package datasource;

import java.lang.reflect.Field;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.sql.SQLException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import javax.sql.DataSource;

import org.apache.log4j.Logger;

/\*\*

\* 数据库基本操作

\*

\* @author wqc

\* @version 1.0

\* @date 2014-11-06

\*

\*/

public class Dao {

protected Logger logger = Logger.getLogger("datasource");

/\*\*

\* 根据数据源创建一个Dao

\*

\* @param id

\* {@link String} DataSource的id

\*/

public Dao(String id) {

this.dataSource = C3p0.getInstance().getDataSource(id);

if(dataSource == null) {

logger.error("DataSource:" + id + " is null");

throw new NullPointerException("DataSource is null");

}

}

protected DataSource dataSource;

public DataSource getDataSource() {

return dataSource;

}

/\*\*

\* 查询Bean类

\*

\* @param clazz

\* {@link Class} JavaBean 如实现了构造方法,则必须实现公开的空参数的构造方法

\* public JavaBean() {}

\* @param sql

\* {@link String} sql语句 { select \* from table where id=? }

\* @param params

\* {@link Object[]} 参数

\* @return {@link List<T>} 查询结果

\* @throws SQLException

\*/

@SuppressWarnings("unchecked")

public <T> List<T> queryBean(Class<? super T> clazz, String sql, Object[] params)

throws SQLException {

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

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

Object param = params[i];

stmt.setObject(i + 1, param);

}

logger.info("executeQueryBean: " + sql);

ResultSet rs = stmt.executeQuery();

int count = 0;

int maxCount = 1000;

List<T> list = new ArrayList<T>();

try {

while (rs.next()) {

Class<?> clazz\_tmp = clazz;

T t = (T) clazz.newInstance();

do {

Field[] fileds = clazz.getDeclaredFields();

for (Field field : fileds) {

try {

String fieldName = field.getName();

Object value = rs.getObject(fieldName);

if (value != null) {

field.setAccessible(true);

field.set(t, value);

}

} catch (Exception e) {

// logger.error("set field error", e);

}

}

} while (!(clazz\_tmp = clazz\_tmp.getSuperclass()).equals(Object.class));

list.add(t);

count++;

if(count >= maxCount)

break;

}

} catch (Throwable e) {

logger.error("foreach ResultSet error", e);

} finally {

rs.close();

stmt.close();

con.close();

}

return list;

}

/\*\*

\* 查询

\*

\* @param sql

\* {@link String} sql语句 { select \* from table where id=? }

\* @param params

\* {@link Object[]} 参数

\* @return {@link List<Map>} 查询结果

\* @throws SQLException

\*/

public List<Map<String, Object>> queryMap(String sql, Object[] params) throws SQLException {

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

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

Object param = params[i];

stmt.setObject(i + 1, param);

}

logger.info("executeQueryMap: " + sql);

ResultSetMetaData rsmd = stmt.getMetaData();

ResultSet rs = stmt.executeQuery();

int count = 0;

int maxCount = 1000;

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

List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();

try {

for (int i = 0; i < rsmd.getColumnCount(); i++) {

String columnName = rsmd.getColumnName(i + 1);

metas.add(columnName);

}

while (rs.next()) {

Map<String, Object> map = new HashMap<String, Object>();

for (String columnName : metas) {

Object value = rs.getObject(columnName);

if (value != null)

map.put(columnName, value);

}

list.add(map);

count++;

if(count >= maxCount)

break;

}

} catch (Throwable e) {

logger.error("foreach ResultSet error", e);

} finally {

rs.close();

stmt.close();

con.close();

}

return list;

}

public long getLastInsertId(Connection con) throws SQLException {

PreparedStatement stmt = con.prepareStatement("select @@IDENTITY as value");

ResultSet rs = stmt.executeQuery();

rs.next();

long id = rs.getLong(1);

return id;

}

/\*\*

\* 插入数据

\*

\* @param sql

\* {@link String} sql语句 { insert into table(id) values(?) }

\* @param params

\* {@link Object}[] 参数

\* @return {@link long} 主键

\* @throws SQLException

\*/

public long insert(String sql, Object[] params) throws SQLException {

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

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

stmt.setObject(i + 1, params[i]);

}

logger.info("executeInsert: " + sql);

stmt.execute();

long id = getLastInsertId(con);

stmt.close();

con.close();

return id;

}

/\*\*

\* 插入数据

\*

\* @param sql

\* {@link String} sql语句 { insert into table(id) values(#id#) }

\* @param t

\* {@link T} JavaBean

\* @return {@link long} 主键

\* @throws SQLException

\*/

public <T> long insert(String sql, T t) throws SQLException {

Class<?> clazz = t.getClass();

Field[] fileds = clazz.getDeclaredFields();

for (Field field : fileds) {

try {

String fieldName = field.getName();

field.setAccessible(true);

Object value = field.get(t);

if(value == null)

continue;

String value\_string = getValueString(value);

String replace = "#" + fieldName + "#";

sql = sql.replaceAll(replace, value\_string);

} catch (IllegalArgumentException | SecurityException | IllegalAccessException e) {

logger.error("get field error", e);

}

}

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

logger.info("executeInsert: " + sql);

try {

stmt.execute();

long id = getLastInsertId(con);

return id;

} catch (Throwable e) {

throw e;

} finally {

stmt.close();

con.close();

}

}

/\*\*

\* 修改JavaBean

\*

\* @param sql

\* {@link String} sql语句 { update table set column=#column# where id=#id# }

\* @param t

\* {@link T} JavaBean

\* @return {@link int} 影响行数

\* @throws SQLException

\*/

public <T> int update(String sql, T t) throws SQLException {

Class<?> clazz = t.getClass();

do {

Field[] fileds = clazz.getDeclaredFields();

for (Field field : fileds) {

try {

String fieldName = field.getName();

field.setAccessible(true);

Object value = field.get(t);

if (value == null)

continue;

String value\_string = getValueString(value);

String replace = "#" + fieldName + "#";

sql = sql.replaceAll(replace, value\_string);

} catch (IllegalArgumentException | SecurityException

| IllegalAccessException e) {

logger.error("get field error", e);

}

}

} while (!(clazz = clazz.getSuperclass()).equals(Object.class));

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

logger.info("executeUpdate: " + sql);

try {

int count = stmt.executeUpdate();

return count;

} catch (Throwable e) {

throw e;

} finally {

stmt.close();

con.close();

}

}

/\*\*

\* 批量修改JavaBean

\*

\* @param sql

\* {@link String} sql语句 { update table set column=? where id=? }

\* @param c

\* {@link Collection<T>} JavaBean集合

\* @param filedNames

\* {@link String}[] JavaBean 属性名称(按sql语句的顺序排列) new String[]{"column", "id"}

\* @return {@link int[]}

\* @throws SQLException

\*/

public <T> int[] updateBatch(String sql, Collection<T> c, String[] filedNames) throws SQLException {

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

for (T t : c) {

Class<?> clazz = t.getClass();

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

try {

String fieldName = filedNames[i];

Field field = clazz.getDeclaredField(fieldName);

field.setAccessible(true);

Object value = field.get(t);

if (value == null)

continue;

stmt.setObject(i + 1, value);

} catch (IllegalArgumentException | SecurityException | IllegalAccessException | NoSuchFieldException e) {

logger.error("get field error", e);

}

}

stmt.addBatch();

}

logger.info("executeUpdateBatch: " + sql);

try {

int[] counts = stmt.executeBatch();

return counts;

} catch (Throwable e) {

throw e;

} finally {

stmt.close();

con.close();

}

}

/\*\*

\* 删除

\*

\* @param sql

\* {@link String} sql语句 { delete from table where id=? }

\* @param params

\* {@link Object}[] 参数

\* @return {@link int} 影响行数

\* @throws SQLException

\*/

public int delete(String sql, Object[] params) throws SQLException {

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

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

stmt.setObject(i + 1, params[i]);

}

logger.info("executeDelete: " + sql);

try {

int count = stmt.executeUpdate();

return count;

} catch (Throwable e) {

throw e;

} finally {

stmt.close();

con.close();

}

}

/\*\*

\* 删除

\*

\* @param sql

\* {@link String} sql语句 { delete from table where id=#id# }

\* @param t

\* {@link T} JavaBean

\* @return {@link int} 影响行数

\* @throws SQLException

\*/

public <T> int delete(String sql, T t) throws SQLException {

Class<?> clazz = t.getClass();

Field[] fileds = clazz.getDeclaredFields();

for (Field field : fileds) {

try {

String fieldName = field.getName();

field.setAccessible(true);

Object value = field.get(t);

if(value == null)

continue;

String value\_string = getValueString(value);

String replace = "#" + fieldName + "#";

sql = sql.replaceAll(replace, value\_string);

} catch (IllegalArgumentException | SecurityException | IllegalAccessException e) {

logger.error("get field error", e);

}

}

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

logger.info("executeDelete: " + sql);

try {

int count = stmt.executeUpdate();

return count;

} catch (Throwable e) {

throw e;

} finally {

stmt.close();

con.close();

}

}

/\*\*

\* 执行sql语句

\*

\* @param sql

\* {@link String} sql语句

\* @return {@link boolean} 是否执行成功

\* @throws SQLException

\*/

public boolean execute(String sql) throws SQLException {

Connection con = dataSource.getConnection();

PreparedStatement stmt = con.prepareStatement(sql);

logger.info("execute: ");

try {

boolean success = stmt.execute();

return success;

} catch (Throwable e) {

throw e;

} finally {

stmt.close();

con.close();

}

}

protected String getValueString(Object value) {

String value\_string = "";

if (value instanceof String) {

value\_string = "'" + value.toString() + "'";

} else if (value instanceof Date) {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

value\_string = "'" + sdf.format((Date) value) + "'";

} else {

value\_string = value.toString();

}

return value\_string;

}

public static void main(String[] args) {

try {

} catch (Throwable e) {

e.printStackTrace();

}

}

}

class TestBean {

private long id;

// public TestBean(long id) {

// this.id = id;

// }

// public TestBean() {}

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

}

package frame.http;

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.util.HashMap;

import java.util.Map;

import org.apache.log4j.Logger;

import org.json.JSONObject;

import service.account.Account;

import com.sun.net.httpserver.Headers;

import com.sun.net.httpserver.HttpExchange;

import frame.Cache;

import frame.Command;

import frame.Factory;

public abstract class HttpCmd extends Command {

protected Logger logger = Logger.getLogger("http");

protected Map<String, String> result = new HashMap<String, String>();

protected HttpExchange http;

protected String method;

protected String params\_string;

protected boolean verify\_session = false;

public void start() {

Cache cache = Cache.getInstance();

if (verify\_session) {

JSONObject params = getJSONObject();

long id = params.optLong("id", 0);

String session = params.optString("session", "");

if (id == 0 || session.equals("")) {

Map<String, String> result = new HashMap<String, String>();

result.put("ret", "-1");

result.put("msg", "id or session is null");

response(result);

return;

}

Account account = cache.get("Account\_id" + id);

if (account == null) {

Map<String, String> result = new HashMap<String, String>();

result.put("ret", "-2");

result.put("msg", "session is expire");

response(result);

return;

}

if (!account.getSession().equals(session)) {

Map<String, String> result = new HashMap<String, String>();

result.put("ret", "-3");

result.put("msg", "session error");

response(result);

return;

}

} else {

}

super.start();

}

protected void response() {

response(result);

}

protected void response(Map<String, String> result) {

String resultString = new JSONObject(result).toString();

response(resultString);

}

protected void response(String result) {

try {

logger.info(result);

byte[] bytes = result.getBytes("utf-8");

Headers responseHeaders = http.getResponseHeaders();

responseHeaders.set("Content-type", "text/plain; charset=utf-8");

responseHeaders.set("Access-Control-Allow-Origin", "\*");

OutputStream os = http.getResponseBody();

http.sendResponseHeaders(200, bytes.length);

os.write(bytes);

os.flush();

} catch (Throwable e) {

logger.error("response", e);

}

}

public String readString() {

try {

if (params\_string != null) {

return params\_string;

}

InputStream is = http.getRequestBody();

BufferedReader br = new BufferedReader(new InputStreamReader(is));

StringBuffer sb = new StringBuffer();

String s;

while ((s = br.readLine()) != null) {

sb.append(s);

}

logger.info(sb.toString());

params\_string = sb.toString();

return params\_string;

} catch (Throwable e) {

logger.error("readString" ,e);

}

return "";

}

protected Map<String, String> getParams() {

String params\_string = readString();

Map<String, String> map = new HashMap<String, String>();

if (params\_string == null || params\_string.equals(""))

return map;

String[] key\_values\_string = params\_string.split("&");

for (String key\_values : key\_values\_string) {

int i = key\_values.indexOf("=");

String key = key\_values.substring(0, i);

String value = key\_values.substring(i + 1);

map.put(key, value);

}

return map;

}

protected JSONObject getJSONObject() {

try {

String params\_string = readString();

if ("".equals(params\_string)) {

return new JSONObject();

}

return new JSONObject(params\_string);

} catch (Throwable e) {

logger.error("getJSONObject", e);

}

return new JSONObject();

}

@SuppressWarnings({ "rawtypes", "unchecked" })

public static void register(String name, Class<? extends HttpCmd> clazz) {

HttpCmdFactory httpCmdFactory = new HttpCmdFactory();

httpCmdFactory.factory = new Factory(clazz);

HttpCmdFactory.factorys.put(name, httpCmdFactory);

}

}

package frame.http;

import java.util.HashMap;

import java.util.Map;

import frame.Factory;

public class HttpCmdFactory {

protected static Map<String, HttpCmdFactory> factorys = new HashMap<String, HttpCmdFactory>();

protected Factory<? extends HttpCmd> factory;

public HttpCmd createInstance() {

return factory.createInstance();

}

public Factory<? extends HttpCmd> getFactory() {

return factory;

}

}

package frame.http;

import java.io.IOException;

import java.io.OutputStream;

import java.io.PrintStream;

import org.apache.log4j.Logger;

import com.sun.net.httpserver.Headers;

import com.sun.net.httpserver.HttpExchange;

public class HttpHandler implements com.sun.net.httpserver.HttpHandler {

protected Logger logger = Logger.getLogger("http");

@Override

public void handle(HttpExchange http) throws IOException {

try {

String uri = http.getRequestURI().getPath();

if (uri.contains("!"))

uri = uri.split("!")[0];

String cmd\_id = uri;

if(cmd\_id == null || cmd\_id.equals(""))

return;

if(cmd\_id.trim().equals("/favicon.ico")) {

http.sendResponseHeaders(200, 0);

} else {

HttpCmdFactory factory = HttpCmdFactory.factorys.get(cmd\_id);

if (factory == null) {

http.sendResponseHeaders(404, 0);

return;

}

HttpCmd cmd = factory.createInstance();

if (cmd == null) {

http.sendResponseHeaders(404, 0);

return;

}

String remoteAddress = http.getRemoteAddress().toString();

String method = http.getRequestMethod();

String requestURI = http.getRequestURI().toString();

logger.info(remoteAddress + "\t" + method + "\t" + requestURI);

cmd.http = http;

cmd.method = method;

cmd.start();

}

} catch (Throwable e) {

logger.error("Handler error", e);

responseError(http, e);

} finally {

http.close();

}

}

protected void responseError(HttpExchange http, Throwable t) {

try {

Headers responseHeaders = http.getResponseHeaders();

responseHeaders.set("Content-type", "text/plain; charset=utf-8");

responseHeaders.set("Access-Control-Allow-Origin", "\*");

OutputStream os = http.getResponseBody();

http.sendResponseHeaders(200, 100000);

t.printStackTrace(new PrintStream(os));

os.flush();

} catch (Throwable e) {

logger.error("response", e);

}

}

}

package frame.http;

import java.net.InetSocketAddress;

import java.util.concurrent.Executors;

import frame.Server;

public class HttpServer extends Server {

public HttpServer(int port) {

super(port);

}

protected com.sun.net.httpserver.HttpServer server;

public void bootstrap() {

try {

server = com.sun.net.httpserver.HttpServer.create(new InetSocketAddress(port), 0);

server.setExecutor(Executors.newCachedThreadPool());

server.createContext("/", new HttpHandler());

server.start();

} catch (Throwable e) {

e.printStackTrace();

}

}

public void shutdown() {

server.stop(0);

}

}

package frame.socket;

import java.nio.charset.Charset;

import org.apache.log4j.Logger;

import org.jboss.netty.buffer.ChannelBuffer;

import org.jboss.netty.buffer.ChannelBuffers;

import org.jboss.netty.channel.Channel;

import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.handler.codec.frame.FrameDecoder;

import org.json.JSONObject;

public class Decoder extends FrameDecoder {

private Logger logger = Logger.getLogger("socket");

protected Object decode(ChannelHandlerContext ctx, Channel channel,

ChannelBuffer buffer) throws Exception {

if (!channel.isReadable() || buffer.readableBytes() < 9) {

return null;

}

byte[] msg = new byte[8];

buffer.getBytes(0, msg);

String str = new String(msg, Charset.forName("utf-8"));

if (str.startsWith("<policy")) {

String xml = "<cross-domain-policy> <allow-access-from domain=\"\*\" to-ports=\"\*\"/></cross-domain-policy> \0";

byte[] xmlarray = xml.getBytes(Charset.forName("utf-8"));

ChannelBuffer anquanHeader = ChannelBuffers.dynamicBuffer();

anquanHeader.writeBytes(xmlarray);

channel.write(anquanHeader);

return null;

}

// TGW

if (str.startsWith("GET")) {

buffer.clear();

return null;

}

byte b = buffer.readByte();

logger.debug("Decoder: " + b);

buffer.markReaderIndex();

Object message = null;

switch (b) {

case 0:

message = org.jboss.netty.handler.timeout.IdleState.READER\_IDLE;

break;

case 1:

message = decoderJSONObject(ctx, channel, buffer);

break;

case 2:

break;

default:

logger.error("error from: " + channel.getRemoteAddress());

logger.error("Decoder: " + b);

return null;

}

return message;

}

private Object decoderJSONObject(ChannelHandlerContext ctx, Channel channel, ChannelBuffer buffer) {

try {

// Short

if (buffer.readableBytes() < 8) {

logger.error("decoderJSONObject buffer.readableBytes() < 8");

return null;

}

int head = buffer.readInt(); // length

if (head < 0) {

logger.error("decoderJSONObject: head < 0");

buffer.resetReaderIndex();

logger.error(new String(buffer.array(), "UTF-8").trim());

return null;

}

if (buffer.readableBytes() < head) {

logger.error("decoderJSONObject: buffer.readableBytes() < head,buffer.readableBytes()="

+ buffer.readableBytes() + ",head=" + head);

buffer.resetReaderIndex();

return null;

}

byte[] body = new byte[head];

buffer.readBytes(body);

String jsonString = new String(body, "UTF-8");

if (!jsonString.startsWith("{")) {

logger.error("not json: " + jsonString);

logger.error("from: " + channel.getRemoteAddress().toString());

buffer.resetReaderIndex();

logger.error(new String(buffer.array(), "UTF-8").trim());

return null;

}

try {

return new JSONObject(jsonString);

} catch (Throwable e) {

logger.error("new JSONObject(jsonString)", e);

channel.close();

return null;

}

} catch (Throwable e) {

logger.error("decoderJSONObject", e);

}

return null;

}

}

package frame.socket;

import org.apache.log4j.Logger;

import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.handler.timeout.IdleState;

import org.jboss.netty.handler.timeout.IdleStateHandler;

import org.jboss.netty.util.HashedWheelTimer;

public class HeartBeat extends IdleStateHandler {

private static final org.jboss.netty.util.Timer timer = new HashedWheelTimer();

private static final int readerIdleTimeSeconds = 100;

private static final int writerIdleTimeSeconds = 0;

private static final int allIdleTimeSeconds = 0;

protected Logger logger = Logger.getLogger("socket");

public HeartBeat() {

super(timer, readerIdleTimeSeconds, writerIdleTimeSeconds, allIdleTimeSeconds);

}

@Override

protected void channelIdle(ChannelHandlerContext ctx, IdleState state,

long lastActivityTimeMillis) throws Exception {

super.channelIdle(ctx, state, lastActivityTimeMillis);

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

if (ctx.getChannel().isConnected()) {

ctx.getChannel().close();

logger.info("IdleState.READER\_IDLE " + readerIdleTimeSeconds + " " + ctx.getChannel().getRemoteAddress());

}

}

}

}

package frame.socket;

import java.util.HashMap;

import java.util.Map;

import org.apache.log4j.Logger;

import org.json.JSONObject;

import frame.Command;

import frame.Factory;

import frame.Session;

public abstract class SocketCmd extends Command {

protected Logger logger = Logger.getLogger("socket");

protected Map<String, String> result = new HashMap<String, String>();

protected SocketHandler handler;

protected Session session;

protected JSONObject json;

protected boolean verify\_session = false;

@SuppressWarnings({ "rawtypes", "unchecked" })

public static void register(String name, Class<? extends SocketCmd> clazz) {

SocketCmdFactory socketCmdFactory = new SocketCmdFactory();

socketCmdFactory.factory = new Factory(clazz);

SocketCmdFactory.factorys.put(name, socketCmdFactory);

}

public void response(Map<String, String> result) {

handler.getChannel().write(result);

}

public SocketHandler getHandler() {

return handler;

}

public void setHandler(SocketHandler handler) {

this.handler = handler;

}

}

package frame.socket;

import java.util.HashMap;

import java.util.Map;

import frame.Factory;

public class SocketCmdFactory {

protected static Map<String, SocketCmdFactory> factorys = new HashMap<String, SocketCmdFactory>();

protected Factory<? extends SocketCmd> factory;

public SocketCmd createInstance() {

return factory.createInstance();

}

public Factory<? extends SocketCmd> getFactory() {

return factory;

}

public static Map<String, SocketCmdFactory> getFactorys() {

return factorys;

}

}

package frame.socket;

import java.nio.ByteOrder;

import java.util.Map;

import org.apache.log4j.Logger;

import org.jboss.netty.buffer.ChannelBuffer;

import org.jboss.netty.buffer.DynamicChannelBuffer;

import org.jboss.netty.channel.Channel;

import org.jboss.netty.channel.ChannelHandlerContext;

import org.jboss.netty.channel.ChannelStateEvent;

import org.jboss.netty.channel.Channels;

import org.jboss.netty.channel.ExceptionEvent;

import org.jboss.netty.channel.MessageEvent;

import org.jboss.netty.channel.SimpleChannelHandler;

import org.json.JSONObject;

@SuppressWarnings("unchecked")

public abstract class SocketHandler extends SimpleChannelHandler {

protected Logger logger = Logger.getLogger("socket");

protected long id;

/\*\*

\* 通信管道

\*/

protected Channel channel;

/\*\*

\* 异常处理

\*/

public void exceptionCaught(ChannelHandlerContext ctx, ExceptionEvent e)

throws Exception {

try {

if (channel.isConnected()) {

channel.close();

}

Throwable t = e.getCause();

String message = t.getMessage();

if (message != null && message.equals("Connection reset by peer")) {

logger.error("Connection reset by peer");

return;

}

if (t.getClass().getSimpleName().equals("ClosedChannelException")) {

logger.error("ClosedChannelException", t);

return;

}

if (message != null && message.equals("远程主机强迫关闭了一个现有的连接。")) {

logger.error("远程主机强迫关闭了一个现有的连接。");

return;

}

logger.error("exceptionCaught", t);

} catch (Throwable t) {

logger.error("exceptionCaught error", t);

}

}

/\*\*

\* 收到消息

\*/

public void messageReceived(ChannelHandlerContext ctx, MessageEvent e)

throws Exception {

try {

Object o = e.getMessage();

if (o == null)

return;

if (o instanceof org.jboss.netty.handler.timeout.IdleState) {

logger.debug("HeartBet " + channel.getRemoteAddress().toString());

} else if (o instanceof JSONObject) {

JSONObject json = (JSONObject) o;

String cmd\_id = json.optString("cmd");

logger.info("request: " + id);

logger.info(json.toString());

Map<String, SocketCmdFactory> factorys = SocketCmdFactory.factorys;

SocketCmdFactory factory = factorys.get(cmd\_id);

if (factory == null) {

logger.error("not factory " + cmd\_id);

return;

}

SocketCmd cmd = factory.createInstance();

if (cmd == null) {

logger.error("not cmd " + cmd\_id);

return;

}

cmd.handler = this;

cmd.json = json;

cmd.start();

}

} catch (Throwable t) {

if (channel.isConnected()) {

channel.close();

}

logger.error("messageReceived error", t);

}

}

/\*\*

\* 发送事件

\*/

public void writeRequested(ChannelHandlerContext ctx, MessageEvent e)

throws Exception {

try {

if (!channel.isConnected())

return;

Object object = e.getMessage();

if (object == null)

return;

ChannelBuffer buffer = new DynamicChannelBuffer(ByteOrder.BIG\_ENDIAN, 1024);

if (object instanceof Map) {

// 发送的数据

Map<String, Object> map = (Map<String, Object>) object;

// json对象

final JSONObject jsonObject = new JSONObject(map);

// json字符串

final String jsonString = jsonObject.toString();

// 封装json

byte[] body = jsonString.getBytes("UTF-8");

int head = body.length;

buffer.writeByte(1);

buffer.writeInt(head);

buffer.writeBytes(body);

Channels.write(ctx, e.getFuture(), buffer);

logger.info("respones: " + id);

logger.info(jsonString);

} else if (object instanceof ChannelBuffer) {

logger.debug("response: " + id + " ChannelBuffer");

ChannelBuffer buffer\_src = (ChannelBuffer) object;

buffer.writeByte(2);

buffer.writeBytes(buffer\_src.array());

Channels.write(ctx, e.getFuture(), buffer);

}

} catch (Throwable t) {

if (channel.isConnected()) {

channel.close();

}

logger.error("writeRequested error", t);

}

}

/\*\*

\* 客户端断连

\*/

@Override

public void channelDisconnected(ChannelHandlerContext ctx,

ChannelStateEvent e) throws Exception {

try {

logger.error("channelDisconnected " + channel.getRemoteAddress().toString());

if (channel.isConnected()) {

channel.close();

}

} catch (Throwable t) {

logger.error("channelDisconnected error", t);

}

}

/\*\*

\* 断连事件

\*/

@Override

public void channelClosed(ChannelHandlerContext ctx, ChannelStateEvent e)

throws Exception {

try {

logger.error("channelClosed " + channel.getRemoteAddress().toString());

} catch (Throwable t) {

logger.error("channelClosed error", t);

}

}

/\*\*

\* 建立连接

\*/

@Override

public void channelConnected(ChannelHandlerContext ctx, ChannelStateEvent e)

throws Exception {

try {

channel = ctx.getChannel();

id = channel.getId();

logger.error("channelConnected " + channel.getRemoteAddress().toString());

} catch (Throwable t) {

logger.error("channelConnected error", t);

}

}

public Channel getChannel() {

return channel;

}

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

}

package frame.socket;

import java.net.InetSocketAddress;

import java.util.concurrent.Executors;

import org.jboss.netty.bootstrap.ServerBootstrap;

import org.jboss.netty.channel.ChannelFactory;

import org.jboss.netty.channel.ChannelPipeline;

import org.jboss.netty.channel.ChannelPipelineFactory;

import org.jboss.netty.channel.Channels;

import org.jboss.netty.channel.socket.nio.NioServerSocketChannelFactory;

import frame.Server;

public class SocketServer extends Server {

public SocketServer(int port) {

super(port);

}

protected ServerBootstrap serverBootstrap;

public void bootstrap() {

bootstrap(SocketHandler.class);

}

public void bootstrap(Class<? extends SocketHandler> clazz) {

serverBootstrap = new ServerBootstrap();

ChannelPipelineFactory channelPipelineFactory = new ChannelPipelineFactory() {

private Decoder decoder = new Decoder();

private HeartBeat heartBeat = new HeartBeat();

@Override

public ChannelPipeline getPipeline() throws Exception {

ChannelPipeline p = Channels.pipeline();

p.addLast("Decoder", decoder);

p.addLast("HeartBeat", heartBeat);

p.addLast("Handler", clazz.newInstance());

return p;

}

};

ChannelFactory channelFactory = new NioServerSocketChannelFactory(Executors.newCachedThreadPool(), Executors.newCachedThreadPool());

serverBootstrap.setPipelineFactory(channelPipelineFactory);

serverBootstrap.setFactory(channelFactory);

serverBootstrap.setOption("client.tcpNodelay", true);

serverBootstrap.setOption("client.keepAlive", true);

serverBootstrap.bind(new InetSocketAddress(port));

}

public void shutdown() {

serverBootstrap.releaseExternalResources();

}

}

package frame;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import org.apache.log4j.Logger;

import com.danga.MemCached.MemCachedClient;

import com.danga.MemCached.SockIOPool;

public class Cache {

protected Cache() {

// init("120.25.80.235:11211");

}

protected static Cache instance = new Cache();

public static Cache getInstance() {

return instance;

}

protected Logger logger = Logger.getLogger("cache");

protected List<String> addrs = new ArrayList<String>();

protected MemCachedClient client;

public boolean set(String key, Object value, int seconds) {

try {

if (value == null)

return false;

if (seconds <= 0) {

return client.set(key, value);

} else {

Date date = new Date(System.currentTimeMillis() + seconds \* 1000);

return client.set(key, value, date);

}

} catch (Throwable e) {

logger.error("set", e);

}

return false;

}

public <T> T get(String key) {

try {

Object o = client.get(key);

if (o == null)

return null;

@SuppressWarnings("unchecked")

T t = (T) o;

return t;

} catch (Throwable e) {

logger.error("get", e);

}

return null;

}

public boolean delete(String key) {

try {

return client.delete(key);

} catch (Throwable e) {

logger.error("delete", e);

}

return false;

}

public void init(String... addrs) {

try {

SockIOPool sockIOPool = SockIOPool.getInstance();

// memcached服务器地址

sockIOPool.setServers(addrs);

// 容错开关

sockIOPool.setFailover(true);

// 连接失败恢复开关

sockIOPool.setFailback(true);

// 心跳开关

sockIOPool.setAliveCheck(true);

// 初始连接、最小连接、最大连接、最大处理时间

sockIOPool.setInitConn(2);

sockIOPool.setMinConn(2);

sockIOPool.setMaxConn(2);

sockIOPool.setMaxIdle(60000);

// 主线程休眠时间

sockIOPool.setMaintSleep(30);

// tcp连接超时

sockIOPool.setNagle(false);

sockIOPool.setSocketTO(1000);

sockIOPool.setSocketConnectTO(3000);

// 初始化

sockIOPool.initialize();

client = new MemCachedClient();

// 压缩设置

// client.setCompressEnable(true);

// client.setCompressThreshold(1024);

// 设置原始类型为string(可加快速度)

client.setPrimitiveAsString(false);

// 编码

client.setDefaultEncoding("utf-8");

} catch (Throwable e) {

logger.error("init", e);

}

}

public void shutdown() {

try {

} catch (Throwable e) {

logger.error("shutdown", e);

}

}

}

package frame;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.net.URI;

import java.net.URISyntaxException;

import java.net.URL;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.zip.ZipEntry;

import java.util.zip.ZipInputStream;

public class ClassLoader {

public synchronized static void loadClassesFromPath() {

if (!isLoad) {

try {

URL resource = java.lang.ClassLoader.getSystemResource("");

if (resource == null)

resource = ClassLoader.class.getResource("");

URI uri = resource.toURI();

if (String.valueOf(uri).indexOf("jar!") > 0) {

resource = ClassLoader.class.getProtectionDomain()

.getCodeSource().getLocation();

uri = resource.toURI();

}

String property = new File(uri).getPath();

String[] paths = property.split(";");

for (String path : paths) {

File file = new File(path);

if (file.isFile() && path.endsWith(".jar")) {

listClassesInZip(file, "/");

} else if (file.isDirectory()) {

listClassesInDirectory(path + File.separatorChar, file);

}

}

} catch (URISyntaxException e) {

}

}

}

private synchronized static void listClassesInDirectory(String rootPath,

File file) {

File[] subFiles = file.listFiles();

for (File subFile : subFiles) {

if (subFile.canRead()) {

if (subFile.isFile()) {

String path = subFile.getPath();

if (path.endsWith(".class")) {

try {

String className = getClassName(path.substring(rootPath.length()));

CLASSES.add(Class.forName(className));

} catch (Throwable e) {

}

} else if (path.endsWith(".jar")) {

listClassesInZip(subFile, "/");

}

} else if (subFile.isDirectory()) {

listClassesInDirectory(rootPath, subFile);

}

}

}

}

private synchronized static void listClassesInZip(File jarFile,

String specPath) {

ZipInputStream in = null;

try {

in = new ZipInputStream(new FileInputStream(jarFile));

ZipEntry ze = null;

while ((ze = in.getNextEntry()) != null) {

if (ze.isDirectory()) {

continue;

} else {

try {

String name = ze.getName();

if (!name.endsWith(".class"))

continue;

if (specPath.substring(1).length() > 0

&& name.indexOf(specPath.substring(1)) < 0)

continue;

String className = getClassName(name);

CLASSES.add(Class.forName(className));

} catch (Throwable e) {

}

}

}

} catch (Throwable e) {

} finally {

if (in != null) {

try {

in.close();

} catch (IOException e) {

}

}

}

}

private static String getClassName(String path) {

String className = path.replace('/', '.').replace('\\', '.').replaceAll(".class", "");

return className;

}

public static List<Class<?>> getSubClasses(Class<?> clazz) {

List<Class<?>> subClasses = SUB\_CLASSES\_MAP.get(clazz);

if (subClasses == null) {

subClasses = new ArrayList<Class<?>>(10);

for (Class<?> tmpClass : CLASSES) {

if (clazz.isAssignableFrom(tmpClass)

&& !tmpClass.isAssignableFrom(clazz)) {

subClasses.add(tmpClass);

}

}

SUB\_CLASSES\_MAP.put(clazz, subClasses);

}

return Collections.unmodifiableList(subClasses);

}

public static final List<Class<?>> CLASSES = new ArrayList<Class<?>>(200);

private static final Map<Class<?>, List<Class<?>>> SUB\_CLASSES\_MAP = new HashMap<Class<?>, List<Class<?>>>();

private static boolean isLoad;

}

**package** frame;

**public** **abstract** **class** Command {

**protected** **int** id;

**protected** String name;

**protected** Command parent;

**protected** Command son;

**public** **void** start() {

init();

execute();

**if** (son != **null**)

son.start();

**if** (parent != **null**)

parent.callback();

}

**public** **void** init() {}

**public** **void** execute() {}

**public** **void** callback() {}

}

**package** frame;

**public** **class** Factory<T> {

/\*\*

\* 实例Class

\*/

**protected** Class<T> instance\_class;

**public** Factory(Class<T> clazz) {

instance\_class = clazz;

}

**public** T createInstance() {

**try** {

T t = instance\_class.newInstance();

**return** t;

} **catch** (Throwable e) {

e.printStackTrace();

}

**return** **null**;

}

**public** Class<T> getInstance\_class() {

**return** instance\_class;

}

**public** **void** setInstance\_class(Class<T> instance\_class) {

**this**.instance\_class = instance\_class;

}

}

package frame;

import java.io.File;

import java.io.IOException;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

import java.net.URL;

import java.net.URLClassLoader;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* 依赖包加载器

\*

\* @author wqc

\* @date 2014-10-31

\* @version 1.0

\*

\*/

public class JarLoader {

protected static List<String> jars\_path = new ArrayList<String>();

protected static URLClassLoader urlClassLoader;

protected static Method addURL;

static {

try {

urlClassLoader = (URLClassLoader) java.lang.ClassLoader.getSystemClassLoader();

addURL = URLClassLoader.class.getDeclaredMethod("addURL",new Class[] { URL.class });

addURL.setAccessible(true);

} catch (Throwable e) {

e.printStackTrace();

}

}

public static void load(String lib) throws IOException {

File file = new File(lib);

if (file.isDirectory()) {

File[] files = file.listFiles();

for (File f : files) {

String path = f.getPath();

load(path);

}

} else if (file.isFile()) {

String path = file.getPath();

if (path.endsWith("jar") || path.endsWith("zip")) {

loadJar(file.toURI().toURL().toString());

}

}

}

public static void loadJar(String path) {

try {

if (jars\_path.contains(path))

return;

jars\_path.add(path);

addURL.invoke(urlClassLoader, new URL(path));

System.out.println("load jar: " + path);

} catch (Throwable e) {

e.printStackTrace();

}

}

public static void addusr\_paths(String path) {

try {

Field field;

String paths[];

field = java.lang.ClassLoader.class.getDeclaredField("usr\_paths");

field.setAccessible(true);

paths = (String[]) field.get(null);

for (String string : paths) {

if (string.equals(path))

return;

}

String[] pathArray = new String[paths.length + 1];

System.arraycopy(paths, 0, pathArray, 0, paths.length);

pathArray[paths.length] = path;

field.set(null, pathArray);

} catch (Throwable e) {

e.printStackTrace();

}

}

}

package frame;

import java.util.concurrent.ScheduledFuture;

import java.util.concurrent.ScheduledThreadPoolExecutor;

import java.util.concurrent.TimeUnit;

public class ScheduledExecutor {

// 定时任务执行者

public static final ScheduledThreadPoolExecutor scheduler = new ScheduledThreadPoolExecutor(Runtime.getRuntime().availableProcessors());

static {

Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {

public void run() {

scheduler.shutdownNow();

}

}

));

}

/\*\*

\* 以固定速率执行周期任务

\*

\* @param runnable

\* 线程

\* @param initialDelay

\* 在initialDelay之后开始执行

\* @param period

\* 执行间隔

\* @param unit

\* 时间单位

\*

\* @return ScheduledFuture

\*/

public static ScheduledFuture<?> scheduleAtFixedRate(Runnable runnable,

long initialDelay, long period, TimeUnit unit) {

return scheduler.scheduleAtFixedRate(runnable, initialDelay, period,

unit);

}

/\*\*

\* 执行启动延时任务

\*

\* @param runnable

\* 线程

\* @param delay

\* 在delay之后开始执行(单位:毫秒)

\* @return ScheduledFuture

\*/

public static ScheduledFuture<?> schedule(Runnable runnable, long delay) {

return scheduler.schedule(runnable, delay, TimeUnit.MILLISECONDS);

}

/\*\*

\* 启动线程

\*

\* @param Runnable

\* 线程

\*/

public static void execute(Runnable runnable) {

scheduler.execute(runnable);

}

/\*\*

\* 取消定时任务

\*

\* @param scheduled

\* 任务

\* @return boolean 是否取消

\*/

public static boolean cancel(ScheduledFuture<?> scheduled) {

if (scheduled == null)

return false;

return scheduled.cancel(false);

}

}

**package** frame;

**public** **abstract** **class** Server {

**public** **static** String *host* = "localhost";

**protected** **int** port;

**public** Server(**int** port) {

**this**.port = port;

}

**public** **abstract** **void** bootstrap();

**public** **abstract** **void** shutdown();

}

package util;

import java.io.InputStream;

import java.net.HttpURLConnection;

import java.net.URL;

import java.security.KeyStore;

import java.security.Security;

import java.security.cert.CertificateException;

import java.security.cert.X509Certificate;

import javax.net.ssl.HostnameVerifier;

import javax.net.ssl.HttpsURLConnection;

import javax.net.ssl.SSLContext;

import javax.net.ssl.SSLSession;

import javax.net.ssl.SSLSocketFactory;

import javax.net.ssl.TrustManager;

import javax.net.ssl.TrustManagerFactory;

import javax.net.ssl.X509TrustManager;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.xpath.XPath;

import javax.xml.xpath.XPathConstants;

import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import com.sun.xml.internal.messaging.saaj.util.ByteInputStream;

public class ConfigXml {

protected String httpUrl;

protected ByteInputStream bis;

protected DocumentBuilderFactory document\_builder\_factory;

protected DocumentBuilder document\_builder;

protected Document document;

protected static XPath xpath = XPathFactory.newInstance().newXPath();

public ConfigXml(String httpUrl) {

try {

this.httpUrl = httpUrl;

URL url = new URL(httpUrl);

HttpURLConnection huc = (HttpURLConnection) url.openConnection();

if (httpUrl.startsWith("https")) {

HttpsURLConnection hucs = (HttpsURLConnection) huc;

TrustManager[] tm = { new MyX509TrustManager() };

SSLContext sslContext = SSLContext.getInstance("SSL", "SunJSSE");

sslContext.init(null, tm, new java.security.SecureRandom());

SSLSocketFactory ssf = sslContext.getSocketFactory();

hucs.setSSLSocketFactory(ssf);

HttpsURLConnection.setDefaultSSLSocketFactory(ssf);

hucs.setHostnameVerifier(new TrustAnyHostnameVerifier());

}

huc.setRequestMethod("GET");

huc.setDoOutput(false);

huc.setDoInput(true);

huc.setConnectTimeout(5000);

huc.setReadTimeout(5000);

huc.setRequestProperty("Content-type", "text/plain;charset=UTF-8");

huc.connect();

int response\_code = huc.getResponseCode();

if (response\_code != 200) {

throw new RuntimeException("ResponseCode=" + response\_code);

}

int countent\_length = huc.getContentLength();

byte[] datas = new byte[countent\_length];

InputStream is = huc.getInputStream();

is.read(datas);

this.bis = new ByteInputStream(datas, countent\_length);

document\_builder\_factory = DocumentBuilderFactory.newInstance();

document\_builder = document\_builder\_factory.newDocumentBuilder();

document = document\_builder.parse(bis);

} catch (Throwable e) {

e.printStackTrace();

throw new RuntimeException(e.getMessage());

}

}

public ConfigXml(InputStream is) {

try {

int length = is.available();

byte[] datas = new byte[length];

is.read(datas);

this.bis = new ByteInputStream(datas, length);

document\_builder\_factory = DocumentBuilderFactory.newInstance();

document\_builder = document\_builder\_factory.newDocumentBuilder();

document = document\_builder.parse(bis);

} catch (Throwable e) {

e.printStackTrace();

throw new RuntimeException(e.getMessage());

}

}

public String getValue(String expression) {

try {

Object o = xpath.compile(expression).evaluate(document, XPathConstants.NODE);

Node node = Node.class.cast(o);

return node.getTextContent().trim();

} catch (Exception e) {

e.printStackTrace();

}

return "";

}

public NodeList getNodeList(String expression) {

try {

Object o = xpath.compile(expression).evaluate(document, XPathConstants.NODESET);

NodeList node\_list = NodeList.class.cast(o);

return node\_list;

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

public static String getValue(Node node, String expression) {

try {

Object o = xpath.compile(expression).evaluate(node, XPathConstants.NODE);

Node n = Node.class.cast(o);

return n.getTextContent().trim();

} catch (Exception e) {

e.printStackTrace();

}

return "";

}

public static NodeList getNodeList(Node node, String expression) {

try {

Object o = xpath.compile(expression).evaluate(node, XPathConstants.NODESET);

NodeList l = NodeList.class.cast(o);

return l;

} catch (Throwable e) {

e.printStackTrace();

}

return null;

}

public static class MyX509TrustManager implements

javax.net.ssl.TrustManager, javax.net.ssl.X509TrustManager {

X509TrustManager sunJSSEX509TrustManager;

MyX509TrustManager() throws Exception {

// create a "default" JSSE X509TrustManager.

System.setProperty("javax.net.ssl.trustStore", "\*.keystore");

System.setProperty("java.protocol.handler.pkgs",

"com.sun.net.ssl.internal.www.protocol");

Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());

System.setProperty("java.protocol.handler.pkgs", "javax.net.ssl");

KeyStore ks = KeyStore.getInstance("JKS");

// ks.load(new

// FileInputStream("trustedCerts"),"passphrase".toCharArray());

TrustManagerFactory tmf = TrustManagerFactory.getInstance(

"SunX509", "SunJSSE");

tmf.init(ks);

TrustManager tms[] = tmf.getTrustManagers();

/\*

\* Iterate over the returned trustmanagers, look for an instance of

\* X509TrustManager. If found, use that as our "default" trust

\* manager.

\*/

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

if (tms[i] instanceof X509TrustManager) {

sunJSSEX509TrustManager = (X509TrustManager) tms[i];

return;

}

}

/\*

\* Find some other way to initialize, or else we have to fail the

\* constructor.

\*/

throw new Exception("Couldn't initialize");

}

/\*

\* Delegate to the default trust manager.

\*/

public void checkClientTrusted(X509Certificate[] chain, String authType)

throws CertificateException {

try {

sunJSSEX509TrustManager.checkClientTrusted(chain, authType);

} catch (CertificateException excep) {

// do any special handling here, or rethrow exception.

}

}

/\*

\* Delegate to the default trust manager.

\*/

public void checkServerTrusted(X509Certificate[] chain, String authType)

throws CertificateException {

try {

sunJSSEX509TrustManager.checkServerTrusted(chain, authType);

} catch (CertificateException excep) {

/\*

\* Possibly pop up a dialog box asking whether to trust the cert

\* chain.

\*/

}

}

/\*

\* Merely pass this through.

\*/

public X509Certificate[] getAcceptedIssuers() {

return sunJSSEX509TrustManager.getAcceptedIssuers();

}

}

public static class TrustAnyHostnameVerifier implements HostnameVerifier {

@Override

public boolean verify(String arg0, SSLSession arg1) {

return true;

}

}

}

package util;

import java.lang.reflect.Field;

import java.lang.reflect.Modifier;

import org.json.JSONObject;

public class JSONUtil {

public static String encodeJSONString(Object bean) {

return encodeJSON(bean).toString();

}

public static JSONObject encodeJSON(Object bean) {

JSONObject json = new JSONObject();

Class<?> clazz = bean.getClass();

do {

Field[] fileds = clazz.getDeclaredFields();

for (Field field : fileds) {

try {

if (Modifier.isStatic(field.getModifiers()))

continue;

String fieldName = field.getName();

field.setAccessible(true);

Object value = field.get(bean);

json.put(fieldName, value);

} catch (Exception e) {

}

}

} while (!(clazz = clazz.getSuperclass()).equals(Object.class));

return json;

}

}