package com.cet.pq.harmonic.trend.calculate;

import com.cet.pq.harmonic.trend.model.dto.CalculateResultDTO;

import com.fasterxml.jackson.core.type.TypeReference;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.isoftstone.bussiness.harmonicTracing.model.Jddaona;

import com.isoftstone.bussiness.harmonicTracing.model.Xddaona;

import com.isoftstone.bussiness.zhejiang.mulNewton;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.io.IOException;

import java.math.BigDecimal;

import java.text.DecimalFormat;

import java.util.\*;

public class Calculate {

private static final Logger LOGGER = LoggerFactory.getLogger(Calculate.class);

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

String dataStr = "{\n" +

" \"xddns\": [\n" +

" {\n" +

" \"nodeIdStart\": 1,\n" +

" \"nodeIdEnd\": 2,\n" +

" \"fReal\": 0.0,\n" +

" \"fImag\": 0.0,\n" +

" \"sReal\": 0.0,\n" +

" \"sImag\": 0.0,\n" +

" \"s\": 0.0,\n" +

" \"logTime\": null,\n" +

" \"harmonicNum\": 0,\n" +

" \"x\": 1.3,\n" +

" \"b\": 1.4,\n" +

" \"hc\": 0.0,\n" +

" \"r\": 1.2\n" +

" },\n" +

" {\n" +

" \"nodeIdStart\": 1,\n" +

" \"nodeIdEnd\": 3,\n" +

" \"fReal\": 0.0,\n" +

" \"fImag\": 0.0,\n" +

" \"sReal\": 0.0,\n" +

" \"sImag\": 0.0,\n" +

" \"s\": 0.0,\n" +

" \"logTime\": null,\n" +

" \"harmonicNum\": 0,\n" +

" \"x\": 1.3,\n" +

" \"b\": 1.4,\n" +

" \"hc\": 0.0,\n" +

" \"r\": 1.2\n" +

" }\n" +

" ],\n" +

" \"jddns\": [\n" +

" {\n" +

" \"nodeId\": 1,\n" +

" \"nodeName\": \"太真负荷\",\n" +

" \"vm\": 0.0,\n" +

" \"vi\": 0.0,\n" +

" \"vb\": 0.0,\n" +

" \"p\": 0.0,\n" +

" \"pb\": 0.0,\n" +

" \"q\": 0.0,\n" +

" \"qb\": 0.0,\n" +

" \"g\": 0.0,\n" +

" \"b\": 0.0,\n" +

" \"iReal\": 0.0,\n" +

" \"iIMag\": 0.0,\n" +

" \"iRealB\": 0.0,\n" +

" \"iIMagB\": 0.0,\n" +

" \"sfcl\": \"1\",\n" +

" \"logTime\": null,\n" +

" \"vReal\": 0.0,\n" +

" \"vIMag\": 0.0,\n" +

" \"vRms\": 0.0,\n" +

" \"harmonicNum\": 0,\n" +

" \"vHyl\": 0.0,\n" +

" \"type\": \"1\",\n" +

" \"iRealH\": 0.0,\n" +

" \"iIMagH\": 0.0,\n" +

" \"iRmsH\": 0.0,\n" +

" \"pH\": 0.0,\n" +

" \"qH\": 0.0,\n" +

" \"sRealH\": 0.0,\n" +

" \"sIMagH\": 0.0,\n" +

" \"sH\": 0.0,\n" +

" \"vRealZl\": 0.0,\n" +

" \"vIMagZl\": 0.0,\n" +

" \"vRmsZl\": 0.0\n" +

" },\n" +

" {\n" +

" \"nodeId\": 2,\n" +

" \"nodeName\": \"古田负荷\",\n" +

" \"vm\": 0.0,\n" +

" \"vi\": 0.0,\n" +

" \"vb\": 0.0,\n" +

" \"p\": 0.0,\n" +

" \"pb\": 0.0,\n" +

" \"q\": 0.0,\n" +

" \"qb\": 0.0,\n" +

" \"g\": 0.0,\n" +

" \"b\": 0.0,\n" +

" \"iReal\": 0.0,\n" +

" \"iIMag\": 0.0,\n" +

" \"iRealB\": 0.0,\n" +

" \"iIMagB\": 0.0,\n" +

" \"sfcl\": \"1\",\n" +

" \"logTime\": null,\n" +

" \"vReal\": 0.0,\n" +

" \"vIMag\": 0.0,\n" +

" \"vRms\": 0.0,\n" +

" \"harmonicNum\": 0,\n" +

" \"vHyl\": 0.0,\n" +

" \"type\": \"1\",\n" +

" \"iRealH\": 0.0,\n" +

" \"iIMagH\": 0.0,\n" +

" \"iRmsH\": 0.0,\n" +

" \"pH\": 0.0,\n" +

" \"qH\": 0.0,\n" +

" \"sRealH\": 0.0,\n" +

" \"sIMagH\": 0.0,\n" +

" \"sH\": 0.0,\n" +

" \"vRealZl\": 0.0,\n" +

" \"vIMagZl\": 0.0,\n" +

" \"vRmsZl\": 0.0\n" +

" },\n" +

" {\n" +

" \"nodeId\": 3,\n" +

" \"nodeName\": \"仙霞负荷\",\n" +

" \"vm\": 7225.0,\n" +

" \"vi\": 85.0,\n" +

" \"vb\": 18.960707704068394,\n" +

" \"p\": 170.0,\n" +

" \"pb\": 1.7,\n" +

" \"q\": 170.0,\n" +

" \"qb\": 1.7,\n" +

" \"g\": 0.0,\n" +

" \"b\": 0.0,\n" +

" \"iReal\": -502.03208813096137,\n" +

" \"iIMag\": -89.7985661737794,\n" +

" \"iRealB\": -3313.4117816643443,\n" +

" \"iIMagB\": -592.670536746944,\n" +

" \"sfcl\": \"1\",\n" +

" \"logTime\": null,\n" +

" \"vReal\": 0.0,\n" +

" \"vIMag\": 0.0,\n" +

" \"vRms\": 0.0,\n" +

" \"harmonicNum\": 0,\n" +

" \"vHyl\": 0.0,\n" +

" \"type\": \"1\",\n" +

" \"iRealH\": 0.0,\n" +

" \"iIMagH\": 0.0,\n" +

" \"iRmsH\": 0.0,\n" +

" \"pH\": 0.0,\n" +

" \"qH\": 0.0,\n" +

" \"sRealH\": 0.0,\n" +

" \"sIMagH\": 0.0,\n" +

" \"sH\": 0.0,\n" +

" \"vRealZl\": 0.0,\n" +

" \"vIMagZl\": 0.0,\n" +

" \"vRmsZl\": 0.0\n" +

" }\n" +

" ]\n" +

" }";

System.out.println(dataStr);

ObjectMapper objectMapper = new ObjectMapper();

Map map = objectMapper.readValue(dataStr, Map.class);

Object jddns = map.get("jddns");

Object xddns = map.get("xddns");

List<Jddaona> jddaonas = objectMapper.readValue(objectMapper.writeValueAsString(jddns), new TypeReference<List<Jddaona>>() {

});

List<Xddaona> xddaonas = objectMapper.readValue(objectMapper.writeValueAsString(xddns), new TypeReference<List<Xddaona>>() {

});

Calculate calculate = new Calculate();

calculate.calculate(jddaonas, xddaonas, "2023-01-11 11:00:00", 5);

}

public CalculateResultDTO calculate(List<Jddaona> jddns, List<Xddaona> xddns, String logTimeStr, Integer harmonicNum) {

int n = jddns.size();

int k = 0; // 不被检测的负荷数

double min;

Double[][] G = new Double[n][n];

Double[][] B = new Double[n][n];

List<Xddaona> dnjz = new ArrayList();

DaonJuzhengGrow daonJuzheng = new DaonJuzhengGrow();

daonJuzheng.jsdaonajuzheng(jddns, xddns, G, B, dnjz, n, k, harmonicNum);

Double[] p = new Double[n - k];

Double[] q = new Double[n - k];

Double[] vm = new Double[n - k];

Double[] vi = new Double[n - k];

for (int j = 0; j < n - k; ++j) {

p[j] = ((Jddaona) jddns.get(j)).getPb();

q[j] = ((Jddaona) jddns.get(j)).getQb();

vm[j] = ((Jddaona) jddns.get(j)).getVb();

vi[j] = ((Jddaona) jddns.get(j)).getVi();

}

Double[] x = new Double[n + k - 1];

Double[] ub = new Double[n + k - 1];

Double[] lb = new Double[n + k - 1];

min = -0.3D;

double max = 0.3D;

BigDecimal xx;

for (int i = 0; i < k \* 2; ++i) {

ub[i] = 0.3D;

lb[i] = -0.3D;

xx = (new BigDecimal(min + Math.random() \* (max - min))).setScale(5, 4);

x[i] = xx.doubleValue();

}

min = 0.0D;

max = 6.29D;

for (int i = 2 \* k; i < n + k - 1; ++i) {

ub[i] = 6.29D;

lb[i] = 0.0D;

xx = (new BigDecimal(min + Math.random() \* (max - min))).setScale(5, 4);

x[i] = xx.doubleValue();

}

mulNewton newton = new mulNewton();

// MulNewton newton = new MulNewton();

Double[] bestX = newton.newton(x, n, k, p, q, G, B, vm, vi, lb, ub);

Double[] cz = new Double[n];

Double[] fz = new Double[n];

for (int i = 0; i < n - k; ++i) {

cz[i] = vm[i] \* Math.cos(vi[i] + bestX[i + 2 \* k]);

fz[i] = vm[i] \* Math.sin(vi[i] + bestX[i + 2 \* k]);

}

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

cz[n - k + i] = bestX[i];

fz[n - k + i] = bestX[i + k];

}

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

double i\_real = 0.0D;

double i\_imag = 0.0D;

for (int j = 0; j < n; ++j) {

i\_real = i\_real + cz[i] \* G[i][j] - fz[i] \* B[i][j];

i\_imag = i\_imag + cz[i] \* B[i][j] - fz[i] \* G[i][j];

}

double i\_rms = Math.sqrt(Math.pow(i\_real, 2.0D) + Math.pow(i\_imag, 2.0D));

double v\_rms = Math.sqrt(Math.pow(cz[i], 2.0D) + Math.pow(fz[i], 2.0D));

double s\_real = cz[i] \* i\_real + fz[i] \* i\_imag;

double s\_imag = -cz[i] \* i\_imag + fz[i] \* i\_real;

double s\_rms = Math.sqrt(Math.pow(s\_real, 2.0D) + Math.pow(s\_imag, 2.0D));

((Jddaona) jddns.get(i)).setLogTime(logTimeStr);

((Jddaona) jddns.get(i)).setHarmonicNum(harmonicNum);

((Jddaona) jddns.get(i)).setvIMag(fz[i]);

((Jddaona) jddns.get(i)).setvReal(cz[i]);

((Jddaona) jddns.get(i)).setvRms(v\_rms);

((Jddaona) jddns.get(i)).setiIMagH(i\_imag);

((Jddaona) jddns.get(i)).setiRealH(i\_real);

((Jddaona) jddns.get(i)).setiRmsH(i\_rms);

((Jddaona) jddns.get(i)).setpH(s\_real);

((Jddaona) jddns.get(i)).setqH(s\_imag);

((Jddaona) jddns.get(i)).setsH(s\_rms);

this.getContent((Jddaona) jddns.get(i));

// this.derivedFunctionDao.saveJddn((Jddaona)jddns.get(i));

}

daonJuzheng.clzjs(jddns, dnjz);

daonJuzheng.gxdjs(jddns, dnjz);

Iterator var77 = dnjz.iterator();

while (var77.hasNext()) {

Xddaona param = (Xddaona) var77.next();

param.setLogTime(logTimeStr);

param.setHarmonicNum(harmonicNum);

// this.derivedFunctionDao.saveXdda(param);

}

CalculateResultDTO calculateResultDTO = new CalculateResultDTO();

calculateResultDTO.setDnjz(dnjz);

calculateResultDTO.setJddns(jddns);

return calculateResultDTO;

}

private void getContent(Jddaona jddaona) {

double value = 0.0D;

if (null != STATION\_LIST.get(String.valueOf(jddaona.getNodeId()))) {

if (jddaona.getNodeId() != 13002L && jddaona.getNodeId() != 13001L) {

value = Chufa(jddaona.getvRms(), 127.17D);

} else {

value = Chufa(jddaona.getvRms(), 317.92D);

}

} else if (jddaona.getHarmonicNum() == 5) {

value = 0.1225D;

} else if (jddaona.getHarmonicNum() == 7) {

value = 0.0685D;

} else if (jddaona.getHarmonicNum() == 11) {

value = 0.0265D;

} else {

value = 0.0D;

}

jddaona.setvHyl(value);

}

private static double Chufa(double a, double b) {

DecimalFormat dF = new DecimalFormat("0.00");

return Double.valueOf(dF.format((double) ((float) a) / b));

}

}

package com.cet.pq.harmonic.trend.calculate;

import com.isoftstone.bussiness.harmonicTracing.model.Jddaona;

import com.isoftstone.bussiness.harmonicTracing.model.Xddaona;

import com.isoftstone.bussiness.zhejiang.DaonJuzheng;

import java.util.List;

/\*\*

\* 原算法代码增强，提高容错能力

\*/

public class DaonJuzhengGrow extends DaonJuzheng {

public void jsdaonajuzheng(List<Jddaona> jddns, List<Xddaona> xddns, Double[][] G, Double[][] B, List<Xddaona> dnjz, int n, int k, int harmonicNum) {

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

long nodeid = ((Jddaona)jddns.get(i)).getNodeId();

double R = 0.0;

double X = 0.0;

if (((Jddaona)jddns.get(i)).getPb() == 0.0 && ((Jddaona)jddns.get(i)).getQb() == 0.0) {

((Jddaona)jddns.get(i)).setG(0.0);

((Jddaona)jddns.get(i)).setB(0.0);

} else {

R = Math.pow(((Jddaona)jddns.get(i)).getVb(), 2.0) \* ((Jddaona)jddns.get(i)).getPb() / (Math.pow(((Jddaona)jddns.get(i)).getPb(), 2.0) + Math.pow(((Jddaona)jddns.get(i)).getQb(), 2.0));

X = (double)harmonicNum \* Math.pow(((Jddaona)jddns.get(i)).getVb(), 2.0) \* ((Jddaona)jddns.get(i)).getQb() / (Math.pow(((Jddaona)jddns.get(i)).getPb(), 2.0) + Math.pow(((Jddaona)jddns.get(i)).getQb(), 2.0));

R = R / (Math.pow(R, 2.0) + Math.pow(X, 2.0));

X = X / (Math.pow(R, 2.0) + Math.pow(X, 2.0));

((Jddaona)jddns.get(i)).setG(Double.isNaN(R) ? 0D : R);

((Jddaona)jddns.get(i)).setB(Double.isNaN(X) ? 0D : X);

}

double y\_real = ((Jddaona)jddns.get(i)).getG();

double y\_flam = ((Jddaona)jddns.get(i)).getB();

for(int j = 0; j < n; ++j) {

if (j != i) {

long nodeid\_j = ((Jddaona)jddns.get(j)).getNodeId();

Xddaona db = new Xddaona();

db.setNodeIdStart(nodeid);

db.setNodeIdEnd(nodeid\_j);

double g = 0.0;

double b = 0.0;

double b2 = 0.0;

for(int m = 0; m < xddns.size(); ++m) {

if (nodeid\_j == ((Xddaona)xddns.get(m)).getNodeIdStart() && nodeid == ((Xddaona)xddns.get(m)).getNodeIdEnd() || nodeid\_j == ((Xddaona)xddns.get(m)).getNodeIdEnd() && nodeid == ((Xddaona)xddns.get(m)).getNodeIdStart()) {

((Xddaona)xddns.get(m)).setX(((Xddaona)xddns.get(m)).getX() \* (double)harmonicNum);

g = ((Xddaona)xddns.get(m)).getR() / (Math.pow(((Xddaona)xddns.get(m)).getR(), 2.0) + Math.pow(((Xddaona)xddns.get(m)).getX(), 2.0));

b = -((Xddaona)xddns.get(m)).getX() / (Math.pow(((Xddaona)xddns.get(m)).getR(), 2.0) + Math.pow(((Xddaona)xddns.get(m)).getX(), 2.0));

b2 = ((Xddaona)xddns.get(m)).getB() \* (double)harmonicNum;

break;

}

}

G[i][j] = -g;

B[i][j] = -b;

db.setfReal(-g);

db.setfImag(-b);

dnjz.add(db);

y\_real = y\_real + g + b2;

y\_flam += b;

}

}

Xddaona zj = new Xddaona();

zj.setNodeIdStart(nodeid);

zj.setNodeIdEnd(nodeid);

G[i][i] = y\_real;

B[i][i] = y\_flam;

zj.setfReal(y\_real);

zj.setfImag(y\_real);

dnjz.add(zj);

}

}

}

package com.cet.pq.harmonic.trend.calculate;

import com.cet.pq.harmonic.trend.constant.HarmonicMapConstant;

import com.cet.pq.harmonic.trend.model.LineResult;

import com.cet.pq.harmonic.trend.model.dto.SubstationDTO;

import com.cet.pq.harmonic.trend.model.dto.TrendResultDTO;

import com.cet.pq.harmonic.trend.service.LineResultService;

import org.apache.commons.collections4.CollectionUtils;

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

import java.util.\*;

import java.util.stream.Collectors;

/\*\*

\* 预治理

\*/

public abstract class Imitate {

@Autowired

protected LineResultService lineResultService;

/\*\*

\* 预治理算法

\* @param substationCodes 增加装置的负荷

\* @param trendResultDTOS 拓扑图结构

\* @param substationDTOMap 负荷map，id为key

\* @param overLineResults 超标监测点

\* @param overLineResultsAfter 治理后超标监测点

\*/

public void zhiLi(List<String> substationCodes, List<TrendResultDTO> trendResultDTOS, Map<String, SubstationDTO> substationDTOMap

, List<LineResult> overLineResults, List<LineResult> overLineResultsAfter) {

String startSubstationCode = null;

String endSubstationCode = null;

SubstationDTO substationDTO = null;

Map<Long, List<LineResult>> lineResultListMap = null;

List<LineResult> zhiliLineResultList = null;

Set<Long> changeNodeIds = new HashSet<>();

if (CollectionUtils.isNotEmpty(overLineResults)) {

lineResultListMap = overLineResults.stream().collect(Collectors.groupingBy(LineResult::getNodeId));

}

//关联负荷治理

for (TrendResultDTO item : trendResultDTOS) {

startSubstationCode = item.getStartSubstationCode();

endSubstationCode = item.getEndSubstationCode();

if (substationCodes.contains(startSubstationCode)) {

substationDTO = substationDTOMap.get(endSubstationCode);

} else if (substationCodes.contains(endSubstationCode)) {

substationDTO = substationDTOMap.get(startSubstationCode);

}

if (substationDTO == null) {

continue;

}

relationZhili(substationDTO);

Long substationDTOId = substationDTO.getId();

changeNodeIds.add(substationDTOId);

if (lineResultListMap == null || lineResultListMap.isEmpty()) {

continue;

}

zhiliLineResultList = lineResultListMap.get(substationDTOId);

if (CollectionUtils.isNotEmpty(zhiliLineResultList)) {

relationZhiliLine(zhiliLineResultList);

}

}

// 加装置负荷治理

for (String code : substationCodes) {

substationDTO = substationDTOMap.get(code);

if (substationDTO == null) {

continue;

}

zhiLi(substationDTO);

Long substationDTOId = substationDTO.getId();

changeNodeIds.add(substationDTOId);

if (lineResultListMap == null || lineResultListMap.isEmpty()) {

continue;

}

zhiliLineResultList = lineResultListMap.get(substationDTOId);

if (CollectionUtils.isNotEmpty(zhiliLineResultList)) {

zhiliLine(zhiliLineResultList);

changeNodeIds.addAll(zhiliLineResultList.stream().map(LineResult::getLineId).collect(Collectors.toList()));

}

}

if (CollectionUtils.isNotEmpty(overLineResults) && CollectionUtils.isNotEmpty(changeNodeIds)) {

overLineResults.forEach(item -> {

if (changeNodeIds.contains(item.getNodeId())) {

overLineResultsAfter.add(item);

}

});

}

}

/\*\*

\* 治理前说明

\*

\* @param lineResults 超标的监测点

\* @param harmonicNum 谐波次数

\* @return

\*/

public String instructionsBefor(List<LineResult> lineResults, int harmonicNum) {

if (CollectionUtils.isEmpty(lineResults)) {

return null;

}

List<String> lineInstructionsList = lineResults.stream().map(lineResult -> {

StringBuffer buffer = new StringBuffer(lineResult.getName())

.append(HarmonicMapConstant.HARMINOC\_NAME\_MAP.get(harmonicNum))

.append(getTypeName())

.append("为")

.append(getValue(lineResult))

.append(getUnit());

return buffer.toString();

}).collect(Collectors.toList());

String result = String.join("、", lineInstructionsList);

return result + "，不满足国标要求。";

}

/\*\*

\* 治理后说明

\*

\* @param overLineResults 超标且已治理的监测点

\* @param harmonicNum

\* @return

\*/

public String instructionsAfter(List<LineResult> overLineResults, Integer harmonicNum) {

if (CollectionUtils.isEmpty(overLineResults)) {

return null;

}

List<String> lineInstructionsList = overLineResults.stream().map(lineResult -> {

StringBuffer buffer = new StringBuffer(lineResult.getName())

.append(HarmonicMapConstant.HARMINOC\_NAME\_MAP.get(harmonicNum))

.append(getTypeName())

.append("为")

.append(getValue(lineResult))

.append(getUnit())

.append("（↓）");

return buffer.toString();

}).collect(Collectors.toList());

String result = String.join("、", lineInstructionsList);

return "经投入SVG后，" + result + "，满足国标要求。";

}

/\*\*

\* 获取单位

\*

\* @return

\*/

protected abstract String getUnit();

/\*\*

\* 获取超标的值

\*

\* @return

\*/

protected abstract Double getValue(LineResult lineResult);

/\*\*

\* 获取类型名称

\*

\* @return

\*/

protected abstract String getTypeName();

/\*\*

\* 获取超标监测点

\*

\* @param overSubstationDTOS

\* @param logTime

\* @return

\*/

public List<LineResult> overLineList(List<SubstationDTO> overSubstationDTOS, Date logTime, Integer harmonicNum, Long mapId) {

if (CollectionUtils.isEmpty(overSubstationDTOS)) {

return null;

}

List<Long> substationIds = overSubstationDTOS.stream().map(SubstationDTO::getId).collect(Collectors.toList());

return overLineListBySubstationId(substationIds, logTime, harmonicNum, mapId);

}

/\*\*

\* 获取超标监测点

\*

\* @param substationId

\* @param logTime

\* @return

\*/

protected abstract List<LineResult> overLineListBySubstationId(List<Long> substationId, Date logTime, Integer harmonicNum, Long mapId);

/\*\*

\* 获取超标负荷

\*

\* @param substationDTOS

\* @return

\*/

public abstract List<SubstationDTO> getOverData(List<SubstationDTO> substationDTOS);

/\*\*

\* 加装置负荷治理

\*

\* @param substationDTO

\*/

protected abstract void zhiLi(SubstationDTO substationDTO);

/\*\*

\* 加装置负荷下监测点治理

\*

\* @param overLineResults

\*/

protected abstract void zhiliLine(List<LineResult> overLineResults);

/\*\*

\* 治理关联监测点

\* @param lineResults

\*/

protected abstract void relationZhiliLine(List<LineResult> lineResults);

/\*\*

\* 关联负荷治理

\*

\* @param substationDTO

\*/

abstract void relationZhili(SubstationDTO substationDTO);

}

package com.cet.pq.harmonic.trend.calculate;

import cn.hutool.core.util.NumberUtil;

import com.cet.pq.harmonic.trend.constant.HarmonicMapConstant;

import com.cet.pq.harmonic.trend.constant.IndexType;

import com.cet.pq.harmonic.trend.model.LineResult;

import com.cet.pq.harmonic.trend.model.dto.SubstationDTO;

import org.apache.commons.collections4.CollectionUtils;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

/\*\*

\* 电流治理

\*/

@Component("imitateCurrent")

@ConfigurationProperties(prefix = "zhili.current")

public class ImitateCurrent extends Imitate{

private final static String UNIT = "A";

/\*\*

\* 关联负荷的系数

\*/

private double relationCoefficient;

/\*\*

\* 加装置负荷的系数

\*/

private double itselfCoefficient;

public double getRelationCoefficient() {

return relationCoefficient;

}

public void setRelationCoefficient(double relationCoefficient) {

this.relationCoefficient = relationCoefficient;

}

public double getItselfCoefficient() {

return itselfCoefficient;

}

public void setItselfCoefficient(double itselfCoefficient) {

this.itselfCoefficient = itselfCoefficient;

}

@Override

protected void relationZhiliLine(List<LineResult> lineResults) {

if (CollectionUtils.isEmpty(lineResults)) {

return;

}

lineResults.forEach(lineResult -> {

lineResult.setCurrent(lineResult.getCurrent() \* relationCoefficient);

});

}

@Override

void relationZhili(SubstationDTO substationDTO) {

substationDTO.setElectric(substationDTO.getElectric() \* relationCoefficient);

}

@Override

protected void zhiliLine(List<LineResult> lineResults) {

if (CollectionUtils.isEmpty(lineResults)) {

return;

}

lineResults.forEach(lineResult -> {

lineResult.setCurrent(lineResult.getCurrent() \* itselfCoefficient);

});

}

/\*\*

\* 预治理算法

\*/

public void zhiLi(SubstationDTO substationDTO) {

substationDTO.setElectric(substationDTO.getElectric() \* itselfCoefficient);

}

@Override

protected String getUnit() {

return UNIT;

}

@Override

protected Double getValue(LineResult lineResult) {

Double value = lineResult.getCurrent();

return value == null ? 0D : NumberUtil.round(value, 2).doubleValue() ;

}

@Override

protected String getTypeName() {

return IndexType.CURRENT.getLable();

}

@Override

protected List<LineResult> overLineListBySubstationId(List<Long> substationId, Date logTime, Integer harmonicNum, Long mapId) {

return lineResultService.currentOverList(substationId, logTime, harmonicNum, mapId);

}

@Override

public List<SubstationDTO> getOverData(List<SubstationDTO> substationDTOS) {

List<SubstationDTO> result = new ArrayList<>();

substationDTOS.forEach(substationDTO -> {

if (substationDTO.getStateI().equals(HarmonicMapConstant.STATE\_OVER)) {

result.add(substationDTO);

}

});

return result;

}

}

package com.cet.pq.harmonic.trend.calculate;

import cn.hutool.core.util.NumberUtil;

import com.cet.pq.harmonic.trend.constant.HarmonicMapConstant;

import com.cet.pq.harmonic.trend.constant.IndexType;

import com.cet.pq.harmonic.trend.model.LineResult;

import com.cet.pq.harmonic.trend.model.dto.SubstationDTO;

import org.apache.commons.collections4.CollectionUtils;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

/\*\*

\* 电压治理

\*/

@Component("imitateVoltage")

@ConfigurationProperties(prefix = "zhili.voltage")

public class ImitateVoltage extends Imitate {

private final static String UNIT = "%";

/\*\*

\* 关联负荷的系数

\*/

private double relationCoefficient;

/\*\*

\* 加装置负荷的系数

\*/

private double itselfCoefficient;

public double getRelationCoefficient() {

return relationCoefficient;

}

public void setRelationCoefficient(double relationCoefficient) {

this.relationCoefficient = relationCoefficient;

}

public double getItselfCoefficient() {

return itselfCoefficient;

}

public void setItselfCoefficient(double itselfCoefficient) {

this.itselfCoefficient = itselfCoefficient;

}

@Override

protected void relationZhiliLine(List<LineResult> lineResults) {

if (CollectionUtils.isEmpty(lineResults)) {

return;

}

lineResults.forEach(lineResult -> {

lineResult.setVoltage(lineResult.getVoltage() \* relationCoefficient);

});

}

@Override

void relationZhili(SubstationDTO substationDTO) {

substationDTO.setVoltclass(substationDTO.getVoltclass() \* relationCoefficient);

}

@Override

protected void zhiliLine(List<LineResult> lineResults) {

if (CollectionUtils.isEmpty(lineResults)) {

return;

}

lineResults.forEach(lineResult -> {

lineResult.setVoltage(lineResult.getVoltage() \* itselfCoefficient);

});

}

public void zhiLi(SubstationDTO substationDTO) {

substationDTO.setVoltclass(substationDTO.getVoltclass() \* itselfCoefficient);

}

@Override

protected String getUnit() {

return UNIT;

}

@Override

protected Double getValue(LineResult lineResult) {

Double value = lineResult.getVoltage();

return value == null ? 0D : NumberUtil.round(value, 5).doubleValue() ;

}

@Override

protected String getTypeName() {

return IndexType.VOLTAGE.getLable() + "含有率";

}

@Override

protected List<LineResult> overLineListBySubstationId(List<Long> substationId, Date logTime, Integer harmonicNum, Long mapId) {

return lineResultService.voltageOverList(substationId, logTime, harmonicNum, mapId);

}

@Override

public List<SubstationDTO> getOverData(List<SubstationDTO> substationDTOS) {

List<SubstationDTO> result = new ArrayList<>();

substationDTOS.forEach(substationDTO -> {

if (substationDTO.getStateV().equals(HarmonicMapConstant.STATE\_OVER)) {

result.add(substationDTO);

}

});

return result;

}

}

package com.cet.pq.harmonic.trend.calculate;

import Jama.Matrix;

import com.alibaba.fastjson.JSON;

import com.isoftstone.bussiness.zhejiang.DerivedFunction;

public class MulNewton {

public MulNewton() {

}

public Double[] newton(Double[] x, int cs, int k, Double[] p, Double[] q, Double[][] g, Double[][] b, Double[] Va, Double[] Vi, Double[] lb, Double[] ub) {

double eps = 1.0E-8;

double lamda = 2.0;

int n = -1;

double tol = 1.0;

int N = 1000;

int symx = x.length;

Matrix ally = new Matrix(1, N, 0.0);

Matrix allx = new Matrix(symx, N, 0.0);

double[][] x1 = new double[symx][1];

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

x1[i][0] = x[i];

}

Matrix x0 = new Matrix(x1);

while(tol > eps) {

++n;

DerivedFunction derivedFunction1 = new DerivedFunction();

double fx = derivedFunction1.func(x, cs, k, p, q, g, b, Va, Vi);

DerivedFunction derivedFunction = new DerivedFunction((xx) -> {

return derivedFunction1.func(xx, cs, k, p, q, g, b, Va, Vi);

});

Double[] dfx = derivedFunction.apply(x);

double[][] dfx1 = new double[dfx.length][1];

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

dfx1[i][0] = dfx[i];

}

Matrix df = new Matrix(dfx1);

Matrix eye = new Matrix(symx, symx, 0.0);

for(int i = 0; i < eye.getRowDimension(); ++i) {

eye.set(i, i, lamda);

}

Matrix times = df.times(df.transpose());

// int rank1 = df.rank();

// int rank2 = eye.rank();

// int rank3 = times.rank();

// Matrix plus = times.plus(eye);

// int rank = plus.rank();

// Matrix inverse = plus.inverse();

Matrix df2 = df.times(df.transpose()).plus(eye).inverse();

// Matrix df2 = df.times(df.transpose()).plus(eye);

Matrix df3 = df2.times(df).times(fx);

Matrix r = x0.minus(df3);

int i;

for(i = 0; i < symx; ++i) {

if (r.get(i, 0) < lb[i]) {

r.set(i, 0, lb[i] / 2.0);

}

if (r.get(i, 0) > ub[i]) {

r.set(i, 0, ub[i] / 2.0);

}

}

tol = r.minus(x0).norm2();

if (tol > eps) {

lamda = 1.1 \* lamda;

}

if (n >= N) {

System.out.println("迭代步数太多，可能不收敛！");

break;

}

for(i = 0; i < symx; ++i) {

allx.set(i, n, x0.get(i, 0));

}

ally.set(0, n, tol);

for(i = 0; i < symx; ++i) {

x[i] = r.get(i, 0);

x0.set(i, 0, r.get(i, 0));

}

}

Double[] bestX = new Double[symx + 1];

int i;

for(i = 0; i < 2 \* k; ++i) {

bestX[i] = allx.get(i, n);

}

for(i = 2 \* k; i < symx; ++i) {

bestX[i + 1] = allx.get(i, n);

}

bestX[2 \* k] = 0.0;

System.out.println(JSON.toJSONString(bestX));

return bestX;

}

}

package com.cet.pq.harmonic.trend.calculate;

import com.cet.pq.harmonic.trend.model.dto.XddaonaDTO;

import lombok.extern.slf4j.Slf4j;

import java.math.BigDecimal;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@Slf4j

public class RXB {

/\*\*

\* 默认电导率(S/m)

\*/

private static Double DEFAULT\_P = 0.03D;

/\*\*

\* 默认单位长度线路电抗 (单位：Ω/m)

\*/

private static Double DEFAULT\_REACTANCE\_LENGTH = 0.00040D;

/\*\*

\* 默认型号

\*/

private static String DEFAULT\_MODEL = "JL/G1A-400/50";

/\*\*

\* 型号横截面

\*/

private static Map<String, Double> MODEL\_J\_MAP = new HashMap<String, Double>() {

{

this.put("JL/G1A-50/8", 56.30D);

this.put("JL/G1A-70/10", 79.39D);

this.put("JL/G1A-95/15", 109.73D);

this.put("JL/G1A-120/20", 134.49D);

this.put("JL/G1A-150/20", 164.50D);

this.put("JL/G1A-185/25", 211.28D);

this.put("JL/G1A-185/30", 210.93D);

this.put("JL/G1A-240/30", 275.96D);

this.put("JL/G1A-240/40", 277.74D);

this.put("JL/G1A-300/40", 338.99D);

this.put("JL/G1A-400/35", 425.24D);

this.put("JL/G1A-400/50", 451.54D);

this.put("JL/G1A-400/65", 464.00D);

this.put("JL/G1A-500/45", 531.68D);

this.put("JL/G1A-500/65", 565.00D);

this.put("JL/G1A-630/45", 674.30D);

this.put("JL/G1A-630/55", 696.22D);

this.put("JL/G1A-720/50", 775.41D);

this.put("JL/G1A-720/90", 817.1D);

this.put("JL/G1A-800/55", 870.60D);

this.put("JL/G1A-900/40", 939D);

this.put("JL/G1A-900/75", 975D);

this.put("JL/G1A-1000/45", 1043D);

this.put("JL/G1A-1120/50", 1167D);

}

};

/\*\*

\*

\* @param length （m） 导线长度 单位m

\* @return Ω

\*/

public static double r(double length) {

return r(length, DEFAULT\_MODEL, DEFAULT\_P);

}

/\*\*

\* 计算电阻

\* @param length（m） 导线长度 单位m

\* @param modelName 型号

\* @return Ω

\*/

public static double r(double length, String modelName) {

return r(length, modelName, DEFAULT\_P);

}

/\*\*

\* 计算电阻

\* @param length 导线长度 单位m

\* @param modelName 型号

\* @param p 电导率(S/m)

\* @return Ω

\*/

public static double r(double length, String modelName,double p) {

double j = getj(modelName);

return r(length, j, p);

}

/\*\*

\* 计算电阻

\* @param length 导线长度 单位m

\* @param j 横截面积 单位mm²

\* @param p 电导率(S/m)

\* @return Ω

\*/

public static double r(double length, double j,double p) {

double r = 0D;

if (j != 0) {

r = p\*length/j;

}

return r;

}

/\*\*

\* 计算电抗

\* @param length（m） 导线长度 单位m

\* @return Ω

\*/

public static double x(double length) {

return x(DEFAULT\_REACTANCE\_LENGTH, length);

}

/\*\*

\* 计算电抗

\* @param reactanceLength 默认单位长度线路电抗 (单位：Ω/m)

\* @param length （m） 导线长度 单位m

\* @return Ω

\*/

public static double x(double reactanceLength, double length) {

return reactanceLength\*length;

}

/\*\*

\* 计算电纳

\* @param length（m） 导线长度 单位m

\* @return Ω

\*/

public static double b(double length) {

double x = x(length);

return bByX(x);

}

/\*\*

\* 计算电纳

\* @param length（m） 导线长度 单位m

\* @param reactanceLength 默认单位长度线路电抗 (单位：Ω/m)

\* @return S

\*/

public static double b(double length, double reactanceLength) {

double x = x(reactanceLength, length);

return bByX(x);

}

/\*\*

\* 计算电纳

\* @param x 电抗 单位：Ω

\* @return S

\*/

public static double bByX(double x) {

if (x == 0) {

return 0D;

}

return 1/x;

}

/\*\*

\* 获取横截面积

\* @param modelName

\* @return 单位 mm²

\*/

private static double getj(String modelName) {

Double j = MODEL\_J\_MAP.get(modelName);

if (j == null) {

j = MODEL\_J\_MAP.get(DEFAULT\_MODEL);

}

return j;

}

/\*\*

\* 获取长度

\* @return 长度 单位 m

\*/

public static double getLeng() {

return Math.random()\*10\*1000;

}

public static void main(String[] args) {

for (int i = 0; i < 100; i++) {

double leng = RXB.getLeng();

double r = r(leng);

double x = x(leng);

double b = bByX(x);

// 0.00256、0.018495、 0.0723

// 0.00011、0.000795、 0.001091287

log.info("长度：{}；电阻：{}；电抗：{}；电纳：{}",

leng,

new BigDecimal(r/1000).setScale(5, BigDecimal.ROUND\_HALF\_UP),

new BigDecimal(x/1000).setScale(5, BigDecimal.ROUND\_HALF\_UP),

new BigDecimal(b/1000).setScale(5, BigDecimal.ROUND\_HALF\_UP)

);

}

}

/\*\*

\* 批量设置线段间的r、x、b

\* @param xddaonaDTOS

\*/

public static void set(List<XddaonaDTO> xddaonaDTOS) {

xddaonaDTOS.forEach(item -> {

double leng = RXB.getLeng();

double r = r(leng);

double x = x(leng);

double b = b(x);

item.setR(r / 1000);

item.setX(x / 1000);

item.setB(b / 1000);

});

}

}

package com.cet.pq.inventoryservice.service.impl;

import com.alibaba.excel.write.metadata.WriteSheet;

import com.cet.pq.common.constant.ExcelConstant;

import com.cet.pq.common.constant.TableName;

import com.cet.pq.common.exception.CommonManagerException;

import com.cet.pq.common.model.ConditionBlock;

import com.cet.pq.common.model.PageResult;

import com.cet.pq.common.model.Result;

import com.cet.pq.common.model.pqobject.Substation;

import com.cet.pq.common.time.DateUtils;

import com.cet.pq.common.utils.JsonTransferUtils;

import com.cet.pq.common.utils.ModelServiceUtils;

import com.cet.pq.common.utils.ParseDataUtil;

import com.cet.pq.inventoryservice.common.constants.ModelImportConstants;

import com.cet.pq.inventoryservice.common.utils.EasyExcelUtils;

import com.cet.pq.inventoryservice.model.excelmodel.SubstationExcel;

import com.cet.pq.inventoryservice.model.log.OperationType;

import com.cet.pq.inventoryservice.model.log.SubOperationType;

import com.cet.pq.inventoryservice.service.KeyColumnService;

import com.cet.pq.inventoryservice.service.SubStationExcelService;

import com.cet.pq.inventoryservice.service.SubstationInventoryService;

import org.apache.commons.collections4.CollectionUtils;

import org.springframework.beans.BeanUtils;

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

import org.springframework.stereotype.Component;

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.text.SimpleDateFormat;

import java.util.\*;

import java.util.concurrent.ConcurrentHashMap;

import java.util.stream.Collectors;

@Component

public class SubStationServiceImpl implements SubStationExcelService {

@Autowired

SubstationInventoryService substationInventoryService;

@Autowired

private KeyColumnService keyColumnService;

@Override

public void downLoadModel(HttpServletResponse response) throws IOException {

EasyExcelUtils.getModelFile(response, ModelImportConstants.SUBSTATION\_BOOK\_MODEL\_NAME);

}

@Override

public List<String> importSubstationData(MultipartFile excel) {

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

// 1、获取excel数据

List<SubstationExcel> substationExcelList = EasyExcelUtils.readExcel(excel, SubstationExcel.class, ModelImportConstants.SUBSTATION\_BOOK\_SHEET\_NAME, ModelImportConstants.SUBSTATION\_BOOK\_ROW\_START);

//2、去掉已经存在的重复数据

List<Map<String, Object>> existSubstationList = ModelServiceUtils.querySingleModel(null, TableName.SUBSTATION, null, null, null).getData();

Iterator<SubstationExcel> it = substationExcelList.iterator();

while (it.hasNext()) {

SubstationExcel substation = it.next();

List<Map<String, Object>> filterExistProvince = existSubstationList.stream()

.filter(ec -> ParseDataUtil.parseString(ec.get("name")).equals(substation.getName())).collect(Collectors.toList());

if (CollectionUtils.isNotEmpty(filterExistProvince)) {

String msg = "数据导入出错，错误原因：数据名称或者代码已经存在：" + substation.getRowid();

errorMsgList.add(msg);

it.remove();

}

}

//3、查询所有的单位

if (CollectionUtils.isEmpty(substationExcelList)) {

return errorMsgList;

}

List<String> companyNameList = substationExcelList.stream().map(SubstationExcel::getCompany).collect(Collectors.toList());

if (CollectionUtils.isEmpty(companyNameList)) {

throw new CommonManagerException("管理单位信息不能为空");

}

List<Map<String, Object>> cityCompanyList = ModelServiceUtils.queryWithChildren(null, TableName.CITYCOMPANY,

Arrays.asList(new ConditionBlock("name", ConditionBlock.OPERATOR\_IN, companyNameList)), null, null);

List<Map<String, Object>> countyCompanyList = ModelServiceUtils.queryWithChildren(null, TableName.COUNTYCOMPANY,

Arrays.asList(new ConditionBlock("name", ConditionBlock.OPERATOR\_IN, companyNameList)), null, null);

// 3、查询所有的行政区域、

List<String> areaNameList = substationExcelList.stream().map(SubstationExcel::getArea).collect(Collectors.toList());

List<Map<String, Object>> cityAreaList = ModelServiceUtils.queryWithChildren(null, TableName.CITY, Arrays.asList(new ConditionBlock("name", ConditionBlock.OPERATOR\_IN, areaNameList)), null,

null);

List<Map<String, Object>> ditrictAreaList = ModelServiceUtils.queryWithChildren(null, TableName.DISTRICT, Arrays.asList(new ConditionBlock("name", ConditionBlock.OPERATOR\_IN, areaNameList)),

null, null);

// 4、逐条处理导入的数据

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

substationExcelList.forEach(s -> {

try {

// 4.1检查管理单位是否存在

Long substationId = validArea(s, cityAreaList, ditrictAreaList);

// 4.2检查行政区域是否存在

s.setId(substationId);

validCompany(s, cityCompanyList, countyCompanyList, descriptions);

} catch (Exception e) {

String msg = "导入负荷电站失败，失败原因：" + e.getMessage() + ",失败数据行数：" + s.getRowid();

errorMsgList.add(msg);

}

});

//添加日志

keyColumnService.saveLogs(descriptions, OperationType.SUBSTATION\_LOG.getId(), SubOperationType.INSERT\_LOG.getId());

return errorMsgList;

}

@Override

public void export(HttpServletResponse response, String modelName, long modelId) throws IOException {

PageResult<Object> pageResult = substationInventoryService.querySubstation(modelName, modelId, 0, 9999,"",null);

if (pageResult.getData() == null) {

return;

}

List<Map<String, Object>> dataList = (List<Map<String, Object>>) pageResult.getData();

List<SubstationExcel> excelList = JsonTransferUtils.transferList(dataList, SubstationExcel.class);

for (int i = 0; i < dataList.size(); i++) {

Map<String, Object> line = dataList.get(i);

SubstationExcel substationExcel = excelList.get(i);

// 子级数据

List<Map<String, Object>> citycompany\_model = (List<Map<String, Object>>) line.get("citycompany\_model");

List<Map<String, Object>> countycompany\_model = (List<Map<String, Object>>) line.get("countycompany\_model");

List<Map<String, Object>> district\_model = (List<Map<String, Object>>) line.get("district\_model");

// 行号

substationExcel.setRowid(String.valueOf(i + 1));

// 赋值剩余子级字段

if (citycompany\_model != null) {

substationExcel.setCompany((String) citycompany\_model.get(0).get("name"));

}

if (countycompany\_model != null) {

substationExcel.setCompany((String) countycompany\_model.get(0).get("name"));

}

if (district\_model != null) {

substationExcel.setArea((String) district\_model.get(0).get("name"));

}

excelList.set(i, substationExcel);

}

// 设置导出的writeSheet

WriteSheet writeSheet = new WriteSheet();

writeSheet.setSheetNo(0);

writeSheet.setNeedHead(Boolean.FALSE);

writeSheet.setSheetName(ModelImportConstants.SUBSTATION\_BOOK\_SHEET\_NAME);

// 组装导出的文件名称

Date date = new Date();

SimpleDateFormat format = new SimpleDateFormat(DateUtils.DF\_YYYY\_MM\_DD\_HH\_MM\_SS);

String dateStr = format.format(date);

String fileName = ModelImportConstants.SUBSTATION\_BOOK\_EXPORT\_FILE\_NAME + "-" + dateStr + "." + ExcelConstant.EXCEL\_TYPE\_XLSX;

//写入模板

EasyExcelUtils.writeByModel(response, fileName, excelList, SubstationExcel.class, ModelImportConstants.SUBSTATION\_BOOK\_MODEL\_NAME, writeSheet);

}

/\*\*

\* 检查管理单位是否存在并写入关联关系

\*

\* @param substation

\* @param cityCompanyList

\* @param countyCompanyList

\*/

private void validCompany(SubstationExcel substation, List<Map<String, Object>> cityCompanyList, List<Map<String, Object>> countyCompanyList, List<String> descriptions) {

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

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

if (CollectionUtils.isNotEmpty(cityCompanyList)) {

cityCompanyFilter = cityCompanyList.stream().filter(cc -> ParseDataUtil.parseString(cc.get("name")).trim().equals(substation.getCompany().trim())).collect(Collectors.toList());

}

if (CollectionUtils.isNotEmpty(countyCompanyList)) {

countyCompanyFilter = countyCompanyList.stream().filter(coc -> ParseDataUtil.parseString(coc.get("name")).trim().equals(substation.getCompany().trim())).collect(Collectors.toList());

}

if (CollectionUtils.isEmpty(cityCompanyFilter) && CollectionUtils.isEmpty(countyCompanyFilter)) {

String errMsg = "导入负荷电站出错，错误原因：管理单位不存在，错误数据行数：" + substation.getRowid();

throw new CommonManagerException(errMsg);

} else if (CollectionUtils.isNotEmpty(cityCompanyFilter) && CollectionUtils.isNotEmpty(countyCompanyFilter)) {

String errMsg = "导入负荷电站出错，错误原因：管理单位存在多个同名单位，请修改单位台账，错误数据行数：" + substation.getRowid();

throw new CommonManagerException(errMsg);

}

Substation substationVo = new Substation();

BeanUtils.copyProperties(substation, substationVo);

substationVo.setCommissioningdate(null == substation.getCommissioningdate() ? null : substation.getCommissioningdate().getTime());

substationVo.setUpdatetime(new Date().getTime());

Map<String, Object> company = new ConcurrentHashMap<>();

if (CollectionUtils.isNotEmpty(cityCompanyFilter)) {

company = cityCompanyFilter.get(0);

company.put("modelLabel", TableName.CITYCOMPANY);

company.put("substation\_model", Arrays.asList(substationVo));

ModelServiceUtils.write(Arrays.asList(company));

} else if (CollectionUtils.isNotEmpty(countyCompanyFilter)) {

company = countyCompanyFilter.get(0);

company.put("modelLabel", TableName.COUNTYCOMPANY);

company.put("substation\_model", Arrays.asList(substationVo));

ModelServiceUtils.write(Arrays.asList(company));

}

descriptions.add("新建负荷电站:" + substationVo.getName());

}

/\*\*

\* 检查行政区域是否存在并写入关联关系

\*

\* @param substation

\* @param cityAreaList

\* @param ditrictAreaList

\*/

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

private Long validArea(SubstationExcel substation, List<Map<String, Object>> cityAreaList, List<Map<String, Object>> ditrictAreaList) {

List<Map<String, Object>> cityAreaFilter = cityAreaList.stream().filter(c -> ParseDataUtil.parseString(c.get("name")).equals(substation.getArea())).collect(Collectors.toList());

List<Map<String, Object>> districtAreaFilter = ditrictAreaList.stream().filter(d -> ParseDataUtil.parseString(d.get("name")).equals(substation.getArea())).collect(Collectors.toList());

if (CollectionUtils.isEmpty(cityAreaFilter) && CollectionUtils.isEmpty(districtAreaFilter)) {

String errMsg = "导入负荷电站出错，错误原因：行政区域不存在，错误数据行数：" + substation.getRowid();

throw new CommonManagerException(errMsg);

} else if (CollectionUtils.isNotEmpty(cityAreaFilter) && CollectionUtils.isNotEmpty(districtAreaFilter)) {

String errMsg = "导入负荷电站出错，错误原因：行政区域存在重复，请更改台账信息，错误数据行数：" + substation.getRowid();

throw new CommonManagerException(errMsg);

}

Substation substationVo = new Substation();

BeanUtils.copyProperties(substation, substationVo);

substationVo.setCommissioningdate(null == substation.getCommissioningdate() ? null : substation.getCommissioningdate().getTime());

substationVo.setUpdatetime(new Date().getTime());

Long substationId = null;

Map<String, Object> area = new ConcurrentHashMap<>();

Result res = null;

if (CollectionUtils.isNotEmpty(cityAreaFilter)) {

area = cityAreaFilter.get(0);

area.put("modelLabel", TableName.CITY);

area.put("substation\_model", Arrays.asList(substationVo));

res = ModelServiceUtils.write(Arrays.asList(area));

} else if (CollectionUtils.isNotEmpty(districtAreaFilter)) {

area = districtAreaFilter.get(0);

area.put("modelLabel", TableName.DISTRICT);

area.put("substation\_model", Arrays.asList(substationVo));

res = ModelServiceUtils.write(Arrays.asList(area));

}

if (res != null && res.getCode() == 0) {

Map<String, Object> district = (Map<String, Object>) ParseDataUtil.parseList(res.getData()).get(0);

Map<String, Object> substationRes = (Map<String, Object>) ParseDataUtil.parseList(district.get("substation\_model")).get(0);

if (null != substationRes) {

substationId = ParseDataUtil.parseLong(substationRes.get("id"));

}

}

return substationId;

}

}

package com.cet.pq.inventoryservice.service.impl;

import com.alibaba.excel.write.metadata.WriteSheet;

import com.cet.pq.anlysis.model.complianceevaluation.ComprehensiveEvaluationTableParam;

import com.cet.pq.anlysis.service.PowerQualityEvaluationIndexService;

import com.cet.pq.common.constant.ColumnName;

import com.cet.pq.common.constant.CommonConstant;

import com.cet.pq.common.constant.ExcelConstant;

import com.cet.pq.common.constant.TableName;

import com.cet.pq.common.enums.UnitLevelEnum;

import com.cet.pq.common.exception.CommonManagerException;

import com.cet.pq.common.feign.ModelDataService;

import com.cet.pq.common.handle.SessionHandler;

import com.cet.pq.common.model.\*;

import com.cet.pq.common.model.auth.ModelNode;

import com.cet.pq.common.model.auth.User;

import com.cet.pq.common.time.DateUtils;

import com.cet.pq.common.utils.\*;

import com.cet.pq.inventoryservice.common.constants.ModelImportConstants;

import com.cet.pq.inventoryservice.common.constants.TableColumnConstants;

import com.cet.pq.inventoryservice.common.utils.EasyExcelUtils;

import com.cet.pq.inventoryservice.model.excelmodel.UnitExcel;

import com.cet.pq.inventoryservice.model.log.OperationType;

import com.cet.pq.inventoryservice.model.log.SubOperationType;

import com.cet.pq.inventoryservice.service.CompanyInventoryService;

import com.cet.pq.inventoryservice.service.KeyColumnService;

import com.cet.pq.inventoryservice.service.UnitExcelService;

import org.apache.commons.collections4.CollectionUtils;

import org.apache.commons.lang3.StringUtils;

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

import org.springframework.stereotype.Component;

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

import java.text.SimpleDateFormat;

import java.util.\*;

import java.util.stream.Collectors;

import static com.cet.pq.common.handle.SessionHandler.getSession;

import static com.cet.pq.common.handle.SessionHandler.getUser;

@Component

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

/\*\*

\* @author yezhuang

\* @date 2022/11/8

\*/

public class UnitExcelServiceImpl implements UnitExcelService {

@Autowired

private ModelDataService modelDataService;

@Autowired

private KeyColumnService keyColumnService;

@Autowired

private CompanyInventoryService companyInventoryService;

@Autowired

private PowerQualityEvaluationIndexService powerQualityEvaluationIndexService;

private static final String PROVINCE\_LEVEL = "省级";

private static final String CITY\_LEVEL = "地市级";

private static final String COUNTY\_LEVEL = "县级";

private static List<Integer> SENSITIVEUSERLIST = Arrays.asList(2401, 2402, 2403, 2404, 2405, 2406,

2407, 2408, 2409, 2410, 2411);

private static final String SUFFIX\_MODEL\_SINGLE = "\_model\_single";

private static final String SUFFIX\_MODEL = "\_model";

private static final String MODELLABEL = "modelLabel";

private static final String CHILDREN = "children";

private static final String DEPTH = "depth";

@Override

public List<String> imporUnitBook(MultipartFile excel) {

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

// 1、获取Excel数据

List<UnitExcel> unitElist = EasyExcelUtils.readExcel(excel, UnitExcel.class, ModelImportConstants.UNIT\_BOOK\_SHEET\_NAME, ModelImportConstants.UNIT\_BOOK\_ROW\_START);

// 2、校验数据库中的重复数据

ResultWithTotal<List<Map<String, Object>>> hasExistProvinceComPanyRes = ModelServiceUtils.querySingleModel(null, TableName.PROVINCECOMPANY, null, null, null);

ResultWithTotal<List<Map<String, Object>>> hasExistCityComPanyRes = ModelServiceUtils.querySingleModel(null, TableName.CITYCOMPANY, null, null, null);

ResultWithTotal<List<Map<String, Object>>> hasExistCountyComPanyRes = ModelServiceUtils.querySingleModel(null, TableName.COUNTYCOMPANY, null, null, null);

// 去掉数据库中已经存在的重复数据

List<Map<String, Object>> existProvinceCompanyList = hasExistProvinceComPanyRes.getData();

List<Map<String, Object>> existCityCompanyList = hasExistCityComPanyRes.getData();

List<Map<String, Object>> existCountyCompanyList = hasExistCountyComPanyRes.getData();

Iterator<UnitExcel> it = unitElist.iterator();

while (it.hasNext()) {

UnitExcel c = it.next();

List<Map<String, Object>> filterExistProvinceCompany = existProvinceCompanyList.stream()

.filter(ec -> ParseDataUtil.parseString(ec.get("code")) .equals( c.getCode()) || ParseDataUtil.parseString(ec.get("name")).equals(c.getName())).collect(Collectors.toList());

List<Map<String, Object>> filterExistCityComPany = existCityCompanyList.stream()

.filter(ec -> ParseDataUtil.parseString(ec.get("code")) .equals( c.getCode()) || ParseDataUtil.parseString(ec.get("name")).equals(c.getName())).collect(Collectors.toList());

List<Map<String, Object>> filterExistCountyComPany = existCountyCompanyList.stream()

.filter(ec -> ParseDataUtil.parseString(ec.get("code")) .equals( c.getCode()) || ParseDataUtil.parseString(ec.get("name")).equals(c.getName())).collect(Collectors.toList());

if (CollectionUtils.isNotEmpty(filterExistProvinceCompany) || CollectionUtils.isNotEmpty(filterExistCityComPany) || CollectionUtils.isNotEmpty(filterExistCountyComPany)) {

String msg = "数据导入出错，错误原因：数据名称或者代码已经存在：" + c.getRowid();

errorMsg.add(msg);

it.remove();

}

}

// 2、先插入省级数据

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

List<UnitExcel> provinceCompanylist = unitElist.stream().filter(u -> u.getLevel().equals(UnitLevelEnum.provinceLevel.getUnitLevel())).collect(Collectors.toList());

provinceCompanylist.forEach(provinceCompany -> {

provinceCompany.setModelLabel(TableName.PROVINCECOMPANY);

provinceCompany.setUpdatetime(System.currentTimeMillis());

descriptions.add("新建管理单位:" + provinceCompany.getName());

});

if (CollectionUtils.isNotEmpty(provinceCompanylist)) {

Result res = modelDataService.write(provinceCompanylist);

CommonUtils.checkResult(res, "省级单位插入错误");

}

// 3、插入地级：需要查询上级是否存在

List<UnitExcel> cityCompanylist = unitElist.stream().filter(u -> u.getLevel().equals(UnitLevelEnum.cityLevel.getUnitLevel())).collect(Collectors.toList());

writeCityCompany(cityCompanylist, errorMsg, descriptions);

// 4、插入县级：需要查询上级是否存在

List<UnitExcel> countyCompanylist = unitElist.stream().filter(u -> u.getLevel().equals(UnitLevelEnum.countyLevel.getUnitLevel())).collect(Collectors.toList());

writeCountyCompany(countyCompanylist, errorMsg, descriptions);

//添加日志

keyColumnService.saveLogs(descriptions, OperationType.COMMANY\_LOG.getId(), SubOperationType.INSERT\_LOG.getId());

return errorMsg;

}

/\*\*

\* 插入地级管理单位信息

\*

\* @param cityCompanylist

\* @return

\*/

private void writeCityCompany(List<UnitExcel> cityCompanylist, List<String> errorMsg, List<String> descriptions) {

List<String> provinceCompanyNameList = cityCompanylist.stream().map(UnitExcel::getSupername).collect(Collectors.toList());

List<ConditionBlock> filter = Arrays.asList(new ConditionBlock("name", ConditionBlock.OPERATOR\_IN, provinceCompanyNameList));

ResultWithTotal<List<Map<String, Object>>> res = ModelServiceUtils.querySingleModel(null, TableName.PROVINCECOMPANY, filter, null, null);

List<Map<String, Object>> provinceCompanyList = res.getData();

provinceCompanyList.forEach(p -> {

p.put(MODELLABEL, TableName.PROVINCECOMPANY);

p.put("citycompany\_model", new ArrayList<>());

});

cityCompanylist.forEach(c -> {

c.setModelLabel(TableName.CITYCOMPANY);

c.setUpdatetime(System.currentTimeMillis());

Boolean flag = Boolean.FALSE;

for (Map<String, Object> provincecompany : provinceCompanyList) {

if (ParseDataUtil.parseString(provincecompany.get("name")).equals(c.getSupername())) {

flag = Boolean.TRUE;

List<Object> citycompanyList = ParseDataUtil.parseList(provincecompany.get("citycompany\_model"));

citycompanyList.add(c);

provincecompany.put("citycompany\_model", citycompanyList);

}

}

if (!flag) {

String msg = "数据导入出错，错误原因：上级单位不存在,错误数据行：" + c.getRowid();

errorMsg.add(msg);

}

descriptions.add("新建管理单位:" + c.getName());

});

if (CollectionUtils.isNotEmpty(provinceCompanyList)) {

Result cityCompanyres = ModelServiceUtils.write(provinceCompanyList);

CommonUtils.checkResult(cityCompanyres, "地级管理单位导入错误");

}

}

/\*\*

\* 插入县级管理单位信息

\*

\* @param

\* @return

\*/

private void writeCountyCompany(List<UnitExcel> countyCompanylist, List<String> errorMsg, List<String> descriptions) {

List<String> cityCompanyNameList = countyCompanylist.stream().map(UnitExcel::getSupername).collect(Collectors.toList());

List<ConditionBlock> filter = Arrays.asList(new ConditionBlock("name", ConditionBlock.OPERATOR\_IN, cityCompanyNameList));

ResultWithTotal<List<Map<String, Object>>> res = ModelServiceUtils.querySingleModel(null, TableName.CITYCOMPANY, filter, null, null);

List<Map<String, Object>> cityCompanyList = res.getData();

cityCompanyList.forEach(p -> {

p.put(MODELLABEL, TableName.CITYCOMPANY);

p.put("countycompany\_model", new ArrayList<>());

});

countyCompanylist.forEach(c -> {

c.setModelLabel(TableName.COUNTYCOMPANY);

c.setUpdatetime(System.currentTimeMillis());

Boolean flag = Boolean.FALSE;

for (Map<String, Object> citycompany : cityCompanyList) {

if (ParseDataUtil.parseString(citycompany.get("name")).equals(c.getSupername())) {

flag = Boolean.TRUE;

List<Object> countycompanyList = ParseDataUtil.parseList(citycompany.get("countycompany\_model"));

countycompanyList.add(c);

citycompany.put("countycompany\_model", countycompanyList);

}

}

if (!flag) {

String msg = "数据导入出错，错误原因：上级单位不存在,错误数据行：" + c.getRowid();

errorMsg.add(msg);

}

descriptions.add("新建管理单位:" + c.getName());

});

if (CollectionUtils.isNotEmpty(cityCompanyList)) {

Result countyRes = ModelServiceUtils.write(cityCompanyList);

CommonUtils.checkResult(countyRes, "县级单位导入错误");

}

}

@Override

public void downLoadUnitModelFile(HttpServletResponse response) throws IOException {

EasyExcelUtils.getModelFile(response, ModelImportConstants.UNIT\_BOOK\_MODEL\_NAME);

}

@Override

public void exportUnitBook(String modelName, long modelId, HttpServletResponse response) throws IOException {

List<Map<String, Object>> result = companyInventoryService.queryCompany(modelName, modelId);

if (CollectionUtils.isEmpty(result)) {

return;

}

List<UnitExcel> unitExcelList = new ArrayList<>();

for (Map<String, Object> data : result) {

String modelLabel = (String) data.get(MODELLABEL);

// 行号

int rowid = 1;

// 查询节点为省

if (TableColumnConstants.TABLE\_PROVINCE\_COMPANY.equals(modelLabel)) {

// 省级单位

UnitExcel provinceCompany = MapUtil.mapToObject(data, UnitExcel.class);

provinceCompany.setRowid(String.valueOf(rowid));

provinceCompany.setLevel(PROVINCE\_LEVEL);

List<Map<String, Object>> netcompany\_model = (List<Map<String, Object>>) data.get("netcompany\_model");

if (netcompany\_model != null) {

provinceCompany.setSupername((String) netcompany\_model.get(0).get("name"));

}

unitExcelList.add(provinceCompany);

// 地市级单位

List<Map<String, Object>> citycompany\_model = (List<Map<String, Object>>) data.get("citycompany\_model");

if (CollectionUtils.isNotEmpty(citycompany\_model)) {

for (Map<String, Object> map : citycompany\_model) {

rowid++;

UnitExcel cityCompany = MapUtil.mapToObject(map, UnitExcel.class);

cityCompany.setRowid(String.valueOf(rowid));

cityCompany.setLevel(CITY\_LEVEL);

cityCompany.setSupername(provinceCompany.getName());

unitExcelList.add(cityCompany);

// 县级单位

List<Map<String, Object>> countycompany\_model = (List<Map<String, Object>>) map.get("countycompany\_model");

if (CollectionUtils.isNotEmpty(countycompany\_model)) {

for (Map<String, Object> county : countycompany\_model) {

rowid++;

UnitExcel countyCompany = MapUtil.mapToObject(county, UnitExcel.class);

countyCompany.setRowid(String.valueOf(rowid));

countyCompany.setLevel(COUNTY\_LEVEL);

countyCompany.setSupername(cityCompany.getName());

unitExcelList.add(countyCompany);

}

}

}

}

}

// 查询节点为地市级

if (TableColumnConstants.TABLE\_CITY\_COMPANY.equals(modelLabel)) {

// 地市级

UnitExcel cityCompany = MapUtil.mapToObject(data, UnitExcel.class);

cityCompany.setRowid(String.valueOf(rowid));

cityCompany.setLevel(CITY\_LEVEL);

List<Map<String, Object>> provincecompany\_model = (List<Map<String, Object>>) data.get("provincecompany\_model");

if (provincecompany\_model != null) {

cityCompany.setSupername((String) provincecompany\_model.get(0).get("name"));

}

unitExcelList.add(cityCompany);

// 县级单位

List<Map<String, Object>> countycompany\_model = (List<Map<String, Object>>) data.get("countycompany\_model");

if (CollectionUtils.isNotEmpty(countycompany\_model)) {

for (Map<String, Object> county : countycompany\_model) {

rowid++;

UnitExcel countyCompany = MapUtil.mapToObject(county, UnitExcel.class);

countyCompany.setRowid(String.valueOf(rowid));

countyCompany.setLevel(COUNTY\_LEVEL);

countyCompany.setSupername(cityCompany.getName());

unitExcelList.add(countyCompany);

}

}

}

// 查询节点为县级

if (TableColumnConstants.TABLE\_COUNTY\_COMPANY.equals(modelLabel)) {

UnitExcel countyCompany = MapUtil.mapToObject(data, UnitExcel.class);

countyCompany.setRowid(String.valueOf(rowid));

countyCompany.setLevel(COUNTY\_LEVEL);

List<Map<String, Object>> citycompany\_model = (List<Map<String, Object>>) data.get("citycompany\_model");

if (citycompany\_model != null) {

countyCompany.setSupername((String) citycompany\_model.get(0).get("name"));

}

unitExcelList.add(countyCompany);

}

}

// 设置导出的writeSheet

WriteSheet writeSheet = new WriteSheet();

writeSheet.setSheetNo(0);

writeSheet.setNeedHead(Boolean.FALSE);

writeSheet.setSheetName(ModelImportConstants.UNIT\_BOOK\_SHEET\_NAME);

// 组装导出的文件名称

Date date = new Date();

SimpleDateFormat format = new SimpleDateFormat(DateUtils.DF\_YYYY\_MM\_DD\_HH\_MM\_SS);

String dateStr = format.format(date);

String fileName = ModelImportConstants.UNIT\_BOOK\_EXPORT\_FILE\_NAME + "-" + dateStr + "." + ExcelConstant.EXCEL\_TYPE\_XLSX;

//写入模板

EasyExcelUtils.writeByModel(response, fileName, unitExcelList, UnitExcel.class, ModelImportConstants.UNIT\_BOOK\_MODEL\_NAME, writeSheet);

}

@Override

public void deleteUnitInventory(Long modelId, String modelLabel) {

// 校验是否为管理员

AuthUtils.cheakAuthRole(1L);

// 查询节点下是否存在节点

String subModelLabel = StringUtils.EMPTY;

if (TableName.PROVINCE.equals(modelLabel)) {

subModelLabel = TableName.CITY;

} else if (TableName.CITY.equals(modelLabel)) {

subModelLabel = TableName.DISTRICT;

} else if (TableName.PROVINCECOMPANY.equals(modelLabel)) {

subModelLabel = TableName.CITYCOMPANY;

} else if (TableName.CITYCOMPANY.equals(modelLabel)) {

subModelLabel = TableName.COUNTYCOMPANY;

}

List<Map<String, Object>> relations = ModelServiceUtils.getRelations(Collections.singletonList(modelId), modelLabel, null, null,

Arrays.asList(new SingleModelConditionDTO(subModelLabel), new SingleModelConditionDTO(TableName.SUBSTATION)));

if (CollectionUtils.isEmpty(relations)) {

throw new CommonManagerException("该节点已经被删除");

} else {

List<Map<String, Object>> sub = (List<Map<String, Object>>) relations.get(0).get(subModelLabel + SUFFIX\_MODEL);

if (!CollectionUtils.isEmpty(sub)) {

throw new CommonManagerException("该节点存在子节点，无法删除");

}

List<Map<String, Object>> substation = (List<Map<String, Object>>) relations.get(0).get(TableName.SUBSTATION + SUFFIX\_MODEL);

if (!CollectionUtils.isEmpty(substation)) {

throw new CommonManagerException("该节点关联负荷电站，无法删除");

}

}

// 删除节点

Result<Object> result = modelDataService.delete(modelLabel, Collections.singletonList(modelId));

ParamUtils.checkResultGeneric(result);

//添加日志

if (modelLabel.equals(TableName.CITYCOMPANY) || modelLabel.equals(TableName.PROVINCECOMPANY) || modelLabel.equals(TableName.COUNTYCOMPANY)) {

keyColumnService.saveLog("删除管理单位:" + ParseDataUtil.parseString(relations.get(0).get("name")),

OperationType.COMMANY\_LOG.getId(), SubOperationType.DELETE\_LOG.getId());

} else if (modelLabel.equals(TableName.CITY) || modelLabel.equals(TableName.PROVINCE) || modelLabel.equals(TableName.DISTRICT)) {

keyColumnService.saveLog("删除行政区域:" + ParseDataUtil.parseString(relations.get(0).get("name")),

OperationType.AREA\_LOG.getId(), SubOperationType.DELETE\_LOG.getId());

}

}

@Override

public void deleteSubstation(Long id) {

// 校验是否为管理员

AuthUtils.cheakAuthRole(1L);

// 查询负荷电站是否关联监测点和终端

List<SingleModelConditionDTO> subConditions = Arrays.asList(new SingleModelConditionDTO(TableName.LINE),

new SingleModelConditionDTO(TableName.PQTERMINAL), new SingleModelConditionDTO(TableName.OFFLINETEST));

List<Map<String, Object>> relations = ModelServiceUtils.getRelations(Collections.singletonList(id), TableName.SUBSTATION, null, null,

subConditions);

if (CollectionUtils.isNotEmpty(relations)) {

List<Map<String, Object>> lines = ParseDataUtil.parseList(relations.get(0).get(TableName.LINE + SUFFIX\_MODEL));

if (CollectionUtils.isNotEmpty(lines)) {

throw new CommonManagerException("该负荷电站关联监测点，无法删除");

}

List<Map<String, Object>> pqTerminals = ParseDataUtil.parseList(relations.get(0).get(TableName.PQTERMINAL + SUFFIX\_MODEL));

if (CollectionUtils.isNotEmpty(pqTerminals)) {

throw new CommonManagerException("该负荷电站关联终端，无法删除");

}

List<Map<String, Object>> offlineTests = ParseDataUtil.parseList(relations.get(0).get(TableName.OFFLINETEST + SUFFIX\_MODEL));

if (CollectionUtils.isNotEmpty(offlineTests)) {

throw new CommonManagerException("该负荷电站关联在线监测，无法删除");

}

}

Result<Object> result = modelDataService.delete(TableName.SUBSTATION, Collections.singletonList(id));

ParamUtils.checkResultGeneric(result);

//添加日志

keyColumnService.saveLog("删除负荷电站:" + ParseDataUtil.parseString(relations.get(0).get("name")),

OperationType.SUBSTATION\_LOG.getId(), SubOperationType.DELETE\_LOG.getId());

}

/\*\*

\* 通用节点树接口

\*

\* @param isCompany

\* @return

\*/

@Override

public List<Map<String, Object>> getTreeNode(Boolean isCompany) {

HttpSession session = getSession();

User user = getUser();

List<Map<String, Object>> relationTree = ParseDataUtil.parseList(session.getAttribute(CommonConstant.USER\_RELATION\_TREE + user.getId() + isCompany));

if (CollectionUtils.isNotEmpty(relationTree)) {

return relationTree;

}

ModelNode userModel = SessionHandler.getUserModel();

String modelLabel = userModel.getModelLabel();

Long modelId = userModel.getId();

List<SingleModelConditionDTO> subConditions = new ArrayList<>(Arrays.asList(

new SingleModelConditionDTO(TableName.SUBSTATION),

new SingleModelConditionDTO(TableName.LINE)));

if (userModel.getModelLabel().equals(TableName.NETCOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCECOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.COUNTRY;

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCE));

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

// 如果授权区域为省级

} else if (userModel.getModelLabel().equals(TableName.PROVINCECOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.PROVINCE;

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

// 如果授权区域是地市级

} else if (userModel.getModelLabel().equals(TableName.CITYCOMPANY)) {

if (isCompany) {

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.CITY;

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

}

relationTree = ModelServiceUtils.getRelationTree(Arrays.asList(modelId), modelLabel, null, subConditions);

setDepth(relationTree);

session.setAttribute(CommonConstant.USER\_RELATION\_TREE + user.getId() + isCompany, relationTree);

return relationTree;

}

/\*\*

\* 离线测试台账节点树

\*

\* @param isCompany

\* @return

\*/

@Override

public List<Map<String, Object>> getOfflineTree(Boolean isCompany) {

ModelNode userModel = SessionHandler.getUserModel();

String modelLabel = userModel.getModelLabel();

Long modelId = userModel.getId();

List<SingleModelConditionDTO> subConditions = new ArrayList<>(Arrays.asList(

new SingleModelConditionDTO(TableName.SUBSTATION),

new SingleModelConditionDTO(TableName.OFFLINETEST)));

// 如果授权区域为网级

if (userModel.getModelLabel().equals(TableName.NETCOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCECOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.COUNTRY;

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCE));

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

} else if (userModel.getModelLabel().equals(TableName.PROVINCECOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.PROVINCE;

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

// 如果授权区域是地市级

} else if (userModel.getModelLabel().equals(TableName.CITYCOMPANY)) {

if (isCompany) {

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.CITY;

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

}

List<Map<String, Object>> relationTree = ModelServiceUtils.getRelationTree(Arrays.asList(modelId), modelLabel, null, subConditions);

setDepth(relationTree);

return relationTree;

}

@Override

public List<Map<String, Object>> getInterferenceSourceTree(Boolean isCompany, String label) {

ModelNode userModel = SessionHandler.getUserModel();

String modelLabel = userModel.getModelLabel();

Long modelId = userModel.getId();

List<SingleModelConditionDTO> subConditions = new ArrayList<>(Arrays.asList(

new SingleModelConditionDTO(label)));

// 如果授权区域为网级

if (userModel.getModelLabel().equals(TableName.NETCOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCECOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.COUNTRY;

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCE));

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

} else if (userModel.getModelLabel().equals(TableName.PROVINCECOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.PROVINCE;

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

// 如果授权区域是地市级

} else if (userModel.getModelLabel().equals(TableName.CITYCOMPANY)) {

if (isCompany) {

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.CITY;

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

}

List<Map<String, Object>> relationTree = ModelServiceUtils.getRelationTree(Arrays.asList(modelId), modelLabel, null, subConditions);

setDepth(relationTree);

return relationTree;

}

@Override

public List<Map<String, Object>> getAuthorizationTree() {

List<SingleModelConditionDTO> subConditions1 = new ArrayList<>(Arrays.asList(new SingleModelConditionDTO(TableName.PROVINCECOMPANY),

new SingleModelConditionDTO(TableName.CITYCOMPANY)));

List<SingleModelConditionDTO> subConditions2 = new ArrayList<>(Arrays.asList(new SingleModelConditionDTO(TableName.CITYCOMPANY)));

List<Map<String, Object>> relationTree = ModelServiceUtils.getRelationTree(Collections.singletonList(null), TableName.NETCOMPANY, null, subConditions1);

//先从网级查 如果为空 从省级查

if (CollectionUtils.isEmpty(relationTree)) {

relationTree = ModelServiceUtils.getRelationTree(Collections.singletonList(null), TableName.PROVINCECOMPANY, null, subConditions2);

}

return relationTree;

}

@Override

public List<Map<String, Object>> getTerminalTree(Boolean isCompany) {

ModelNode userModel = SessionHandler.getUserModel();

String modelLabel = userModel.getModelLabel();

Long modelId = userModel.getId();

List<SingleModelConditionDTO> subConditions = new ArrayList<>(Arrays.asList(

new SingleModelConditionDTO(TableName.SUBSTATION),

new SingleModelConditionDTO(TableName.PQTERMINAL)));

// 如果授权区域为网级

if (userModel.getModelLabel().equals(TableName.NETCOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCECOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.COUNTRY;

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCE));

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

} else if (userModel.getModelLabel().equals(TableName.PROVINCECOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.PROVINCE;

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

// 如果授权区域是地市级

} else if (userModel.getModelLabel().equals(TableName.CITYCOMPANY)) {

if (isCompany) {

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.CITY;

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

}

List<Map<String, Object>> relationTree = ModelServiceUtils.getRelationTree(Arrays.asList(modelId), modelLabel, null, subConditions);

setDepth(relationTree);

return relationTree;

}

@Override

public List<Map<String, Object>> getAdministrativeTree() {

Map<String, Object> countTree = CountTreeUtils.getCountTree(null, null);

List<Map<String, Object>> areaTreeList = CountTreeUtils.transCompany2AreaTree(countTree);

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

areaTreeList.forEach(areaTree -> {

Map<String, Object> temp = transModel2Children(areaTree);

mapList.add(temp);

});

setDepth(mapList);

return mapList;

}

@Override

public List<Map<String, Object>> getAdministrativeBookTree() {

ModelNode userModel = SessionHandler.getUserModel();

List<Map<String, Object>> mapList = null;

List<SingleModelConditionDTO> subConditions = new ArrayList<>(

Arrays.asList(new SingleModelConditionDTO(TableName.DISTRICT), new SingleModelConditionDTO(TableName.SUBSTATION), new SingleModelConditionDTO(TableName.LINE))

);

if (userModel.getModelLabel().equals(TableName.NETCOMPANY)) {

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCE));

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

mapList = ModelServiceUtils.getRelationTree(null, TableName.COUNTRY, null, subConditions);

} else if (userModel.getModelLabel().equals(TableName.PROVINCECOMPANY)) {

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

//查询所有省的children树

mapList = ModelServiceUtils.getRelationTree(null, TableName.PROVINCE, null, subConditions);

//获得所授权管理单位的所有负荷电站树

List<Map<String, Object>> countTree = getAuthorSubstation(userModel);

for (Map<String, Object> map : mapList) {

//获得省行政区域的负荷电站树

List<Map<String, Object>> substationChildren = getSubstation(userModel, map);

List<Long> ids = new ArrayList<>();

//获得省行政区域的负荷电站id

substationChildren.forEach(substation ->

ids.add(ParseDataUtil.parseLong(substation.get("id")))

);

Optional<Map<String, Object>> res = countTree.stream().filter(sub -> ids.contains(ParseDataUtil.parseLong(sub.get("id")))).findFirst();

if (res.isPresent()) {

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

newMapList.add(map);

setDepth(newMapList);

return newMapList;

}

}

} else {

mapList = ModelServiceUtils.getRelationTree(null, TableName.CITY, null, subConditions);

List<Map<String, Object>> countTree = getAuthorSubstation(userModel);

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

for (Map<String, Object> map : mapList) {

List<Map<String, Object>> substationChildren = getSubstation(userModel, map);

List<Long> ids = new ArrayList<>();

substationChildren.forEach(substation ->

ids.add(ParseDataUtil.parseLong(substation.get("id")))

);

Optional<Map<String, Object>> res = countTree.stream().filter(sub -> ids.contains(ParseDataUtil.parseLong(sub.get("id")))).findFirst();

if (res.isPresent()) {

//考虑到一个市级单位下的多个负荷电站可挂在多个市级行政区域 这里可能返回多棵树

newMapList.add(map);

}

}

mapList = newMapList;

}

setDepth(mapList);

return mapList;

}

@Override

public List<Map<String, Object>> getLevelInfo() {

ModelNode userModel = SessionHandler.getUserModel();

List<SingleModelConditionDTO> subConditions = new ArrayList<>(Arrays.asList(

new SingleModelConditionDTO(TableName.PROVINCECOMPANY),

new SingleModelConditionDTO(TableName.CITYCOMPANY),

new SingleModelConditionDTO(TableName.COUNTYCOMPANY),

new SingleModelConditionDTO(TableName.SUBSTATION)

));

String table = TableName.PROVINCECOMPANY;

ModelIdPairDTO modelIdPairDTO = new ModelIdPairDTO();

Long id = null;

if (TableName.PROVINCECOMPANY.equals(userModel.getModelLabel())) {

subConditions.get(0).setFilter(new ConditionBlockCompose(Collections.singletonList(new ConditionBlock(ColumnName.ID, ConditionBlock.OPERATOR\_EQ, userModel.getId()))));

id = userModel.getId();

} else if (TableName.CITYCOMPANY.equals(userModel.getModelLabel())) {

subConditions.get(1).setFilter(new ConditionBlockCompose(Collections.singletonList(new ConditionBlock(ColumnName.ID, ConditionBlock.OPERATOR\_EQ, userModel.getId()))));

modelIdPairDTO.setId(userModel.getId());

modelIdPairDTO.setModelLabel(userModel.getModelLabel());

} else if (TableName.COUNTYCOMPANY.equals(userModel.getModelLabel())) {

subConditions.get(2).setFilter(new ConditionBlockCompose(Collections.singletonList(new ConditionBlock(ColumnName.ID, ConditionBlock.OPERATOR\_EQ, userModel.getId()))));

modelIdPairDTO.setId(userModel.getId());

modelIdPairDTO.setModelLabel(userModel.getModelLabel());

}

QueryCondition queryCondition = ModelServiceUtils.getQueryCondition(Collections.singletonList(id), table, null, modelIdPairDTO, null, null, false, subConditions);

queryCondition.setAllowInterRelation(false);

queryCondition.setTreeReturnEnable(true);

ResultWithTotal<List<Map<String, Object>>> result = modelDataService.query(queryCondition);

ParamUtils.checkResultGeneric(result, queryCondition);

List<Map<String, Object>> data = result.getData();

setDepth(data);

return data;

}

/\*\*

\* 敏感用户节点树接口

\*

\* @param isCompany

\* @return

\*/

@Override

public List<Map<String, Object>> getSensitiveUserTree(Boolean isCompany) {

ModelNode userModel = SessionHandler.getUserModel();

String modelLabel = userModel.getModelLabel();

Long modelId = userModel.getId();

List<SingleModelConditionDTO> subConditions = new ArrayList<>();

SingleModelConditionDTO singleModelConditionDTO = new SingleModelConditionDTO(TableName.INTERFERENCESOURCE);

singleModelConditionDTO.setFilter(new ConditionBlockCompose(Arrays.asList(new ConditionBlock(ColumnName.OBJECTTYPE, ConditionBlock.OPERATOR\_IN, SENSITIVEUSERLIST))));

subConditions.add(singleModelConditionDTO);

// 如果授权区域为网级

if (userModel.getModelLabel().equals(TableName.NETCOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCECOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.COUNTRY;

subConditions.add(new SingleModelConditionDTO(TableName.PROVINCE));

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

} else if (userModel.getModelLabel().equals(TableName.PROVINCECOMPANY)) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.PROVINCE;

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

// 如果授权区域是地市级

} else if (userModel.getModelLabel().equals(TableName.CITYCOMPANY)) {

if (isCompany) {

subConditions.add(new SingleModelConditionDTO(TableName.COUNTYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.CITY;

subConditions.add(new SingleModelConditionDTO(TableName.DISTRICT));

}

}

List<Map<String, Object>> relationTree = ModelServiceUtils.getRelationTree(Arrays.asList(modelId), modelLabel, null, subConditions);

setDepth(relationTree);

return relationTree;

}

/\*\*

\* 测试仪器节点数接口 省-市-测试仪器

\*

\* @param isCompany

\* @return

\*/

@Override

public List<Map<String, Object>> getTestInstrument(Boolean isCompany) {

ModelNode userModel = SessionHandler.getUserModel();

String modelLabel = userModel.getModelLabel();

Long modelId = userModel.getId();

List<SingleModelConditionDTO> subConditions = new ArrayList<>(Arrays.asList(new SingleModelConditionDTO(TableName.TESTINSTRUMENT)));

if (TableName.PROVINCECOMPANY.equals(userModel.getModelLabel())) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.PROVINCE;

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

}

// 如果授权区域是地市级

} else if (TableName.CITYCOMPANY.equals(userModel.getModelLabel())&&!isCompany) {

//行政区域

modelId = null;

modelLabel = TableName.CITY;

}

List<Map<String, Object>> relationTree = ModelServiceUtils.getRelationTree(Arrays.asList(modelId), modelLabel, null, subConditions);

setDepth(relationTree);

return relationTree;

}

/\*\*

\* 治理设备节点数接口 省-市-负荷电站-治理设备

\*

\* @param isCompany

\* @return

\*/

@Override

public List<Map<String, Object>> getGovernanceDevice(Boolean isCompany) {

ModelNode userModel = SessionHandler.getUserModel();

String modelLabel = userModel.getModelLabel();

Long modelId = userModel.getId();

List<SingleModelConditionDTO> subConditions = new ArrayList<>(Arrays.asList(

new SingleModelConditionDTO(TableName.GOVERNANCEDEVICE),

new SingleModelConditionDTO(TableName.SUBSTATION)));

if (TableName.PROVINCECOMPANY.equals(userModel.getModelLabel())) {

if (isCompany) {

// 如果是管理单位

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

} else {

modelId = null;

modelLabel = TableName.PROVINCE;

subConditions.add(new SingleModelConditionDTO(TableName.CITY));

}

// 如果授权区域是地市级

} else if (TableName.CITYCOMPANY.equals(userModel.getModelLabel())&&!isCompany) {

//行政区域

modelId = null;

modelLabel = TableName.CITY;

}

List<Map<String, Object>> relationTree = ModelServiceUtils.getRelationTree(Arrays.asList(modelId), modelLabel, null, subConditions);

setDepth(relationTree);

return relationTree;

}

/\*\*

\* 数据分析节点树接口(区域下无监测点则过滤整个区域节点)

\* @param isCompany

\* @return

\*/

@Override

public List<Map<String, Object>> getAnalysisTreeNode(Boolean isCompany) {

List<Map<String, Object>> treeNode = getTreeNode(isCompany);

//过滤区域无监测点节点

removeAreaWithoutLine(treeNode);

//获取监测点符合性评价

Long starttime =DateUtils.getFirstDayOfMonth().getTime();//获取本月1号

Long endtime = DateUtils.getNextMonthFirst(new Date(starttime)).getTime();//获取下月1号

ComprehensiveEvaluationTableParam param = new ComprehensiveEvaluationTableParam();

param.setModelLabel(ParseDataUtil.parseString(treeNode.get(0).get(MODELLABEL)));

param.setModelId(ParseDataUtil.parseLong(treeNode.get(0).get(ColumnName.ID)));

param.setAggregationCycle(14);

param.setCompany(true);

param.setType(2);

param.setStarttime(starttime);

param.setEndtime(endtime);

Map<Long, Double> resultMap = powerQualityEvaluationIndexService.setLineElectricQualityResult(param);

getLineQualifiedvalue(treeNode,resultMap);

return treeNode;

}

private void getLineQualifiedvalue(List<Map<String, Object>> treeNode,Map<Long, Double> resultMap) {

Iterator<Map<String, Object>> it = treeNode.iterator();

while(it.hasNext()){

Map<String, Object> next = it.next();

if(next.get(MODELLABEL).equals(TableName.LINE)){

Long id = ParseDataUtil.parseLong(next.get(ColumnName.ID));

if(resultMap.containsKey(id)){

next.put("qualifiedvalue",resultMap.get(id));

}else {

next.put("qualifiedvalue",null);

}

}

if(!next.get(MODELLABEL).equals(TableName.LINE)&&next.containsKey(CHILDREN)){

getLineQualifiedvalue(ParseDataUtil.parseList(next.get(CHILDREN)),resultMap);

}

}

}

private void removeAreaWithoutLine(List<Map<String, Object>> treeNode) {

Iterator<Map<String, Object>> it = treeNode.iterator();

while(it.hasNext()){

Map<String, Object> next = it.next();

if(!isAreaHasLine(next)){

it.remove();

}

if(!next.get(MODELLABEL).equals(TableName.SUBSTATION)&&next.containsKey(CHILDREN)){

removeAreaWithoutLine(ParseDataUtil.parseList(next.get(CHILDREN)));

}

}

}

private boolean isAreaHasLine(Map<String, Object> map) {

boolean flag = true;

if (map.containsKey(MODELLABEL)) {

if (map.get(MODELLABEL).equals(TableName.PROVINCECOMPANY) || map.get(MODELLABEL).equals(TableName.CITYCOMPANY) || map.get(MODELLABEL).equals(TableName.COUNTYCOMPANY)) {

List<Map<String, Object>> subStationList = getSubstationMapList(map);

Optional<Map<String, Object>> hasChildren = subStationList.stream().filter(station -> station.containsKey(CHILDREN)).findFirst();

if(!hasChildren.isPresent()){

return false;

}

} else if (map.get(MODELLABEL).equals(TableName.SUBSTATION)&&!map.containsKey(CHILDREN)) {

flag = false;

}

}

return flag;

}

private List<Map<String, Object>> getSubstationMapList(Map<String, Object> map) {

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

if (TableName.PROVINCECOMPANY.equals(map.get(MODELLABEL))) {

if (map.containsKey(TableName.CITYCOMPANY + SUFFIX\_MODEL\_SINGLE)) {

//获得市

List<Map<String, Object>> cityTree = ParseDataUtil.parseList(map.get(CHILDREN));

cityTree.forEach(city -> {

if (city.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE) && !city.containsKey(TableName.COUNTYCOMPANY + SUFFIX\_MODEL\_SINGLE)) {

List<Map<String, Object>> citySubTree = ParseDataUtil.parseList(city.get(CHILDREN));

subStation.addAll(citySubTree);

} else {

List<Map<String, Object>> citySubTree = ParseDataUtil.parseList(city.get(CHILDREN));

//既包含负荷电站和区

citySubTree.forEach(x -> {

if (x.get(MODELLABEL).equals(TableName.SUBSTATION)) {

subStation.add(x);

} else if (x.get(MODELLABEL).equals(TableName.COUNTYCOMPANY)&&x.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE)) {

List<Map<String, Object>> disSubTree = ParseDataUtil.parseList(x.get(CHILDREN));

subStation.addAll(disSubTree);

}

});

}

});

}

}else if(TableName.CITYCOMPANY.equals(map.get(MODELLABEL))){

if (map.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE) && !map.containsKey(TableName.COUNTYCOMPANY + SUFFIX\_MODEL\_SINGLE)) {

List<Map<String, Object>> citySubTree = ParseDataUtil.parseList(map.get(CHILDREN));

subStation.addAll(citySubTree);

} else {

List<Map<String, Object>> citySubTree = ParseDataUtil.parseList(map.get(CHILDREN));

//既包含负荷电站和区

citySubTree.forEach(x -> {

if (x.get(MODELLABEL).equals(TableName.SUBSTATION)) {

subStation.add(x);

} else if (x.get(MODELLABEL).equals(TableName.COUNTYCOMPANY)&&x.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE)) {

List<Map<String, Object>> disSubTree = ParseDataUtil.parseList(x.get(CHILDREN));

subStation.addAll(disSubTree);

}

});

}

}else if(TableName.COUNTYCOMPANY.equals(map.get(MODELLABEL))&&map.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE)){

List<Map<String, Object>> disSubTree = ParseDataUtil.parseList(map.get(CHILDREN));

subStation.addAll(disSubTree);

}

return subStation;

}

private List<Map<String, Object>> getSubstation(ModelNode userModel, Map<String, Object> map) {

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

if (TableName.PROVINCECOMPANY.equals(userModel.getModelLabel())) {

if (map.containsKey(TableName.CITY + SUFFIX\_MODEL\_SINGLE)) {

//获得市

List<Map<String, Object>> cityTree = ParseDataUtil.parseList(map.get(CHILDREN));

cityTree.forEach(city -> {

if (city.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE) && !city.containsKey(TableName.DISTRICT + SUFFIX\_MODEL\_SINGLE)) {

List<Map<String, Object>> citySubTree = ParseDataUtil.parseList(city.get(CHILDREN));

subStation.addAll(citySubTree);

} else {

List<Map<String, Object>> citySubTree = ParseDataUtil.parseList(city.get(CHILDREN));

//既包含负荷电站和区

citySubTree.forEach(x -> {

if (x.get(MODELLABEL).equals(TableName.SUBSTATION)) {

subStation.add(x);

} else if (x.get(MODELLABEL).equals(TableName.DISTRICT)&&x.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE)) {

List<Map<String, Object>> disSubTree = ParseDataUtil.parseList(x.get(CHILDREN));

subStation.addAll(disSubTree);

}

});

}

});

}

} else {

if (map.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE) && !map.containsKey(TableName.DISTRICT + SUFFIX\_MODEL\_SINGLE)) {

List<Map<String, Object>> citySubTree = ParseDataUtil.parseList(map.get(CHILDREN));

subStation.addAll(citySubTree);

} else {

List<Map<String, Object>> citySubTree = ParseDataUtil.parseList(map.get(CHILDREN));

//既包含负荷电站和区

citySubTree.forEach(x -> {

if (x.get(MODELLABEL).equals(TableName.SUBSTATION)) {

subStation.add(x);

} else if (x.get(MODELLABEL).equals(TableName.DISTRICT)&&x.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL\_SINGLE)) {

List<Map<String, Object>> disSubTree = ParseDataUtil.parseList(x.get(CHILDREN));

subStation.addAll(disSubTree);

}

});

}

}

return subStation;

}

private List<Map<String, Object>> getAuthorSubstation(ModelNode userModel) {

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

List<SingleModelConditionDTO> subConditions = new ArrayList<>(Arrays.asList(new SingleModelConditionDTO(TableName.COUNTYCOMPANY), new SingleModelConditionDTO(TableName.SUBSTATION)));

if (TableName.PROVINCECOMPANY.equals(userModel.getModelLabel())) {

subConditions.add(new SingleModelConditionDTO(TableName.CITYCOMPANY));

List<Map<String, Object>> interRelationsTree = ModelServiceUtils.getInterRelations(Arrays.asList(userModel.getId()), userModel.getModelLabel(), null, null, subConditions, Boolean.TRUE);

List<Map<String, Object>> cityCompanyTree = ParseDataUtil.parseList(interRelationsTree.get(0).get(TableName.CITYCOMPANY + SUFFIX\_MODEL));

cityCompanyTree.forEach(cityCompany -> {

if (cityCompany.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL)) {

List<Map<String, Object>> citysubTree = ParseDataUtil.parseList(cityCompany.get(TableName.SUBSTATION + SUFFIX\_MODEL));

subStationTree.addAll(citysubTree);

}

if (cityCompany.containsKey(TableName.COUNTYCOMPANY + SUFFIX\_MODEL)) {

List<Map<String, Object>> countyCompanyTree = ParseDataUtil.parseList(cityCompany.get(TableName.COUNTYCOMPANY + SUFFIX\_MODEL));

countyCompanyTree.forEach(countyCompany -> {

if (countyCompany.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL)) {

List<Map<String, Object>> countysubTree = ParseDataUtil.parseList(countyCompany.get(TableName.SUBSTATION + SUFFIX\_MODEL));

subStationTree.addAll(countysubTree);

}

});

}

});

} else {

List<Map<String, Object>> interRelationsTree = ModelServiceUtils.getInterRelations(Arrays.asList(userModel.getId()), userModel.getModelLabel(), null, null, subConditions, Boolean.TRUE);

if (interRelationsTree.get(0).containsKey(TableName.SUBSTATION + SUFFIX\_MODEL)) {

List<Map<String, Object>> citysubTree = ParseDataUtil.parseList(interRelationsTree.get(0).get(TableName.SUBSTATION + SUFFIX\_MODEL));

subStationTree.addAll(citysubTree);

}

if (interRelationsTree.get(0).containsKey(TableName.COUNTYCOMPANY + SUFFIX\_MODEL)) {

List<Map<String, Object>> countyCompanyTree = ParseDataUtil.parseList(interRelationsTree.get(0).get(TableName.COUNTYCOMPANY + SUFFIX\_MODEL));

countyCompanyTree.forEach(countyCompany -> {

if (countyCompany.containsKey(TableName.SUBSTATION + SUFFIX\_MODEL)) {

List<Map<String, Object>> countysubTree = ParseDataUtil.parseList(countyCompany.get(TableName.SUBSTATION + SUFFIX\_MODEL));

subStationTree.addAll(countysubTree);

}

});

}

}

return subStationTree;

}

private Map<String, Object> transModel2Children(Map<String, Object> areaTree) {

Long id = ParseDataUtil.parseLong(areaTree.get("id"));

String modelLabel = ParseDataUtil.parseString(areaTree.get(MODELLABEL));

areaTree.put("tree\_id", modelLabel + "\_" + id);

for (String key : areaTree.keySet()) {

if (key.endsWith(SUFFIX\_MODEL) && !"pqterminal\_model".equals(key)) {

List<Map<String, Object>> subList = ParseDataUtil.parseList(areaTree.get(key));

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

subList.forEach(subTree -> {

Map<String, Object> subNewTree = transModel2Children(subTree);

newSubList.add(subNewTree);

});

areaTree.put(CHILDREN, CollectionUtils.isEmpty(newSubList) ? new ArrayList<>() : newSubList);

areaTree.remove(key);

break;

}

}

return areaTree;

}

public void setDepth(List<Map<String, Object>> relationTree) {

if (null != relationTree && CollectionUtils.isNotEmpty(relationTree)) {

for (Map<String, Object> map : relationTree) {

if (map.containsKey(MODELLABEL)) {

String modelLabel = ParseDataUtil.parseString(map.get(MODELLABEL));

if (modelLabel.equals(TableName.NETCOMPANY) || modelLabel.equals(TableName.COUNTRY)) {

map.put(DEPTH, 0);

} else if (modelLabel.equals(TableName.PROVINCECOMPANY) || modelLabel.equals(TableName.PROVINCE)) {

map.put(DEPTH, 1);

} else if (modelLabel.equals(TableName.CITYCOMPANY) || modelLabel.equals(TableName.CITY)) {

map.put(DEPTH, 2);

} else if (modelLabel.equals(TableName.COUNTYCOMPANY) || modelLabel.equals(TableName.DISTRICT)) {

map.put(DEPTH, 3);

} else if (modelLabel.equals(TableName.SUBSTATION) || modelLabel.equals(TableName.WINDPOWERSTATION) || modelLabel.equals(TableName.PHOTOVOLTAICSTATION)

|| modelLabel.equals(TableName.ELECTRICRAILWAY) || modelLabel.equals(TableName.INTERFERENCESOURCE) || modelLabel.equals(TableName.TESTINSTRUMENT)) {

map.put(DEPTH, 4);

} else if (modelLabel.equals(TableName.LINE) || modelLabel.equals(TableName.OFFLINETEST) || modelLabel.equals(TableName.PQTERMINAL)

|| modelLabel.equals(TableName.GOVERNANCEDEVICE)) {

map.put(DEPTH, 5);

}

}

if (map.containsKey(CHILDREN)) {

setDepth((List<Map<String, Object>>) map.get(CHILDREN));

}

}

}

}

}

package com.cet.pq.inventoryservice.service.impl;

import com.cet.pq.common.constant.ColumnName;

import com.cet.pq.common.constant.CommonConstant;

import com.cet.pq.common.constant.ErrorCode;

import com.cet.pq.common.constant.TableName;

import com.cet.pq.common.encode.CryptoUtils;

import com.cet.pq.common.encode.GbLimitManager;

import com.cet.pq.common.encode.sm2.SM2Utils;

import com.cet.pq.common.encode.sm2.SecurityUtils;

import com.cet.pq.common.encode.sm4.CSm4;

import com.cet.pq.common.encode.sm4.CodeUtil;

import com.cet.pq.common.enums.ObjectTypeEnum;

import com.cet.pq.common.exception.CommonManagerException;

import com.cet.pq.common.feign.ModelDataService;

import com.cet.pq.common.feign.PecNodeService;

import com.cet.pq.common.model.\*;

import com.cet.pq.common.model.pecnode.\*;

import com.cet.pq.common.model.realtime.Line;

import com.cet.pq.common.model.realtime.PecDeviceExtend;

import com.cet.pq.common.model.realtime.PqDefaultLimit;

import com.cet.pq.common.model.realtime.QuantityLimit;

import com.cet.pq.common.time.DateUtils;

import com.cet.pq.common.utils.\*;

import com.cet.pq.inventoryservice.model.PQTerminal;

import com.cet.pq.inventoryservice.model.PecDeviceExtendVo;

import com.cet.pq.inventoryservice.model.SubstationVo;

import com.cet.pq.inventoryservice.model.keycolumn.LinePqterminal;

import com.cet.pq.inventoryservice.model.line.\*;

import com.cet.pq.inventoryservice.model.log.OperationType;

import com.cet.pq.inventoryservice.model.log.SubOperationType;

import com.cet.pq.inventoryservice.model.pqterminal.AddPqterminalParams;

import com.cet.pq.inventoryservice.model.pqterminal.Pqterminal;

import com.cet.pq.inventoryservice.model.regularcheck.AddRegularcheckParams;

import com.cet.pq.inventoryservice.model.regularcheck.Regularcheck;

import com.cet.pq.inventoryservice.model.unit.CityCompany;

import com.cet.pq.inventoryservice.model.unit.CountyCompany;

import com.cet.pq.inventoryservice.model.unit.SubstationUpUnit;

import com.cet.pq.inventoryservice.model.vo.LineVo;

import com.cet.pq.inventoryservice.service.KeyColumnService;

import com.cet.pq.inventoryservice.service.MonitorRelatedService;

import com.cet.pq.pqadvanceservice.model.governance.Constant;

//import com.codingapi.txlcn.tc.annotation.LcnTransaction;

import lombok.val;

import org.apache.commons.lang3.StringUtils;

import org.apache.commons.lang3.math.NumberUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.util.CollectionUtils;

import java.nio.charset.StandardCharsets;

import java.util.\*;

import java.util.concurrent.ConcurrentHashMap;

import java.util.stream.Collectors;

/\*\*

\* @author hammer

\*/

@Service

public class MonitorServiceImpl implements MonitorRelatedService {

private static Logger logger = LoggerFactory.getLogger(MonitorServiceImpl.class);

public static final int USEFUL\_CHANNEL = 60;

@Autowired

private PecNodeService pecNodeService;

@Autowired

private ModelDataService modelDataService;

@Autowired

private KeyColumnService keyColumnService;

private final String publicKey = "04455b991880b7b0176fcf63a69b83f873c16fca8fed9a6586ce93eda6324dad29b2f79b9cc306a145f5023a875d29b4d938511258251e08bed202b28de98f4e97";

private final String privateKey = "701adf736f6312a4c75f5058cf35a780987a2b603212153f023348339e7e8ee4";

/\*\*

\* 厂站类型

\*/

private static final Long STATION\_TYPE = 269615104L;

/\*\*

\* 查询监测终端

\*

\* @param modelName

\* @param modelId

\* @param index

\* @param limit

\* @return

\*/

@Override

public PageResult<Object> queryPqterminal(String modelName, Long modelId, int index, int limit, String name, String vendor,String equipmentmodel) {

AuthUtils.checkAuth(modelId, modelName, true);

QueryCondition queryConditionLine;

if (TableName.PQTERMINAL.equals(modelName)) {

queryConditionLine = new QueryCondition(modelId, TableName.PQTERMINAL);

FlatQueryConditionDTO flatQueryConditionDTO = new FlatQueryConditionDTO();

ArrayList<ConditionBlock> filters = new ArrayList<>();

if (!StringUtils.isEmpty(name)) {

filters.add(new ConditionBlock("name", ConditionBlock.OPERATOR\_LIKE, name));

}

if (!StringUtils.isEmpty(vendor)) {

filters.add(new ConditionBlock("vendor", ConditionBlock.OPERATOR\_LIKE, vendor));

}

if (!StringUtils.isEmpty(equipmentmodel)) {

filters.add(new ConditionBlock("equipmentmodel", ConditionBlock.OPERATOR\_LIKE, equipmentmodel));

}

flatQueryConditionDTO.setFilter(new ConditionBlockCompose(filters));

flatQueryConditionDTO.setOrders(Collections.singletonList(new Order(ColumnName.UPDATE\_TIME, "desc", 1)));

if (filters.size() > 0) {

queryConditionLine.setRootCondition(flatQueryConditionDTO);

}

} else {

ModelIdPairDTO treeNode = new ModelIdPairDTO(modelId, modelName);

FlatQueryConditionDTO flatQueryConditionDTO = new FlatQueryConditionDTO();

flatQueryConditionDTO.setTreeNode(treeNode);

//flatQueryConditionDTO.setPage(new Page(index,limit));

flatQueryConditionDTO.setOrders(Collections.singletonList(new Order(ColumnName.UPDATE\_TIME, "desc", 1)));

ArrayList<ConditionBlock> filters = new ArrayList<>();

if (!StringUtils.isEmpty(name)) {

filters.add(new ConditionBlock("name", ConditionBlock.OPERATOR\_LIKE, name));

}

if (!StringUtils.isEmpty(vendor)) {

filters.add(new ConditionBlock("vendor", ConditionBlock.OPERATOR\_LIKE, vendor));

}

if (!StringUtils.isEmpty(equipmentmodel)) {

filters.add(new ConditionBlock("equipmentmodel", ConditionBlock.OPERATOR\_LIKE, equipmentmodel));

}

if (filters.size() > 0) {

flatQueryConditionDTO.setFilter(new ConditionBlockCompose(filters));

}

queryConditionLine = new QueryCondition(TableName.PQTERMINAL, flatQueryConditionDTO);

}

List<SingleModelConditionDTO> subLayerConditions = Arrays.asList(

new SingleModelConditionDTO(TableName.SUBSTATION),

new SingleModelConditionDTO(TableName.PROVINCE),

new SingleModelConditionDTO(TableName.CITY),

new SingleModelConditionDTO(TableName.DISTRICT),

new SingleModelConditionDTO(TableName.COUNTYCOMPANY),

new SingleModelConditionDTO(TableName.CITYCOMPANY),

new SingleModelConditionDTO(TableName.PROVINCECOMPANY),

new SingleModelConditionDTO(TableName.PECDEVICEEXTEND)

);

List<SingleModelConditionDTO> subLayerConditions1 = Arrays.asList(

new SingleModelConditionDTO(TableName.LINE)

);

queryConditionLine.setSubLayerConditions(subLayerConditions);

queryConditionLine.setAllowInterRelation(false);

ResultWithTotal<List<Map<String, Object>>> queryResult = modelDataService.query(queryConditionLine);

queryConditionLine.setSubLayerConditions(subLayerConditions1);

ResultWithTotal<List<Map<String, Object>>> queryResult1 = modelDataService.query(queryConditionLine);

List<Map<String, Object>> terminalList = queryResult.getData();

//对没有负荷电站的终端进行过滤

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

terminalList.forEach(terminal -> {

if (terminal.containsKey(TableName.SUBSTATION + ColumnName.\_MODEL)) {

newterminalList.add(terminal);

}

});

//结果过滤

int deleteNum = ResultFilterUtils.lineOrTerminalFilterByCompany(newterminalList, modelName, modelId);

List<Map<String, Object>> lineOrList = queryResult1.getData();

newterminalList.forEach(terminal -> {

List<Map<String, Object>> lines = lineOrList.stream().filter(lineOr -> ParseDataUtil.parseLong(terminal.get("id")).equals(

ParseDataUtil.parseLong(lineOr.get("id")))).collect(Collectors.toList());

if (!CollectionUtils.isEmpty(lines)) {

Map<String, Object> lineOr = lines.get(0);

List<Map<String, Object>> line\_model = ParseDataUtil.parseList(lineOr.get(TableName.LINE + ColumnName.\_MODEL));

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(line\_model))) {

terminal.put(TableName.LINE + ColumnName.\_MODEL, line\_model);

long isUploadNum = line\_model.stream().filter(line -> (boolean) line.get("isupload")).count();

if (isUploadNum > 0) {

terminal.put("isupload", Boolean.TRUE);

} else {

terminal.put("isupload", Boolean.FALSE);

}

} else {

terminal.put("isupload", Boolean.FALSE);

}

}

});

List<Map<String, Object>> listPaging = CommonUtils.getListPaging(newterminalList, (index / limit) + 1, limit);

PageResult result = PageResult.success(listPaging);

result.setPageNum(index / limit + 1);

result.setPageSize(limit);

result.setTotal(newterminalList.size());

return result;

}

/\*\*

\* 添加监测终端

\*

\* @param addPqterminalParams

\* @return

\*/

@Override

public void addPqterminal(AddPqterminalParams addPqterminalParams) {

//校验是否为管理员

AuthUtils.cheakAuthRole(1L);

//查重

ResultWithTotal<List<Map<String, Object>>> pqResult = ModelServiceUtils.querySingleModel(null, TableName.PQTERMINAL, null, null, null);

if (pqResult.getCode() == Result.SUCCESS\_CODE) {

List<String> names = pqResult.getData().stream().map(t -> ParseDataUtil.parseString(t.get("name"))).collect(Collectors.toList());

List<String> codes = pqResult.getData().stream().map(t -> ParseDataUtil.parseString(t.get(ColumnName.TERMINALCODE))).collect(Collectors.toList());

if (names.contains(addPqterminalParams.getPqterminal().getName())) {

throw new CommonManagerException("已存在相同的终端名称");

}

if (codes.contains(addPqterminalParams.getPqterminal().getTerminalcode())) {

throw new CommonManagerException("已存在相同的终端编号");

}

}

Long id = addPqterminalParams.getId();

String modelLabel = addPqterminalParams.getModelLabel();

Pqterminal pqterminal = addPqterminalParams.getPqterminal();

HashMap<String, Object> params = new HashMap<>(16);

params.put("id", id);

params.put("modelLabel", modelLabel);

params.put("pqterminal\_model", Arrays.asList(pqterminal));

Result<Object> result = modelDataService.write(Arrays.asList(params));

ParamUtils.checkResultGeneric(result);

//添加日志

keyColumnService.saveLog("新建终端:" + pqterminal.getName(), OperationType.PQTERMINAL\_LOG.getId(), SubOperationType.INSERT\_LOG.getId());

}

/\*\*

\* 添加定检管理

\*

\* @param addRegularcheckParams

\* @return

\*/

@Override

public void addRegularcheck(AddRegularcheckParams addRegularcheckParams) {

//校验是否为管理员

AuthUtils.cheakAuthRole(1L);

//获取终端

List<ConditionBlock> filter = new ArrayList<>();

List<Pqterminal> pqterminals = ModelServiceUtils.querySingleModel(Arrays.asList(addRegularcheckParams.getId()), TableName.PQTERMINAL, filter, null, null, Pqterminal.class);

if (pqterminals == null) {

throw new CommonManagerException("检测终端编号错误");

}

//终端编号

String terminalcode = pqterminals.get(0).getTerminalcode();

Long planchecktime = null;

if (pqterminals.get(0).getPlanchecktime() != null) {

planchecktime = Long.parseLong(pqterminals.get(0).getPlanchecktime().toString());

}

Double checkcycle=null;

if(pqterminals.get(0).getCheckcycle()!=null){

checkcycle = Double.parseDouble(pqterminals.get(0).getCheckcycle().toString())\*4;

}

if (planchecktime == null || checkcycle == null) {

return;

}

Integer checkcycleMonth = checkcycle.intValue();

List<Regularcheck> list = ModelServiceUtils.querySingleModel(null, TableName.REGULARCHECK, null, null, null, Regularcheck.class);

//获取定检状态

Map<String,Object> itemmap = getCheckstatus(terminalcode, pqterminals.get(0).getPlanchecktime(), checkcycleMonth, list);

if(Constant.STRING\_ONE.equals(itemmap.get(ColumnName.CHECKSTATUS).toString()) || Constant.STRING\_THREE.equals(itemmap.get(ColumnName.CHECKSTATUS).toString()))

{

throw new CommonManagerException("请勿重复提交");

}else if(Constant.STRING\_ZERO.equals(itemmap.get(ColumnName.REGULARENABLE).toString())){

throw new CommonManagerException("未到达定检处理操作时间");

}

Regularcheck regularcheck = addRegularcheckParams.getRegularcheck();

regularcheck.setTerminalcode(terminalcode);

//定检结果异常，默认未整改

if(regularcheck.getCheckresult().equals(1))

{

regularcheck.setRectificationstatus(0);

}

HashMap<String, Object> params = new HashMap<>(16);

params.put("id", addRegularcheckParams.getId());

params.put("modelLabel", addRegularcheckParams.getModelLabel());

params.put("regularcheck\_model", Arrays.asList(regularcheck));

Result<Object> result = modelDataService.write(Arrays.asList(params));

ParamUtils.checkResultGeneric(result);

//添加日志

keyColumnService.saveLog("新建终端定检管理:" + regularcheck.getTerminalcode(), OperationType.REGULARCHECK\_LOG.getId(), SubOperationType.INSERT\_LOG.getId());

}

/\*\*

\* 修改定检管理

\*

\* @param regularchecknew

\* @return

\*/

@Override

public void updateRegularcheck(Regularcheck regularchecknew) {

//校验是否为管理员

AuthUtils.cheakAuthRole(1L);

List<Regularcheck> regularchecks = ModelServiceUtils.querySingleModel(Arrays.asList(regularchecknew.getId()), TableName.REGULARCHECK, null,

null, null, Regularcheck.class);

if(regularchecks==null){

throw new CommonManagerException("请选择定检管理");

}

Regularcheck regularcheck = regularchecks.get(0);

//是否异常+未整改

if(!(regularcheck.getCheckresult()==1 && regularcheck.getRectificationstatus()==0))

{

throw new CommonManagerException("请检查定检状态");

}

if(regularcheck.getCompletechecktime()>regularchecknew.getActualfinishdate())

{

throw new CommonManagerException("实际完成整改日期不能小于完成定检日期");

}

//整改状态

regularcheck.setRectificationstatus(1);

//实际完成整改日期

regularcheck.setActualfinishdate(regularchecknew.getActualfinishdate());

//备注

regularcheck.setRemark(regularchecknew.getRemark());

Result<Object> result = modelDataService.write(Arrays.asList(regularcheck));

ParamUtils.checkResultGeneric(result);

//添加日志

keyColumnService.saveLog("修改终端定检管理:" + regularcheck.getTerminalcode(), OperationType.REGULARCHECK\_LOG.getId(), SubOperationType.INSERT\_LOG.getId());

}

/\*\*

\* 获取定检状态 1按期完成，2按期未完成,3超期完成,4超期未完全

\* planchecktime 计划定检时间

\* checkcycleMonth 周期（年）\*12/3 =1/3周期（月）

\* list 定检管理集合

\* \*\*/

public Map<String,Object> getCheckstatus(String terminalcode,Long planchecktime,Integer checkcycleMonth,List<Regularcheck> list){

//当前时间

Long presentDate = System.currentTimeMillis();

//开始时间

Long starttime = DateUtils.caculateMonth(planchecktime, -checkcycleMonth);

//截止时间

Long deadline = starttime;

Map<String,Object> map = new HashMap<>(16);

if(presentDate >= deadline){

while (presentDate >= deadline) {

starttime = deadline;

planchecktime = DateUtils.caculateMonth(deadline, checkcycleMonth);

deadline = DateUtils.caculateMonth(deadline, 3 \* checkcycleMonth);

}

Long finalDeadline = deadline;

Long finalStarttime = starttime;

List<Regularcheck> list2 = list.stream()

.filter(l -> l.getTerminalcode().equals(terminalcode) && l.getCompletechecktime() >= finalStarttime && l.getCompletechecktime() < finalDeadline)

.collect(Collectors.toList());

if (list2 != null && list2.size()>0) {

map.put("regularcheck\_id",list2.get(0).getId());

map.put("completechecktime",list2.get(0).getCompletechecktime());

map.put("checkresult",list2.get(0).getCheckresult());

map.put("defectcause",list2.get(0).getDefectcause());

map.put("rectificationmeasures",list2.get(0).getRectificationmeasures());

map.put("plannedcompletiondate",list2.get(0).getPlannedcompletiondate());

map.put("rectificationstatus",list2.get(0).getRectificationstatus());

map.put("actualfinishdate",list2.get(0).getActualfinishdate());

map.put("remark",list2.get(0).getRemark());

map.put(ColumnName.REGULARENABLE,0);//已完成，定检处理不可操作

//获取完成定检时间

val completechecktime = list2.get(0).getCompletechecktime();

if(completechecktime>planchecktime){

//超期完成

map.put(ColumnName.CHECKSTATUS,3);

map.put("checkstatus$text","超期完成");

}else {

//按期完成

map.put(ColumnName.CHECKSTATUS,1);

map.put("checkstatus$text","按期完成");

}

return map;

}

}

map.put("regularcheck\_id",null);

map.put("completechecktime",null);

map.put("checkresult",null);

map.put("defectcause",null);

map.put("rectificationmeasures",null);

map.put("plannedcompletiondate",null);

map.put("rectificationstatus",null);

map.put("actualfinishdate",null);

map.put("remark",null);

if(presentDate>planchecktime){

//超期未完成

map.put(ColumnName.CHECKSTATUS,4);

map.put("checkstatus$text","超期未完成");

}else {

//按期未完成

map.put(ColumnName.CHECKSTATUS,2);

map.put("checkstatus$text","按期未完成");

}

if (presentDate < starttime) {

map.put(ColumnName.REGULARENABLE, 0);

} else {

map.put(ColumnName.REGULARENABLE, 1);

}

return map;

}

/\*\*

\* 查询终端定检管理

\*

\* @param modelName

\* @param modelId

\* @param index

\* @param limit

\* @return

\*/

@Override

public PageResult<Object> queryRegularcheck(String modelName, Long modelId, int index, int limit, String name, String vendor,String equipmentmodel) {

AuthUtils.checkAuth(modelId, modelName, true);

QueryCondition queryConditionLine;

if (TableName.PQTERMINAL.equals(modelName)) {

queryConditionLine = new QueryCondition(modelId, TableName.PQTERMINAL);

FlatQueryConditionDTO flatQueryConditionDTO = new FlatQueryConditionDTO();

ArrayList<ConditionBlock> filters = new ArrayList<>();

if (!StringUtils.isEmpty(name)) {

filters.add(new ConditionBlock("name", ConditionBlock.OPERATOR\_LIKE, name));

}

if (!StringUtils.isEmpty(vendor)) {

filters.add(new ConditionBlock("vendor", ConditionBlock.OPERATOR\_LIKE, vendor));

}

if (!StringUtils.isEmpty(equipmentmodel)) {

filters.add(new ConditionBlock("equipmentmodel", ConditionBlock.OPERATOR\_LIKE, equipmentmodel));

}

flatQueryConditionDTO.setFilter(new ConditionBlockCompose(filters));

flatQueryConditionDTO.setOrders(Collections.singletonList(new Order(ColumnName.UPDATE\_TIME, "desc", 1)));

if (filters.size() > 0) {

queryConditionLine.setRootCondition(flatQueryConditionDTO);

}

} else {

ModelIdPairDTO treeNode = new ModelIdPairDTO(modelId, modelName);

FlatQueryConditionDTO flatQueryConditionDTO = new FlatQueryConditionDTO();

flatQueryConditionDTO.setTreeNode(treeNode);

//flatQueryConditionDTO.setPage(new Page(index,limit));

flatQueryConditionDTO.setOrders(Collections.singletonList(new Order(ColumnName.UPDATE\_TIME, "desc", 1)));

ArrayList<ConditionBlock> filters = new ArrayList<>();

if (!StringUtils.isEmpty(name)) {

filters.add(new ConditionBlock("name", ConditionBlock.OPERATOR\_LIKE, name));

}

if (!StringUtils.isEmpty(vendor)) {

filters.add(new ConditionBlock("vendor", ConditionBlock.OPERATOR\_LIKE, vendor));

}

if (!StringUtils.isEmpty(equipmentmodel)) {

filters.add(new ConditionBlock("equipmentmodel", ConditionBlock.OPERATOR\_LIKE, equipmentmodel));

}

if (filters.size() > 0) {

flatQueryConditionDTO.setFilter(new ConditionBlockCompose(filters));

}

queryConditionLine = new QueryCondition(TableName.PQTERMINAL, flatQueryConditionDTO);

}

List<SingleModelConditionDTO> subLayerConditions = Arrays.asList(

new SingleModelConditionDTO(TableName.SUBSTATION),

new SingleModelConditionDTO(TableName.PROVINCE),

new SingleModelConditionDTO(TableName.CITY),

new SingleModelConditionDTO(TableName.DISTRICT),

new SingleModelConditionDTO(TableName.COUNTYCOMPANY),

new SingleModelConditionDTO(TableName.CITYCOMPANY),

new SingleModelConditionDTO(TableName.PROVINCECOMPANY),

new SingleModelConditionDTO(TableName.PECDEVICEEXTEND)

);

List<SingleModelConditionDTO> subLayerConditions1 = Arrays.asList(

new SingleModelConditionDTO(TableName.LINE)

);

queryConditionLine.setSubLayerConditions(subLayerConditions);

queryConditionLine.setAllowInterRelation(false);

ResultWithTotal<List<Map<String, Object>>> queryResult = modelDataService.query(queryConditionLine);

queryConditionLine.setSubLayerConditions(subLayerConditions1);

ResultWithTotal<List<Map<String, Object>>> queryResult1 = modelDataService.query(queryConditionLine);

List<Map<String, Object>> terminalList = queryResult.getData();

//对没有负荷电站的终端进行过滤

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

terminalList.forEach(terminal -> {

if (terminal.containsKey(TableName.SUBSTATION + ColumnName.\_MODEL)) {

newterminalList.add(terminal);

}

});

//结果过滤

int deleteNum = ResultFilterUtils.lineOrTerminalFilterByCompany(newterminalList, modelName, modelId);

List<Map<String, Object>> lineOrList = queryResult1.getData();

newterminalList.forEach(terminal -> {

List<Map<String, Object>> lines = lineOrList.stream().filter(lineOr -> ParseDataUtil.parseLong(terminal.get("id")).equals(

ParseDataUtil.parseLong(lineOr.get("id")))).collect(Collectors.toList());

if (!CollectionUtils.isEmpty(lines)) {

Map<String, Object> lineOr = lines.get(0);

List<Map<String, Object>> line\_model = ParseDataUtil.parseList(lineOr.get(TableName.LINE + ColumnName.\_MODEL));

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(line\_model))) {

terminal.put(TableName.LINE + ColumnName.\_MODEL, line\_model);

long isUploadNum = line\_model.stream().filter(line -> (boolean) line.get("isupload")).count();

if (isUploadNum > 0) {

terminal.put("isupload", Boolean.TRUE);

} else {

terminal.put("isupload", Boolean.FALSE);

}

} else {

terminal.put("isupload", Boolean.FALSE);

}

}

});

List<Regularcheck> regularchecks = ModelServiceUtils.querySingleModel(null, TableName.REGULARCHECK, null, null, null, Regularcheck.class);

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

newterminalList.forEach(terminal -> {

if(terminal.get(ColumnName.TERMINALCODE)==null){

return;

}

String terminalcode = terminal.get(ColumnName.TERMINALCODE).toString();

Long planchecktime = null;

if (terminal.get(ColumnName.PLANCHECKTIME) != null) {

planchecktime = Long.parseLong(terminal.get(ColumnName.PLANCHECKTIME).toString());

}

Double checkcycle=null;

if(terminal.get(ColumnName.CHECKCYCLE)!=null){

checkcycle = Double.parseDouble(terminal.get(ColumnName.CHECKCYCLE).toString())\*4;

}

if (planchecktime == null || checkcycle == null) {

return;

}

Integer checkcycleMonth = checkcycle.intValue();

//获取定检状态

Map<String,Object> itemmap = getCheckstatus(terminalcode, planchecktime, checkcycleMonth, regularchecks);

itemmap.put("id",terminal.get("id"));

itemmap.put("modelLabel","pqterminal");

itemmap.put(ColumnName.TERMINALCODE,terminal.get(ColumnName.TERMINALCODE));

itemmap.put("name",terminal.get("name"));

itemmap.put("master",terminal.get("master"));

itemmap.put("equipmentmodel",terminal.get("equipmentmodel"));

itemmap.put(ColumnName.PLANCHECKTIME,terminal.get(ColumnName.PLANCHECKTIME));

itemmap.put("substation\_name",null);

itemmap.put("citycompany\_name",null);

List<Map<String,Object>> substation\_list = ParseDataUtil.parseList(terminal.get("substation\_model"));

if (substation\_list != null) {

Map<String,Object> substation\_model = substation\_list.get(0);

itemmap.put("substation\_name",substation\_model.get("name"));

List<Map<String,Object>> citycompany\_list = ParseDataUtil.parseList(substation\_model.get("citycompany\_model"));

if(citycompany\_list!=null){

Map<String,Object> citycompany\_model = citycompany\_list.get(0);

itemmap.put("citycompany\_name",citycompany\_model.get("name"));

}

}

regularcheck.add(itemmap);

});

regularcheck.sort(new Comparator<Map<String, Object>>() {

@Override

public int compare(Map<String, Object> o1, Map<String, Object> o2) {

// 对比是否超期

if ((ParseDataUtil.parseInteger(o1.get(ColumnName.CHECKSTATUS)) <= CommonConstant.NUMBER\_TWO && ParseDataUtil.parseInteger(o2.get(ColumnName.CHECKSTATUS)) <= CommonConstant.NUMBER\_TWO)

|| (ParseDataUtil.parseInteger(o1.get(ColumnName.CHECKSTATUS)) > CommonConstant.NUMBER\_TWO && ParseDataUtil.parseInteger(o2.get(ColumnName.CHECKSTATUS)) > CommonConstant.NUMBER\_TWO)) {

//o1是否异常未整改

Boolean r1= ParseDataUtil.parseInteger(o1.get("checkresult"))==1 && ParseDataUtil.parseInteger(o1.get("rectificationstatus"))==0;

//o2是否异常未整改

Boolean r2= ParseDataUtil.parseInteger(o2.get("checkresult"))==1 && ParseDataUtil.parseInteger(o2.get("rectificationstatus"))==0;

//对比是否未整改

//o1和o2都异常未整改，或都非异常未整改

if ((r1 && r2) || (!r1 && !r2)) {

//对比计划定检时间

if(ParseDataUtil.parseLong(o1.get(ColumnName.PLANCHECKTIME))<ParseDataUtil.parseLong(o2.get(ColumnName.PLANCHECKTIME)))

{

return -1;

}else{

return 1;

}

} else if (r1) {

return -1;//o1异常未整改

} else {

return 1;//o2异常未整改

}

}else {

// 对比是否超期

if (ParseDataUtil.parseInteger(o1.get(ColumnName.CHECKSTATUS)) >= ParseDataUtil.parseInteger(o2.get(ColumnName.CHECKSTATUS))) {

return -1;

}

else{

return 1;

}

}

}

});

List<Map<String, Object>> listPaging = CommonUtils.getListPaging(regularcheck, (index / limit) + 1, limit);

PageResult result = PageResult.success(listPaging);

result.setPageNum(index / limit + 1);

result.setPageSize(limit);

result.setTotal(regularcheck.size());

return result;

}

/\*\*

\* 查询监测点

\*

\* @param modelName

\* @param modelId

\* @param index

\* @param limit

\* @return

\*/

@Override

public PageResult<Object> queryMonitor(String modelName, Long modelId, int index, int limit, String name) {

AuthUtils.checkAuth(modelId, modelName, true);

QueryCondition queryConditionLine;

if (TableName.LINE.equals(modelName)) {

queryConditionLine = new QueryCondition(modelId, TableName.LINE);

if (!StringUtils.isEmpty(name)) {

FlatQueryConditionDTO flatQueryConditionDTO = new FlatQueryConditionDTO();

flatQueryConditionDTO.setFilter(new ConditionBlockCompose(Arrays.asList(new ConditionBlock("name", ConditionBlock.OPERATOR\_LIKE, name))));

queryConditionLine.setRootCondition(flatQueryConditionDTO);

}

} else {

ModelIdPairDTO treeNode = new ModelIdPairDTO(modelId, modelName);

FlatQueryConditionDTO flatQueryConditionDTO = new FlatQueryConditionDTO();

flatQueryConditionDTO.setTreeNode(treeNode);

//flatQueryConditionDTO.setPage(new Page(index,limit));

if (!StringUtils.isEmpty(name)) {

flatQueryConditionDTO.setFilter(new ConditionBlockCompose(Arrays.asList(new ConditionBlock("name", ConditionBlock.OPERATOR\_LIKE, name))));

}

queryConditionLine = new QueryCondition(TableName.LINE, flatQueryConditionDTO);

}

List<SingleModelConditionDTO> subLayerConditions = Arrays.asList(

new SingleModelConditionDTO(TableName.SUBSTATION),

new SingleModelConditionDTO(TableName.PROVINCE),

new SingleModelConditionDTO(TableName.CITY),

new SingleModelConditionDTO(TableName.DISTRICT),

new SingleModelConditionDTO(TableName.COUNTYCOMPANY),

new SingleModelConditionDTO(TableName.CITYCOMPANY),

new SingleModelConditionDTO(TableName.PROVINCECOMPANY)

);

List<SingleModelConditionDTO> subLayerConditions1 = Arrays.asList(

new SingleModelConditionDTO(TableName.PECDEVICEEXTEND),

new SingleModelConditionDTO(TableName.ELECTRICRAILWAY),

new SingleModelConditionDTO(TableName.PHOTOVOLTAICSTATION),

new SingleModelConditionDTO(TableName.WINDPOWERSTATION),

new SingleModelConditionDTO(TableName.INTERFERENCESOURCE),

new SingleModelConditionDTO(TableName.DEVICENODE),

new SingleModelConditionDTO(TableName.PQTERMINAL)

);

queryConditionLine.setSubLayerConditions(subLayerConditions);

queryConditionLine.setAllowInterRelation(false);

//模型服务最多支持查10层相关的数据，分开查询

ResultWithTotal<List<Map<String, Object>>> queryResult = modelDataService.query(queryConditionLine);

queryConditionLine.setSubLayerConditions(subLayerConditions1);

ResultWithTotal<List<Map<String, Object>>> queryResult1 = modelDataService.query(queryConditionLine);

List<Map<String, Object>> lineList = queryResult.getData();

//监测点过滤

int deleteNum = ResultFilterUtils.lineOrTerminalFilterByCompany(lineList, modelName, modelId);

//过滤无负荷电站无市级的监测点

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

lineList.forEach(terminal -> {

if (terminal.containsKey(TableName.SUBSTATION + ColumnName.\_MODEL)) {

newlineList.add(terminal);

}

});

List<Map<String, Object>> lineOrList = queryResult1.getData();

newlineList.forEach(line -> {

List<Map<String, Object>> collect = lineOrList.stream().filter(line1 -> ParseDataUtil.parseLong(line.get("id")).equals(

ParseDataUtil.parseLong(line1.get("id")))).collect(Collectors.toList());

if (!CollectionUtils.isEmpty(collect)) {

Map<String, Object> lineOr = collect.get(0);

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(lineOr.get(TableName.PECDEVICEEXTEND + ColumnName.\_MODEL)))) {

line.put(TableName.PECDEVICEEXTEND + ColumnName.\_MODEL, ParseDataUtil.parseList(lineOr.get(TableName.PECDEVICEEXTEND + ColumnName.\_MODEL)));

}

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(lineOr.get(TableName.ELECTRICRAILWAY + ColumnName.\_MODEL)))) {

line.put(TableName.ELECTRICRAILWAY + ColumnName.\_MODEL, ParseDataUtil.parseList(lineOr.get(TableName.ELECTRICRAILWAY + ColumnName.\_MODEL)));

}

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(lineOr.get(TableName.PHOTOVOLTAICSTATION + ColumnName.\_MODEL)))) {

line.put(TableName.PHOTOVOLTAICSTATION + ColumnName.\_MODEL, ParseDataUtil.parseList(lineOr.get(TableName.PHOTOVOLTAICSTATION + ColumnName.\_MODEL)));

}

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(lineOr.get(TableName.WINDPOWERSTATION + ColumnName.\_MODEL)))) {

line.put(TableName.WINDPOWERSTATION + ColumnName.\_MODEL, ParseDataUtil.parseList(lineOr.get(TableName.WINDPOWERSTATION + ColumnName.\_MODEL)));

}

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(lineOr.get(TableName.INTERFERENCESOURCE + ColumnName.\_MODEL)))) {

line.put(TableName.INTERFERENCESOURCE + ColumnName.\_MODEL, ParseDataUtil.parseList(lineOr.get(TableName.INTERFERENCESOURCE + ColumnName.\_MODEL)));

}

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(lineOr.get(TableName.DEVICENODE + ColumnName.\_MODEL)))) {

line.put(TableName.DEVICENODE + ColumnName.\_MODEL, ParseDataUtil.parseList(lineOr.get(TableName.DEVICENODE + ColumnName.\_MODEL)));

}

if (!CollectionUtils.isEmpty(ParseDataUtil.parseList(lineOr.get(TableName.PQTERMINAL + ColumnName.\_MODEL)))) {

line.put(TableName.PQTERMINAL + ColumnName.\_MODEL, ParseDataUtil.parseList(lineOr.get(TableName.PQTERMINAL + ColumnName.\_MODEL)));

}

}

});

List<Map<String, Object>> listPaging = CommonUtils.getListPaging(newlineList, (index / limit) + 1, limit);

PageResult result = PageResult.success(listPaging);

result.setPageNum(index / limit + 1);

result.setPageSize(limit);

result.setTotal(newlineList.size());

return result;

}

/\*\*

\* 修改监测点

\*

\* @param updateLineParams

\* @return

\*/

@Override

@Transactional(rollbackFor = Exception.class,transactionManager = "matterhornTransactionManager")

// @LcnTransaction

public void updateMonitor(UpdateLineParams updateLineParams) {

//校验是否为管理员

AuthUtils.cheakAuthRole(1L);

Line line = updateLineParams.getLine();

List<Map<String, Object>> relations = ModelServiceUtils.getRelations(null, TableName.LINE, null, null,

Collections.singletonList(new SingleModelConditionDTO(TableName.PECDEVICEEXTEND)));

List<Line> lineList = JsonTransferUtils.transferList(relations, Line.class);

List<Long> lineIds = lineList.stream().map(Line::getId).collect(Collectors.toList());

List<String> lineCodes = lineList.stream().filter(l -> !l.getId().equals(line.getId())).map(Line::getCode).collect(Collectors.toList());

if (!lineIds.contains(line.getId())) {

throw new CommonManagerException("该监测点已删除");

}

if (lineCodes.contains(line.getCode())) {

throw new CommonManagerException("监测点编码已存在");

}

line.setObject\_label(getObjectLabel(line.getObjecttype()));

Result<Object> result = modelDataService.write(Collections.singletonList(line));

ParamUtils.checkResultGeneric(result);

//更改关联关系

lineList = lineList.stream().filter(l -> l.getId().equals(line.getId())).collect(Collectors.toList());

Line oldLine = lineList.get(0);

String oldObjectLabel = oldLine.getObject\_label();

Long oldObjectId = oldLine.getObject\_id();

if (oldObjectId != null && (!oldObjectId.equals(line.getObject\_id()) || !oldObjectLabel.equals(line.getObject\_label()))) {

//更新

Map<String, Object> moveOutParams = new HashMap<>(16);

moveOutParams.put("moveout\_flag", Boolean.TRUE);

moveOutParams.put("id", oldObjectId);

moveOutParams.put("modelLabel", oldObjectLabel);

Map<String, Object> moveInParams = new HashMap<>(16);

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

params.add(moveOutParams);

if (line.getObject\_id() != null && StringUtils.isNotEmpty(line.getObject\_label())) {

moveInParams.put("id", line.getObject\_id());

moveInParams.put("modelLabel", getObjectLabel(line.getObjecttype()));

params.add(moveInParams);

}

Map<String, Object> writeParams = new HashMap<>(16);

writeParams.put("id", line.getId());

writeParams.put("modelLabel", TableName.LINE);

writeParams.put("children", params);

result = modelDataService.write(Collections.singletonList(writeParams));

ParamUtils.checkResultGeneric(result);

} else {

//新增

//保存监测点和干扰源的关系

if (oldObjectId == null && line.getObject\_id() != null && StringUtils.isNotEmpty(line.getObject\_label())) {

Map<String, Object> interferenceMap = new ConcurrentHashMap<>();

Map<String, Object> interference = new ConcurrentHashMap<>();

interferenceMap.put("id", line.getId());

interferenceMap.put("modelLabel", TableName.LINE);

interference.put("id", line.getObject\_id());

interference.put("modelLabel", line.getObject\_label());

interferenceMap.put(line.getObject\_label() + ColumnName.\_MODEL, Collections.singletonList(interference));

result = modelDataService.write(Collections.singleton(interferenceMap));

ParamUtils.checkResultGeneric(result);

}

}

//修改监测点电压等级、接线方式或容量相关参数，国标限值需要重算

Boolean isReca = false;

if (!oldLine.getVoltclass().equals(line.getVoltclass()) || !line.getUpvoltdeviation().equals(oldLine.getUpvoltdeviation()) ||

!line.getLowvoltdeviation().equals(oldLine.getLowvoltdeviation()) || !line.getShortcircuitcapacity().equals(oldLine.getShortcircuitcapacity()) ||

!line.getUserprotocolcapacity().equals(oldLine.getUserprotocolcapacity()) || !line.getSupplyequipmentcapacity().equals(oldLine.getSupplyequipmentcapacity())) {

isReca = true;

}

if (!CollectionUtils.isEmpty(line.getPecdeviceextend\_model()) && !CollectionUtils.isEmpty(oldLine.getPecdeviceextend\_model())) {

PecDeviceExtend pecDeviceExtend = line.getPecdeviceextend\_model().get(0);

PecDeviceExtend oldPecDeviceExtend = oldLine.getPecdeviceextend\_model().get(0);

if (!pecDeviceExtend.getWiredtype().equals(oldPecDeviceExtend.getWiredtype())) {

isReca = true;

}

}

// 保存国标限值

if (isReca && updateLineParams.getRecalculationParam() != null) {

ResultWithTotal<List<Map<String, Object>>> listResultWithTotal;

if (!line.getPecdeviceextend\_model().get(0).getWiredtype().equals(oldLine.getPecdeviceextend\_model().get(0).getWiredtype())) {

listResultWithTotal = gbLimitByWiredType(line.getPecdeviceextend\_model().get(0).getWiredtype());

//修改接线方式，需要删除之前的限值，重新保存新的限值

ResultWithTotal<List<Map<String, Object>>> quantityLimitResult = ModelServiceUtils.querySingleModel(null, TableName.QUANTITYLIMIT,

Collections.singletonList(new ConditionBlock(ColumnName.LINE\_ID, ConditionBlock.OPERATOR\_EQ, line.getId())), null, null);

if (quantityLimitResult.getCode() == Result.SUCCESS\_CODE && !CollectionUtils.isEmpty(quantityLimitResult.getData())) {

List<Map<String, Object>> data = quantityLimitResult.getData();

if (!CollectionUtils.isEmpty(data)) {

List<Long> quantityLimitIds = data.stream().map(t -> ParseDataUtil.parseLong(t.get("id"))).collect(Collectors.toList());

Result<Object> deleteQuantityLimit = modelDataService.delete(TableName.QUANTITYLIMIT, quantityLimitIds);

if (deleteQuantityLimit.getCode() != Result.SUCCESS\_CODE) {

throw new CommonManagerException("删除物理量限值信息失败");

}

}

}

} else {

listResultWithTotal = gbLimitByNodeId(oldLine.getId(), 0);

}

List<QuantityLimit> limits = JsonTransferUtils.transferList(listResultWithTotal.getData(), QuantityLimit.class);

List<PqReportParam> pqReportParams = new ArrayList<>();

limits.forEach(quantityLimit -> {

PqReportParam pqReportParam = new PqReportParam(quantityLimit.getDataid(), quantityLimit.getUplimit(),

quantityLimit.getLowlimit(), quantityLimit.getDataname());

pqReportParams.add(pqReportParam);

});

RecalculationParam recalculationParam = updateLineParams.getRecalculationParam();

recalculationParam.setReportParams(pqReportParams);

Result<List<PqReportParam>> listResult = recalculationGBLimit(recalculationParam);

ParamUtils.checkResultGeneric(listResult);

recalculationParam.setReportParams(listResult.getData());

Result<Object> saveGbLimit = saveGbLimit(recalculationParam);

ParamUtils.checkResultGeneric(saveGbLimit);

}

//更新pqnodestatus状态

updateNodeStatus(line);

//添加日志

String pecDeviceUpdateLog = StringUtils.EMPTY;

if (!CollectionUtils.isEmpty(lineList.get(0).getPecdeviceextend\_model()) && !CollectionUtils.isEmpty(updateLineParams.getLine().getPecdeviceextend\_model())) {

pecDeviceUpdateLog = CommonUtils.getUpdateLog(lineList.get(0).getPecdeviceextend\_model().get(0), updateLineParams.getLine().getPecdeviceextend\_model().get(0));

}

keyColumnService.saveLog("修改监测点:" + ParseDataUtil.parseString(updateLineParams.getLine().getName()) +

CommonUtils.getUpdateLog(lineList.get(0), updateLineParams.getLine()) + pecDeviceUpdateLog,

OperationType.LINE\_LOG.getId(), SubOperationType.MODIFY\_LOG.getId());

}

/\*\*

\* @Description: 更新pqnodestatus状态

\*\*/

private void updateNodeStatus(Line line) {

List<List<Object>> writeDatas = new ArrayList<>();

List<List<Object>> filterWriteDatas = new ArrayList<>();

FlatWriteData flatWriteData = new FlatWriteData();

flatWriteData.setModelLabel(TableName.PQNODESTATUS);

List<String> writeProperty = Collections.singletonList(ColumnName.MONITORSTATUS);

flatWriteData.setWriteProperty(writeProperty);

flatWriteData.setWriteMethod("update");

flatWriteData.setFilter(Collections.singletonList(ColumnName.PQMONITOR\_ID));

writeDatas.add(Collections.singletonList(line.getStatus()));

filterWriteDatas.add(Collections.singletonList(line.getId()));

flatWriteData.setFilterData(filterWriteDatas);

flatWriteData.setWriteData(writeDatas);

ModelServiceUtils.flatWrite(flatWriteData);

}

@Transactional(rollbackFor = Exception.class,transactionManager = "matterhornTransactionManager")

// @LcnTransaction

@Override

public Result<Object> addMonitor(AddLineParams addLineParams) {

//校验是否为管理员

AuthUtils.cheakAuthRole(1L);

//校验监测点名称和编码

LineVo oldLine = addLineParams.getLine();

List<LineVo> lineVos = ModelServiceUtils.querySingleModel(null, TableName.LINE, null, null, null, LineVo.class);

List<String> lineCodes = lineVos.stream().map(LineVo::getCode).collect(Collectors.toList());

if (lineCodes.contains(oldLine.getCode())) {

throw new CommonManagerException("监测点编码已存在");

}

// 1.判断厂站是否存在，不存在直接抛异常 nodeType = 269615104 - OK

Result<SystemNode> systemNodeResult = pecNodeService.getSpecifiedNode(STATION\_TYPE,

addLineParams.getStationID());

// 检查返回结果异常

checkResult(systemNodeResult);

// 检查是否存在，不存在抛异常

checkStationHave(systemNodeResult.getData());

// 创建设备返回设备ID

Long deviceId = getDeviceId(addLineParams);

if (0 == deviceId) {

throw new CommonManagerException("保存设备异常");

}

// 3,获取创建成功的设备id,设备创建成功了，写入PecDeviceExtend和监测点模型

Line line = saveLineAndPecDeviceExtend(addLineParams, deviceId);

// 4，写入监测点-设备管联的模型measuredby，写入限制模型QuantityLimit

Result<Object> result = saveMeasuredByAndQuantityLimit(line, deviceId, addLineParams.getSubstation\_id(),

addLineParams.getPqterminal\_id(), addLineParams.getRecalculationParam(), addLineParams.getPecDeviceExtend().getWiredtype());

checkResult(result);

//添加日志

keyColumnService.saveLog("新建监测点:" + line.getName(), OperationType.LINE\_LOG.getId(), SubOperationType.INSERT\_LOG.getId());

return result;

}

@Override

public Result<List<PqReportParam>> recalculationGBLimit(RecalculationParam recalculationParam) {

Result<List<PqReportParam>> result = new Result<>();

List<PqReportParam> reportParams = recalculationParam.getReportParams();

GbRecalcPara gbRecalcPara = recalculationParam.getGbRecalcPara();

List<PqReportParam> pqReportParamList = GbLimitManager.recalcGBLimit(reportParams, gbRecalcPara);

result.setData(pqReportParamList);

return result;

}

private void checkResultWithTotal(ResultWithTotal resultWithTotal) {

if (Result.SUCCESS\_CODE != resultWithTotal.getCode()) {

throw new CommonManagerException(resultWithTotal.getMsg());

}

}

@Override

public ResultWithTotal<List<Map<String, Object>>> gbLimitByWiredType(int wiredType) {

QueryCondition condition = new QueryCondition();

condition.setRootLabel("pqdefaultlimit");

FlatQueryConditionDTO rootCondition = new FlatQueryConditionDTO();

ConditionBlockCompose filter = new ConditionBlockCompose();

List<ConditionBlock> expressions = new ArrayList<>();

ConditionBlock conditionBlock = new ConditionBlock("wiredtype", ConditionBlock.OPERATOR\_EQ, wiredType);

expressions.add(conditionBlock);

filter.setExpressions(expressions);

rootCondition.setFilter(filter);

condition.setRootCondition(rootCondition);

return modelDataService.query(condition);

}

@Override

public ResultWithTotal<List<Map<String, Object>>> gbLimitByNodeId(Long nodeId, int wiredType) {

QueryCondition condition = new QueryCondition();

condition.setRootLabel("quantitylimit");

FlatQueryConditionDTO rootCondition = new FlatQueryConditionDTO();

ConditionBlockCompose filter = new ConditionBlockCompose();

List<ConditionBlock> expressions = new ArrayList<>();

ConditionBlock conditionBlock = new ConditionBlock("line\_id", ConditionBlock.OPERATOR\_EQ, nodeId);

expressions.add(conditionBlock);

filter.setExpressions(expressions);

rootCondition.setFilter(filter);

condition.setRootCondition(rootCondition);

ResultWithTotal<List<Map<String, Object>>> query = modelDataService.query(condition);

if (CollectionUtils.isEmpty(query.getData())) {

return gbLimitByWiredType(wiredType);

}

return query;

}

@Override

public Result<Object> saveGbLimit(RecalculationParam recalculationParam) {

GbRecalcPara gbRecalcPara = recalculationParam.getGbRecalcPara();

List<QuantityLimit> quantityLimits = getQuantityLimits(gbRecalcPara.getLineId(), recalculationParam, gbRecalcPara.getWiredtype());

if (CollectionUtils.isEmpty(quantityLimits)) {

return Result.success();

}

FlatWriteData flatWriteData = new FlatWriteData();

flatWriteData.setModelLabel(TableName.QUANTITYLIMIT);

List<List<Object>> writeDatas = new ArrayList<>();

List<List<Object>> filterWriteDatas = new ArrayList<>();

if (quantityLimits.get(0).getId() == null) {

List<Long> ids = ModelServiceUtils.querySingleModel(null, TableName.QUANTITYLIMIT, Collections.singletonList(

new ConditionBlock(ColumnName.LINE\_ID, ConditionBlock.OPERATOR\_EQ, quantityLimits.get(0).getLineId())), null, null, QuantityLimit.class)

.stream().map(QuantityLimit::getId).collect(Collectors.toList());

if (!CollectionUtils.isEmpty(ids)) {

Result<Object> deleteResult = ModelServiceUtils.delete(ids, TableName.QUANTITYLIMIT);

if (!deleteResult.getCode().equals(Result.SUCCESS\_CODE)) {

throw new CommonManagerException("更新异常");

}

}

List<String> writeProperty = Arrays.asList("uplimit", "quantitymaptemplate\_id", "lowlimit", "line\_id", "dataid", "dataname", "updatetime");

flatWriteData.setWriteProperty(writeProperty);

flatWriteData.setWriteMethod("insert");

quantityLimits.forEach(quantityLimit -> {

List<Object> writeData = new ArrayList<>();

writeData.add(quantityLimit.getUplimit());

writeData.add(quantityLimit.getQuantitymaptemplateId());

writeData.add(quantityLimit.getLowlimit());

writeData.add(quantityLimit.getLineId());

writeData.add(quantityLimit.getDataid());

writeData.add(quantityLimit.getDataname());

writeData.add(quantityLimit.getUpdatetime());

writeDatas.add(writeData);

});

} else {

List<String> writeProperty = Arrays.asList("uplimit", "quantitymaptemplate\_id", "lowlimit", "line\_id", "dataid", "dataname", "updatetime");

flatWriteData.setWriteProperty(writeProperty);

flatWriteData.setWriteMethod("update");

flatWriteData.setFilter(Collections.singletonList("id"));

quantityLimits.forEach(quantityLimit -> {

List<Object> writeData = new ArrayList<>();

List<Object> filterWriteData = new ArrayList<>();

filterWriteData.add(quantityLimit.getId());

writeData.add(quantityLimit.getUplimit());

writeData.add(quantityLimit.getQuantitymaptemplateId());

writeData.add(quantityLimit.getLowlimit());

writeData.add(quantityLimit.getLineId());

writeData.add(quantityLimit.getDataid());

writeData.add(quantityLimit.getDataname());

writeData.add(quantityLimit.getUpdatetime());

writeDatas.add(writeData);

filterWriteDatas.add(filterWriteData);

});

}

flatWriteData.setFilterData(filterWriteDatas);

flatWriteData.setWriteData(writeDatas);

//更新监测点上下偏差

saveVoltdeviation(recalculationParam.getGbRecalcPara());

return modelDataService.flatWrite(flatWriteData);

}

/\*\*

\* @Description: 保存电压上下偏差

\*\*/

private void saveVoltdeviation(GbRecalcPara gbRecalcPara) {

//查询监测点是否存在

List<Line> lines = ModelServiceUtils.querySingleModel(Collections.singletonList(gbRecalcPara.getLineId()), TableName.LINE, null, null, null, Line.class);

if (CollectionUtils.isEmpty(lines)) {

logger.error("限值重算，该监测点不存在.");

return;

}

Line line = lines.get(0);

if (null == line.getLowvoltdeviation()) {

line.setLowvoltdeviation(0D);

}

if (null == line.getUpvoltdeviation()) {

line.setUpvoltdeviation(0D);

}

//判断是否修改属性值,修改则保存

if (!line.getVoltclass().equals(gbRecalcPara.getRatedVolt()) || !line.getUpvoltdeviation().equals(gbRecalcPara.getUpvoltdeviation()) ||

!line.getLowvoltdeviation().equals(gbRecalcPara.getLowvoltdeviation()) || !line.getShortcircuitcapacity().equals(gbRecalcPara.getShortCircuitCapacity()) ||

!line.getUserprotocolcapacity().equals(gbRecalcPara.getUserProtocolCapacity()) || !line.getSupplyequipmentcapacity().equals(gbRecalcPara.getSupplyEquipmentCapacity())) {

FlatWriteData flatWriteData = new FlatWriteData();

flatWriteData.setModelLabel(TableName.LINE);

List<String> writeProperty = Arrays.asList("upvoltdeviation", "lowvoltdeviation", "voltclass", "userprotocolcapacity",

"supplyequipmentcapacity", "shortcircuitcapacity");

flatWriteData.setWriteProperty(writeProperty);

flatWriteData.setWriteMethod("update");

List<Object> writeData = new ArrayList<>();

writeData.add(gbRecalcPara.getUpvoltdeviation());

writeData.add(gbRecalcPara.getLowvoltdeviation());

writeData.add(gbRecalcPara.getRatedVolt());

writeData.add(gbRecalcPara.getUserProtocolCapacity());

writeData.add(gbRecalcPara.getSupplyEquipmentCapacity());

writeData.add(gbRecalcPara.getShortCircuitCapacity());

List<List<Object>> writeDatas = new ArrayList<>();

writeDatas.add(writeData);

List<Object> filterWriteData = new ArrayList<>();

filterWriteData.add(gbRecalcPara.getLineId());

List<List<Object>> filterWriteDatas = new ArrayList<>();

filterWriteDatas.add(filterWriteData);

flatWriteData.setFilter(Collections.singletonList("id"));

flatWriteData.setFilterData(filterWriteDatas);

flatWriteData.setWriteData(writeDatas);

modelDataService.flatWrite(flatWriteData);

}

}

@Transactional(rollbackFor = Exception.class,transactionManager = "matterhornTransactionManager")

// @LcnTransaction

@Override

public Result<Object> deleteMonitoringInventory(Long id) {

//校验是否为管理员

AuthUtils.cheakAuthRole(1L);

//查询设备信息

List<Map<String, Object>> relations = ModelServiceUtils.getRelations(Collections.singletonList(id), TableName.LINE, null, null,

Collections.singletonList(new SingleModelConditionDTO(TableName.PECDEVICEEXTEND)));

List<Long> deviceId = new ArrayList<>();

List<Long> pecdeviceextendId = new ArrayList<>();

if (CollectionUtils.isEmpty(relations)) {

throw new CommonManagerException("该监测点已经被删除");

} else {

List<Map<String, Object>> devices = ParseDataUtil.parseList(relations.get(0).get("pecdeviceextend\_model"));

if (!CollectionUtils.isEmpty(devices)) {

pecdeviceextendId = devices.stream().map(t -> ParseDataUtil.parseLong(t.get("id"))).collect(Collectors.toList());

deviceId = devices.stream().map(t -> ParseDataUtil.parseLong(t.get("deviceid"))).collect(Collectors.toList());

}

}

//删除监测点

Result<Object> deleteLine = modelDataService.delete(TableName.LINE, Collections.singletonList(id));

if (deleteLine.getCode() != Result.SUCCESS\_CODE) {

throw new CommonManagerException("删除监测点失败");

}

//删除监测点关联的设备

Result<Object> deletePecdeviceextend = modelDataService.delete(TableName.PECDEVICEEXTEND, pecdeviceextendId);

if (deletePecdeviceextend.getCode() != Result.SUCCESS\_CODE) {

throw new CommonManagerException("删除设备扩展信息失败");

}

//删除监测点关联的measureby

List<ConditionBlock> filters = Arrays.asList(

new ConditionBlock(ColumnName.MONITOREDID, ConditionBlock.OPERATOR\_EQ, id),

new ConditionBlock(ColumnName.MONITOREDLABEL, ConditionBlock.OPERATOR\_EQ, TableName.LINE));

ResultWithTotal<List<Map<String, Object>>> measurebyResult = ModelServiceUtils.querySingleModel(null, TableName.MEASURED\_BY, filters, null, null);

if (measurebyResult.getCode() == Result.SUCCESS\_CODE && !CollectionUtils.isEmpty(measurebyResult.getData())) {

List<Map<String, Object>> data = measurebyResult.getData();

if (!CollectionUtils.isEmpty(data)) {

List<Long> measurebyIds = data.stream().map(t -> ParseDataUtil.parseLong(t.get("id"))).collect(Collectors.toList());

Result<Object> deleteMeasureby = modelDataService.delete(TableName.MEASURED\_BY, measurebyIds);

if (deleteMeasureby.getCode() != Result.SUCCESS\_CODE) {

throw new CommonManagerException("删除被监测关系信息失败");

}

}

}

//删除监测点关联的QuantityLimit

ResultWithTotal<List<Map<String, Object>>> quantityLimitResult = ModelServiceUtils.querySingleModel(null, TableName.QUANTITYLIMIT,

Collections.singletonList(new ConditionBlock(ColumnName.LINE\_ID, ConditionBlock.OPERATOR\_EQ, id)), null, null);

if (quantityLimitResult.getCode() == Result.SUCCESS\_CODE && !CollectionUtils.isEmpty(quantityLimitResult.getData())) {

List<Map<String, Object>> data = quantityLimitResult.getData();

if (!CollectionUtils.isEmpty(data)) {

List<Long> quantityLimitIds = data.stream().map(t -> ParseDataUtil.parseLong(t.get("id"))).collect(Collectors.toList());

Result<Object> deleteQuantityLimit = modelDataService.delete(TableName.QUANTITYLIMIT, quantityLimitIds);

if (deleteQuantityLimit.getCode() != Result.SUCCESS\_CODE) {

throw new CommonManagerException("删除物理量限值信息失败");

}

}

}

//删除检测点关联的设备信息

Result<Object> deleteDevice = pecNodeService.deleteDevice(deviceId);

if (deleteDevice.getCode() != Result.SUCCESS\_CODE) {

throw new CommonManagerException("删除设备失败");

}

//添加日志

keyColumnService.saveLog("删除监测点:" + ParseDataUtil.parseString(relations.get(0).get("name")),

OperationType.LINE\_LOG.getId(), SubOperationType.DELETE\_LOG.getId());

return Result.success();

}

@Override

public Result<Object> updateMonitoringTerminal(PQTerminal pqTerminal) {

//校验是否为管理员

AuthUtils.cheakAuthRole(1L);

String originalName = StringUtils.EMPTY;

List<PQTerminal> originalPqTerminal = new ArrayList<>();

if (ParseDataUtil.parseLong(pqTerminal.getId()) != 0L) {

//更新

ResultWithTotal<List<Map<String, Object>>> originalModel = ModelServiceUtils.querySingleModel(Collections.singletonList(pqTerminal.getId()),

TableName.PQTERMINAL, null, null, null);

if (CollectionUtils.isEmpty(originalModel.getData())) {

throw new CommonManagerException("该管理单位已被删除.");

}

originalPqTerminal = JsonTransferUtils.transferList(originalModel.getData(), PQTerminal.class);

originalName = ParseDataUtil.parseString(originalModel.getData().get(0).get("name"));

}

//查重

if (!originalName.equals(pqTerminal.getName())) {

ResultWithTotal<List<Map<String, Object>>> result = ModelServiceUtils.querySingleModel(null, TableName.PQTERMINAL, null, null, null);

if (result.getCode() == Result.SUCCESS\_CODE) {

List<String> names = result.getData().stream().map(t -> ParseDataUtil.parseString(t.get("name"))).collect(Collectors.toList());

if (names.contains(pqTerminal.getName())) {

throw new CommonManagerException("已存在相同的终端名称.");

}

}

}

//更新终端设备

Result<Object> pqTerminalResult = modelDataService.write(Collections.singleton(pqTerminal));

if (pqTerminalResult.getCode() != Result.SUCCESS\_CODE) {

throw new CommonManagerException("更新终端设备失败");

}

//查询关联的设备信息

List<Map<String, Object>> relations = ModelServiceUtils.getRelations(Collections.singletonList(pqTerminal.getId()), TableName.PQTERMINAL, null, null,

Collections.singletonList(new SingleModelConditionDTO(TableName.LINE)));

if (CollectionUtils.isEmpty(relations)) {

throw new CommonManagerException("该终端设备已经被删除");

} else {

List<List<Map<String, Object>>> lines = relations.stream().map(t -> (List<Map<String, Object>>) t.get("line\_model")).collect(Collectors.toList());

if (!CollectionUtils.isEmpty(lines) && !CollectionUtils.isEmpty(lines.get(0))) {

List<Long> lineIds = lines.get(0).stream().map(t -> ParseDataUtil.parseLong(t.get(ColumnName.ID))).collect(Collectors.toList());

List<Map<String, Object>> relations1 = ModelServiceUtils.getRelations(lineIds, TableName.LINE, null, null,

Collections.singletonList(new SingleModelConditionDTO(TableName.PECDEVICEEXTEND)));

if (CollectionUtils.isEmpty(relations1)) {

return Result.success();

}

List<List<Map<String, Object>>> devices = relations1.stream().map(t -> (List<Map<String, Object>>) t.get("pecdeviceextend\_model")).collect(Collectors.toList());

List<Long> deviceIds = new ArrayList<>();

devices.forEach(t -> {

deviceIds.add(ParseDataUtil.parseLong(t.get(0).get("deviceid")));

});

// 批量更新设备节点信息

List<DeviceNode> deviceNodes = new ArrayList<>();

deviceIds.forEach(t -> {

DeviceNode deviceNode = new DeviceNode();

CommunicationParam communicationParam = new CommunicationParam(pqTerminal.getPort(), pqTerminal.getIp());

deviceNode.setCommunicationParam(communicationParam);

deviceNode.setDeviceId(t);

deviceNodes.add(deviceNode);

});

if (deviceNodes != null)

{

Result<Object> deviceResult = pecNodeService.updateDevices(deviceNodes);

if (deviceResult.getCode() != Result.SUCCESS\_CODE) {

throw new CommonManagerException("更新设备节点信息失败");

}

}

}

}

//添加日志

keyColumnService.saveLog("修改终端:" + pqTerminal.getName() + CommonUtils.getUpdateLog(originalPqTerminal.get(0),

pqTerminal), OperationType.PQTERMINAL\_LOG.getId(), SubOperationType.MODIFY\_LOG.getId());

//更新关联设备的ip和端口

return Result.success();

}

@Override

public Result<Object> deleteMonitoringTerminal(Long id) {

//校验是否为管理员

AuthUtils.cheakAuthRole(1L);

//查看终端和监测点的关联

List<Map<String, Object>> relations = ModelServiceUtils.getRelations(Collections.singletonList(id), TableName.PQTERMINAL, null, null,

Collections.singletonList(new SingleModelConditionDTO(TableName.LINE)));

if (!CollectionUtils.isEmpty(relations.get(0))) {

List<Map<String, Object>> lines = (List<Map<String, Object>>) relations.get(0).get(TableName.LINE + ColumnName.\_MODEL);

if (!CollectionUtils.isEmpty(lines)) {

throw new CommonManagerException("该终端关联监测点，无法删除");

}

}

Result<Object> result = modelDataService.delete(TableName.PQTERMINAL, Collections.singletonList(id));

//添加日志

if (Result.SUCCESS\_CODE == result.getCode()) {

keyColumnService.saveLog("删除终端:" + ParseDataUtil.parseString(relations.get(0).get("name")),

OperationType.PQTERMINAL\_LOG.getId(), SubOperationType.DELETE\_LOG.getId());

}

return result;

}

@Override

public Result<Object> batchRecalculationGBLimit(List<Long> ids) {

List<Long> successLineIdList = new ArrayList<>();

List<Long> failLineIdList = new ArrayList<>();

Map<String, List<Long>> map = new HashMap<>(2);

map.put("success", successLineIdList);

map.put("fail", failLineIdList);

List<Line> lineList;

if (CollectionUtils.isEmpty(ids)) {

lineList = JsonTransferUtils.transferList(ModelServiceUtils.getInterRelations(null, TableName.LINE, null, null,

Collections.singletonList(new SingleModelConditionDTO(TableName.PECDEVICEEXTEND)), false), Line.class);

} else {

lineList = JsonTransferUtils.transferList(ModelServiceUtils.getInterRelations(ids, TableName.LINE, null, null,

Collections.singletonList(new SingleModelConditionDTO(TableName.PECDEVICEEXTEND)), false), Line.class);

}

ids = lineList.stream().map(Line::getId).collect(Collectors.toList());

lineList.forEach(line -> {

logger.info("start recalculate line {}.", line.getName());

try {

if (!CollectionUtils.isEmpty(line.getPecdeviceextend\_model())) {

Integer wiredtype = line.getPecdeviceextend\_model().get(0).getWiredtype();

GbRecalcPara gbRecalcPara = new GbRecalcPara(line.getVoltclass(), line.getShortcircuitcapacity(), line.getUserprotocolcapacity(),

line.getSupplyequipmentcapacity(), line.getId(), wiredtype, line.getUpvoltdeviation(), line.getLowvoltdeviation());

ResultWithTotal<List<Map<String, Object>>> result = gbLimitByWiredType(wiredtype);

if (!CollectionUtils.isEmpty(result.getData())) {

batchSaveRecalculation(successLineIdList, line, gbRecalcPara, result);

} else {

logger.info("fail save line {}, error: {}", line.getName(), "获取默认限值失败, " + result.getMsg());

}

} else {

logger.info("fail save line {}, error: {}", line.getName(), "监测点扩展设备为空.");

}

} catch (Exception e) {

logger.error("fail save line {}, error: {}", line.getName(), e);

}

});

ids.forEach(id -> {

if (!successLineIdList.contains(id)) {

failLineIdList.add(id);

}

});

return Result.success(map);

}

/\*\*

\* @param successLineIdList

\* @param line

\* @param gbRecalcPara

\* @param result

\* @Description: 批量保存限制重算值

\* @Author: gongtong

\*\*/

private void batchSaveRecalculation(List<Long> successLineIdList, Line line, GbRecalcPara gbRecalcPara, ResultWithTotal<List<Map<String, Object>>> result) {

List<PqReportParam> pqReportParams = transMapToPqReportParam(result.getData());

RecalculationParam recalculationParam = new RecalculationParam(pqReportParams, gbRecalcPara);

Result<List<PqReportParam>> listResult = recalculationGBLimit(recalculationParam);

if (listResult.getCode() == Result.SUCCESS\_CODE) {

List<PqReportParam> reportParams = listResult.getData();

recalculationParam.setReportParams(reportParams);

Result<Object> saveGbLimitResult = saveGbLimit(recalculationParam);

if (saveGbLimitResult.getCode() == Result.SUCCESS\_CODE) {

successLineIdList.add(line.getId());

logger.info("success save line {}.", line.getName());

} else {

logger.info("fail save line {}, error: {}", line.getName(), "保存限值失败, " + saveGbLimitResult.getMsg());

}

} else {

logger.info("fail recalculate line {}, error: {}", line.getName(), "限值重算失败, " + listResult.getMsg());

}

}

private List<PqReportParam> transMapToPqReportParam(List<Map<String, Object>> mapList) {

List<PqReportParam> reportParams = new ArrayList<>();

mapList.forEach(map -> {

Long dataID = ParseDataUtil.parseLong(map.get("dataid"));

Double limitHigh = ParseDataUtil.parseDouble(map.get("limithigh"));

Double limitLow = ParseDataUtil.parseDouble(map.get("limitlow"));

String dataName = ParseDataUtil.parseString(map.get("dataname"));

PqReportParam pPqReportParam = new PqReportParam(dataID, limitHigh, limitLow, dataName);

reportParams.add(pPqReportParam);

});

return reportParams;

}

private Channel channelParamStr(String drvName, AddLineParams addLineParams) {

// 构造BasicParam入参

BasicParam basicParam = new BasicParam();

// nodename

basicParam.setDrvName(addLineParams.getChannelBean().getDrvName());

//并行

basicParam.setPortalMode("PARALLEL");

//ETHERNET-以太网

basicParam.setPortType("ETHERNET");

// 获取上级单位

String unitName = getUpUnitName(addLineParams.getSubstation\_id());

String nodeName = unitName + "\_" + drvName;

// 获取通道名称

Channel channel = getChannelByDrvName(nodeName, addLineParams.getStationID());

if(null == channel){

throw new CommonManagerException("参数异常");

}

if (null != channel.getNodeId()) {

return channel;

}

basicParam.setNodeName(channel.getNodeName());

ChannelParam channelParam = new ChannelParam();

channelParam.setBasicParam(basicParam);

// 构造CommunicationParam入参

CommunicationParam communicationParam = new CommunicationParam();

QueryCondition queryCommCondition = new QueryCondition();

queryCommCondition.setRootLabel("pqterminal");

queryCommCondition.setRootId(addLineParams.getPqterminal\_id());

List<Map<String, Object>> commData = modelDataService.query(queryCommCondition).getData();

if (!CollectionUtils.isEmpty(commData) && null != commData.get(0).get(ColumnName.PORT)) {

communicationParam.setPort(commData.get(0).get(ColumnName.PORT).hashCode());

}

if (!CollectionUtils.isEmpty(commData) && null != commData.get(0).get(ColumnName.IP)) {

communicationParam.setRemoteAddr(commData.get(0).get(ColumnName.IP).toString());

}

channelParam.setCommunicationParam(communicationParam);

// 当没找空闲通道时创建通道

List<ChannelParam> channelParamList = new ArrayList<>();

channelParamList.add(channelParam);

Result<Object> channelResult = pecNodeService.addChannels(addLineParams.getStationID(), channelParamList);

checkResult(channelResult);

List<Integer> channelIdList = JsonTransferUtils

.transferJsonString(JsonTransferUtils.toJsonString(channelResult.getData()), Integer.class);

if (CollectionUtils.isEmpty(channelIdList)) {

throw new CommonManagerException("保存通道异常");

}

addLineParams.getChannelBean().setChannelID(Long.valueOf(channelIdList.get(0)));

channel.setNodeId(Long.valueOf(channelIdList.get(0)));

return channel;

}

@SuppressWarnings("deprecation")

private Channel getChannelByDrvName(String nodeName, Long stationID) {

// 判断新建的通道名是否已存在，如不存在直接新建，如存在则自增1以“\_N”的方式重新构造通道名称新建

Result<List<StationAndChannel>> channelResult = pecNodeService.getStations();

checkResult(channelResult);

List<StationAndChannel> stationAndChannels = channelResult.getData();

if (CollectionUtils.isEmpty(stationAndChannels)) {

throw new CommonManagerException("查找厂站异常");

}

Optional<StationAndChannel> stationAndChannel = stationAndChannels.stream().filter(x -> x.getStation().getNodeId().compareTo(stationID) == 0).findFirst();

if (!stationAndChannel.isPresent()) {

return null;

}

List<Channel> channels = stationAndChannel.get().getChannels();

Channel channelGoal = new Channel();

if (CollectionUtils.isEmpty(channels)) {

channelGoal.setNodeName(nodeName);

return channelGoal;

}

String channelName = nodeName;

channels = channels.stream().filter(x -> x.getNodeName().contains(channelName)).collect(Collectors.toList());

if (CollectionUtils.isEmpty(channels)) {

channelGoal.setNodeName(nodeName);

return channelGoal;

}

Map<Integer, Channel> channelMap = new ConcurrentHashMap<>();

for (Channel channel : channels) {

String[] names = channel.getNodeName().split("\_");

if (names.length > 2) {

if (NumberUtils.isNumber(names[2])) {

channelMap.put(Integer.valueOf(names[2]), channel);

}

} else {

channelMap.put(0, channel);

}

}

int i;

for (i = 0; i < channelMap.size(); i++) {

Channel channel = channelMap.get(i);

// 查看通道下的设备是否有空闲，设备节点数量<60。无空闲先创建通道，有空闲就在此通道下创建设备

Result<List<Device>> deviceResult = pecNodeService.getDevicesByChannelId(channel.getNodeId());

checkResult(deviceResult);

List<Device> devices = deviceResult.getData();

if (CollectionUtils.isEmpty(devices) || devices.size() < USEFUL\_CHANNEL) {

return channel;

}

}

if (0 != i) {

nodeName = nodeName + "\_" + i;

}

channelGoal.setNodeName(nodeName);

return channelGoal;

}

private String getUpUnitName(Long substationId) {

QueryCondition queryBasicCondition = new QueryCondition();

queryBasicCondition.setRootLabel("substation");

// 查询负荷电站上级单位的名称

queryBasicCondition.setRootId(substationId);

List<SingleModelConditionDTO> subLayerConditions = new ArrayList<>();

SingleModelConditionDTO unitModelCondition = new SingleModelConditionDTO();

unitModelCondition.setModelLabel("countycompany");

subLayerConditions.add(unitModelCondition);

SingleModelConditionDTO unitUpModelCondition = new SingleModelConditionDTO();

unitUpModelCondition.setModelLabel("citycompany");

subLayerConditions.add(unitUpModelCondition);

queryBasicCondition.setSubLayerConditions(subLayerConditions);

ResultWithTotal<List<Map<String, Object>>> resultWithTotal = modelDataService.query(queryBasicCondition);

if (Result.SUCCESS\_CODE != resultWithTotal.getCode()) {

throw new CommonManagerException(resultWithTotal.getMsg());

}

List<SubstationUpUnit> list = JsonTransferUtils.transferList(resultWithTotal.getData(), SubstationUpUnit.class);

if (CollectionUtils.isEmpty(list)) {

throw new CommonManagerException("无法找到该负荷电站");

}

SubstationUpUnit substationUpUnit = list.get(0);

List<CountyCompany> countycompany\_model = substationUpUnit.getCountycompany\_model();

List<CityCompany> citycompany\_model = substationUpUnit.getCitycompany\_model();

if (!CollectionUtils.isEmpty(countycompany\_model)) {

return countycompany\_model.get(0).getName();

}

if (!CollectionUtils.isEmpty(citycompany\_model)) {

return citycompany\_model.get(0).getName();

}

throw new CommonManagerException("该负荷电站没有关联上级单位");

}

private void setDeviceNode(List<DeviceNode> deviceNodes, AddLineParams addLineParams) {

DeviceNode deviceNode = new DeviceNode();

// 要确定通道是否已经创建（如果新建了通道，此处需要在新建通道后将通道id设置到addLineParams中）

if (null != addLineParams.getChannelBean().getChannelID()) {

deviceNode.setChannelId(addLineParams.getChannelBean().getChannelID().intValue());

}

if (null != addLineParams.getPecDeviceExtend().getBranchindex()) {

deviceNode.setCircuitId(addLineParams.getPecDeviceExtend().getBranchindex());

}

deviceNode.setComParam(null);

// 通信ID

if (null != addLineParams.getDeviceNode().getCommunicationId()) {

deviceNode.setCommunicationId(addLineParams.getDeviceNode().getCommunicationId().intValue());

}

CommunicationParam communicationParam = new CommunicationParam();

List<PQTerminal> pqtemernalList = getPortAndIp(addLineParams.getPqterminal\_id());

if (!CollectionUtils.isEmpty(pqtemernalList)) {

communicationParam.setRemoteAddr(pqtemernalList.get(0).getIp());

communicationParam.setPort(pqtemernalList.get(0).getPort());

}

deviceNode.setCommunicationParam(communicationParam);

CustomInfParamList ptCustomInfo = new CustomInfParamList();

ptCustomInfo.setSection("PT\_CT");

ptCustomInfo.setKey("PT");

ptCustomInfo.setValue(addLineParams.getPecDeviceExtend().getPtratio());

List<CustomInfParamList> customInfParamList = new ArrayList<>();

customInfParamList.add(ptCustomInfo);

CustomInfParamList ctCustomInfo = new CustomInfParamList();

ctCustomInfo.setSection("PT\_CT");

ctCustomInfo.setKey("CT");

ctCustomInfo.setValue(addLineParams.getPecDeviceExtend().getCtratio());

customInfParamList.add(ctCustomInfo);

CustomInfParamList voltCustomInfo = new CustomInfParamList();

voltCustomInfo.setSection("PT\_CT");

voltCustomInfo.setKey("VoltClass");

//voltagelevel

Result<List<IdTextPair>> enumResult = modelDataService.getEnumrationByModel("voltagelevel");

if (Result.SUCCESS\_CODE != enumResult.getCode()) {

throw new CommonManagerException("获取voltagelevel异常");

}

List<IdTextPair> voltagelevels = enumResult.getData();

Optional<IdTextPair> idTextPair = voltagelevels.stream().filter(x ->

x.getId().compareTo(addLineParams.getLine().getVoltclass()) == 0).findFirst();

if (idTextPair.isPresent()) {

voltCustomInfo.setValue(idTextPair.get().getText());

}

customInfParamList.add(voltCustomInfo);

writeInf(addLineParams, customInfParamList);

deviceNode.setCustomInfParamList(customInfParamList);

DeviceAccessParam deviceAccessParam = new DeviceAccessParam();

deviceAccessParam.setDeviceTypeName(addLineParams.getDeviceNode().getDeviceTypeName());

deviceAccessParam.setDataLogMapName(addLineParams.getDeviceNode().getDataLogMapName());

deviceAccessParam.setInfData(null);

if (null == addLineParams.getPecDeviceExtend().getBranchindex()) {

addLineParams.getPecDeviceExtend().setBranchindex(1);

}

deviceAccessParam.setInfFileName(addLineParams.getDeviceNode().getDataLogMapName() + "\_" + addLineParams.getPecDeviceExtend().getBranchindex() + ".inf");

deviceAccessParam.setParaSetMapName(null);

deviceAccessParam.setSnfData(null);

deviceAccessParam.setSnfFileName(addLineParams.getDeviceNode().getDataLogMapName() + ".snf");

deviceAccessParam.setWaveMapName(null);

deviceNode.setDeviceAccessParam(deviceAccessParam);

deviceNode.setNodeName(addLineParams.getLine().getName());

//ETHERNET-以太网

deviceNode.setPortType("ETHERNET");

deviceNodes.add(deviceNode);

}

private List<PQTerminal> getPortAndIp(Long pqterminalId) {

QueryCondition queryCondition = new QueryCondition();

queryCondition.setRootLabel("pqterminal");

FlatQueryConditionDTO rootCondition = new FlatQueryConditionDTO();

ConditionBlockCompose filter = new ConditionBlockCompose();

List<ConditionBlock> expressions = new ArrayList<>();

ConditionBlock idConditon = new ConditionBlock("id", ConditionBlock.OPERATOR\_EQ, pqterminalId);

expressions.add(idConditon);

filter.setExpressions(expressions);

rootCondition.setFilter(filter);

queryCondition.setRootCondition(rootCondition);

return JsonTransferUtils.transferList(modelDataService.query(queryCondition).getData(), PQTerminal.class);

}

@Override

public Long getDeviceId(AddLineParams addLineParams) {

// 非0时直接使用此通道ID，指定为0时表示需要自动选择通道或者自动创建通道

List<DeviceNode> deviceNode = new ArrayList<>();

// 2.查找空闲通道，如果有空闲通道则创建设备。如果没有空闲通道则创建通道再创建设备

ChannelBean channelBean = addLineParams.getChannelBean();

Long channelID = channelBean.getChannelID();

String drvName = channelBean.getDrvName();

if (null == channelID) {

channelID = 0L;

}

if (0 == channelID.longValue()) {

// 创建通道，在新通道下创建设备

return saveDeviceChannelUnusefulCondition(addLineParams, drvName, deviceNode);

} else {

// 查看通道下的设备是否有空闲，设备节点数量<60。无空闲先创建通道，有空闲就在此通道下创建设备

Result<List<Device>> deviceResult = pecNodeService.getDevicesByChannelId(channelID);

checkResult(deviceResult);

List<Device> devices = deviceResult.getData();

if (CollectionUtils.isEmpty(devices) || devices.size() < USEFUL\_CHANNEL) {

// 拼接deviceNode入参

setDeviceNode(deviceNode, addLineParams);

// 创建设备

Result<Object> saveDeviceResult = pecNodeService.addDevices(deviceNode);

checkResult(saveDeviceResult);

List<Long> idList = JsonTransferUtils

.transferJsonString(JsonTransferUtils.toJsonString(saveDeviceResult.getData()), Long.class);

if (CollectionUtils.isEmpty(idList)) {

throw new CommonManagerException("保存设备异常");

}

//写入inf文件

//writeInf(addLineParams);

return idList.get(0);

}

if (!CollectionUtils.isEmpty(devices) && devices.size() >= USEFUL\_CHANNEL) {

// 创建通道，在新通道下创建设备

return saveDeviceChannelUnusefulCondition(addLineParams, drvName, deviceNode);

}

}

return 0L;

}

public void writeInf(AddLineParams addLineParams, List<CustomInfParamList> customInfParamList) {

DeviceVo deviceNode = addLineParams.getDeviceNode();

//终端识别码

String sbm = deviceNode.getSbm();

String sbmMd5 = deviceNode.getSbmMd5();

//终端秘钥

String my = deviceNode.getMy();

String myMd5 = deviceNode.getMyMd5();

//md5校验

Boolean sbmMd5Check = checkMd5(sbm, sbmMd5);

Boolean myMd5Check = checkMd5(my, myMd5);

if (!(sbmMd5Check && myMd5Check)) {

throw new CommonManagerException("md5加密错误");

}

//sm2解密

try {

byte[] temp = SecurityUtils.hexStringToBytes(privateKey);

String sbmSm2 = new String(SM2Utils.decrypt(temp, SecurityUtils.hexStringToBytes(sbm)), StandardCharsets.UTF\_8);

String mySm2 = new String(SM2Utils.decrypt(temp, SecurityUtils.hexStringToBytes(my)), StandardCharsets.UTF\_8);

if (StringUtils.isEmpty(sbmSm2) && StringUtils.isEmpty(mySm2)) {

return;

}

//sm4加密

//csm4

byte[] gm4Buffer = CodeUtil.StringToSmBuffer(sbmSm2);

byte[] cetKeyBuffer = CodeUtil.StringToSmBuffer(generateSM4Key());

byte[] miyaoBuffer = CodeUtil.StringToSmBuffer(mySm2);

byte[] jmGm4Buffer = CSm4.EncriptB(miyaoBuffer, gm4Buffer);

byte[] jmMiyaoBuffer = CSm4.EncriptB(cetKeyBuffer, miyaoBuffer);

String strGM4 = CodeUtil.BufferToHexString(jmGm4Buffer);

String strGMKey = CodeUtil.BufferToHexString(jmMiyaoBuffer);

logger.info("GM4: {}，GM4Key： {}", strGM4, strGMKey);

CustomInfParamList gm4CustomInfParam = new CustomInfParamList();

gm4CustomInfParam.setKey("GM4");

gm4CustomInfParam.setSection("Main");

gm4CustomInfParam.setValue(strGM4);

customInfParamList.add(gm4CustomInfParam);

CustomInfParamList gM4KeyCustomInfParam = new CustomInfParamList();

gM4KeyCustomInfParam.setKey("GM4Key");

gM4KeyCustomInfParam.setSection("Main");

gM4KeyCustomInfParam.setValue(strGMKey);

customInfParamList.add(gM4KeyCustomInfParam);

} catch (Exception e) {

throw new CommonManagerException("密钥解密失败，请重新保存！", e);

}

}

//产生对称秘钥

public String generateSM4Key() {

return "cet-electric.com";

}

private boolean checkMd5(String sbm, String sbmMd5) {

return sbmMd5.equals(CryptoUtils.encodeMD5(sbm));

}

private Result<Object> saveMeasuredByAndQuantityLimit(Line line, Long deviceId, Long id, Long pqterminalId, RecalculationParam recalculationParam, Integer wiredtype) {

SubstationVo substationVo = new SubstationVo();

substationVo.setId(id);

List<Line> line\_model = new ArrayList<>();

line\_model.add(line);

substationVo.setLine\_model(line\_model);

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

list.add(substationVo);

MeasuredbyVo measuredbyVo = new MeasuredbyVo();

measuredbyVo.setMonitoredid(line.getId());

measuredbyVo.setMonitoredlabel(TableName.LINE);

measuredbyVo.setMeasuredby(deviceId);

list.add(measuredbyVo);

List<QuantityLimit> quantityLimits = getQuantityLimits(line.getId(), recalculationParam, wiredtype);

if (!CollectionUtils.isEmpty(quantityLimits)) {

list.addAll(quantityLimits);

}

LinePqterminal linePqterminal = new LinePqterminal();

linePqterminal.setId(line.getId());

List<com.cet.pq.inventoryservice.model.keycolumn.Pqterminal> pqterminalModel = new ArrayList<>();

com.cet.pq.inventoryservice.model.keycolumn.Pqterminal pqterminal = new com.cet.pq.inventoryservice.model.keycolumn.Pqterminal();

pqterminal.setId(pqterminalId);

pqterminalModel.add(pqterminal);

linePqterminal.setPqterminal\_model(pqterminalModel);

list.add(linePqterminal);

//保存监测点和干扰源的关系

if (line.getObject\_id() != null && StringUtils.isNotEmpty(line.getObject\_label())) {

Map<String, Object> interferenceMap = new ConcurrentHashMap<>();

Map<String, Object> interference = new ConcurrentHashMap<>();

interferenceMap.put("id", line.getId());

interferenceMap.put("modelLabel", TableName.LINE);

interference.put("id", line.getObject\_id());

interference.put("modelLabel", line.getObject\_label());

interferenceMap.put(line.getObject\_label() + ColumnName.\_MODEL, Collections.singletonList(interference));

modelDataService.write(Collections.singleton(interferenceMap));

}

return modelDataService.write(list);

}

private List<QuantityLimit> getQuantityLimits(Long id, RecalculationParam recalculationParam, Integer wiredtype) {

//如果修改了接线方式，则取recalculationParam的限值

List<Map<String, Object>> relations = ModelServiceUtils.getRelations(Collections.singletonList(id), TableName.LINE, null, null,

Collections.singletonList(new SingleModelConditionDTO(TableName.PECDEVICEEXTEND)));

List<Line> lineList = JsonTransferUtils.transferList(relations, Line.class);

Boolean isWiredTypeChanged = false;

if (!CollectionUtils.isEmpty(lineList) && !CollectionUtils.isEmpty(lineList.get(0).getPecdeviceextend\_model()) && wiredtype != null) {

isWiredTypeChanged = !wiredtype.equals(lineList.get(0).getPecdeviceextend\_model().get(0).getWiredtype());

}

List<QuantityLimit> limits = null;

if (!isWiredTypeChanged && CollectionUtils.isEmpty(recalculationParam.getReportParams())) {

ResultWithTotal<List<Map<String, Object>>> resultWithTotal = gbLimitByNodeId(id, 0);

checkResultWithTotal(resultWithTotal);

limits = JsonTransferUtils.transferList(resultWithTotal.getData(), QuantityLimit.class);

}

List<PqReportParam> reportParams = recalculationParam.getReportParams();

if (CollectionUtils.isEmpty(limits)) {

if (CollectionUtils.isEmpty(reportParams)) {

ResultWithTotal<List<Map<String, Object>>> limitResult = gbLimitByWiredType(wiredtype);

checkResultWithTotal(limitResult);

List<PqDefaultLimit> pqDefaultLimits = JsonTransferUtils.transferList(limitResult.getData(), PqDefaultLimit.class);

setRecalculationParam(pqDefaultLimits, recalculationParam);

//重算国标限值

Result<List<PqReportParam>> recalculationResult = recalculationGBLimit(recalculationParam);

checkResult(recalculationResult);

if (!CollectionUtils.isEmpty(recalculationResult.getData())) {

return recalculationLimitCoverQuantityLimit(recalculationResult.getData(), id);

}

return null;

} else {

return reportParamsConverQuantityLimit(reportParams, limits, id);

}

} else {

return reportParamsConverQuantityLimit(reportParams, limits, id);

}

}

private List<QuantityLimit> recalculationLimitCoverQuantityLimit(List<PqReportParam> pqReportParams, Long id) {

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

pqReportParams.forEach(pqReportParam -> {

QuantityLimit quantityLimit = new QuantityLimit();

quantityLimit.setUplimit(ParseDataUtil.parseDouble(pqReportParam.getLimitHigh()).equals(Double.NaN) ? null : pqReportParam.getLimitHigh());

quantityLimit.setLowlimit(ParseDataUtil.parseDouble(pqReportParam.getLimitLow()).equals(Double.NaN) ? null : pqReportParam.getLimitLow());

quantityLimit.setLineId(id);

quantityLimit.setDataid(pqReportParam.getDataID());

quantityLimit.setDataname(pqReportParam.getDataName());

list.add(quantityLimit);

});

return list;

}

private void setRecalculationParam(List<PqDefaultLimit> pqDefaultLimits, RecalculationParam recalculationParam) {

List<PqReportParam> pqReportParams = new ArrayList<>();

pqDefaultLimits.forEach(pqDefaultLimit -> {

PqReportParam pqReportParam = new PqReportParam();

pqReportParam.setLimitHigh(pqDefaultLimit.getLimithigh());

pqReportParam.setLimitLow(pqDefaultLimit.getLimitlow());

pqReportParam.setDataID(pqDefaultLimit.getDataid());

pqReportParam.setDataName(pqDefaultLimit.getDataname());

pqReportParams.add(pqReportParam);

});

recalculationParam.setReportParams(pqReportParams);

}

private List<QuantityLimit> reportParamsConverQuantityLimit(List<PqReportParam> reportParams, List<QuantityLimit> list, Long id) {

List<QuantityLimit> limits = new ArrayList<>();

if (CollectionUtils.isEmpty(list)) {

reportParamToQuantityLimit(reportParams, id, limits);

} else {

quantityLimitSetLimitLowAndUp(reportParams, list);

return list;

}

return limits;

}

private void quantityLimitSetLimitLowAndUp(List<PqReportParam> reportParams, List<QuantityLimit> list) {

for (QuantityLimit quantityLimit : list) {

Optional<PqReportParam> pqReportParam = reportParams.stream().filter(

x -> x.getDataID().compareTo(quantityLimit.getDataid()) == 0).findFirst();

if (pqReportParam.isPresent()) {

quantityLimit.setLowlimit(pqReportParam.get().getLimitLow());

quantityLimit.setUplimit(pqReportParam.get().getLimitHigh());

}

}

for (PqReportParam pqReportParam : reportParams) {

Optional<QuantityLimit> param = list.stream().filter(

x -> pqReportParam.getDataID().compareTo(x.getDataid()) == 0).findFirst();

if (!param.isPresent()) {

QuantityLimit quantityLimit = new QuantityLimit();

quantityLimit.setUplimit(pqReportParam.getLimitHigh());

quantityLimit.setLowlimit(pqReportParam.getLimitLow());

quantityLimit.setLineId(list.get(0).getLineId());

quantityLimit.setDataid(pqReportParam.getDataID());

quantityLimit.setDataname(pqReportParam.getDataName());

list.add(quantityLimit);

}

}

}

private void reportParamToQuantityLimit(List<PqReportParam> reportParams, Long id, List<QuantityLimit> limits) {

for (PqReportParam pqReportParam : reportParams) {

QuantityLimit quantityLimit = new QuantityLimit();

quantityLimit.setUplimit(pqReportParam.getLimitHigh());

quantityLimit.setLowlimit(pqReportParam.getLimitLow());

quantityLimit.setLineId(id);

quantityLimit.setDataid(pqReportParam.getDataID());

quantityLimit.setDataname(pqReportParam.getDataName());

limits.add(quantityLimit);

}

}

private Line saveLineAndPecDeviceExtend(AddLineParams addLineParams, Long deviceId) {

LineVo lineVo = addLineParams.getLine();

Line line = new Line();

line.setIsupload(lineVo.getIsupload());

line.setStatus(lineVo.getStatus());

line.setUpvoltdeviation(lineVo.getUpvoltdeviation());

line.setUpdatetime(lineVo.getUpdatetime());

line.setPqmonitortype(lineVo.getPqmonitortype());

line.setNeutralpointgroundtype(lineVo.getNeutralpointgroundtype());

line.setLowvoltdeviation(lineVo.getLowvoltdeviation());

line.setIsspeciallinepower(lineVo.getIsspeciallinepower());

line.setIntegrityparaset\_id(lineVo.getIntegrityparaset\_id());

line.setVoltagetransformertype(lineVo.getVoltagetransformertype());

line.setId(lineVo.getId());

line.setVoltclass(lineVo.getVoltclass());

line.setUserprotocolcapacity(lineVo.getUserprotocolcapacity());

line.setSupplyequipmentcapacity(lineVo.getSupplyequipmentcapacity());

line.setShortcircuitcapacity(lineVo.getShortcircuitcapacity());

line.setObjecttype(lineVo.getObjecttype());

line.setObject\_label(getObjectLabel(lineVo.getObjecttype()));

line.setObject\_id(lineVo.getObject\_id());

line.setObject\_insname(lineVo.getObject\_insname());

line.setName(lineVo.getName());

line.setIntergrityparaset\_id(lineVo.getIntegrityparaset\_id());

line.setCode(lineVo.getCode());

line.setPqterminal\_id(addLineParams.getPqterminal\_id());

PecDeviceExtendVo pecDeviceExtendVo = addLineParams.getPecDeviceExtend();

PecDeviceExtend pecDeviceExtend = new PecDeviceExtend();

pecDeviceExtend.setWiredtype(pecDeviceExtendVo.getWiredtype());

pecDeviceExtend.setStatus(pecDeviceExtendVo.getStatus());

pecDeviceExtend.setStationid(addLineParams.getStationID());

pecDeviceExtend.setPtratio(pecDeviceExtendVo.getPtratio());

pecDeviceExtend.setProtocaltype(pecDeviceExtendVo.getProtocaltype());

pecDeviceExtend.setDrivertypename(pecDeviceExtendVo.getDrivertypename());

pecDeviceExtend.setName(pecDeviceExtendVo.getName());

pecDeviceExtend.setMetertype(pecDeviceExtendVo.getMetertype());

pecDeviceExtend.setIntegrityrateschemeId(pecDeviceExtendVo.getIntegrityratescheme\_id());

pecDeviceExtend.setInputvolt(pecDeviceExtendVo.getInputvolt());

pecDeviceExtend.setInputcurrent(pecDeviceExtendVo.getInputcurrent());

pecDeviceExtend.setFullscalevalue(pecDeviceExtendVo.getFullscalevalue());

pecDeviceExtend.setEnergytype(pecDeviceExtendVo.getEnergytype());

pecDeviceExtend.setDevicesstatus(pecDeviceExtendVo.getDevicesstatus());

pecDeviceExtend.setDeviceid(deviceId);

pecDeviceExtend.setDataloginterval(pecDeviceExtendVo.getDataloginterval());

pecDeviceExtend.setCtratio(pecDeviceExtendVo.getCtratio());

pecDeviceExtend.setCommissioningdate(pecDeviceExtendVo.getCommissioningdate());

// 2.查找空闲通道，如果有空闲通道则创建设备。如果没有空闲通道则创建通道再创建设备

ChannelBean channelBean = addLineParams.getChannelBean();

Long channelID = channelBean.getChannelID();

pecDeviceExtend.setChannelid(channelID);

pecDeviceExtend.setChannelname(channelBean.getDrvName());

pecDeviceExtend.setBranchindex(pecDeviceExtendVo.getBranchindex());

pecDeviceExtend.setAccessdate(pecDeviceExtendVo.getAccessdate());

pecDeviceExtend.setDatalogmapname(pecDeviceExtendVo.getDatalogmapname());

pecDeviceExtend.setStationname(pecDeviceExtendVo.getStationname());

pecDeviceExtend.setDevicetypename(pecDeviceExtendVo.getDevicetypename());

List<PecDeviceExtend> pecdeviceextend\_model = new ArrayList<>();

pecdeviceextend\_model.add(pecDeviceExtend);

line.setPecdeviceextend\_model(pecdeviceextend\_model);

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

list.add(line);

Result<Object> result = modelDataService.write(list);

checkResult(result);

List<Line> lines = JsonTransferUtils.transferJsonString(JsonTransferUtils.toJsonString(result.getData()),

Line.class);

if (CollectionUtils.isEmpty(lines)) {

throw new CommonManagerException("保存监测点异常");

}

return lines.get(0);

}

private String getObjectLabel(Integer objectType) {

if (ObjectTypeEnum.getValueByName(Constant.ELECTRIFIEDRAILWAY).equals(objectType)) {

return "electricrailway";

} else if (ObjectTypeEnum.getValueByName(Constant.PHOTOVOLTAIC\_POWERSTATION).equals(objectType)) {

return "photovoltaicstation";

} else if (ObjectTypeEnum.getValueByName(Constant.WINDFARM).equals(objectType)) {

return "windpowerstation";

} else {

return "interferencesource";

}

}

private Long saveDeviceChannelUnusefulCondition(AddLineParams addLineParams, String drvName, List<DeviceNode> deviceNode) {

// 拼接channelParamList入参。新建通道的命名规则：“负荷电站所属上级单位”\_“驱动类型名称”

Channel channelParam = channelParamStr(drvName, addLineParams);

if(null == channelParam.getNodeId()){

throw new CommonManagerException("参数异常！");

}

addLineParams.getChannelBean().setChannelID(channelParam.getNodeId());

addLineParams.getChannelBean().setDrvName(channelParam.getNodeName());

// 拼接deviceNode入参

setDeviceNode(deviceNode, addLineParams);

Result<Object> deviceResult = pecNodeService.addDevices(deviceNode);

String json = JsonTransferUtils.toJsonString(deviceNode);

checkResult(deviceResult);

List<Long> devices = JsonTransferUtils

.transferJsonString(JsonTransferUtils.toJsonString(deviceResult.getData()), Long.class);

if (CollectionUtils.isEmpty(devices)) {

throw new CommonManagerException("保存设备异常");

}

return devices.get(0);

}

private void checkStationHave(SystemNode data) {

if (null == data) {

throw new CommonManagerException("该厂站不存在！");

}

}

private <T> void checkResult(Result<T> result) {

if (Result.SUCCESS\_CODE != result.getCode()) {

if (ErrorCode.PEC\_DUPLICATE\_NAME\_CODE == result.getCode()) {

throw new CommonManagerException(ErrorCode.PEC\_DUPLICATE\_NAME\_DESC);

}

throw new CommonManagerException(result.getMsg());

}

}

}

package com.cet.pq.harmonic.trend.constant;

import com.google.common.collect.Lists;

import java.text.DecimalFormat;

import java.util.\*; \*/

int STATE\_NORMAL = 1;

/\*\*

\* 状态：超标

\*/

int STATE\_OVER = 2;

/\*\*

\* 状态：没数据

\*/

int STATE\_NO\_DATA = 3;

/\*\*

\* line 表中的状态，运行

\*/

int LINE\_STATUS\_RUNNING = 1;

/\*\*

\* line 表中的状态，停用

\*/

int LINE\_STATUS\_STOP = 4;

/\*\*

\* 谐波次数

\*/

List<Integer> HARMINOC\_NUMS = Lists.newArrayList(5, 7, 11, 13);

Map<Integer, String> HARMINOC\_NAME\_MAP = new HashMap<Integer, String>() {

{

this.put(5, "五次谐波");

this.put(7, "七次谐波");

this.put(11, "十一次谐波");

this.put(13, "十三次谐波");

}

};

List<String> DRILL\_DOWN\_MAP\_PARAMS = new ArrayList<String>(){

};

/\*\*

\* 超标判定指标 不可改负荷，默认0~2三项为电压、后3~5三项为电流

\*/

List<String> OVER\_PARAMS = new ArrayList<String>() { };

}

package com.cet.pq.harmonic.trend.constant;

public enum IndexType {

CURRENT(1, "电流"),

VOLTAGE(2, "电压");

private Integer code;

private String lable;

IndexType(Integer code, String lable) {

this.code = code;

this.lable = lable;

}

public Integer getCode() {

return code;

}

public String getLable() {

return lable;

}

}

package com.cet.pq.harmonic.trend.constant;

import com.cet.pq.config.SwaggerDisplayEnum;

import lombok.Getter;

/\*\*

\* 全网谐波潮流图层级

\*/

@Getter

@SwaggerDisplayEnum

public enum MapLevelEnum {

NET(1, "全网"),

PROVINCE(2, "省"),

CITY(3, "市");

private Integer code;

private String lable;

MapLevelEnum(Integer code, String lable) {

this.code = code;

this.lable = lable;

}

/\*\*

\* 单个枚举的展示

\*/

@Override

public String toString() {

return code + "-" + lable;

}

}

package com.cet.pq.harmonic.trend.constant;

public interface SubstationConstant {

/\*\*

\* 虚拟负荷

\*/

int TYPE\_VIRTUAL = 0;

/\*\*

\* 负荷电站

\*/

int SUBSTATION = 1;

/\*\*

\* 负荷压器

\*/

int TRANSFORMER = 2;

/\*\*

\* 换流站

\*/

int CONVERTER\_STATION = 4;

/\*\*

\* 发电站

\*/

int POWER\_STATION = 5;

/\*\*

\* 开关

\*/

int SWITCH\_STATION = 6;

}

package com.cet.pq.harmonic.trend.constant;

import com.cet.pq.config.SwaggerDisplayEnum;

import lombok.Getter;

/\*\*

\* 谐波潮流方向

\*/

@Getter

@SwaggerDisplayEnum

public enum TrendDirectionEnum {

POSITIVE(1, "正向"),

REVERSE(2, "反向");

private Integer code;

private String lable;

TrendDirectionEnum(Integer code, String lable) {

this.code = code;

this.lable = lable;

}

}

package com.cet.pq.harmonic.trend.controller;

import com.cet.pq.common.model.Result;

import com.cet.pq.harmonic.trend.constant.HarmonicMapConstant;

import com.cet.pq.harmonic.trend.model.Substation;

import com.cet.pq.harmonic.trend.model.dto.\*;

import com.cet.pq.harmonic.trend.model.query.\*;

import com.cet.pq.harmonic.trend.service.\*;

import io.swagger.annotations.Api;

import io.swagger.annotations.ApiOperation;

import org.apache.commons.collections4.CollectionUtils;

import org.apache.commons.lang3.StringUtils;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.stream.Collectors;

@Api("谐波潮流")

@RestController

@RequestMapping("/harmonicSource")

public class HarmonicSourceController {

@Autowired

private HarmonicSourceService harmonicSourceService;

@Autowired

private DrillDownMapService drillDownMapService;

@Autowired

private NwSubstationService nwSubstationService;

@Autowired

private CalculateService calculate;

@Autowired

private WholeNetService wholeNetService;

@ApiOperation("算法测试")

@PostMapping("/test/caculate")

public void Calculate(@RequestBody CalculateQuery query) {

// calculateServiceOld.calculate(query);

calculate.running(query);

}

@ApiOperation("贡献度排名")

@PostMapping({"/contributionSort"})

@ResponseBody

public Result<List<ContributionDTO>> contributionSort(@RequestBody HarmonicTrendQuery query) {

return Result.success(this.harmonicSourceService.contributionSort(query));

}

/\*@ApiOperation("源定位图")

@PostMapping({"/localtionMap"})

@ResponseBody

public TrendMapDataDTO localtionMap(@RequestBody HarmonicTrendQuery query) {

return this.harmonicSourceService.trendMapData(query, HarmonicMapConstant.TYPE\_LOCALTION\_MAP);

}\*/

@ApiOperation("源定位图")

@PostMapping({"/localtionMap"})

@ResponseBody

public Result<List<LocaltionMapDTO>> localtionMap(@RequestBody HarmonicTrendQuery query) {

return Result.success(this.harmonicSourceService.localtionMap(query));

}

@ApiOperation("源定位图相关负荷")

@PostMapping({"/localtionMapSubstationList"})

@ResponseBody

public Result<List<SubstationDTO>> localtionMapSubstationList(@RequestBody HarmonicTrendQuery query) {

return Result.success(this.harmonicSourceService.localtionMapSubstation(query, HarmonicMapConstant.TYPE\_LOCALTION\_MAP));

}

@ApiOperation("总览图")

@PostMapping({"/overviewMap"})

@ResponseBody

public Result<TrendMapDataDTO> overviewMap(@RequestBody OverviewMapQuery query) {

return wholeNetService.queryOrCreateTrendResult(query);

}

@ApiOperation("总览图,生成拓扑结构")

@PostMapping({"/overviewMapPsr"})

@ResponseBody

public Result<TrendMapDataDTO> overviewMapPsr(@RequestBody OverviewMapPsrQuery query) {

Result result = this.wholeNetService.init(query);

if (result.getCode().equals(Result.SUCCESS\_CODE)) {

return wholeNetService.queryOrCreateTrendResult(query);

}

return result;

}

@ApiOperation("预治理")

@PostMapping({"/imitate"})

@ResponseBody

public Result<ImitateDTO> imitate(@RequestBody ImitateQuery query) {

return Result.success(this.harmonicSourceService.imitate(query));

}

@ApiOperation("下钻图")

@PostMapping({"/drillDownMap"})

@ResponseBody

public Result<List<DrillDownMapDTO>> drillDownMap(@RequestBody HarmonicTrendQuery query) {

return Result.success(this.drillDownMapService.drillDownMap(query));

}

@ApiOperation("重要负荷")

@PostMapping({"/mainSubstation"})

@ResponseBody

public Result<List<SubstationDictDTO>> mainSubstation() {

return Result.success(this.nwSubstationService.selectMainSubstation());

}

@ApiOperation("关联负荷")

@PostMapping({"/relationSubstationList"})

@ResponseBody

public Result<List<SubstationDictDTO>> relationSubstation(@RequestParam() String mainSubstationCode) {

List<Substation> substations = this.nwSubstationService.selectRelationSubstation(mainSubstationCode, HarmonicMapConstant.TYPE\_LOCALTION\_MAP);

if (CollectionUtils.isEmpty(substations)) {

return null;

}

List<SubstationDictDTO> dictDTOS = substations.stream()

.filter(item -> !StringUtils.equals(item.getCode(), mainSubstationCode))

.map(item -> {

SubstationDictDTO dictDTO = new SubstationDictDTO();

dictDTO.setSubstationId(item.getId());

dictDTO.setSubstationCode(item.getCode());

dictDTO.setSubstationName(item.getName());

return dictDTO;

}).collect(Collectors.toList());

return Result.success(dictDTOS);

}

@ApiOperation("贡献度图表")

@PostMapping({"/contributionChart"})

@ResponseBody

public Result<ContributionChartDTO> contributionChart(@RequestBody ContributionChartQuery query) {

return Result.success(this.harmonicSourceService.contributionChart(query));

}

}

package com.cet.pq.harmonic.trend.datachange;

import com.cet.pq.harmonic.trend.constant.HarmonicMapConstant;

import com.cet.pq.harmonic.trend.model.Mation;

import com.cet.pq.harmonic.trend.model.dto.NodeInfoDTO;

import com.google.common.collect.Lists;

import org.apache.commons.collections4.CollectionUtils;

import java.util.List;

import java.util.Objects;

import java.util.stream.Collectors;

public class MationChange {

public static NodeInfoDTO toNodeInfoDTO(Mation mation) {

// List<Double> voltageRateList = Lists.newArrayList(mation.getVoltageRateA(), mation.getVoltageRateB(), mation.getVoltageRateC());

// List<Double> currentList = Lists.newArrayList(mation.getCurrentA(), mation.getCurrentB(), mation.getCurrentC());

List<Double> activePowerList = Lists.newArrayList(mation.getActivePowerA(), mation.getActivePowerB(), mation.getActivePowerC());

List<Double> reactivePowerList = Lists.newArrayList(mation.getReactivePowerA(), mation.getReactivePowerB(), mation.getReactivePowerC());

// voltageRateList = voltageRateList.stream().filter(Objects::nonNull).collect(Collectors.toList());

// currentList = currentList.stream().filter(Objects::nonNull).collect(Collectors.toList());

activePowerList = activePowerList.stream().filter(Objects::nonNull).collect(Collectors.toList());

reactivePowerList = reactivePowerList.stream().filter(Objects::nonNull).collect(Collectors.toList());

// Double voltageRate = CollectionUtils.isEmpty(voltageRateList) ? null : voltageRateList.stream().mapToDouble(Double::doubleValue).average().getAsDouble();

// Double current = CollectionUtils.isEmpty(currentList) ? null : currentList.stream().mapToDouble(Double::doubleValue).average().getAsDouble();

Double activePower = CollectionUtils.isEmpty(activePowerList) ? null : activePowerList.stream().mapToDouble(Double::doubleValue).average().getAsDouble();

Double reactivePower = CollectionUtils.isEmpty(reactivePowerList) ? null : reactivePowerList.stream().mapToDouble(Double::doubleValue).average().getAsDouble();

NodeInfoDTO nodeInfoDTO = new NodeInfoDTO();

// nodeInfoDTO.setVoltclass(voltageRate == null ? null : Double.valueOf(HarmonicMapConstant.decimalFormat.format(voltageRate)));

// nodeInfoDTO.setElectric(current == null ? null : Double.valueOf(HarmonicMapConstant.decimalFormat.format(current)));

nodeInfoDTO.setVoltclass(mation.getVoltage());

nodeInfoDTO.setElectric(mation.getCurrent());

nodeInfoDTO.setActivePower(activePower == null ? null : Double.valueOf(HarmonicMapConstant.decimalFormat.format(activePower)));

nodeInfoDTO.setReactivePower(reactivePower == null ? null : Double.valueOf(HarmonicMapConstant.decimalFormat.format(reactivePower)));

nodeInfoDTO.setVoltageOver(mation.getVoltageOver());

nodeInfoDTO.setCurrentOver(mation.getCurrentOver());

return nodeInfoDTO;

}

}

package com.cet.pq.harmonic.trend.datachange;

import com.cet.pq.harmonic.trend.constant.HarmonicMapConstant;

import com.cet.pq.harmonic.trend.model.SubstationRelation;

import com.cet.pq.harmonic.trend.model.dto.SubstationInfoDTO;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

/\*\*

\* SubstationRelation 数据格式转换

\*/

public class SubstationRelationChange {

public static List<SubstationInfoDTO> toSubstation(List<SubstationRelation> substationRelations) {

List<SubstationInfoDTO> substationInfoDTOS = new ArrayList<>();

substationInfoDTOS.addAll( substationRelations.stream().map(item-> {

SubstationInfoDTO substationInfoDTO = new SubstationInfoDTO();

substationInfoDTO.setSubstationCode(item.getStartSubstationCode());

substationInfoDTO.setSubstationName(item.getStartSubstation());

substationInfoDTO.setLevel(item.getStartSubstationLevel());

substationInfoDTO.setState(HarmonicMapConstant.STATE\_NORMAL);

return substationInfoDTO;

}).collect(Collectors.toList()));

substationInfoDTOS.addAll( substationRelations.stream().map(item -> {

SubstationInfoDTO substationInfoDTO = new SubstationInfoDTO();

substationInfoDTO.setSubstationCode(item.getEndSubstationCode());

substationInfoDTO.setSubstationName(item.getEndSubstation());

substationInfoDTO.setLevel(item.getEndSubstationLevel());

substationInfoDTO.setState(HarmonicMapConstant.STATE\_NORMAL);

return substationInfoDTO;

}).collect(Collectors.toList()));

substationInfoDTOS = substationInfoDTOS.stream().distinct().collect(Collectors.toList());

return substationInfoDTOS;

}

}

package com.cet.pq.harmonic.trend.datachange;

import com.cet.pq.harmonic.trend.constant.IndexType;

import com.cet.pq.harmonic.trend.model.dto.\*;

import org.apache.commons.collections4.CollectionUtils;

import java.util.\*;

import java.util.concurrent.atomic.AtomicReference;

public class TrendResultDTOChange {

public static TrendMapDataDTO toTrendMapDataDTO(List<TrendResultDTO> list) {

if (CollectionUtils.isEmpty(list)) {

return null;

}

List<SubstationInfoDTO> substationInfos = new ArrayList<>();

List<LineInfoDTO> lineInfos = new ArrayList<>();

Set<String> substationInfoCodeSet = new HashSet<>();

list.forEach(item -> {

Integer startSubstationState = item.getStartSubstationState();

Integer endSubstationState = item.getEndSubstationState();

if (!substationInfoCodeSet.contains(item.getStartSubstationCode())) {

SubstationInfoDTO substationInfo = new SubstationInfoDTO();

substationInfos.add(substationInfo);

substationInfo.setSubstationCode(item.getStartSubstationCode());

substationInfo.setSubstationName(item.getStartSubstationName());

substationInfo.setVoltclass(item.getStartTrendVoltclass());

substationInfo.setElectric(item.getStartTrendElectric());

substationInfo.setLevel(item.getStartSubstationLevel());

substationInfo.setState(item.getStartSubstationState());

substationInfo.setDevId(item.getStartSubstationDevId());

substationInfoCodeSet.add(item.getStartSubstationCode());

}

if (!substationInfoCodeSet.contains(item.getEndSubstationCode())) {

SubstationInfoDTO substationInfo = new SubstationInfoDTO();

substationInfos.add(substationInfo);

substationInfo.setSubstationCode(item.getEndSubstationCode());

substationInfo.setSubstationName(item.getEndSubstationName());

substationInfo.setVoltclass(item.getEndTrendVoltclass());

substationInfo.setElectric(item.getEndTrendElectric());

substationInfo.setLevel(item.getEndSubstationLevel());

substationInfo.setState(item.getEndSubstationState());

substationInfo.setDevId(item.getEndSubstationDevId());

substationInfoCodeSet.add(item.getEndSubstationCode());

}

LineInfoDTO lineInfo = new LineInfoDTO();

lineInfos.add(lineInfo);

lineInfo.setStartSubstationCode(item.getStartSubstationCode());

lineInfo.setStartSubstationDevId(item.getStartSubstationDevId());

lineInfo.setEndSubstationCode(item.getEndSubstationCode());

lineInfo.setEndSubstationDevId(item.getEndSubstationDevId());

// 有功功率从高到低即为潮流方向

Double startSubstationP = item.getStartSubstationP();

Double endSubstationP = item.getEndSubstationP();

lineInfo.setTrendDirection(item.decideTrendDirection(startSubstationP, endSubstationP));

lineInfo.setTrendValue(item.getTrendValue());

lineInfo.setStartSubstationLevel(item.getStartSubstationLevel());

lineInfo.setEndSubstationLevel(item.getEndSubstationLevel());

lineInfo.setState(LineInfoDTO.getLineState(startSubstationState, endSubstationState));

});

TrendMapDataDTO trendMapDataDTO = new TrendMapDataDTO();

trendMapDataDTO.setLineInfos(lineInfos);

trendMapDataDTO.setSubstationInfos(substationInfos);

return trendMapDataDTO;

}

public static List<SubstationDTO> toSubstationDTOs(List<TrendResultDTO> list) {

List<SubstationDTO> substationDTOS = new ArrayList<>();

Set<String> substationCodeSet = new HashSet<>();

list.forEach(item -> {

if (!substationCodeSet.contains(item.getStartSubstationCode())) {

SubstationDTO substationDTO = new SubstationDTO();

substationDTOS.add(substationDTO);

substationDTO.setId(item.getStartSubstationId());

substationDTO.setCode(item.getStartSubstationCode());

substationDTO.setName(item.getStartSubstationName());

substationDTO.setVoltclass(item.getStartTrendVoltclass());

substationDTO.setElectric(item.getStartTrendElectric());

substationDTO.setLevel(item.getStartSubstationLevel());

substationDTO.setState(item.getStartSubstationState());

substationDTO.setStateI(item.getStartSubstationStateI());

substationDTO.setStateV(item.getStartSubstationStateV());

substationCodeSet.add(item.getStartSubstationCode());

}

if (!substationCodeSet.contains(item.getEndSubstationCode())) {

SubstationDTO substationDTO = new SubstationDTO();

substationDTOS.add(substationDTO);

substationDTO.setId(item.getEndSubstationId());

substationDTO.setCode(item.getEndSubstationCode());

substationDTO.setName(item.getEndSubstationName());

substationDTO.setVoltclass(item.getEndTrendVoltclass());

substationDTO.setElectric(item.getEndTrendElectric());

substationDTO.setLevel(item.getEndSubstationLevel());

substationDTO.setState(item.getEndSubstationState());

substationDTO.setStateI(item.getEndSubstationStateI());

substationDTO.setStateV(item.getEndSubstationStateV());

substationCodeSet.add(item.getEndSubstationCode());

}

});

return substationDTOS;

}

public static List<LocaltionMapDTO> toLocaltionMapDTO(List<TrendResultDTO> list, Integer indexType) {

List<LocaltionMapDTO> results = new ArrayList<>();

list.forEach(item -> {

LocaltionMapDTO localtionMapDTO = new LocaltionMapDTO();

localtionMapDTO.setCode(item.getStartSubstationCode() + "-" + item.getEndSubstationCode());

localtionMapDTO.setStartTrendElectric(item.getStartTrendElectric());

localtionMapDTO.setStartTrendVoltclass(item.getStartTrendVoltclass());

localtionMapDTO.setStartSubstationLevel(item.getStartSubstationLevel());

localtionMapDTO.setEndTrendVoltclass(item.getEndTrendVoltclass());

localtionMapDTO.setEndSubstationLevel(item.getEndSubstationLevel());

localtionMapDTO.setEndTrendElectric(item.getEndTrendElectric());

localtionMapDTO.setTrendValue(item.getTrendValue());

localtionMapDTO.setTrendDirection(1); // todo 暂定

Integer state = null;

if (indexType == null) {

Integer startSubstationState = item.getStartSubstationState();

Integer endSubstationState = item.getEndSubstationState();

state = LineInfoDTO.getLineState(startSubstationState, endSubstationState);

} else if (indexType.equals(IndexType.CURRENT.getCode())) {

Integer startSubstationState = item.getStartSubstationStateI();

Integer endSubstationState = item.getEndSubstationStateI();

state = LineInfoDTO.getLineState(startSubstationState, endSubstationState);

} else if (indexType.equals(IndexType.VOLTAGE.getCode())) {

Integer startSubstationState = item.getStartSubstationStateV();

Integer endSubstationState = item.getEndSubstationStateV();

state = LineInfoDTO.getLineState(startSubstationState, endSubstationState);

} else {

Integer startSubstationState = item.getStartSubstationState();

Integer endSubstationState = item.getEndSubstationState();

state = LineInfoDTO.getLineState(startSubstationState, endSubstationState);

}

localtionMapDTO.setState(state);

results.add(localtionMapDTO);

});

return results;

}

public static Map<String, SubstationDTO> toSubstationDTOMap(List<TrendResultDTO> list) {

Map<String, SubstationDTO> substationDTOMap = new HashMap<>();

AtomicReference<SubstationDTO> substationDTO = new AtomicReference<>();

list.forEach(item -> {

if (!substationDTOMap.containsKey(item.getStartSubstationCode())) {

substationDTO.set(new SubstationDTO());

substationDTO.get().setId(item.getStartSubstationId());

substationDTO.get().setCode(item.getStartSubstationCode());

substationDTO.get().setName(item.getStartSubstationName());

substationDTO.get().setVoltclass(item.getStartTrendVoltclass());

substationDTO.get().setElectric(item.getStartTrendElectric());

substationDTO.get().setLevel(item.getStartSubstationLevel());

substationDTOMap.put(item.getStartSubstationCode(), substationDTO.get());

}

if (!substationDTOMap.containsKey(item.getEndSubstationCode())) {

substationDTO.set(new SubstationDTO());

substationDTO.get().setId(item.getEndSubstationId());

substationDTO.get().setCode(item.getEndSubstationCode());

substationDTO.get().setName(item.getEndSubstationName());

substationDTO.get().setVoltclass(item.getEndTrendVoltclass());

substationDTO.get().setElectric(item.getEndTrendElectric());

substationDTO.get().setLevel(item.getEndSubstationLevel());

substationDTOMap.put(item.getEndSubstationCode(), substationDTO.get());

}

});

return substationDTOMap;

}

public static void substationDTOMapToTrendResultDTO(List<TrendResultDTO> trendResultDTOS, Map<String, SubstationDTO> substationDTOMap) {

SubstationDTO substationDTO = null;

String startSubstationCode = null;

String endSubstationCode = null;

for (TrendResultDTO item : trendResultDTOS) {

startSubstationCode = item.getStartSubstationCode();

endSubstationCode = item.getEndSubstationCode();

substationDTO = substationDTOMap.get(startSubstationCode);

if (substationDTO != null) {

item.setStartSubstationId(substationDTO.getId());

item.setStartSubstationCode(substationDTO.getCode());

item.setStartSubstationName(substationDTO.getName());

item.setStartTrendVoltclass(substationDTO.getVoltclass());

item.setStartTrendElectric(substationDTO.getElectric());

item.setStartSubstationLevel(substationDTO.getLevel());

}

substationDTO = substationDTOMap.get(endSubstationCode);

if (substationDTO != null) {

item.setEndSubstationId(substationDTO.getId());

item.setEndSubstationCode(substationDTO.getCode());

item.setEndSubstationName(substationDTO.getName());

item.setEndTrendVoltclass(substationDTO.getVoltclass());

item.setEndTrendElectric(substationDTO.getElectric());

item.setEndSubstationLevel(substationDTO.getLevel());

}

}

}

}

package com.cet.pq.harmonic.trend.datachange;

import com.cet.pq.harmonic.trend.model.WholeNetRelation;

import com.cet.pq.harmonic.trend.model.dto.XddaonaDTO;

import org.apache.commons.collections4.CollectionUtils;

import java.util.List;

import java.util.stream.Collectors;

public class WholeNetRelationChange {

public static List<XddaonaDTO> toXddaonaDTO(List<WholeNetRelation> relations) {

if (CollectionUtils.isEmpty(relations)) {

return null;

}

return relations.stream().map(item -> {

XddaonaDTO xddaonaDTO = new XddaonaDTO();

xddaonaDTO.setMapId(item.getMapId());

xddaonaDTO.setNodeIdStart(item.getStartDevId());

// xddaonaDTO.setNodeCodeStart(item.getStartDevId());

xddaonaDTO.setNodeNameStart(item.getStartDevName());

xddaonaDTO.setNodeIdEnd(item.getEndDevId());

// xddaonaDTO.setNodeCodeEnd(item.getEndDevId());

xddaonaDTO.setNodeNameEnd(item.getEndDevName());

return xddaonaDTO;

}).collect(Collectors.toList());

}

}

package com.cet.pq.harmonic.trend.datachange;

import com.cet.pq.harmonic.trend.constant.HarmonicMapConstant;

import com.cet.pq.harmonic.trend.model.dto.JddaonaDTO;

import com.cet.pq.harmonic.trend.model.dto.XddaonaDTO;

import com.isoftstone.bussiness.harmonicTracing.model.Jddaona;

import java.util.ArrayList;

import java.util.List;

import java.util.TreeSet;

import java.util.stream.Collectors;

import static java.util.Comparator.comparingLong;

import static java.util.stream.Collectors.collectingAndThen;

import static java.util.stream.Collectors.toCollection;

public class XddaonaDTOChange {

public static List<Jddaona> toJddaona(List<XddaonaDTO> xddaonaDTOS) {

List<Jddaona> jddaonas = new ArrayList<>();

jddaonas.addAll(xddaonaDTOS.stream().map(item -> {

JddaonaDTO jddaona = new JddaonaDTO();

jddaona.setNodeId(item.getNodeIdStart());

jddaona.setNodeName(item.getNodeNameStart());

jddaona.setNodeCode(item.getNodeCodeStart());

jddaona.setState(HarmonicMapConstant.STATE\_NORMAL);

jddaona.setMapId(item.getMapId());

return jddaona;

}).collect(Collectors.toList()));

jddaonas.addAll(xddaonaDTOS.stream().map(item -> {

JddaonaDTO jddaona = new JddaonaDTO();

jddaona.setNodeId(item.getNodeIdEnd());

jddaona.setNodeName(item.getNodeNameEnd());

jddaona.setNodeCode(item.getNodeCodeEnd());

jddaona.setState(HarmonicMapConstant.STATE\_NORMAL);

jddaona.setMapId(item.getMapId());

return jddaona;

}).collect(Collectors.toList()));

jddaonas = jddaonas.stream().distinct().collect(Collectors.toList());

return jddaonas;

}

public static List<JddaonaDTO> toJddaonaDTO(List<XddaonaDTO> xddaonaDTOS) {

List<JddaonaDTO> jddaonas = new ArrayList<>();

jddaonas.addAll(xddaonaDTOS.stream().map(item -> {

JddaonaDTO jddaona = new JddaonaDTO();

jddaona.setNodeId(item.getNodeIdStart());

jddaona.setNodeName(item.getNodeNameStart());

jddaona.setNodeCode(item.getNodeCodeStart());

jddaona.setState(HarmonicMapConstant.STATE\_NORMAL);

jddaona.setMapId(item.getMapId());

return jddaona;

}).collect(Collectors.toList()));

jddaonas.addAll(xddaonaDTOS.stream().map(item -> {

JddaonaDTO jddaona = new JddaonaDTO();

jddaona.setNodeId(item.getNodeIdEnd());

jddaona.setNodeName(item.getNodeNameEnd());

jddaona.setNodeCode(item.getNodeCodeEnd());

jddaona.setState(HarmonicMapConstant.STATE\_NORMAL);

jddaona.setMapId(item.getMapId());

return jddaona;

}).collect(Collectors.toList()));

jddaonas = jddaonas.stream().collect(

collectingAndThen(

toCollection(() -> new TreeSet<>(comparingLong(JddaonaDTO::getNodeId))), ArrayList::new)

);

return jddaonas;

}

}