

/\*\*

\* 允许入网指令，设备入网成功后结果在回调newDevice\_Callback()中返回

\*/

public int permitJoin();

/\*\*

\* --------------------------------以下是回调方法----------------------------------

\* -

\*\*/

/\*\*

\* 这是一个测试回调方法

\*

\* @param result

\*/

public void callback(String result);

/\*\*

\* 获取所有设备回调方法

\*

\* @param dinfo

\*/

public void newDevice\_CallBack(List<DeviceInfo> dinfos);

/\*\*

\* 允许入网回调

\*/

public void newDevice\_Callback(DeviceInfo info);

/\*\*

\* 本地查找网关返回

\*

\* @param ip

\*/

public void newGate\_CallBack(String ip, String sn);

/\*\*

\* 获取设备的值回调方法

\*/

public void getDeviceLevel\_CallBack(int level, byte[] uId);

/\*\*

\* 获取所有场景回调方法

\*

\* @param dinfo

\*/

public void newSence\_CallBack(List<SenceInfo> sinfos);

/\*\*

\* 获取网关信息返回

\*/

public void getGateWayInfo\_CallBack(GateWayInfo gainfo);

/\*\*

\* 获取设备状态回调,通过state来判断设备状态（>0 开、<= 0关）

\*

\* @param state

\* @param uId

\*/

public void getDeviceState\_CallBack(int state, byte[] uId);

/\*\*

\* 获取设备色调与饱和度回调

\*

\* @param hue

\* 色调

\* @param sat

\* 饱和度

\* @param uId

\*/

public void getDeviceHueAndSat\_CallBack(int hue, int sat, byte[] uId);

/\*\*

\* 获取设备色温值回调

\*

\* @param colorTemperature

\* @param uId

\*/

public void getColorTemperature\_CallBack(int colorTemperature, byte[] uId);

/\*\*

\* 获取场景成员回调

\*/

public void getSenceMember\_CallBack(int sceneId, String sceneName, List<byte[]> uids);

/\*\*

\* 获取所有任务回调

\*/

public void newTask\_CallBack(List<TaskInfo> tasks);

/\*\*

\* 获取指定场景定时任务 当调用getTimerTaskBySceneId方法后回调

\*/

public void getTimerTaskBySceneId\_CallBack(List<TaskInfo> tasks);

/\*\*

\* 获取指定设备联动任务 当条用getSensorTaskByUid方法后回调

\*/

public void getSensorTaskByUid\_CallBack(List<TaskInfo> tasks);

/\*\*

\* 当调用getTaskInfo()时返回

\*

\* @param taskId

\* task's id

\* @param taskName

\* task's name

\* @param action1

\* first task's action

\* @param action2

\* second task's action

\*/

public void getTimerTaskDetails\_CallBack(TaskInfo taskInfo, TaskTimerAction timerAction, short sceneId);

/\*\*

\* 当调用getTaskInfo()时返回，返回获取设备联动任务

\*

\* @param taskName

\* 任务名

\* @param deviceAction

\* 设备联动时，设备动作信息

\* @param senceName

\* 联动执行场景

\* @param isAlarm

\* 是否报警标志

\*/

public void getDeviceTaskDetails\_CallBack(TaskInfo taskInfo, TaskDeviceAction deviceAction, short sceneId,

byte isAlarm);

/\*\*

\* 当调用addSence后返回

\*

\* @param sceneId

\* 场景id

\*

\* @param senceName

\* 场景名称

\*/

public void addSence\_callBack(byte sceneId, String senceName);

/\*\*

\* 当调用ChangeDeviceName后返回

\*

\* @param uid

\* uid

\*

\* @param deviceName

\* 设备名称

\*/

public void ChangeDeviceName\_callBack(byte[] uid, String deviceName);

/\*\*

\* 传感器推送数据回调

\*

\* @param uid

\* 设备uid

\* @param 参数

\* 请参考recentValue对照表

\*/

public void sensorReport\_callBack(byte[] uid, byte[] recentValue, byte[] deviceId, short attrId);

/\*\*

\* 温湿度传感器推送回调方法

\*

\* @param temperature

\* 温度

\* @param humidity

\* 湿度

\* @param battery

\* 电量 0正常，1低电量

\*/

public void TemperatureAndHumidityReport\_callBack(Double temperature, Double humidity, Integer battery);

/\*\*

\* 紧急按钮推送回调方法

\*

\* @param state

\* 状态

\*/

public void EmergencyButtonReprot\_callBack(byte state, byte[] uid);

/\*\*

\* 燃气传感器推送回调方法

\*

\* @param state

\* 状态

\*/

public void GasSensorReprot\_callBack(byte state, byte[] uid);

/\*\*

\* 烟雾传感器推送回调方法

\*

\* @param state

\* 状态

\*/

public void FireSensorReprot\_callBack(byte state, byte[] uid);

/\*\*

\* 一氧化碳传感器推送回调方法

\*

\* @param state

\* 状态

\*/

public void CoSensorReprot\_callBack(byte state, byte[] uid);

/\*\*

\* 多功能遥控器推送回调方法

\*

\* @param state

\* 状态

\*/

public void KeyFobReprot\_callBack(byte state, byte[] uid);

/\*\*

\* 客户端离线后回调方法

\*/

public void clientClose();

/\*\*

\* 客户端建立链接回调方法

\*/

public void clientActive();

/\*\*

\* 获取设备定时任务回调方法

\*/

public void getDeviceTimingTask\_CallBack(List<TimingInfo> timingInfos);

/\*\*

\* 心跳响应回调方法

\*/

public void sendHeartbeat\_CallBack();

}

package com.b\_noble.n\_life.handler;

import java.util.ArrayList;

import java.util.List;

import com.b\_noble.n\_life.BaseApplication;

import com.b\_noble.n\_life.Serial;

import com.b\_noble.n\_life.handler.processing.HandlerProcessing;

import com.b\_noble.n\_life.info.DeviceInfo;

import com.b\_noble.n\_life.info.TaskDeviceAction;

import com.b\_noble.n\_life.info.TaskTimerAction;

import com.b\_noble.n\_life.utils.Global;

import com.b\_noble.n\_life.utils.Test16Binary;

import android.util.Log;

import io.netty.buffer.ByteBuf;

import io.netty.channel.ChannelHandlerAdapter;

import io.netty.channel.ChannelHandlerContext;

/\*\*

\* @公司：深圳市中泰智丰物联网科技有限公司

\* @项目：智能家居

\* @简述：客户端handler

\* @作者：zhangfan @date：2016-08-02 @版本：V0.0.1

\*/

public class ClientHandler extends ChannelHandlerAdapter {

@Override

public void channelActive(ChannelHandlerContext ctx) throws Exception {

Global.result = 1;

Global.group.add(ctx.channel());

Serial s = BaseApplication.getInstance();

s.clientActive();

//向服务端发送登录验证指令

s.sendLoginCmd();

}

@Override

public void channelRead(ChannelHandlerContext ctx, Object msg) {

ByteBuf buf = (ByteBuf) msg;

int count = buf.readableBytes();

System.out.println(count);

byte[] b = new byte[count];

if (count >= 4) {

byte header = buf.readByte();

short totalLength = buf.readShort();

int responseType = buf.readByte();

if (totalLength == count && totalLength != 36 && totalLength != 37) {

HandlerProcessing.processing(responseType, buf);

}

/\*\*

\* report new device返回

\*/

else if (totalLength == 37) {

HandlerProcessing.reportNewDevice(buf);

}

/\*\*

\* 传感器推送

\*/

else if (totalLength == 36) {

HandlerProcessing.sensorReport(buf);

}

}

/\*\*

\* 接收原始指令

\*/

byte[] yb = new byte[count];

buf.resetReaderIndex();

for (int i = 0; i < count; i++) {

yb[i] = buf.readByte();

}

System.out.println("响应指令：\n" + Test16Binary.bytes2hex03(yb));

}

public void handlerRemoved(ChannelHandlerContext ctx) {

System.out.println("handlerRemoved(链接关闭)");

Global.buf = "";

if (Global.group.size() > 0) {

Global.group.close();

Global.group.clear();

}

BaseApplication.getInstance().clientClose();

}

public void exceptionCaught(ChannelHandlerContext ctx, Throwable throwable) {

// 释放资源

System.out.println("exceptionCaught:" + throwable.getMessage());

Global.buf = "";

if (Global.group.size() > 0) {

Global.group.close();

Global.group.clear();

}

ctx.close();

}

}

package com.b\_noble.n\_life.handler;

import java.util.List;

import io.netty.buffer.ByteBuf;

import io.netty.channel.ChannelHandlerContext;

import io.netty.handler.codec.ByteToMessageDecoder;

import io.netty.handler.codec.DecoderResult;

/\*\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：消息解码类，主要是解决粘包/断包问题

\* 作者：zhangfan

\* 时间：2016-08-02

\* 版本：V0.0.1

\*/

public class Decoder extends ByteToMessageDecoder{

@Override

protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) throws Exception {

int readable = in.readableBytes();

in.markReaderIndex();

byte header = in.readByte();

int frameLength = in.readShort();

in.resetReaderIndex();

if (readable < frameLength) {

return;

}

ByteBuf frame = in.readBytes(frameLength);

out.add(frame);

}

}

package com.b\_noble.n\_life.handler;

import com.b\_noble.n\_life.handler.processing.EncoderProcessing;

import com.b\_noble.n\_life.model.RequestHead;

import com.b\_noble.n\_life.model.RequestMessage;

import com.b\_noble.n\_life.utils.Test16Binary;

import io.netty.buffer.ByteBuf;

import io.netty.channel.ChannelHandlerContext;

import io.netty.handler.codec.MessageToByteEncoder;

/\*\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：请求消息编码类

\* 作者：zhangfan

\* 时间：2016-08-02

\* 版本：V0.0.1

\*/

public class Encoder extends MessageToByteEncoder<Object> {

@Override

protected void encode(ChannelHandlerContext ctx, Object msg, ByteBuf buf) throws Exception {

// TODO Auto-generated method stub

RequestMessage requestMessage = (RequestMessage) msg;

RequestHead head = requestMessage.getHeader();

buf.writeByte(head.getHeader());

buf.writeShort(head.getLength());

buf.writeBytes(head.getSncode().getBytes());

//自生成序列号

buf.writeShort(head.getSeqNumber());

buf.writeByte(head.getControlMark());

buf.writeByte(head.getControlType());

EncoderProcessing.processing(requestMessage, buf);

//tbd 8 bytes

for (int i = 0; i < 10; i++) {

buf.writeByte(0x00);

}

/\*\*

\* 输出发送指令

\*/

byte[] b = new byte[buf.readableBytes()];

for (int i = 0; i < b.length; i++) {

b[i] = buf.getByte(i);

}

System.out.println("发送指令：\n"+Test16Binary.bytes2hex03(b));

}

}

package com.b\_noble.n\_life.handler;

import com.b\_noble.n\_life.utils.Global;

import io.netty.channel.ChannelHandlerContext;

import io.netty.channel.SimpleChannelInboundHandler;

import io.netty.channel.socket.DatagramPacket;

import io.netty.util.CharsetUtil;

/\*\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：UDP广播方式搜索网关handler

\* 作者：zhangfan

\* 时间：2016-08-02

\* 版本：V0.0.1

\*/

public class getGatewayHandler extends SimpleChannelInboundHandler<DatagramPacket> {

@Override

protected void messageReceived(ChannelHandlerContext ctx, DatagramPacket datagramPacket) throws Exception {

String req = datagramPacket.content().toString(CharsetUtil.UTF\_8);

System.out.println("客户端接收："+req);

Global.ips.put(datagramPacket.sender().getHostString(), req.split(":")[1]);

//BaseApplication.getInstance().newGate\_CallBack(datagramPacket.sender().getHostString(),req.split(":")[1]);

}

}

package com.b\_noble.n\_life.handler;

import io.netty.channel.ChannelHandlerContext;

import io.netty.channel.ChannelInitializer;

import io.netty.channel.ChannelPipeline;

import io.netty.channel.socket.SocketChannel;

import io.netty.handler.codec.LengthFieldBasedFrameDecoder;

import io.netty.handler.codec.protobuf.ProtobufDecoder;

/\*\*

\* 公司：深圳市中泰智丰物联网科技有限公司

\* 项目：智能家居

\* 简述：handler初始化类

\* 作者：zhangfan

\* 时间：2016-08-02

\* 版本：V0.0.1

\*/

public class MyClientInitializer extends ChannelInitializer<SocketChannel> {

@Override

protected void initChannel(SocketChannel ch) throws Exception {

ChannelPipeline pipeline = ch.pipeline();

pipeline.addLast("encoder", new Encoder());

pipeline.addLast("decoder", new Decoder());

pipeline.addLast("handler",new ClientHandler());

}

@Override

public void channelRead(ChannelHandlerContext ctx, Object msg)

throws Exception {

System.out.println("---channelRead--- msg="+msg);

super.channelRead(ctx, msg);

}

@Override

public void channelReadComplete(ChannelHandlerContext ctx) throws Exception {

System.out.println("---channelReadComplete---");

super.channelReadComplete(ctx);

}

}