金坤科创智慧人防监测综合管理系统软件V1.0

源代码

北京金坤科创技术有限公司

2017-08-24

前30页

package com.socket;

import com.beans.TemperatureHumidityInfo;

import com.beans.WaterInfo;

import com.common.util.BufferDataInit;

import com.util.global.LocalTime;

import com.util.global.UsedCache;

public class ServerUtils {

public static void realtime\_handler(byte[] received\_data, String[] str) {

if (21 == received\_data.length) {// 接受设备信息

if (received\_data[9] == 0x01) {// 温湿度信息

// 设备地址

int device\_id = received\_data[6];

// 解析温度数据

double temperature = (double) ((short) ((short) (received\_data[11] << 8) | (short) (received\_data[12] & 0xff))) / 10;

// 解析湿度数据

double humidity = (double) ((int) ((short) (received\_data[15] << 8) | (short) (received\_data[16] & 0xff)) & 0xffff) / 10;

// 解析电量数据

double electricity = (double) ((int) ((short) (received\_data[19] << 8) | (short) (received\_data[20] & 0xff)) & 0xffff) / 10;

TemperatureHumidityInfo temHumInfo = new TemperatureHumidityInfo();

if (UsedCache.temHumInfoMap.containsKey(Integer

.toString(device\_id))) {

temHumInfo = UsedCache.temHumInfoMap.get(Integer

.toString(device\_id));

if ("1".equals(temHumInfo.getOnlineStatus())) {

temHumInfo.setEntryTime(LocalTime.getDateTime());

// 把解析数据存入数据库

BufferDataInit.commonService.updateDate(

TemperatureHumidityInfo.class, new String[] {

"temperature='" + temperature + "'",

"humidity='" + humidity + "'",

"electricity='" + electricity + "'", },

"deviceAddress='" + device\_id + "'");

}

} else {

temHumInfo.setOnlineStatus("0");

temHumInfo.setEntryTime(LocalTime.getDateTime());

}

temHumInfo.setDeviceAddress(Integer.toString(device\_id));

temHumInfo.setTemperature(Double.toString(temperature));

temHumInfo.setHumidity(Double.toString(humidity));

temHumInfo.setElectricity(Double.toString(electricity));

// 把温湿度放入缓存

UsedCache.temHumInfoMap.put(Integer.toString(device\_id),

temHumInfo);

} else if (received\_data[9] == 0x0D) {// 水浸信息

// 设备地址

int device\_id = received\_data[6];

// 解析水浸数据

int waterStatus = 1 - (short) received\_data[12];

int oldWaterStatus = 1;

if (!UsedCache.waterInfoMap.isEmpty()) {

//获取有无水状态，用于对比状态变化，有变化则向数据库存储

oldWaterStatus = Integer.parseInt(UsedCache.waterInfoMap

.get(Integer.toString(device\_id)).getWaterStatus());

}

WaterInfo waterInfo = new WaterInfo();

if (UsedCache.waterInfoMap.containsKey(Integer

.toString(device\_id))) {

waterInfo = UsedCache.waterInfoMap.get(Integer

.toString(device\_id));

if ("1".equals(waterInfo.getOnlineStatus())) {

waterInfo.setEntryTime(LocalTime.getDateTime());

}

} else {

waterInfo.setOnlineStatus("0");

waterInfo.setEntryTime(LocalTime.getDateTime());

}

waterInfo.setDeviceAddress(Integer.toString(device\_id));

waterInfo.setWaterStatus(Integer.toString(waterStatus));

// 把水浸信息放入Map缓存

UsedCache.waterInfoMap.put(Integer.toString(device\_id),

waterInfo);

// 把第一次解析数据存入数据库

if (!BufferDataInit.commonService.findByHql(WaterInfo.class,

"deviceAddress='" + device\_id + "'", null).isEmpty()) {

//获取有无水状态，用于对比状态变化，有变化则向数据库存储

if (waterStatus != oldWaterStatus) {// 如果当前水浸值与缓存中水浸值不同，才往数据库中存，这样为了节省数据库空间

BufferDataInit.commonService.saveDate(waterInfo);

}

} else {

BufferDataInit.commonService.saveDate(waterInfo);

}

}

} else if (40 == received\_data.length) {// 接收在线状态查询信息

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

int j = Integer.parseInt(str[i]) % 8 - 1;

int byteNum = (int) Integer.parseInt(str[i]) / 8 + 10;

short a = (short) Math.pow(2, j);

int status = (((short) received\_data[byteNum - 1] & 0xff) & (a & 0xff)) >> j;

if (UsedCache.temHumInfoMap.containsKey(str[i])) {

TemperatureHumidityInfo temHumInfo = UsedCache.temHumInfoMap

.get(str[i]);

if (status == 1) {

temHumInfo.setOnlineStatus("1");

// 更新设备在线状态

BufferDataInit.commonService.updateDate(

TemperatureHumidityInfo.class,

new String[] { "onlineStatus='1'" },

"deviceAddress='"

+ temHumInfo.getDeviceAddress() + "'");

} else {

temHumInfo.setOnlineStatus("0");

// 更新设备在线状态

BufferDataInit.commonService.updateDate(

TemperatureHumidityInfo.class,

new String[] { "onlineStatus='0'" },

"deviceAddress='"

+ temHumInfo.getDeviceAddress() + "'");

}

UsedCache.temHumInfoMap.put(str[i], temHumInfo);

} else if (UsedCache.waterInfoMap.containsKey(str[i])) {

WaterInfo waterInfo = UsedCache.waterInfoMap.get(str[i]);

if (status == 1) {

waterInfo.setOnlineStatus("1");

// 保存水浸信息

// BufferDataInit.commonService.saveDate(waterInfo);

} else {

waterInfo.setOnlineStatus("0");

}

UsedCache.waterInfoMap.put(str[i], waterInfo);

}

}

}

}

}

/\*\*

\* 该类主要用于处理定位服务器传递来的值

\* @author ZHUZHIYONG

\*/

package com.socket;

import java.util.List;

import com.beans.AskHelpInfo;

import com.beans.StationErrorInfo;

import com.beans.alarms.Button1AlarmBean;

import com.beans.alarms.StationAlarmBean;

import com.common.util.BufferDataInit;

import com.form.PositionBean;

import com.ps.Host\_ds\_def;

import com.util.global.DwrScriptSessionManagerUtil;

import com.util.global.LocalTime;

import com.util.global.UsedCache;

public class PositionServerUtils {

/\*\*

\* 处理通过回调函数传递来的定位标签的位置信息

\*

\* @param message

\*/

public static void query\_abs\_pos\_response(String message) {

String[] dataStrings = message.split("\n");

String tag\_mac;

if (dataStrings.length > 1) {

for (int i = 1; i < dataStrings.length; i++) {

String[] single\_datas = dataStrings[i].split(",");

// 获取MAC

tag\_mac = new String(single\_datas[0]);

// 获取定位服务器传来X坐标

Double x\_pos = Double.parseDouble(single\_datas[2]);

// 获取定位服务器传来X坐标

Double y\_pos = Double.parseDouble(single\_datas[3]);

// 获取定位服务器传来X坐标

Double z\_pos = Double.parseDouble(single\_datas[4]);

// 获取定位服务器传来X坐标

int bid = Integer.valueOf(single\_datas[5]);

// 获取基站MAC地址

String ap = single\_datas[6];

// 判断人员map中是否包含该人员信息

if (UsedCache.userInfoMacMap.containsKey(tag\_mac)) {

PositionBean positionBean = new PositionBean();

positionBean.setMac(tag\_mac);

positionBean.setX\_pos(x\_pos);

positionBean.setY\_pos(y\_pos);

positionBean.setZ\_pos(z\_pos);

positionBean.setBid(bid);

positionBean.setNearestAp(ap);

positionBean.setPosTime(LocalTime.getDateTime());

positionBean.setUserInfo(UsedCache.userInfoMacMap

.get(tag\_mac));

if (UsedCache.button1\_alarm\_map.containsKey(tag\_mac)) {

positionBean.setAlarm("1");

} else {

positionBean.setAlarm("0");

}

UsedCache.positionBeanMap.put(tag\_mac, positionBean);// 把定位信息存入缓存

}

}

}

}

/\*\*

\* 处理基站错误事件

\*

\* @param received\_data

\*/

@SuppressWarnings("unchecked")

public static void station\_alarm\_handler(String received\_data) {

boolean station\_error\_changed = false;

String[] received\_data\_array = received\_data

.split(Host\_ds\_def.CMD\_SPLIT);

// [0]为消息头 [1]为消息类型

for (int i = 2; i < received\_data\_array.length; i++) {

String[] single\_datas = received\_data\_array[i].split(",");

// 基站编号,当前是否有故障(Y/N),故障发生或者是恢复的时间戳 CRLF

if (single\_datas.length < 3) {

continue;

}

// 解析到基站MAC

String station\_mac = new String(single\_datas[0]);

// 是否欠电

boolean is\_error = single\_datas[1].equals("Y") ? true : false;

// 封装基站信息所有信息

StationAlarmBean stationAlarmBean = new StationAlarmBean();

if (!UsedCache.m\_station\_map.containsKey(station\_mac)) {

// 系统中无该基站记录，不处理

continue;

}

try {

if (is\_error) { // 基站错误报警

if (!UsedCache.m\_station\_error\_alarm\_map

.containsKey(station\_mac)) {

station\_error\_changed = true;

// 基站位置

stationAlarmBean.setApInfo(UsedCache.m\_station\_map

.get(station\_mac));

stationAlarmBean.setHandle(false);

stationAlarmBean.setAlarmTime(LocalTime.getDateTime());

UsedCache.m\_station\_error\_alarm\_map.put(station\_mac,

stationAlarmBean);

// 添加到基站报警信息表

StationErrorInfo stationErrorInfo = new StationErrorInfo();

stationErrorInfo.setStationNo(UsedCache.m\_station\_map

.get(station\_mac).getBaseNo());

stationErrorInfo.setErrorTime(LocalTime.getDateTime());

stationErrorInfo.setStationMac(station\_mac);

stationErrorInfo.setxLocation(UsedCache.m\_station\_map

.get(station\_mac).getPos\_x());

stationErrorInfo.setyLocotion(UsedCache.m\_station\_map

.get(station\_mac).getPos\_y());

stationErrorInfo.setStationName(UsedCache.m\_station\_map

.get(station\_mac).getBaseName());

stationErrorInfo.setIsHandle("0");

BufferDataInit.commonService.saveDate(stationErrorInfo);

try {

Thread.sleep(2000);// 线程阻塞2秒

} catch (Exception e) {

}

}

} else {

if (UsedCache.m\_station\_error\_alarm\_map

.containsKey(station\_mac)) {

List<StationErrorInfo> list = BufferDataInit.commonService

.findByHqlPage(

StationErrorInfo.class,

"stationMac='"

+ station\_mac

+ "' and isHandle='0' order by errorTime desc",

null, 1, 1);

if (!list.isEmpty()) {

BufferDataInit.commonService

.updateDate(

StationErrorInfo.class,

new String[] {

"isHandle='1'",

"handleTime='"

+ LocalTime

.getDateTime()

+ "'" },

"stationMac='"

+ station\_mac

+ "' and errorTime='"

+ UsedCache.m\_station\_error\_alarm\_map

.get(station\_mac)

.getAlarmTime()

+ "'");

}

UsedCache.m\_station\_error\_alarm\_map.remove(station\_mac);

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

// 推送基站报警

push\_alarm\_to\_browser(false, station\_error\_changed);

}

/\*\*

\* 处理定位标签第一个按键报警事件(02紧急求助报警状态)

\*

\* @param received\_data

\*/

public static void button1\_alarm\_handler(String received\_data) {

boolean askhelp\_changed = false; // 紧急求助是否发生

String[] received\_data\_array = received\_data

.split(Host\_ds\_def.CMD\_SPLIT);

// [0]为消息头 [1]为消息类型

for (int i = 2; i < received\_data\_array.length; i++) {

String[] single\_datas = received\_data\_array[i].split(",");

if (single\_datas.length < 3) {

continue;

}

// 解析报警的TAG\_MAC

String tag\_mac = new String(single\_datas[0]);

// 人员携带的电子安全护照

if (UsedCache.userInfoMacMap.containsKey(tag\_mac)) {

askhelp\_changed = true;

// 生成报警文字信息

String message = "";

if (UsedCache.positionBeanMap.containsKey(tag\_mac)) {

message = "紧急求助，"

+ UsedCache.positionBeanMap.get(tag\_mac)

.getUserInfo().getUserName() + "，在坐标"

+ UsedCache.positionBeanMap.get(tag\_mac).getX\_pos()

+ "米处，呼救，请速去处理！";

}

try {

Button1AlarmBean button1AlarmBean = new Button1AlarmBean();

if (!UsedCache.button1\_alarm\_map.containsKey(tag\_mac)) {

button1AlarmBean.setTag\_mac(tag\_mac);

button1AlarmBean.setPeopleName(UsedCache.userInfoMacMap

.get(tag\_mac).getUserName());

button1AlarmBean

.setAskHelpTime(LocalTime.getDateTime());

button1AlarmBean.setHandled("0");

button1AlarmBean.setAlarm\_voice\_file\_name(message);

// 把紧急求助相关信息放入缓存

UsedCache.button1\_alarm\_map.put(tag\_mac,

button1AlarmBean);

AskHelpInfo askHelpInfo = new AskHelpInfo();

askHelpInfo.setAskHelpTime(button1AlarmBean

.getAskHelpTime());

askHelpInfo.setIsHandled("0");

askHelpInfo.setPeopleName(button1AlarmBean

.getPeopleName());

askHelpInfo.setRegionName(UsedCache.positionBeanMap

.get(tag\_mac).getX\_pos()

+ ","

+ UsedCache.positionBeanMap.get(tag\_mac)

.getY\_pos()

+ ","

+ UsedCache.positionBeanMap.get(tag\_mac)

.getZ\_pos());

// 把紧急求助信息添加到数据库

BufferDataInit.commonService.saveDate(askHelpInfo);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

// 推送报警消息

push\_alarm\_to\_browser(askhelp\_changed, false);

}

/\*\*

\* 处理定位标签第二个按键报警事件(08取消紧急求助报警)

\*

\* @param received\_data

\*/

@SuppressWarnings("unchecked")

public static void button2\_alarm\_handler(String received\_data) {

boolean askhelp\_changed = false;

String[] received\_data\_array = received\_data

.split(Host\_ds\_def.CMD\_SPLIT);

// [0]为消息头 [1]为消息类型

for (int i = 2; i < received\_data\_array.length; i++) {

String[] single\_datas = received\_data\_array[i].split(",");

if (single\_datas.length < 3) {

continue;

}

String tag\_mac = new String(single\_datas[0]);

try {

// 紧急求助缓存是否有此tag\_mac，如有则从缓存removed，同时更新紧急求助信息表

if (UsedCache.button1\_alarm\_map.containsKey(tag\_mac)) {

String peopleName = UsedCache.button1\_alarm\_map.get(tag\_mac).getPeopleName();

askhelp\_changed = true;

// 更新没有处理的求助报警信息

List<AskHelpInfo> list = BufferDataInit.commonService

.findByHql(AskHelpInfo.class, "peopleName='"

+ peopleName

+ "' and askHelpTime like '%"

+ UsedCache.button1\_alarm\_map.get(tag\_mac)

.getAskHelpTime() + "%' ", null);

if (null == list.get(0).getHandleTime()) {

BufferDataInit.commonService

.updateDate(

AskHelpInfo.class,

new String[] {

"isHandled='1'",

"handleTime='"

+ LocalTime

.getDateTime()

+ "'" }, "id='"

+ list.get(0).getId() + "'");

}

// 报警取消后从报警缓存中移除

UsedCache.button1\_alarm\_map.remove(tag\_mac);

}

} catch (Exception e) {

e.printStackTrace();

}

}

// 推送报警消息

push\_alarm\_to\_browser(askhelp\_changed, false);

}

/\*\*

\* 根据参数决定是否要向客户端推送报警信息

\*

\* @param askhelp\_changed

\* 是否有紧急求助事件变更

\* @param station\_error\_changed

\* 是否有基站事件变更

\*/

public static void push\_alarm\_to\_browser(boolean askhelp\_changed,

boolean station\_error\_changed) {

if (askhelp\_changed) {

// 向页面推送紧急求助报警信息和取消紧急求助报警信息

DwrScriptSessionManagerUtil.sendObjectByAttribute(

DwrScriptSessionManagerUtil.SCRIPT\_SESSION\_USERID,

"alarm\_receiver", UsedCache.button1\_alarm\_map,

"load\_button1\_alarm");

}

if (station\_error\_changed) {

// 向页面推送基站报警信息

DwrScriptSessionManagerUtil.sendObjectByAttribute(

DwrScriptSessionManagerUtil.SCRIPT\_SESSION\_USERID,

"alarm\_receiver", UsedCache.m\_station\_error\_alarm\_map,

"load\_station\_alarm");

}

}

}

package com.common.dao.impl;

import java.util.List;

import org.hibernate.HibernateException;

import org.hibernate.Query;

import org.hibernate.SQLQuery;

import org.hibernate.Session;

import org.springframework.orm.hibernate4.HibernateCallback;

import org.springframework.orm.hibernate4.support.HibernateDaoSupport;

import com.common.dao.CommonDao;

/\*\*

\*

\* 说明：万能无敌dao实现类，数据库操作 作者：张彬 时间：2016-1-11

\* sessionFactory.openSession()打开session，要手动关闭session

\* sessionFactory.getCurrentSession()获取当前session，自动关闭session，不能update，delete方法

\* 对数据库的增删改操作使用：sessionFactory.openSession()

\* 对数据库的查询操作使用：sessionFactory.getCurrentSession()

\* 对数据库表进行通用的操作：Object可对类对象自动转换，Object作用是实体类

\*

\* \*/

public class CommonDaoImpl extends HibernateDaoSupport implements CommonDao {

/\*\*

\* 保存数据表

\*/

@Override

public void saveTable(Object obj) {

this.getHibernateTemplate().save(obj);

this.getHibernateTemplate().flush();

}

/\*\*

\* 修改数据表

\*/

@Override

public void updateTable(Object obj) {

this.getHibernateTemplate().update(obj);

this.getHibernateTemplate().flush();

}

/\*\*

\* 更新数据表

\*/

@Override

public void renewTable(Object obj) {

this.getHibernateTemplate().saveOrUpdate(obj);

this.getHibernateTemplate().flush();

}

/\*\*

\* 删除数据表

\*/

@Override

public void deleteTable(Object obj) {

this.getHibernateTemplate().delete(obj);

this.getHibernateTemplate().flush();

}

/\*\*

\* 查找主键为Integer类型的数据

\*/

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

@Override

public Object findByIntegerId(Class className, Integer id) {

Object obj = this.getHibernateTemplate().get(className, id);

return obj;

}

/\*\*

\* 查找主键为Long类型的数据

\*/

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

@Override

public Object findByLongId(Class className, Long id) {

Object obj = this.getHibernateTemplate().get(className, id);

return obj;

}

/\*\*

\* 查找主键为String类型的数据

\*/

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

@Override

public Object findByStringId(Class className, String id) {

Object obj = this.getHibernateTemplate().get(className, id);

return obj;

}

/\*\*

\* 查询数据表中总记录数

\*/

@SuppressWarnings("rawtypes")

@Override

public Long getCountByHql(Class className, String subwhere) {

Object obj = this

.getHibernateTemplate()

.find(" select count(\*) "

+ subwhereToHql(className, subwhere, "")).iterator()

.next();

if (obj == null) {

return Long.valueOf(0);

} else {

return Long.valueOf(obj.toString());

}

}

/\*\*

\* 按hql语句查询对象

\*/

@SuppressWarnings("rawtypes")

@Override

public List findByHql(Class className, String subwhere, String orderby) {

List list = this.getHibernateTemplate().find(

subwhereToHql(className, subwhere, orderby));

return list;

}

/\*\*

\* 按hql分页查询对象集合

\*/

@SuppressWarnings("rawtypes")

@Override

public List findByHqlPage(final Class className, final String subwhere,

final String orderby, final Integer currentPage,

final Integer pageSize) {

return (List) this.getHibernateTemplate().execute(

new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

Query query = session.createQuery(subwhereToHql(

className, subwhere, orderby));

query.setFirstResult((currentPage - 1) \* pageSize);

query.setMaxResults(pageSize);

List list = query.list();

return list;

}

});

}

/\*\*

\* 查询列

\*/

@SuppressWarnings("rawtypes")

@Override

public List findByHqlGetColumn(Class className, String subwhere,

String orderby, String columns) {

List list = this.getHibernateTemplate().find(

"select " + columns

+ subwhereToHql(className, subwhere, orderby));

return list;

}

/\*\*

\* 分页查询列

\*/

@SuppressWarnings("rawtypes")

@Override

public List findByHqlGetColumnPage(final Class className,

final String subwhere, final String orderby, String columns,

final Integer currentPage, final Integer pageSize) {

return (List) this.getHibernateTemplate().execute(

new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

Query query = session.createQuery(" select "

+ subwhereToHql(className, subwhere, orderby));

query.setFirstResult((currentPage - 1) \* pageSize);

query.setMaxResults(pageSize);

List list = query.list();

return list;

}

});

}

/\*

\* 根据查询条件删除数据(non-Javadoc)

\*

\* @see com.common.dao.CommonDao#deleteTable(java.lang.Class,

\* java.lang.String)

\*/

@SuppressWarnings("rawtypes")

@Override

public void deleteTableMoreData(Class className, String subwhere) {

this.getHibernateTemplate().bulkUpdate(

" delete " + subwhereToHql(className, subwhere, ""));

this.getHibernateTemplate().flush();

}

/\*

\* 批量修改数据(non-Javadoc)

\*

\* @see com.common.dao.CommonDao#updateTableMoreData(java.lang.Class,

\* java.lang.String[], java.lang.String)

\*/

@SuppressWarnings("rawtypes")

@Override

public void updateTableMoreData(Class className, String setHql,

String subwhere) {

this.getHibernateTemplate().bulkUpdate(

updwhereToHql(className, setHql, subwhere));

this.getHibernateTemplate().flush();

}

// --------执行sql----------

/\*

\* 批量删除数据

\*/

@Override

public void deleteTableMoreData(final String tableName,

final String subwhere) {

this.getHibernateTemplate().execute(new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

Query query = session.createSQLQuery(" delete "

+ subwhereToSql(tableName, subwhere, ""));

query.executeUpdate();

session.clear();

return true;

}

});

}

/\*

\* 批量修改数据(non-Javadoc)

\*

\* @see com.common.dao.CommonDao#updateTableMoreData(java.lang.String,

\* java.lang.String, java.lang.String)

\*/

@Override

public void updateTableMoreData(final String tableName,

final String setSql, final String subwhere) {

this.getHibernateTemplate().execute(new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

Query query = session.createSQLQuery(updwhereToSql(tableName,

setSql, subwhere));

query.executeUpdate();

session.clear();

return true;

}

});

}

// 根据数据库表名和查询条件，查询数据

@SuppressWarnings("rawtypes")

@Override

public List findBySql(final String tableName, final String subwhere,

final String orderby) {

return (List) this.getHibernateTemplate().execute(

new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

List list = session.createSQLQuery(

"select \* "

+ subwhereToSql(tableName, subwhere,

orderby)).list();

return list;

}

});

}

// 根据数据库表名和查询条件，分页查询数据

@SuppressWarnings("rawtypes")

@Override

public List findBySqlPage(final String tableName, final String subwhere,

final String orderby, final Integer currentPage,

final Integer pageSize) {

return (List) this.getHibernateTemplate().execute(

new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

Query query = session.createSQLQuery(" select \* "

+ subwhereToSql(tableName, subwhere, orderby));

query.setFirstResult((currentPage - 1) \* pageSize);

query.setMaxResults(pageSize);

List list = query.list();

return list;

}

});

}

@SuppressWarnings("rawtypes")

@Override

public List findBySqlGetColumn(final String tableName,

final String columns, final String subwhere, final String orderby) {

return (List) this.getHibernateTemplate().execute(

new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

List list = session.createSQLQuery(

" select "

+ columns

+ subwhereToSql(tableName, subwhere,

orderby)).list();

return list;

}

});

}

@SuppressWarnings("rawtypes")

@Override

public List findBySqlGetColumnPage(final String tableName,

final String columns, final String subwhere, final String orderby,

final Integer currentPage, final Integer pageSize) {

return (List) this.getHibernateTemplate().execute(

new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

Query query = session.createSQLQuery(" select "

+ columns

+ subwhereToSql(tableName, subwhere, orderby));

query.setFirstResult((currentPage - 1) \* pageSize);

query.setMaxResults(pageSize);

// 将查询到的记录保存到list中

List list = query.list();

return list;

}

});

}

/\*

\* 根据数据库表名查sql语句(non-Javadoc)

\*

\* @see com.common.dao.CommonDao#getCountBySql(java.lang.String,

\* java.lang.String)

\*/

@Override

public Long getCountBySql(final String tableName, final String subwhere) {

return (Long) this.getHibernateTemplate().execute(

new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session)

throws HibernateException {

Object obj = session

.createSQLQuery(

" select count(\*) "

+ subwhereToSql(tableName,

subwhere, "")).list()

.iterator().next();

if (obj == null) {

return Long.valueOf(0);

} else {

return Long.valueOf(obj.toString());

}

}

});

}

// 对数据库进行hql语句生成

@SuppressWarnings("rawtypes")

private String subwhereToHql(Class className, String subwhere,

String orderby) {

String hql = " from " + className.getSimpleName() + " ";

// 判断是否有条件

if (subwhere != null && subwhere.trim().length() > 0) {

hql += " where " + subwhere;

}

// 判断是否有排序

if (orderby != null && orderby.trim().length() > 0) {

hql += " order by " + orderby;

}

// 返回hql语句

return hql;

}

// 对数据库进行sql语句生成

private String subwhereToSql(String tableName, String subwhere,

String orderby) {

String sql = " from " + tableName;

// 判断是否有条件

if (subwhere != null && subwhere.trim().length() > 0) {

sql += " where " + subwhere;

}

if (orderby != null && orderby.trim().length() > 0) {

sql += " order by " + orderby;

}

// 返回sql语句

return sql;

}

// Hql修改语句生成

@SuppressWarnings("rawtypes")

private String updwhereToHql(Class className, String setHql, String subwhere) {

String hql = " update " + className.getSimpleName() + " set " + setHql;

// 判断是否有条件

if (subwhere != null && subwhere.trim().length() > 0) {

hql += " where " + subwhere;

}

return hql;

}

// sql修改语句生成

private String updwhereToSql(String tableName, String setSql,

String subwhere) {

String sql = " update " + tableName + " set " + setSql;

// 判断是否有条件

if (subwhere != null && subwhere.trim().length() > 0) {

sql += " where " + subwhere;

}

return sql;

}

@Override

public Object executeSql(final String sql) {

try{

this.getHibernateTemplate().execute(new HibernateCallback<Object>() {

@Override

public Object doInHibernate(Session session) {

SQLQuery createSQLQuery = session.createSQLQuery(sql);

return createSQLQuery.executeUpdate();

}

});

}catch(Exception e) {

System.out.println(e);

}

return null;

}

}

**package** com.common.dao;

**import** java.util.List;

/\*\*

\*

\* 说明：万能无敌dao，数据库接口

\* 作者：张彬

\* 时间：2016-1-11

\*

\* \*/

@SuppressWarnings("rawtypes")

**public** **interface** CommonDao {

//根据实体类，添加数据库信息

**public** **void** saveTable(Object obj);

//根据实体类，修改数据库信息

**public** **void** updateTable(Object obj);

//根据实体类，更新数据库信息

**public** **void** renewTable(Object obj);

//根据实体类，删除数据库信息

**public** **void** deleteTable(Object obj);

//根据实体类，主键id：int 查询

**public** Object findByIntegerId(Class className,Integer id);

//根据实体类，主键id：Long 查询

**public** Object findByLongId(Class className,Long id);

//根据实体类，主键id：String 查询

**public** Object findByStringId(Class className,String id);

//根据实体类，hql查询数量

**public** Long getCountByHql(Class className,String subwhere);

//根据实体类，hql查询条件 查询

**public** List findByHql(Class className,String subwhere,String orderby);

//分页查询：根据实体类，hql查询条件，当前页，显示数查询

**public** List findByHqlPage(Class className,String subwhere,String orderby,Integer currentPage,Integer pageSize);

//根据实体类，hql查询,得到列值

**public** List findByHqlGetColumn(Class className,String subwhere,String orderby,String columns);

//根据实体类，hql分页查询,得到列值

**public** List findByHqlGetColumnPage(Class className,String subwhere,String orderby,String columns,Integer currentPage,Integer pageSize);

//根据实体类和查询条件，批量删除数据

**public** **void** deleteTableMoreData(Class className,String subwhere);

//根据实体类和查询条件，批量修改数据

**public** **void** updateTableMoreData(Class className,String setHql,String subwhere);

//根据数据库表名和查询条件，批量删除数据

**public** **void** deleteTableMoreData(String tableName,String subwhere);

//根据数据库表名和查询条件，批量修改数据

**public** **void** updateTableMoreData(String tableName,String setSql,String subwhere);

//根据数据库表名和查询条件，查询数据

**public** List findBySql(String tableName,String subwhere,String orderby);

//根据数据库表名和查询条件，分页查询数据

**public** List findBySqlPage(String tableName,String subwhere,String orderby,Integer currentPage,Integer pageSize);

//根据数据库表名和查询条件，查询列数据

**public** List findBySqlGetColumn(String tableName,String columns,String subwhere,String orderby);

//根据数据库表名和查询条件，分页查询,得到列值

**public** List findBySqlGetColumnPage(String tableName,String columns,String subwhere,String orderby,Integer currentPage,Integer pageSize);

//根据数据库表名和查询条件，sql查询数量

**public** Long getCountBySql(String tableName,String subwhere);

Object executeSql(String sql);

}

package com.init;

import java.io.IOException;

import java.io.InputStream;

import java.net.BindException;

import java.net.ServerSocket;

import java.util.Properties;

import javax.servlet.ServletContextEvent;

import javax.servlet.ServletContextListener;

import com.common.util.BufferDataInit;

import com.socket.PS\_func\_utils;

import com.socket.PositionThread;

import com.socket.ServerThread;

import com.socket.DistanceThread;

import com.socket.dingshiThread;

import com.util.global.Global;

public class StartInit implements ServletContextListener {

@Override

public void contextDestroyed(ServletContextEvent sce) {

Global.logger.info("系统停止...");

}

@Override

public void contextInitialized(ServletContextEvent sce) {

// 连接service服务

BufferDataInit.initDatabaseService(sce);

// 读取网站相关配置

load\_configs(Global.CONFIG\_FILE);

// 启动网关服务

startSensorServer();

// 启动定位服务

startPositionServer();

// 启动激光测距服务

startDistanceServer();

//启动水浸检测服务

//startWaterServer();

// 开始查询所有定位标签的位置

PS\_func\_utils.query\_ps\_abs\_pos();

// 启动定时线程

dingshi();

}

// 用于定时清空定位人员map和定时出入人防工事打卡

private void dingshi() {

new Thread(new dingshiThread()).start();

}

// 读取配置文件，可在CONFIG配置文件里获取数据并赋给对应的公共常量

public void load\_configs(String config\_file) {

InputStream in = getClass().getResourceAsStream(config\_file);

Properties properties = new Properties();

try {

properties.load(in);

// 网关端口

Global.SERVER\_PORT = Integer.parseInt(properties

.getProperty("server\_port"));

// 温湿度设备编号

Global.device\_id = properties.getProperty("temHumDevice\_id");

// 加载温湿度更新时间单位秒

Global.tempHum = Integer.parseInt(properties.getProperty("tempHum"));

// 监听定位服务器端口

Global.Position\_Port = Integer.parseInt(properties

.getProperty("Position\_Port"));

// 监听来自定位服务器报警的IP地址

Global.PS\_IP = properties.getProperty("ps\_ip");

// 监听报警信息端口号

Global.PS\_Port = Integer

.parseInt(properties.getProperty("ps\_port"));

//激光测距服务器端口

Global.Distance\_Port = Integer.parseInt(properties

.getProperty("Distance\_Port"));

//激光传感器设备编号

Global.laserdevice\_id = properties.getProperty("laserdevice\_id");

//激光测距传感器数据更新时间间隔

Global.distance\_data\_interval = Integer.parseInt(properties

.getProperty("distance\_data\_interval"));

} catch (IOException e) {

e.printStackTrace();

}

}

// 启动监听网关服务

private void startSensorServer() {

try {

// 监听网关端口

ServerSocket serverSocket = new ServerSocket(Global.SERVER\_PORT);

// 初始化ServerThread并开启监听服务

new Thread(new ServerThread(serverSocket)).start();

Global.logger.info("startSensorServer success");

} catch (BindException e) {

Global.logger.info("监听网关启动失败，端口已被占用！");

} catch (Exception e) {

Global.logger.info("监听网关启动异常！");

}

}

// 启动监听报警服务

private void startPositionServer() {

try {

// 初始化ServerSocket

ServerSocket serverSocket = new ServerSocket(Global.Position\_Port);

// 初始化ServerThread

PositionThread positionThread = new PositionThread(serverSocket);

positionThread.start(); // 开启监听服务

Global.logger.info("startPositionServer success");

} catch (IOException e) {

e.printStackTrace();

Global.logger.info("监听定位服务器启动失败，端口已被占用！");

} catch (Exception e) {

Global.logger.info("监听定位服务器启动异常！");

}

}

// 启动监听激光测距传感器服务

private void startDistanceServer() {

try {

// 监听网关端口

ServerSocket serverSocket = new ServerSocket(Global.Distance\_Port);

// 初始化ServerThread并开启监听服务

new Thread(new DistanceThread(serverSocket)).start();

Global.logger.info("startDistanceServer success");

} catch (BindException e) {

Global.logger.info("监听激光测距传感器启动失败，端口已被占用！");

} catch (Exception e) {

Global.logger.info("监听激光测距传感器启动异常！");

}

}

// 启动监听激光测距传感器服务

/\*private void startWaterServer() {

try {

// 监听网关端口

ServerSocket serverSocket = new ServerSocket(Global.Water\_Port);

// 初始化ServerThread并开启监听服务

new Thread(new WaterThread(serverSocket)).start();

Global.logger.info("startWaterServer success");

} catch (BindException e) {

Global.logger.info("监听水浸传感器启动失败，端口已被占用！");

} catch (Exception e) {

Global.logger.info("监听水浸传感器启动异常！");

}

}\*/

}

package com.interceptor;

import java.util.Date;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import org.springframework.stereotype.Component;

import org.springframework.web.servlet.ModelAndView;

import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

import com.util.global.UsedCache;

/\*\*

\* spring 拦截器

\* @author ZHANGBIN

\*

\*/

@Component

public class FullPathInterceptor extends HandlerInterceptorAdapter {

/\*\*

\* 在业务处理器处理请求之前被调用

\* 如果返回false

\* 从当前的拦截器往回执行所有拦截器的afterCompletion(),再退出拦截器链

\* 如果返回true

\* 执行下一个拦截器,直到所有的拦截器都执行完毕

\* 再执行被拦截的Controller

\* 然后进入拦截器链,

\* 从最后一个拦截器往回执行所有的postHandle()

\* 接着再从最后一个拦截器往回执行所有的afterCompletion()

\*/

@Override

public boolean preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) throws Exception {

return true;

}

/\*\*

\* 在业务处理器处理请求执行完成后,生成视图之前执行的动作

\* 可在modelAndView中加入数据，比如当前时间

\*/

@Override

public void postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

//根目录

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

request.setAttribute("basePath", basePath);

//完整地址

/\* String methodPath = request.getServletPath();

String fullPath = basePath+methodPath.substring(1);

request.setAttribute("fullPath", fullPath);

System.out.println(fullPath);\*/

//获取页面活动时间

HttpSession session = request.getSession();

if(UsedCache.loginInfoMap.containsKey(session.getId())

&& UsedCache.loginInfoMap.get(session.getId()) != null

&& UsedCache.loginInfoMap.get(session.getId()).getLoginTime() != null){

UsedCache.loginInfoMap.get(session.getId()).setLoginTime(new Date());

}

}

/\*\*

\* 在DispatcherServlet完全处理完请求后被调用,可用于清理资源等

\*

\* 当有拦截器抛出异常时,会从当前拦截器往回执行所有的拦截器的afterCompletion()

\*/

@Override

public void afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

throws Exception {

}

}

package com.interceptor;

import java.util.Date;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import org.springframework.stereotype.Component;

import org.springframework.web.servlet.ModelAndView;

import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

import com.form.LoginUser;

import com.util.global.Global;

import com.util.global.UsedCache;

/\*\*

\* spring 拦截器

\*

\* @author ZHANGBIN

\*

\*/

@Component

public class SessionInterceptor extends HandlerInterceptorAdapter {

/\*\*

\* 在业务处理器处理请求之前被调用 如果返回false 从当前的拦截器往回执行所有拦截器的afterCompletion(),再退出拦截器链

\* 如果返回true 执行下一个拦截器,直到所有的拦截器都执行完毕 再执行被拦截的Controller 然后进入拦截器链,

\* 从最后一个拦截器往回执行所有的postHandle() 接着再从最后一个拦截器往回执行所有的afterCompletion()

\*/

@Override

public boolean preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) throws Exception {

HttpSession session = request.getSession();

if (session.getId() == null

|| !UsedCache.loginInfoMap.containsKey(session.getId())) {

request.getRequestDispatcher("/WEB-INF/login.jsp").forward(request,

response);

return false;

}

LoginUser login = UsedCache.loginInfoMap.get(session.getId());

if (login == null) {

request.getRequestDispatcher("/WEB-INF/login.jsp").forward(request,

response);

return false;

}

if (login.getLoginTime() != null

&& new Date().getTime() - login.getLoginTime().getTime() > Global.session\_time\_second \* 60 \* 1000) {

request.getRequestDispatcher("/WEB-INF/login.jsp").forward(request,

response);

return false;

}

return true;

}

/\*\*

\* 在业务处理器处理请求执行完成后,生成视图之前执行的动作 可在modelAndView中加入数据，比如当前时间

\*/

@Override

public void postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

}

/\*\*

\* 在DispatcherServlet完全处理完请求后被调用,可用于清理资源等

\*

\* 当有拦截器抛出异常时,会从当前拦截器往回执行所有的拦截器的afterCompletion()

\*/

@Override

public void afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

throws Exception {

}

}

/\*\*

\* 定位标签/基站等原始坐标和转换后坐标的bean

\* @author ZHUZHIYONG

\*/

package com.form;

import com.beans.UserInfo;

public class PositionBean {

private double x\_pos; // x坐标

private double y\_pos; // y坐标

private double z\_pos; // z坐标

private int bid; // 所在建筑物编号

private double x\_pos\_trans; // x坐标 转换后

private double y\_pos\_trans; // y坐标 转换后

private double z\_pos\_trans; // z坐标 转换后

private String mac; // 定位卡MAC

private String nearestAp; // 定位基站MAC

private UserInfo userInfo; // 定位用户对象

private String posTime; // 定位时间

private String alarm; // 报警状态

public PositionBean() {

this.x\_pos = this.y\_pos = this.z\_pos = this.x\_pos\_trans = this.y\_pos\_trans = this.z\_pos\_trans = 0;

this.bid = 0;

}

public double getX\_pos() {

return x\_pos;

}

public void setX\_pos(double x\_pos) {

this.x\_pos = x\_pos;

}

public double getY\_pos() {

return y\_pos;

}

public void setY\_pos(double y\_pos) {

this.y\_pos = y\_pos;

}

public double getZ\_pos() {

return z\_pos;

}

public void setZ\_pos(double z\_pos) {

this.z\_pos = z\_pos;

}

public int getBid() {

return bid;

}

public void setBid(int bid) {

this.bid = bid;

}

public double getX\_pos\_trans() {

return x\_pos\_trans;

}

public void setX\_pos\_trans(double x\_pos\_trans) {

this.x\_pos\_trans = x\_pos\_trans;

}

public double getY\_pos\_trans() {

return y\_pos\_trans;

}

public void setY\_pos\_trans(double y\_pos\_trans) {

this.y\_pos\_trans = y\_pos\_trans;

}

public double getZ\_pos\_trans() {

return z\_pos\_trans;

}

public void setZ\_pos\_trans(double z\_pos\_trans) {

this.z\_pos\_trans = z\_pos\_trans;

}

public String getMac() {

return mac;

}

public void setMac(String mac) {

this.mac = mac;

}

public String getNearestAp() {

return nearestAp;

}

public void setNearestAp(String nearestAp) {

this.nearestAp = nearestAp;

}

public UserInfo getUserInfo() {

return userInfo;

}

public void setUserInfo(UserInfo userInfo) {

this.userInfo = userInfo;

}

public String getPosTime() {

return posTime;

}

public void setPosTime(String posTime) {

this.posTime = posTime;

}

public String getAlarm() {

return alarm;

}

public void setAlarm(String alarm) {

this.alarm = alarm;

}

}

/\*\*

\* 定位标签/基站等原始坐标和转换后坐标的bean

\* @author ZHUZHIYONG

\*/

package com.form;

import com.beans.UserInfo;

public class PositionBean {

private double x\_pos; // x坐标

private double y\_pos; // y坐标

private double z\_pos; // z坐标

private int bid; // 所在建筑物编号

private double x\_pos\_trans; // x坐标 转换后

private double y\_pos\_trans; // y坐标 转换后

private double z\_pos\_trans; // z坐标 转换后

private String mac; // 定位卡MAC

private String nearestAp; // 定位基站MAC

private UserInfo userInfo; // 定位用户对象

private String posTime; // 定位时间

private String alarm; // 报警状态

public PositionBean() {

this.x\_pos = this.y\_pos = this.z\_pos = this.x\_pos\_trans = this.y\_pos\_trans = this.z\_pos\_trans = 0;

this.bid = 0;

}

public double getX\_pos() {

return x\_pos;

}

public void setX\_pos(double x\_pos) {

this.x\_pos = x\_pos;

}

public double getY\_pos() {

return y\_pos;

}

public void setY\_pos(double y\_pos) {

this.y\_pos = y\_pos;

}

public double getZ\_pos() {

return z\_pos;

}

public void setZ\_pos(double z\_pos) {

this.z\_pos = z\_pos;

}

public int getBid() {

return bid;

}

public void setBid(int bid) {

this.bid = bid;

}

public double getX\_pos\_trans() {

return x\_pos\_trans;

}

public void setX\_pos\_trans(double x\_pos\_trans) {

this.x\_pos\_trans = x\_pos\_trans;

}

public double getY\_pos\_trans() {

return y\_pos\_trans;

}

public void setY\_pos\_trans(double y\_pos\_trans) {

this.y\_pos\_trans = y\_pos\_trans;

}

public double getZ\_pos\_trans() {

return z\_pos\_trans;

}

public void setZ\_pos\_trans(double z\_pos\_trans) {

this.z\_pos\_trans = z\_pos\_trans;

}

public String getMac() {

return mac;

}

public void setMac(String mac) {

this.mac = mac;

}

public String getNearestAp() {

return nearestAp;

}

public void setNearestAp(String nearestAp) {

this.nearestAp = nearestAp;

}

public UserInfo getUserInfo() {

return userInfo;

}

public void setUserInfo(UserInfo userInfo) {

this.userInfo = userInfo;

}

public String getPosTime() {

return posTime;

}

public void setPosTime(String posTime) {

this.posTime = posTime;

}

public String getAlarm() {

return alarm;

}

public void setAlarm(String alarm) {

this.alarm = alarm;

}

}