import com.bme.cloud.carbon.common.PageUtils;

import com.bme.cloud.carbon.service.EquipmentService;

import com.bme.cloud.carbon.service.MeasuringEquipmentService;

import com.bme.cloud.common.enums.CarbonTypeEnum;

import com.bme.cloud.common.model.basic.Page;

import com.bme.cloud.common.model.carbon.Equipment;

import com.bme.cloud.common.model.carbon.EquipmentQuery;

import com.bme.cloud.common.model.carbon.MeasuringEquipment;

import com.github.pagehelper.PageInfo;

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

import org.springframework.util.CollectionUtils;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

import java.util.Objects;

import java.util.Set;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/api/process")

public class ProcessController {

@Autowired

private EquipmentService equipmentService;

@Autowired

private MeasuringEquipmentService measuringEquipmentService;

@PostMapping("/page")

public Page<Equipment> getPage(EquipmentQuery equipmentQuery) {

List<Equipment> result = equipmentService.getPageByProcess(equipmentQuery);

if (CollectionUtils.isEmpty(result)) {

return PageUtils.pageEmpty(equipmentQuery.getPageNo(), equipmentQuery.getPageSize());

}

for (Equipment equipment : result) {

List<MeasuringEquipment> measuringEquipments = measuringEquipmentService.getByEquNo(equipment.getEquipmentNo());

if (CollectionUtils.isEmpty(measuringEquipments)) {

continue;

}

Set<Integer> types = measuringEquipments.stream()

.filter(e -> e.getPartakeCalculate() == 1)

.filter(e -> (Objects.nonNull(e.getCarbonEmissionType()) && !CarbonTypeEnum.DEFAULT.getCarbonType().equals(e.getCarbonEmissionType())))

.map(MeasuringEquipment::getCarbonEmissionType).collect(Collectors.toSet());

if (CollectionUtils.isEmpty(types)) {

continue;

}

String carbonEmissionType = types.stream().map(e -> {

CarbonTypeEnum type = CarbonTypeEnum.getByEnumType(e);

return type.getName();

}).collect(Collectors.joining(","));

equipment.setEmissionType(carbonEmissionType);

}

Page<Equipment> page = new Page<>(equipmentQuery.getPageNo(), equipmentQuery.getPageSize());

PageInfo<Equipment> pageInfo = new PageInfo<>(result);

page.setRecords(pageInfo.getList());

page.setTotal(pageInfo.getTotal());

return page;

}

@GetMapping("/getProcessByEquipmentNo")

public List<Equipment> getProcessByEquipmentNo(@RequestParam("customerId") Long customerId, @RequestParam("equipmentNoList") List<String> equipmentNoList){

return equipmentService.getProcessByEquipentNo(customerId, equipmentNoList);

}

@GetMapping("/getEquipmentList")

public List<Equipment> getEquipmentList(@RequestParam("customerId") Long customerId) {

return equipmentService.getEquipmentListV2(customerId);

}

@PostMapping("/list")

List<Equipment> selectEquipmentByProcess(@RequestParam("customerId") Long customerId, @RequestParam("processId") Long processId) {

return equipmentService.selectEquipmentByProcess(customerId,processId);

}

@GetMapping("/getEquipmentListByProcess")

public List<Equipment> getEquipmentListByProcess(@RequestParam("customerId") Long customerId, @RequestParam(value = "processIdList", required = false) List<Integer> processIdList){

return equipmentService.getEquipmentListByProcess(customerId, processIdList);

}

}

import com.bme.cloud.carbon.repository.EquipmentDao;

import com.bme.cloud.common.enums.CarbonTypeEnum;

import com.bme.cloud.common.model.carbon.Equipment;

import com.bme.cloud.common.model.carbon.EquipmentQuery;

import com.bme.cloud.common.model.carbon.MeasuringEquipment;

import com.github.pagehelper.PageHelper;

import org.apache.commons.collections4.CollectionUtils;

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

import org.springframework.stereotype.Service;

import org.springframework.web.bind.annotation.RequestParam;

import java.util.\*;

import java.util.stream.Collectors;

@Service

public class EquipmentService {

@Autowired

private EquipmentDao equipmentDao;

@Autowired

private MeasuringEquipmentService measuringEquipmentService;

public List<Equipment> getPageByProcess(EquipmentQuery equipmentQuery) {

PageHelper.startPage(equipmentQuery.getPageNo(), equipmentQuery.getPageSize());

return equipmentDao.selectPageByProcess(equipmentQuery);

}

public List<Equipment> getProcessByEquipentNo(Long customerId, List<String> equipmentNoList) {

return equipmentDao.selectProcessByEquipentNo(customerId, equipmentNoList);

}

public List<Equipment> getEquipmentList(Long customerId) {

List<Equipment> list = equipmentDao.selectEquipmentListByCustomerId(customerId);

if (CollectionUtils.isEmpty(list)) {

return Collections.emptyList();

}

Map<String, Map<String, List<Equipment>>> maps = list.stream().collect(Collectors.groupingBy(Equipment::getProcess, Collectors.groupingBy(Equipment::getFacility)));

if (Objects.isNull(maps) || maps.size() <= 0) {

return Collections.emptyList();

}

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

maps.forEach((process, temps) -> {

temps.forEach((facility, listTemp) -> {

Equipment equipment = new Equipment();

equipment.setProcess(process);

equipment.setFacility(facility);

if (CollectionUtils.isNotEmpty(listTemp)) {

Equipment equ = listTemp.get(0);

equipment.setFacilitySpecs(equ.getFacilitySpecs());

equipment.setCount(listTemp.size());

equipment.setEquipmentName(equ.getDeviceName());

equipment.setEquipmentNo(equ.getEquipmentNo());

equipment.setBranchFactory(equ.getBranchFactory());

equipment.setEmissionType(equ.getEmissionType());

}

result.add(equipment);

});

});

return result;

}

public List<Equipment> getEquipmentListV2(Long customerId) {

List<Equipment> list = equipmentDao.selectEquipmentListByCustomerId(customerId);

if (CollectionUtils.isEmpty(list)) {

return Collections.emptyList();

}

for (Equipment equipment : list) {

//设备排放类型

List<MeasuringEquipment> measuringEquipments = measuringEquipmentService.getByEquNo(equipment.getEquipmentNo());

if (org.springframework.util.CollectionUtils.isEmpty(measuringEquipments)) {

continue;

}

Set<Integer> types = measuringEquipments.stream()

.filter(e -> e.getPartakeCalculate() == 1)

.filter(e -> (Objects.nonNull(e.getCarbonEmissionType()) && !CarbonTypeEnum.DEFAULT.getCarbonType().equals(e.getCarbonEmissionType())))

.map(MeasuringEquipment::getCarbonEmissionType).collect(Collectors.toSet());

if (CollectionUtils.isEmpty(types)) {

continue;

}

String carbonEmissionType = types.stream().map(e -> {

CarbonTypeEnum type = CarbonTypeEnum.getByEnumType(e);

return type.getName();

}).collect(Collectors.joining(","));

equipment.setEmissionType(carbonEmissionType);

//位置

equipment.setLocation(equipment.getBranchFactory());

//变更情况

equipment.setChangeDesc("无");

}

return list;

}

public List<Equipment> selectEquipmentByProcess(Long customerId, Long processId) {

return equipmentDao.selectEquipmentByProcess(customerId, processId);

}

public List<Equipment> getEquipmentListByProcess(Long customerId, List<Integer> processIdList){

return equipmentDao.getEquipmentListByProcess(customerId, processIdList);

}

}

import com.bme.cloud.carbon.common.PageUtils;

import com.bme.cloud.carbon.service.MeasuringEquipmentService;

import com.bme.cloud.common.model.basic.Page;

import com.bme.cloud.common.model.carbon.CarbonEmission;

import com.bme.cloud.common.model.carbon.EquipmentQuery;

import com.bme.cloud.common.model.carbon.MeasuringEquipment;

import com.bme.cloud.common.support.PageParam;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import io.swagger.annotations.Api;

import io.swagger.annotations.ApiImplicitParam;

import io.swagger.annotations.ApiImplicitParams;

import io.swagger.annotations.ApiOperation;

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

import org.springframework.util.CollectionUtils;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

import java.util.Map;

/\*\*

\* @Description: 设备清单信息接口层

\* @Author: dongJue

\* @CreateDate: MeasuringEquipment

\* @Version: V1.0

\*/

@Api(description = "设备清单信息", value = "设备清单信息")

@RestController

@RequestMapping("/api/carbon")

public class MeasuringEquipmentController {

@Autowired

public MeasuringEquipmentService measuringEquipmentServiceImpl;

/\*\*

\* 根据物料id查询计量设备列表

\*/

@PostMapping("/material/equipment/page")

public Page<MeasuringEquipment> getByMaterialId(@RequestParam("customerId") Long customerId,

@RequestParam("materialId") Integer materialId,

@RequestParam(value = "carbonEmissionType", required = false) Integer carbonEmissionType,

@RequestParam("pageNo") Integer pageNo,

@RequestParam("pageSize") Integer pageSize) {

PageHelper.startPage(pageNo, pageSize, false);

List<MeasuringEquipment> result = measuringEquipmentServiceImpl.getByMaterialId(customerId, materialId, carbonEmissionType);

if (CollectionUtils.isEmpty(result)) {

return PageUtils.pageEmpty(pageNo, pageSize);

}

PageInfo<MeasuringEquipment> pageInfo = new PageInfo<>(result);

return new Page<>(pageSize, pageInfo.getTotal(), pageNo, result);

}

/\*\*

\* @param 对象参数：tMeasuringEquipment

\* @return List<TMeasuringEquipment>

\* @explain 获取匹配设备清单信息

\* @author dongJue

\* @time MeasuringEquipment

\*/

@ApiOperation(value = "条件查询设备清单信息", notes = "条件查询[tMeasuringEquipment],作者：dongJue")

@PostMapping("/queryTMeasuringEquipmentList")

public List<MeasuringEquipment> queryTMeasuringEquipmentList(@RequestBody MeasuringEquipment measuringEquipment) {

List<MeasuringEquipment> list = measuringEquipmentServiceImpl.queryTMeasuringEquipmentList(measuringEquipment);

return list;

}

/\*\*

\* @param 对象参数：AppPage<TMeasuringEquipment>

\* @return PageInfo<TMeasuringEquipment>

\* @explain 分页条件查询设备清单信息

\* @author dongJue

\* @time MeasuringEquipment

\*/

@GetMapping("/getPageTMeasuringEquipment/{pageNum}/{pageSize}")

@ApiOperation(value = "分页查询", notes = "分页查询返回对象[PageInfo<TMeasuringEquipment>]")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "path", name = "pageNum", value = "当前页", required = true, dataType = "int"),

@ApiImplicitParam(paramType = "path", name = "pageSize", value = "页行数", required = true, dataType = "int")

})

public PageInfo<MeasuringEquipment> getTMeasuringEquipmentBySearch(@PathVariable("pageNum") Integer pageNum, @PathVariable("pageSize") Integer pageSize, @ModelAttribute MeasuringEquipment measuringEquipment) {

PageParam<MeasuringEquipment> page = new PageParam<MeasuringEquipment>();

page.setPageNum(pageNum);

page.setPageSize(pageSize);

//其他参数

page.setParam(measuringEquipment);

//分页数据

PageInfo<MeasuringEquipment> pageInfo = measuringEquipmentServiceImpl.getTMeasuringEquipmentBySearch(page);

return pageInfo;

}

@GetMapping("/measuringEquipment/list")

public List<MeasuringEquipment> queryMeasuringEquipmentList(@RequestParam("customerId") long customerId, @RequestParam(required = false, value = "level") Integer level, @RequestParam(required = false, value = "partakeCalculate") Integer partakeCalculate) {

return measuringEquipmentServiceImpl.getMeasuringEquipmentList(customerId, level, partakeCalculate);

}

@GetMapping("/measuringEquipment/status/list")

public List<MeasuringEquipment> queryMeasuringEquipmentStatusList(@RequestParam("customerId") long customerId) {

return measuringEquipmentServiceImpl.getMeasuringEquipmentStausList(customerId);

}

@GetMapping("/measuringEquipmentListByType")

public PageInfo<MeasuringEquipment> queryMeasuringEquipmentListByType(@RequestParam("customerId") long customerId, @RequestParam("measuringEquipmentType") Integer measuringEquipmentType,

@RequestParam("pageNo") Integer pageNo, @RequestParam("pageSize") Integer pageSize) {

return measuringEquipmentServiceImpl.queryMeasuringEquipmentListByType(customerId, measuringEquipmentType, pageNo, pageSize);

}

@GetMapping("/deviceInfo/query")

public MeasuringEquipment queryEquipmentInfo(@RequestParam("customerId") long customerId, @RequestParam("deviceNo") String deviceNo) {

return measuringEquipmentServiceImpl.getEquipmentInfo(customerId, deviceNo);

}

/\*\*

\* @param 对象参数：tCarbonFactorConfig

\* @return List<TCarbonFactorConfig>

\* @explain 获取碳排放清单

\* @author dongJue

\* @time CarbonFactorConfig

\*/

@ApiOperation(value = "查询碳排放类型和物料的统计数量")

@GetMapping("/countByType")

@ApiImplicitParams({

@ApiImplicitParam(name = "customerId", value = "客户ID", required = true, dataType = "Long", paramType = "query")

})

public Map<Integer, List<CarbonEmission>> queryCountByType(@RequestParam("customerId") Long customerId) {

return measuringEquipmentServiceImpl.queryCountByType(customerId);

}

@ApiOperation(value = "查询工序分类,返回工序id和工序名称")

@GetMapping("/getProcessList")

@ApiImplicitParams({

@ApiImplicitParam(name = "customerId", value = "客户ID", required = true, dataType = "Long", paramType = "query")

})

public List<Map<String, Object>> getProcessList(@RequestParam("customerId") Long customerId) {

return measuringEquipmentServiceImpl.getProcessList(customerId);

}

@ApiOperation(value = "查询碳排放监控点位")

@GetMapping("/getCarbonEmissionType")

@ApiImplicitParams({

@ApiImplicitParam(name = "customerId", value = "客户ID", required = true, dataType = "Long", paramType = "query")

})

public List<Map<String, Object>> getCarbonEmissionType(@RequestParam("customerId") Long customerId, @RequestParam(value = "processId", required = false) Long processId) {

return measuringEquipmentServiceImpl.getCarbonEmissionType(customerId, processId);

}

@ApiOperation(value = "查询设施,返回设施id和设施名称，设施类型")

@GetMapping("/getFacilityList")

@ApiImplicitParams({

@ApiImplicitParam(name = "customerId", value = "客户ID", required = true, dataType = "Long", paramType = "query")

})

public List<Map<String, Object>> getFacitlityList(@RequestParam("customerId") Long customerId) {

return measuringEquipmentServiceImpl.getFacilityList(customerId);

}

@PostMapping("/measuringEquipment/page")

public Page<MeasuringEquipment> getPage(EquipmentQuery equipmentQuery) {

List<MeasuringEquipment> result = measuringEquipmentServiceImpl.getPage(equipmentQuery);

if (CollectionUtils.isEmpty(result)) {

return PageUtils.pageEmpty(equipmentQuery.getPageNo(), equipmentQuery.getPageSize());

}

Page<MeasuringEquipment> page = new Page<>(equipmentQuery.getPageNo(), equipmentQuery.getPageSize());

PageInfo<MeasuringEquipment> pageInfo = new PageInfo<>(result);

page.setRecords(pageInfo.getList());

page.setTotal(pageInfo.getTotal());

return page;

}

@GetMapping("/meterDeviceListByEquipment")

public List<MeasuringEquipment> getMeterDeviceListByEquipment(@RequestParam("customerId") Long customerId, @RequestParam("materialId") Integer materialId, @RequestParam("equipmentNoList") List<String> equipmentNoList) {

return measuringEquipmentServiceImpl.getMeterDeviceListByEquipment(customerId, materialId, equipmentNoList);

}

}

package com.bme.cloud.carbon.service.impl;

import com.bme.cloud.carbon.repository.MeasuringEquipmentDao;

import com.bme.cloud.carbon.service.MeasuringEquipmentService;

import com.bme.cloud.common.enums.CarbonTypeEnum;

import com.bme.cloud.common.model.carbon.CarbonEmission;

import com.bme.cloud.common.model.carbon.EquipmentQuery;

import com.bme.cloud.common.model.carbon.MeasuringEquipment;

import com.bme.cloud.common.support.PageParam;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import lombok.extern.slf4j.Slf4j;

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

import org.springframework.stereotype.Service;

import org.springframework.util.CollectionUtils;

import java.util.List;

import java.util.Map;

import static java.util.stream.Collectors.groupingBy;

/\*\*

\* @Description: 设备清单信息——SERVICEIMPL

\* @Author: dongJue

\* @CreateDate: MeasuringEquipment

\* @Version: V1.0

\*/

@Service

@Slf4j

public class MeasuringEquipmentServiceImpl implements MeasuringEquipmentService {

@Autowired

public MeasuringEquipmentDao tMeasuringEquipmentDao;

@Override

public List<MeasuringEquipment> getByMaterialId(Long customerId, Integer materialId, Integer carbonEmissionType) {

return tMeasuringEquipmentDao.selectByMaterialId(customerId, materialId, carbonEmissionType);

}

//查询集合

@Override

public List<MeasuringEquipment> queryTMeasuringEquipmentList(MeasuringEquipment measuringEquipment) {

return tMeasuringEquipmentDao.queryTMeasuringEquipmentList(measuringEquipment);

}

//分页查询

@Override

public PageInfo<MeasuringEquipment> getTMeasuringEquipmentBySearch(PageParam<MeasuringEquipment> page) {

// TODO Auto-generated method stub

PageHelper.startPage(page.getPageNum(), page.getPageSize());

List<MeasuringEquipment> list = tMeasuringEquipmentDao.queryTMeasuringEquipmentList(page.getParam());

PageInfo<MeasuringEquipment> pageInfo = new PageInfo<MeasuringEquipment>(list);

return pageInfo;

}

@Override

public List<MeasuringEquipment> getMeasuringEquipmentList(long customerId, Integer level, Integer partakeCalculate) {

return tMeasuringEquipmentDao.getMeasuringEquipmentList(customerId, level, partakeCalculate);

}

@Override

public List<MeasuringEquipment> getMeasuringEquipmentStausList(long customerId) {

return tMeasuringEquipmentDao.getMeasuringEquipmentStatusList(customerId);

}

@Override

public PageInfo<MeasuringEquipment> queryMeasuringEquipmentListByType(long customerId, Integer measuringEquipmentType, Integer pageNo, Integer pageSize) {

PageHelper.startPage(pageNo, pageSize);

List<MeasuringEquipment> measuringEquipmentListByType = tMeasuringEquipmentDao.getMeasuringEquipmentListByType(customerId, measuringEquipmentType);

return new PageInfo<>(measuringEquipmentListByType);

}

// 根据客户id查询碳排放清单

@Override

public Map<Integer, List<CarbonEmission>> queryCountByType(Long customerId) {

// 根据客户id查询所有的碳排放类型

List<CarbonEmission> carbonEmissions = tMeasuringEquipmentDao.queryCountByType(customerId);

if (CollectionUtils.isEmpty(carbonEmissions)) return null;

// 返回name的字段

for (CarbonEmission emission : carbonEmissions) {

CarbonTypeEnum carbonTypeEnum = CarbonTypeEnum.getByEnumType(emission.getEmissionType());

if (carbonTypeEnum != null) {

emission.setCarbonTypeName(carbonTypeEnum.getName());

}

}

// 根据物料类型分组

Map<Integer, List<CarbonEmission>> carbonEmissionList = carbonEmissions.stream().collect(groupingBy(CarbonEmission::getEmissionType));

return carbonEmissionList;

}

@Override

public MeasuringEquipment getEquipmentInfo(long customerId, String deviceNo) {

return tMeasuringEquipmentDao.getEquipmentInfo(customerId, deviceNo);

}

@Override

public List<Map<String, Object>> getProcessList(Long customerId) {

return tMeasuringEquipmentDao.getProcessList(customerId);

}

@Override

public List<Map<String, Object>> getCarbonEmissionType(Long customerId, Long processId) {

List<Map<String, Object>> list = tMeasuringEquipmentDao.getCarbonEmissionType(customerId, processId);

if (CollectionUtils.isEmpty(list)) return null;

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

if (map.get("carbonType") != null) {

CarbonTypeEnum carbonTypeEnum = CarbonTypeEnum.getByEnumType(Integer.parseInt(map.get("carbonType").toString()));

if (carbonTypeEnum != null) {

map.put("carbonTypeName", carbonTypeEnum.getName());

}

}

}

return list;

}

@Override

public List<Map<String, Object>> getFacilityList(Long customerId) {

return tMeasuringEquipmentDao.getFacilityList(customerId);

}

@Override

public List<MeasuringEquipment> getPage(EquipmentQuery equipmentQuery) {

PageHelper.startPage(equipmentQuery.getPageNo(),equipmentQuery.getPageSize());

return tMeasuringEquipmentDao.getPage(equipmentQuery);

}

@Override

public List<MeasuringEquipment> getByEquNo(String equNo) {

return tMeasuringEquipmentDao.selectByEquNo(equNo);

}

@Override

public List<MeasuringEquipment> getMeterDeviceListByEquipment(Long customerId, Integer materialId, List<String> equipmentNoList) {

return tMeasuringEquipmentDao.getMeterDeviceListByEquipment(customerId, materialId, equipmentNoList);

}

}

package com.bme.cloud.carbon.web;

import com.bme.cloud.carbon.service.LowCarbonService;

import com.bme.cloud.common.model.basic.Page;

import com.bme.cloud.common.model.carbon.\*;

import com.github.pagehelper.PageInfo;

import io.swagger.annotations.ApiImplicitParam;

import io.swagger.annotations.ApiImplicitParams;

import io.swagger.annotations.ApiOperation;

import io.swagger.models.auth.In;

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

import org.springframework.web.bind.annotation.\*;

import java.math.BigDecimal;

import java.util.List;

import java.util.Map;

@RestController

@RequestMapping("/api")

public class LowCarbonController {

@Autowired

private LowCarbonService lowCarbonService;

@GetMapping("/findExpendsList")

public List<Map<String, Object>> findExpendsList(@RequestParam("customerId") Long customerId, @RequestParam(value = "materialId", required = false) Long materialId) {

List<Map<String, Object>> list = lowCarbonService.findExpendsList(customerId, materialId);

return list;

}

@GetMapping("/getMeteingInfo")

public MeasuringEquipment getMeteingInfo(@RequestParam("deviceNo") String deviceNo) {

MeasuringEquipment mes = lowCarbonService.getMeteingInfo(deviceNo);

return mes;

}

@ApiOperation(value = "查询每种物料下显示每个批次对应的累积重量")

@GetMapping("/countByMaterialId")

@ApiImplicitParams({

@ApiImplicitParam(name = "materialId", value = "物料id", required = true, dataType = "Long", paramType = "query"),

@ApiImplicitParam(name = "carbonEmissionType", value = "碳排放类型", required = true, dataType = "Integer", paramType = "query"),

@ApiImplicitParam(name = "customerId", value = "客户ID", required = true, dataType = "Integer", paramType = "query")

})

public List<CarbonEmission> countByMaterialId(@RequestParam("materialId") Long materialId,

@RequestParam("carbonEmissionType") Integer carbonEmissionType,

@RequestParam("customerId") Long customerId) {

return lowCarbonService.countByMaterialId(materialId, carbonEmissionType, customerId);

}

@ApiOperation(value = "查询粗钢产量")

@GetMapping("/getSteelYield")

@ApiImplicitParams({

@ApiImplicitParam(name = "customerId", value = "客户id", required = true, dataType = "Long", paramType = "query"),

@ApiImplicitParam(name = "productName", value = "产品名称", required = true, dataType = "String", paramType = "query"),

@ApiImplicitParam(name = "startTime", value = "开始时间", required = true, dataType = "String", paramType = "query"),

@ApiImplicitParam(name = "endTime", value = "结束时间", required = true, dataType = "String", paramType = "query")

})

public List<SteelYieldData> getSteelYield(@RequestParam("customerId") Long customerId,

@RequestParam("productName") String productName,

@RequestParam("startTime") String startTime,

@RequestParam("endTime") String endTime) {

return lowCarbonService.getSteelYieldDataList(customerId, productName, startTime, endTime);

}

@GetMapping("/getSteelYield/params")

public List<SteelYieldData> getSteelYieldByParams(@RequestParam("customerId") Long customerId,

@RequestParam("productName") String productName,

@RequestParam("equipmentNo") String equipmentNo,

@RequestParam("startTime") String startTime,

@RequestParam("endTime") String endTime) {

return lowCarbonService.getSteelYieldDataListByParams(customerId, productName, equipmentNo, startTime, endTime);

}

@ApiOperation(value = "查询粗钢产量总产量")

@GetMapping("/getSumSteelYield")

@ApiImplicitParams({

@ApiImplicitParam(name = "customerId", value = "客户id", required = true, dataType = "Long", paramType = "query"),

@ApiImplicitParam(name = "productName", value = "产品名称", required = true, dataType = "String", paramType = "query"),

@ApiImplicitParam(name = "startTime", value = "开始时间", required = true, dataType = "String", paramType = "query"),

@ApiImplicitParam(name = "endTime", value = "结束时间", required = true, dataType = "String", paramType = "query")

})

public BigDecimal getSumSteelYield(@RequestParam("customerId") Long customerId,

@RequestParam("productName") String productName,

@RequestParam("startTime") String startTime,

@RequestParam("endTime") String endTime) {

return lowCarbonService.getSumSteelYieldDataList(customerId, productName, startTime, endTime);

}

@GetMapping("/getSteelYieldByPage")

@ApiImplicitParams({

@ApiImplicitParam(name = "customerId", value = "客户id", required = true, dataType = "Long", paramType = "query"),

@ApiImplicitParam(name = "productName", value = "产品名称", required = true, dataType = "String", paramType = "query"),

@ApiImplicitParam(name = "pageNo", value = "页码", required = true, dataType = "Integer", paramType = "query"),

@ApiImplicitParam(name = "pageSize", value = "每页条数", required = true, dataType = "Integer", paramType = "query")

})

public PageInfo<SteelYieldData> getSteelYieldByPage(@RequestParam("customerId") Long customerId,

@RequestParam(required = false, value = "productName") String productName,

@RequestParam("pageNo") Integer pageNo,

@RequestParam("pageSize") Integer pageSize) {

return lowCarbonService.getSteelYieldDataByPage(customerId, productName, pageNo, pageSize);

}

@ApiOperation(value = "产品(粗钢及其他)产量增加")

@PostMapping("/addSteelYield")

public int addSteelYield(@RequestBody SteelYieldData steelYieldData) {

return lowCarbonService.addSteelYield(steelYieldData);

}

@GetMapping("/findEquipmentList")

public List<Map<String, Object>> findEquipmentList(@RequestParam("customerId") Long customerId) {

List<Map<String, Object>> list = lowCarbonService.findEquipmentList(customerId);

return list;

}

@ApiOperation(value = "产品(粗钢及其他)产量修改")

@PostMapping("/updateSteelYield")

public int updateSteelYield(@RequestBody SteelYieldData steelYieldData) {

return lowCarbonService.updateSteelYield(steelYieldData);

}

@ApiOperation(value = "碳排放履约截止日期设置和碳排放额度配置-新增修改")

@PostMapping("/editPromises")

public void editPromises(@RequestBody CustConfig custConfig) {

lowCarbonService.editPromises(custConfig);

}

@ApiOperation(value = "行业吨钢碳CO2、吨钢NOx排放量、吨钢SO2、吨钢颗粒物排放量添加")

@PostMapping("/addEmissionLevel")

public void addEmissionLevel(@RequestBody IndustryTonsteelConfig industryTonsteelConfig) {

lowCarbonService.addEmissionLevel(industryTonsteelConfig);

}

@GetMapping("/getEmissionLevel")

public List<IndustryTonsteelConfig> getEmissionLevel() {

return lowCarbonService.getEmissionLevel();

}

@GetMapping("/findPollutionReducePlan")

public PageInfo<PollutionReducePlan> findPollutionReducePlan(@RequestParam("customerId") Long customerId,

@RequestParam(value = "startTime", required = false) String startTime,

@RequestParam(value = "endTime", required = false) String endTime,

@RequestParam(value = "processId", required = false) String processId,

@RequestParam("pageNo") Integer pageNo,

@RequestParam("pageSize") Integer pageSize) {

return lowCarbonService.findPollutionReducePlan(customerId, processId, startTime, endTime, pageNo, pageSize);

}

@GetMapping("/findProcess")

public List<Map<String, Object>> findProcess(@RequestParam("customerId") Long customerId) {

List<Map<String, Object>> list = lowCarbonService.findProcess(customerId);

return list;

}

@PostMapping("/saveOrUpateCarbonReduction")

public void saveOrUpateCarbonReduction(@RequestBody PollutionReducePlan pollutionReducePlan) {

lowCarbonService.saveOrUpateCarbonReduction(pollutionReducePlan);

}

@GetMapping("/findCustConfig")

public PageInfo<CustConfig> findCustConfig(@RequestParam("customerId") Long customerId,

@RequestParam("pageNo") Integer pageNo,

@RequestParam("pageSize") Integer pageSize) {

return lowCarbonService.findCustConfig(customerId, pageNo, pageSize);

}

@ApiOperation(value = "查询全年每月产值")

@GetMapping("/findProductivity")

@ApiImplicitParams({

@ApiImplicitParam(name = "customerId", value = "客户id", required = true, dataType = "Long", paramType = "query")

})

public List<CarbonView> findProductivity(@RequestParam("customerId") Long customerId, @RequestParam("year") String year) {

return lowCarbonService.findProductivity(customerId, year);

}

}

package com.bme.cloud.carbon.service;

import com.bme.cloud.carbon.repository.LowCarbonDao;

import com.bme.cloud.carbon.repository.OutputDao;

import com.bme.cloud.carbon.repository.SteelYieldDataDao;

import com.bme.cloud.carbon.repository.TruckScaleDataDao;

import com.bme.cloud.carbon.service.impl.BatchInvoiceImpl;

import com.bme.cloud.common.model.alarm.AlarmInfo;

import com.bme.cloud.common.model.basic.Page;

import com.bme.cloud.common.model.carbon.\*;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import io.swagger.models.auth.In;

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

import org.springframework.stereotype.Service;

import org.springframework.web.bind.annotation.RequestParam;

import java.math.BigDecimal;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

@Service

public class LowCarbonService {

@Autowired

private LowCarbonDao lowCarbonDao;

@Autowired

private TruckScaleDataDao truckScaleDataDao;

@Autowired

private BatchInvoiceImpl batchInvoiceImpl;

@Autowired

private SteelYieldDataDao steelYieldDataDao;

@Autowired

private OutputDao outputDao;

public List<Map<String, Object>> findExpendsList(Long customerId, Long materialId) {

return lowCarbonDao.findExpendsList(customerId, materialId);

}

public MeasuringEquipment getMeteingInfo(String deviceNo) {

return lowCarbonDao.getMeteingInfot(deviceNo);

}

public List<CarbonEmission> countByMaterialId(Long materialId, Integer carbonEmissionType, Long customerId) {

List<CarbonEmission> carbonEmissions = new ArrayList<>();

// 只要不是电力的则查询柱状图

if (carbonEmissionType != 3) {

List<String> batchIdList = truckScaleDataDao.selectBatchIds(customerId, materialId);

batchIdList.forEach(batchId -> {

CarbonEmission carbonEmission = truckScaleDataDao.countByMaterialId(materialId, customerId, batchId);

List<BatchInvoice> batchInvoices = batchInvoiceImpl.queryBatchInvoiceList(batchId, customerId);

carbonEmission.setInvoiceImgUrl(batchInvoices.stream().map(BatchInvoice::getBatchNo).collect(Collectors.toList()));

carbonEmissions.add(carbonEmission);

});

}

return carbonEmissions;

}

public List<SteelYieldData> getSteelYieldDataList(Long customerId, String productName, String startTime, String endTime) {

return steelYieldDataDao.getSteelYieldDataList(customerId, productName, startTime, endTime);

}

public List<SteelYieldData> getSteelYieldDataListByParams(Long customerId, String productName, String equipmentNo, String startTime, String endTime) {

return steelYieldDataDao.getSteelYieldDataList(customerId, productName, startTime, endTime);

}

public BigDecimal getSumSteelYieldDataList(Long customerId, String productName, String startTime, String endTime) {

return steelYieldDataDao.getSumSteelYieldDataList(customerId, productName, startTime, endTime);

}

public PageInfo<SteelYieldData> getSteelYieldDataByPage(Long customerId, String productName, Integer pageNo, Integer pageSize) {

PageHelper.startPage(pageNo, pageSize);

List<SteelYieldData> steelYieldDataList = steelYieldDataDao.getSteelYieldDataListByCustomerId(customerId, productName);

return new PageInfo<>(steelYieldDataList);

}

public int addSteelYield(SteelYieldData steelYieldData) {

return steelYieldDataDao.addSteelYieldData(steelYieldData);

}

public int updateSteelYield(SteelYieldData steelYieldData) {

return steelYieldDataDao.updateSteelYieldData(steelYieldData);

}

public void editPromises(CustConfig custConfig) {

Map<String, Object> cust = lowCarbonDao.custConfigCount(custConfig);

Integer count = Integer.parseInt(cust.get("count").toString());

if (count > 0) {

//修改

custConfig.setUpdateTime(new Date());

lowCarbonDao.updateCustConfig(custConfig);

} else {

//新增

custConfig.setUpdateTime(new Date());

custConfig.setCreateTime(new Date());

lowCarbonDao.saveCustConfig(custConfig);

}

}

public void addEmissionLevel(IndustryTonsteelConfig industryTonsteelConfig) {

if (industryTonsteelConfig.getId() == null) {

industryTonsteelConfig.setCreateTime(new Date());

industryTonsteelConfig.setUpdateTime(new Date());

lowCarbonDao.addEmissionLevel(industryTonsteelConfig);

} else {

industryTonsteelConfig.setUpdateTime(new Date());

lowCarbonDao.updateEmissionLevel(industryTonsteelConfig);

}

}

public List<IndustryTonsteelConfig> getEmissionLevel() {

return lowCarbonDao.getEmissionLevel();

}

public PageInfo<PollutionReducePlan> findPollutionReducePlan(Long customerId, String processId, String startTime, String endTime, Integer pageNo, Integer pageSize) {

PageHelper.startPage(pageNo, pageSize);

List<PollutionReducePlan> poll = lowCarbonDao.findPollutionReducePlan(customerId, processId, startTime, endTime);

return new PageInfo<>(poll);

}

public List<Map<String, Object>> findProcess(Long customerId) {

return lowCarbonDao.findProcess(customerId);

}

public void saveOrUpateCarbonReduction(PollutionReducePlan poll) {

if (poll.getId() == null) {

poll.setCreateTime(new Date());

poll.setUpdateTime(new Date());

lowCarbonDao.saveCarbonReduction(poll);

} else {

poll.setUpdateTime(new Date());

lowCarbonDao.upateCarbonReduction(poll);

}

}

public PageInfo<CustConfig> findCustConfig(Long customerId, Integer pageNo, Integer pageSize) {

PageHelper.startPage(pageNo, pageSize);

List<CustConfig> poll = lowCarbonDao.findCustConfig(customerId);

return new PageInfo<>(poll);

}

public List<Map<String, Object>> findEquipmentList(Long customerId) {

return steelYieldDataDao.findEquipmentList(customerId);

}

public List<CarbonView> findProductivity(Long customerId, String year) {

return outputDao.getProductivityByYear(customerId, year);

}

}

package com.bme.cloud.carbon.web;

import com.bme.cloud.carbon.service.CarbonFactorConfigService;

import com.bme.cloud.common.model.carbon.CarbonFactorConfig;

import com.bme.cloud.common.model.carbon.CarbonFactorInfo;

import com.bme.cloud.common.model.carbon.CarbonParams;

import com.bme.cloud.common.model.carbon.MaterialConfig;

import com.bme.cloud.common.model.carbon.ParamNameConfig;

import com.bme.cloud.common.support.PageParam;

import com.github.pagehelper.PageInfo;

import io.swagger.annotations.Api;

import io.swagger.annotations.ApiImplicitParam;

import io.swagger.annotations.ApiImplicitParams;

import io.swagger.annotations.ApiOperation;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

/\*\*

\*

\* @Description: 碳排计算参数表接口层

\* @Author: dongJue

\* @CreateDate: CarbonFactorConfig

\* @Version: V1.0

\*

\*/

@Api(description = "碳排计算参数表",value="碳排计算参数表" )

@RestController

@RequestMapping("/api/carbonFactor")

public class CarbonFactorConfigController {

Logger logger = LoggerFactory.getLogger(this.getClass());

@Autowired

public CarbonFactorConfigService carbonFactorConfigServiceImpl;

/\*\*

\* @explain 查询碳排计算参数表对象 <swagger GET请求>

\* @param 对象参数：id

\* @return tCarbonFactorConfig

\* @author dongJue

\* @time CarbonFactorConfig

\*/

@GetMapping("/getTCarbonFactorConfigById/{id}")

@ApiOperation(value = "获取碳排计算参数表信息", notes = "获取碳排计算参数表信息[tCarbonFactorConfig]，作者：dongJue")

@ApiImplicitParam(paramType="path", name = "id", value = "碳排计算参数表id", required = true, dataType = "Long")

public CarbonFactorConfig getTCarbonFactorConfigById(@PathVariable("id")Long id){

CarbonFactorConfig carbonFactorConfig = carbonFactorConfigServiceImpl.selectByPrimaryKey(id);

return carbonFactorConfig;

}

/\*\*

\* @explain 添加碳排计算参数表对象

\* @param 对象参数：tCarbonFactorConfig

\* @return Boolean

\* @author dongJue

\* @time CarbonFactorConfig

\*/

@PostMapping("/insertSelective")

@ApiOperation(value = "添加碳排计算参数表", notes = "添加碳排计算参数表[tCarbonFactorConfig],作者：dongJue")

public Boolean insertSelective(@RequestBody CarbonFactorConfig carbonFactorConfig){

int rg= carbonFactorConfigServiceImpl.insertSelective(carbonFactorConfig);

if (rg>0) {

return Boolean.TRUE;

}

return Boolean.FALSE;

}

/\*\*

\* @explain 删除碳排计算参数表对象

\* @param 对象参数：id

\* @return Boolean

\* @author dongJue

\* @time CarbonFactorConfig

\*/

@PostMapping("/deleteByPrimaryKey")

@ApiOperation(value = "删除碳排计算参数表", notes = "删除碳排计算参数表,作者：dongJue")

@ApiImplicitParam(paramType="query", name = "id", value = "碳排计算参数表id", required = true, dataType = "Long")

public Boolean deleteByPrimaryKey(@RequestParam("id") Long id){

int reg= carbonFactorConfigServiceImpl.deleteByPrimaryKey(id);

if (reg>0) {

return Boolean.TRUE;

}

return Boolean.FALSE;

}

/\*\*

\* @explain 修改碳排计算参数表对象

\* @param 对象参数：tCarbonFactorConfig

\* @return tCarbonFactorConfig

\* @author dongJue

\* @time CarbonFactorConfig

\*/

@ApiOperation(value = "修改碳排计算参数表", notes = "修改碳排计算参数表[tCarbonFactorConfig],作者：dongJue")

@PostMapping("/updateByPrimaryKeySelective")

public Boolean updateByPrimaryKeySelective(@RequestBody CarbonFactorConfig carbonFactorConfig){

int reg = carbonFactorConfigServiceImpl.updateByPrimaryKeySelective(carbonFactorConfig);

if (reg>0) {

return Boolean.TRUE;

}

return Boolean.FALSE;

}

/\*\*

\* @explain 获取匹配碳排计算参数表

\* @param 对象参数：tCarbonFactorConfig

\* @return List<TCarbonFactorConfig>

\* @author dongJue

\* @time CarbonFactorConfig

\*/

@ApiOperation(value = "条件查询碳排计算参数表", notes = "条件查询[tCarbonFactorConfig],作者：dongJue")

@PostMapping("/queryTCarbonFactorConfigList")

public List<CarbonFactorConfig> queryTCarbonFactorConfigList(@RequestBody CarbonFactorConfig carbonFactorConfig){

List<CarbonFactorConfig> list = carbonFactorConfigServiceImpl.queryTCarbonFactorConfigList(carbonFactorConfig);

return list;

}

/\*\*

\* @explain 分页条件查询碳排计算参数表

\* @param 对象参数：AppPage<TCarbonFactorConfig>

\* @return PageInfo<TCarbonFactorConfig>

\* @author dongJue

\* @time CarbonFactorConfig

\*/

@GetMapping("/getPageTCarbonFactorConfig/{pageNum}/{pageSize}")

@ApiOperation(value = "分页查询", notes = "分页查询返回对象[PageInfo<TCarbonFactorConfig>],作者：边鹏")

@ApiImplicitParams({

@ApiImplicitParam(paramType="path", name = "pageNum", value = "当前页", required = true, dataType = "int"),

@ApiImplicitParam(paramType="path", name = "pageSize", value = "页行数", required = true, dataType = "int")

})

public PageInfo<CarbonFactorConfig> getTCarbonFactorConfigBySearch(@PathVariable("pageNum") Integer pageNum,@PathVariable("pageSize") Integer pageSize, @ModelAttribute CarbonFactorConfig config){

PageParam<CarbonFactorConfig> page =new PageParam<CarbonFactorConfig>();

page.setPageNum(pageNum);

page.setPageSize(pageSize);

//其他参数

page.setParam(config);

//分页数据

PageInfo<CarbonFactorConfig> pageInfo = carbonFactorConfigServiceImpl.getTCarbonFactorConfigBySearch(page);

return pageInfo;

}

@GetMapping("/fuelType/get")

public List<MaterialConfig> getFuelTypeList(){

return carbonFactorConfigServiceImpl.getFuelTypeList();

}

@GetMapping("/paramName/list")

public List<ParamNameConfig> getParamNameConfig(){

return carbonFactorConfigServiceImpl.getParamNameConfigList();

}

@GetMapping("/factor/list")

public PageInfo<CarbonFactorConfig> getFactorListByPage(@RequestParam("customerId") Long customerId, @RequestParam("pageNo") Integer pageNo, @RequestParam("pageSize") Integer pageSize){

return carbonFactorConfigServiceImpl.getFactorListByPage(customerId, pageNo, pageSize);

}

@GetMapping("/factor/listAll")

public List<CarbonFactorInfo> getFactorListByAll(){

return carbonFactorConfigServiceImpl.getFactorListAll();

}

@GetMapping("/param/list")

public List<CarbonParams> getCarbonParams(@RequestParam("factorId") Integer factorId){

return carbonFactorConfigServiceImpl.getCarbonParams(factorId);

}

@PostMapping("/factor/add")

private int addFactorAndParams(@RequestBody CarbonFactorInfo carbonFactorInfo){

return carbonFactorConfigServiceImpl.addFactorAndParams(carbonFactorInfo);

}

@PostMapping("/factor/update")

private int updateFactorAndParams(@RequestBody CarbonFactorInfo carbonFactorInfo){

return carbonFactorConfigServiceImpl.updateFactorAndParams(carbonFactorInfo);

}

@GetMapping("/factor/deleteParams")

public void deleteParams(@RequestParam("id") Integer id){

carbonFactorConfigServiceImpl.getCarbonParams(id);

}

}

/\*\*

\* @filename:TCarbonFactorConfigServiceImpl CarbonFactorConfig

\* @project bme-low-carbon-service V1.0

\* Copyright(c) 2021 dongJue Co. Ltd.

\* All right reserved.

\*/

package com.bme.cloud.carbon.service.impl;

import cn.hutool.core.collection.CollectionUtil;

import cn.hutool.core.date.DateUtil;

import cn.hutool.core.util.StrUtil;

import com.bme.cloud.carbon.repository.CarbonFactorConfigDao;

import com.bme.cloud.carbon.service.CarbonFactorConfigService;

import com.bme.cloud.carbon.service.EsService;

import com.bme.cloud.common.constant.ESCarbonIndexConstants;

import com.bme.cloud.common.model.carbon.CarbonFactorConfig;

import com.bme.cloud.common.model.carbon.CarbonFactorInfo;

import com.bme.cloud.common.model.carbon.CarbonParams;

import com.bme.cloud.common.model.carbon.MaterialConfig;

import com.bme.cloud.common.model.carbon.ParamNameConfig;

import com.bme.cloud.common.model.es.CarbonRelation;

import com.bme.cloud.common.model.es.EsCarbonFactorConfig;

import com.bme.cloud.common.model.es.EsCarbonParams;

import com.bme.cloud.common.support.PageParam;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import org.springframework.beans.BeanUtils;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Objects;

/\*\*

\*

\* @Description: 碳排计算参数表——SERVICEIMPL

\* @Author: dongJue

\* @CreateDate: CarbonFactorConfig

\* @Version: V1.0

\*

\*/

@Service

public class CarbonFactorConfigServiceImpl implements CarbonFactorConfigService {

@Autowired

public CarbonFactorConfigDao tCarbonFactorConfigDao;

@Autowired

private EsService esService;

//查询对象

@Override

public CarbonFactorConfig selectByPrimaryKey(Long id) {

return tCarbonFactorConfigDao.selectByPrimaryKey(id);

}

//删除对象

@Override

public int deleteByPrimaryKey(Long id) {

return tCarbonFactorConfigDao.deleteByPrimaryKey(id);

}

//添加对象

@Override

public int insertSelective(CarbonFactorConfig carbonFactorConfig) {

return tCarbonFactorConfigDao.insertSelective(carbonFactorConfig);

}

//修改对象

@Override

public int updateByPrimaryKeySelective(CarbonFactorConfig carbonFactorConfig) {

return tCarbonFactorConfigDao.updateByPrimaryKeySelective(carbonFactorConfig);

}

//查询集合

@Override

public List<CarbonFactorConfig> queryTCarbonFactorConfigList(CarbonFactorConfig carbonFactorConfig) {

return tCarbonFactorConfigDao.queryTCarbonFactorConfigList(carbonFactorConfig);

}

//分页查询

@Override

public PageInfo<CarbonFactorConfig> getTCarbonFactorConfigBySearch(PageParam<CarbonFactorConfig> page) {

// TODO Auto-generated method stub

PageHelper.startPage(page.getPageNum(),page.getPageSize());

List<CarbonFactorConfig> list=tCarbonFactorConfigDao.queryTCarbonFactorConfigList(page.getParam());

PageInfo<CarbonFactorConfig> pageInfo = new PageInfo<CarbonFactorConfig>(list);

return pageInfo;

}

@Override

public List<MaterialConfig> getFuelTypeList() {

return tCarbonFactorConfigDao.getFuelTypeList();

}

@Override

public List<ParamNameConfig> getParamNameConfigList() {

return tCarbonFactorConfigDao.getParamNameConfigList();

}

@Override

public PageInfo<CarbonFactorConfig> getFactorListByPage(Long customerId, Integer pageNo, Integer pageSize) {

PageHelper.startPage(pageNo, pageSize);

List<CarbonFactorConfig> factorList = tCarbonFactorConfigDao.getFactorList(customerId);

return new PageInfo<>(factorList);

}

@Override

public List<CarbonFactorInfo> getFactorListAll() {

List<CarbonFactorInfo> carbonFactorInfoList = new ArrayList<>();

List<CarbonFactorConfig> factorListAll = tCarbonFactorConfigDao.getFactorListAll();

if (!CollectionUtil.isEmpty(factorListAll)){

factorListAll.forEach(carbonFactorConfig -> {

CarbonFactorInfo carbonFactorInfo = new CarbonFactorInfo();

carbonFactorInfo.setCarbonFactorId(carbonFactorConfig.getId());

carbonFactorInfo.setMaterialId(carbonFactorConfig.getMaterialId());

carbonFactorInfo.setMaterialName(carbonFactorConfig.getMaterialName());

carbonFactorInfo.setCustomerId(carbonFactorConfig.getCustomerId());

carbonFactorInfo.setMaterialUnit(carbonFactorConfig.getMaterialUnit());

carbonFactorInfo.setMaterialType(carbonFactorConfig.getMaterialType());

carbonFactorInfo.setEmissionType(carbonFactorConfig.getEmissionType());

carbonFactorInfo.setCreateTime(carbonFactorConfig.getCreateTime());

carbonFactorInfo.setUpdateTime(carbonFactorConfig.getUpdateTime());

List<CarbonParams> carbonParams = tCarbonFactorConfigDao.getCarbonParams(carbonFactorConfig.getId());

if (!CollectionUtil.isEmpty(carbonParams)){

carbonFactorInfo.setCarbonParamsList(carbonParams);

}

carbonFactorInfoList.add(carbonFactorInfo);

});

}

return carbonFactorInfoList;

}

@Override

public List<CarbonParams> getCarbonParams(Integer factorId) {

return tCarbonFactorConfigDao.getCarbonParams(factorId);

}

@Override

@Transactional(rollbackFor = Exception.class)

public int addFactorAndParams(CarbonFactorInfo carbonFactorInfo) {

CarbonFactorConfig carbonFactorConfig = new CarbonFactorConfig();

carbonFactorConfig.setEmissionType(carbonFactorInfo.getEmissionType());

carbonFactorConfig.setCustomerId(carbonFactorInfo.getCustomerId());

carbonFactorConfig.setCreateTime(carbonFactorInfo.getCreateTime());

carbonFactorConfig.setMaterialId(carbonFactorInfo.getMaterialId());

carbonFactorConfig.setMaterialName(carbonFactorInfo.getMaterialName());

carbonFactorConfig.setMaterialType(carbonFactorInfo.getMaterialType());

carbonFactorConfig.setMaterialUnit(carbonFactorInfo.getMaterialUnit());

//先去做校验是否存在--存在则提示

int count=Integer.parseInt(tCarbonFactorConfigDao.countFactorInfo(carbonFactorConfig).get("count").toString());

if(count>0){

return 99999;

}

int result = tCarbonFactorConfigDao.addCarbonFactorInfo(carbonFactorConfig);

List<CarbonParams> carbonParamsList = carbonFactorInfo.getCarbonParamsList();

List<CarbonParams> addCarbonParamsList = new ArrayList<>();

if (!CollectionUtil.isEmpty(carbonParamsList)){

carbonParamsList.forEach(carbonParams -> {

if (!StrUtil.isEmpty(carbonParams.getParamNo())){

CarbonParams addCarbonParams = new CarbonParams();

BeanUtils.copyProperties(carbonParams, addCarbonParams);

ParamNameConfig paramNameConfig = tCarbonFactorConfigDao.getParamNameConfigByParamNo(carbonParams.getParamNo());

addCarbonParams.setParamName(paramNameConfig.getParamName());

addCarbonParams.setCarbonFactorId(carbonFactorConfig.getId());

addCarbonParams.setCreateTime(new Date());

addCarbonParamsList.add(addCarbonParams);

}

});

if (!CollectionUtil.isEmpty(addCarbonParamsList)){

result = tCarbonFactorConfigDao.addCarbonParams(addCarbonParamsList);

}

}

// 更新es

this.updateOrAddToEs(carbonFactorConfig, addCarbonParamsList);

return result;

}

@Override

@Transactional(rollbackFor = Exception.class)

public int updateFactorAndParams(CarbonFactorInfo carbonFactorInfo) {

CarbonFactorConfig carbonFactorConfig = new CarbonFactorConfig();

carbonFactorConfig.setEmissionType(carbonFactorInfo.getEmissionType());

carbonFactorConfig.setCustomerId(carbonFactorInfo.getCustomerId());

carbonFactorConfig.setCreateTime(carbonFactorInfo.getCreateTime());

carbonFactorConfig.setMaterialId(carbonFactorInfo.getMaterialId());

carbonFactorConfig.setMaterialUnit(carbonFactorInfo.getMaterialUnit());

carbonFactorConfig.setMaterialName(carbonFactorInfo.getMaterialName());

carbonFactorConfig.setMaterialType(carbonFactorInfo.getMaterialType());

carbonFactorConfig.setUpdateTime(new Date());

carbonFactorConfig.setId(carbonFactorInfo.getCarbonFactorId());

int result = tCarbonFactorConfigDao.updateCarbonFactorInfo(carbonFactorConfig);

List<CarbonParams> carbonParamsList = carbonFactorInfo.getCarbonParamsList();

tCarbonFactorConfigDao.deleteCarbonFactorId(carbonFactorInfo.getCarbonFactorId());

List<CarbonParams> addCarbonParamsList = new ArrayList<>();

if (!CollectionUtil.isEmpty(carbonParamsList)){

carbonParamsList.forEach(carbonParams -> {

ParamNameConfig paramNameConfig = tCarbonFactorConfigDao.getParamNameConfigByParamNo(carbonParams.getParamNo());

carbonParams.setParamName(paramNameConfig.getParamName());

carbonParams.setCarbonFactorId(carbonFactorConfig.getId());

Date createTime = carbonParams.getCreateTime();

if (createTime == null){

createTime = new Date();

}

carbonParams.setCreateTime(createTime);

carbonParams.setUpdateTime(new Date());

addCarbonParamsList.add(carbonParams);

});

if (!CollectionUtil.isEmpty(addCarbonParamsList)){

result = tCarbonFactorConfigDao.addCarbonParams(addCarbonParamsList);

}

}

this.updateOrAddToEs(carbonFactorConfig, addCarbonParamsList);

return result;

}

@Override

public void deleteParams(Integer id) {

tCarbonFactorConfigDao.deleteParams(id);

}

private void updateOrAddToEs(CarbonFactorConfig carbonFactorConfig, List<CarbonParams> addCarbonParamsList){

EsCarbonFactorConfig queryResult = esService.query(carbonFactorConfig.getCustomerId(), carbonFactorConfig.getMaterialName(), ESCarbonIndexConstants.CARBON\_FACTOR, EsCarbonFactorConfig.class);

if (queryResult == null){

EsCarbonFactorConfig esCarbonFactorConfig = this.buildCreateEsCarbonFactorConfig(carbonFactorConfig);

esService.createNewDocument(ESCarbonIndexConstants.CARBON\_FACTOR, esCarbonFactorConfig);

} else {

if (!Objects.equals(queryResult.getId(), carbonFactorConfig.getId())){

esService.deleteDocumentInfo(ESCarbonIndexConstants.CARBON\_FACTOR, queryResult.getId().toString());

EsCarbonFactorConfig esCarbonFactorConfig = this.buildCreateEsCarbonFactorConfig(carbonFactorConfig);

esService.createNewDocument(ESCarbonIndexConstants.CARBON\_FACTOR, esCarbonFactorConfig);

} else {

EsCarbonFactorConfig esCarbonFactorConfig = this.buildUpdateEsCarbonFactorConfig(carbonFactorConfig);

esService.updateDocumentInfo(ESCarbonIndexConstants.CARBON\_FACTOR, esCarbonFactorConfig, esCarbonFactorConfig.getId().toString());

}

}

Map<String, EsCarbonParams> paramNameMap = new HashMap<>();

List<EsCarbonParams> esCarbonParamsListFromEs = esService.queryByCarbonFactorId(ESCarbonIndexConstants.CARBON\_FACTOR, queryResult.getId().toString(), EsCarbonParams.class);

if (!CollectionUtil.isEmpty(esCarbonParamsListFromEs)){

esCarbonParamsListFromEs.forEach(esCarbonParams -> paramNameMap.put(esCarbonParams.getParamName(), esCarbonParams));

}

if (!CollectionUtil.isEmpty(addCarbonParamsList)){

addCarbonParamsList.forEach(carbonParams -> {

EsCarbonParams esCarbonParamsTmp = paramNameMap.get(carbonParams.getParamName());

if (esCarbonParamsTmp == null){

EsCarbonParams createEsCarbonParams = this.buildCreateEsCarbonParams(carbonParams);

esService.createNewDocumentWithRoutingParams(ESCarbonIndexConstants.CARBON\_FACTOR, createEsCarbonParams, carbonFactorConfig.getId().toString());

} else {

if (!Objects.equals(esCarbonParamsTmp.getCarbonFactorId(), carbonParams.getCarbonFactorId())){

esService.deleteDocumentInfo(ESCarbonIndexConstants.CARBON\_FACTOR, esCarbonParamsTmp.getId());

EsCarbonParams createEsCarbonParams = this.buildCreateEsCarbonParams(carbonParams);

esService.createNewDocumentWithRoutingParams(ESCarbonIndexConstants.CARBON\_FACTOR, createEsCarbonParams, carbonFactorConfig.getId().toString());

} else {

EsCarbonParams updateEsCarbonParams = this.buildUpdateEsCarbonParams(carbonParams);

updateEsCarbonParams.setId(esCarbonParamsTmp.getId());

esService.updateDocumentInfo(ESCarbonIndexConstants.CARBON\_FACTOR, updateEsCarbonParams, updateEsCarbonParams.getId());

paramNameMap.remove(carbonParams.getParamName());

}

}

});

}

if (!CollectionUtil.isEmpty(paramNameMap)){

paramNameMap.forEach((k,v)-> esService.deleteDocumentInfo(ESCarbonIndexConstants.CARBON\_FACTOR, v.getId()));

}

}

private EsCarbonFactorConfig buildCreateEsCarbonFactorConfig(CarbonFactorConfig carbonFactorConfig){

EsCarbonFactorConfig configParent = new EsCarbonFactorConfig();

configParent.setId(carbonFactorConfig.getId());

configParent.setCreateTime(DateUtil.format(carbonFactorConfig.getCreateTime(), "yyyy-MM-dd HH:mm:ss"));

configParent.setCustomerId(carbonFactorConfig.getCustomerId());

configParent.setEmissionType(carbonFactorConfig.getEmissionType());

configParent.setMaterialId(carbonFactorConfig.getMaterialId());

configParent.setMaterialName(carbonFactorConfig.getMaterialName());

configParent.setMaterialType(carbonFactorConfig.getMaterialType());

configParent.setMaterialUnit(carbonFactorConfig.getMaterialUnit());

CarbonRelation carbonRelationParent = new CarbonRelation();

carbonRelationParent.setName(carbonRelationParent.getParentName());

configParent.setCarbonRelation(carbonRelationParent);

return configParent;

}

private EsCarbonFactorConfig buildUpdateEsCarbonFactorConfig(CarbonFactorConfig carbonFactorConfig){

EsCarbonFactorConfig configParent = new EsCarbonFactorConfig();

configParent.setId(carbonFactorConfig.getId());

configParent.setCreateTime(DateUtil.format(carbonFactorConfig.getCreateTime(), "yyyy-MM-dd HH:mm:ss"));

configParent.setCustomerId(carbonFactorConfig.getCustomerId());

configParent.setEmissionType(carbonFactorConfig.getEmissionType());

configParent.setMaterialId(carbonFactorConfig.getMaterialId());

configParent.setMaterialName(carbonFactorConfig.getMaterialName());

configParent.setMaterialType(carbonFactorConfig.getMaterialType());

configParent.setMaterialUnit(carbonFactorConfig.getMaterialUnit());

return configParent;

}

private EsCarbonParams buildCreateEsCarbonParams(CarbonParams carbonParams){

EsCarbonParams esCarbonParams = new EsCarbonParams();

esCarbonParams.setParamId(carbonParams.getId());

esCarbonParams.setCarbonFactorId(carbonParams.getCarbonFactorId());

esCarbonParams.setParamNo(carbonParams.getParamNo());

esCarbonParams.setParamName(carbonParams.getParamName());

esCarbonParams.setParamUnit(carbonParams.getParamUnit());

esCarbonParams.setParamValue(carbonParams.getParamValue());

esCarbonParams.setCreateTime(DateUtil.format(carbonParams.getCreateTime(), "yyyy-MM-dd HH:mm:ss"));

esCarbonParams.setUpdateTime(DateUtil.format(carbonParams.getUpdateTime(), "yyyy-MM-dd HH:mm:ss"));

CarbonRelation carbonRelationChild = new CarbonRelation();

carbonRelationChild.setName(carbonRelationChild.getChildName());

carbonRelationChild.setParent(String.valueOf(carbonParams.getCarbonFactorId()));

esCarbonParams.setCarbonRelation(carbonRelationChild);

return esCarbonParams;

}

private EsCarbonParams buildUpdateEsCarbonParams(CarbonParams carbonParams){

EsCarbonParams esCarbonParams = new EsCarbonParams();

esCarbonParams.setParamId(carbonParams.getId());

esCarbonParams.setCarbonFactorId(carbonParams.getCarbonFactorId());

esCarbonParams.setParamNo(carbonParams.getParamNo());

esCarbonParams.setParamName(carbonParams.getParamName());

esCarbonParams.setParamUnit(carbonParams.getParamUnit());

esCarbonParams.setParamValue(carbonParams.getParamValue());

esCarbonParams.setCreateTime(DateUtil.format(carbonParams.getCreateTime(), "yyyy-MM-dd HH:mm:ss"));

esCarbonParams.setUpdateTime(DateUtil.format(carbonParams.getUpdateTime(), "yyyy-MM-dd HH:mm:ss"));

return esCarbonParams;

}

}

package com.bme.screen.module.carbon.web;

import cn.hutool.core.collection.CollectionUtil;

import cn.hutool.core.util.NumberUtil;

import com.bme.cloud.common.constant.CarbonConstants;

import com.bme.cloud.common.model.carbon.\*;

import com.bme.cloud.common.support.CommonResult;

import com.bme.cloud.common.util.DateUtils;

import com.bme.screen.module.carbon.feign.LowCarbonService;

import com.bme.screen.module.carbon.feign.LowCarbonStatisticalService;

import com.bme.screen.module.carbon.model.ConsumedMaterialComparePower;

import com.bme.screen.module.carbon.model.ConsumedMaterialInfo;

import com.bme.screen.module.carbon.service.CarbonEmissionService;

import com.bme.screen.module.carbon.service.CarbonService;

import com.bme.screen.module.carbon.service.EquipmentService;

import com.bme.screen.module.carbon.service.HistoryService;

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.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;

import java.math.BigDecimal;

import java.math.RoundingMode;

import java.text.SimpleDateFormat;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.ZoneOffset;

import java.time.format.DateTimeFormatter;

import java.time.temporal.TemporalAdjusters;

import java.util.\*;

import java.util.function.Function;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/carbon")

public class LowCarbonController {

@Autowired

private LowCarbonService lowCarbonService;

@Autowired

private LowCarbonStatisticalService lowCarbonStatisticalService;

@Resource

HistoryService historyService;

@Autowired

private CarbonService carbonService;

@Autowired

private EquipmentService equipmentService;

@Autowired

private CarbonEmissionService carbonEmissionService;

SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");

@ApiOperation("进厂物料和消耗物料对比")

@GetMapping("/getMaterialsCompare1")

public CommonResult getMaterialsCompare1(@RequestParam("customerId") Long customerId, @RequestParam(value = "materialId", required = false) Long materialId) {

/\* //获取进厂物料

List<Map<String,Object>> xpendsListe =lowCarbonService.findExpendsList(customerId,materialId);

//获取消耗物料

List<Map<String,Object>> intoMaterialsList=lowCarbonStatisticalService.findIntoMaterialsList(customerId,materialId);

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

map.put("customerId",customerId);

map.put("intoMaterialsList",intoMaterialsList);

map.put("expendsList",xpendsListe);\*/

return CommonResult.success("查询成功");

}

@ApiOperation("进厂物料和消耗物料对比")

@GetMapping("/getMaterialsCompare")

public CommonResult getMaterialsCompare(@RequestParam("customerId") Long customerId, @RequestParam(value = "materialId", required = false) Long materialId) {

DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

LocalDateTime localDateTime = LocalDateTime.now();

String startTime = localDateTime.with(TemporalAdjusters.firstDayOfMonth()).format(dateTimeFormatter);

String endTime = localDateTime.with(TemporalAdjusters.firstDayOfMonth()).plusMonths(1).format(dateTimeFormatter);

//获取进厂物料

List<ConsumedMaterialDayData> xpendsList = lowCarbonStatisticalService.findIntoMaterialsList(customerId, materialId, startTime, endTime, 1);

Map<String, ConsumedMaterialDayData> xpendsListMap = xpendsList.stream().collect(Collectors.toMap(ConsumedMaterialDayData::getDate, Function.identity(), (k1, k2) -> k1));

int monthOfDays = LocalDate.now().lengthOfMonth();

LocalDate firstDayOfMonth = LocalDate.now().with(TemporalAdjusters.firstDayOfMonth());

DateTimeFormatter dateTimeFormatter2 = DateTimeFormatter.ofPattern("yyyy-MM-dd");

List<Map<String, Object>> xpendsListNew = new ArrayList<>();//进场

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

String date = firstDayOfMonth.plusDays(i).format(dateTimeFormatter2);

ConsumedMaterialDayData consumedMaterialDayData = xpendsListMap.get(date);

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

if (consumedMaterialDayData != null) {

consumedMaterialObjectMap.put(i + 1 + "", consumedMaterialDayData.getTotal());

} else {

consumedMaterialObjectMap.put(i + 1 + "", "");

}

xpendsListNew.add(consumedMaterialObjectMap);

}

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

map.put("customerId", customerId);

map.put("expendsList", xpendsListNew);

List<ConsumedMaterialDayData> intoMaterialsList = lowCarbonStatisticalService.findIntoMaterialsList(customerId, materialId, startTime, endTime, 2);

Map<String, ConsumedMaterialDayData> intoMaterialsListMap = intoMaterialsList.stream().collect(Collectors.toMap(ConsumedMaterialDayData::getDate, Function.identity(), (k1, k2) -> k1));

List<Map<String, Object>> intoMaterialsListNew = new ArrayList<>();//进场

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

String date = firstDayOfMonth.plusDays(i).format(dateTimeFormatter2);

ConsumedMaterialDayData intoMaterialsListObjectMapDayData = intoMaterialsListMap.get(date);

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

if (intoMaterialsListObjectMapDayData != null) {

intoMaterialsListObjectMap.put(i + 1 + "", intoMaterialsListObjectMapDayData.getTotal());

} else {

intoMaterialsListObjectMap.put(i + 1 + "", "");

}

intoMaterialsListNew.add(intoMaterialsListObjectMap);

}

map.put("intoMaterialsList", intoMaterialsListNew);

return CommonResult.success("查询成功", map);

}

@ApiOperation("消耗物料信息")

@GetMapping("/getConsumedMaterial")

public CommonResult getConsumedMaterial(@RequestParam("customerId") Long customerId, @RequestParam("materialId") Integer materialId) {

DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

LocalDateTime localDateTime = LocalDateTime.now();

String startTime = localDateTime.with(TemporalAdjusters.firstDayOfMonth()).format(dateTimeFormatter);

String endTime = localDateTime.with(TemporalAdjusters.firstDayOfMonth()).plusMonths(1).format(dateTimeFormatter);

ConsumedMaterialInfo consumedMaterialInfo = new ConsumedMaterialInfo();

//获取消耗物料

List<ConsumedMaterialDayData> consumedMaterialList = lowCarbonStatisticalService.getConsumedMaterial(customerId, materialId, startTime, endTime);

Map<String, ConsumedMaterialDayData> consumedMaterialMap = consumedMaterialList.stream().collect(Collectors.toMap(ConsumedMaterialDayData::getDate, Function.identity(), (k1, k2) -> k1));

List<SteelYieldData> steelYieldList = lowCarbonService.getSteelYield(customerId, CarbonConstants.CRUDE\_STEEL\_NAME, startTime, endTime);

Map<String, SteelYieldData> steelYieldMap = new HashMap<>();

if (!CollectionUtil.isEmpty(steelYieldList)) {

steelYieldMap = steelYieldList.stream().collect(Collectors.toMap(SteelYieldData::getRegisterDate, Function.identity(), (k1, k2) -> k1));

}

int monthOfDays = LocalDate.now().lengthOfMonth();

LocalDate firstDayOfMonth = LocalDate.now().with(TemporalAdjusters.firstDayOfMonth());

DateTimeFormatter dateTimeFormatter2 = DateTimeFormatter.ofPattern("yyyy-MM-dd");

List<Map<Integer, Object>> consumedMaterialResultList = new ArrayList<>();

List<Map<Integer, Object>> steelYieldResultList = new ArrayList<>();

List<Map<Integer, Object>> tonSteelConsumptionList = new ArrayList<>();

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

String date = firstDayOfMonth.plusDays(i).format(dateTimeFormatter2);

ConsumedMaterialDayData consumedMaterialDayData = consumedMaterialMap.get(date);

Map<Integer, Object> consumedMaterialObjectMap = new HashMap<>();

if (consumedMaterialDayData != null) {

consumedMaterialObjectMap.put(i + 1, consumedMaterialDayData.getTotal());

} else {

consumedMaterialObjectMap.put(i + 1, "");

}

consumedMaterialResultList.add(consumedMaterialObjectMap);

SteelYieldData steelYieldData = steelYieldMap.get(date);

Map<Integer, Object> steelYieldObjectMap = new HashMap<>();

if (steelYieldData != null) {

steelYieldObjectMap.put(i + 1, steelYieldData.getSteelYield());

} else {

steelYieldObjectMap.put(i + 1, "");

}

steelYieldResultList.add(steelYieldObjectMap);

Map<Integer, Object> tonSteelConsumptionObjectMap = new HashMap<>();

if (steelYieldData != null && consumedMaterialDayData != null) {

tonSteelConsumptionObjectMap.put(i + 1, NumberUtil.div(consumedMaterialDayData.getTotal(), steelYieldData.getSteelYield(), 2));

} else {

tonSteelConsumptionObjectMap.put(i + 1, "");

}

tonSteelConsumptionList.add(tonSteelConsumptionObjectMap);

}

consumedMaterialInfo.setConsumedMaterialList(consumedMaterialResultList);

consumedMaterialInfo.setSteelYieldList(steelYieldResultList);

consumedMaterialInfo.setTonSteelConsumptionList(tonSteelConsumptionList);

return CommonResult.success("查询成功", consumedMaterialInfo);

}

@ApiOperation("消耗物料信息/消耗电量")

@GetMapping("/getConsumedMaterialComparePower")

public CommonResult getConsumedMaterialComparePower(@RequestParam("customerId") Long customerId, @RequestParam("materialId") Integer materialId) {

DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

LocalDateTime localDateTime = LocalDateTime.now();

String startTime = localDateTime.with(TemporalAdjusters.firstDayOfMonth()).format(dateTimeFormatter);

String endTime = localDateTime.with(TemporalAdjusters.firstDayOfMonth()).plusMonths(1).format(dateTimeFormatter);

ConsumedMaterialComparePower consumedMaterialComparePower = new ConsumedMaterialComparePower();

//获取消耗物料

List<ConsumedMaterialDayData> consumedMaterialList = lowCarbonStatisticalService.getConsumedMaterial(customerId, materialId, startTime, endTime);

Map<String, ConsumedMaterialDayData> consumedMaterialMap = consumedMaterialList.stream().collect(Collectors.toMap(ConsumedMaterialDayData::getDate, Function.identity(), (k1, k2) -> k1));

List<ConsumedMaterialDayData> consumedPowerList = lowCarbonStatisticalService.getConsumedPower(customerId, 2, startTime, endTime);

Map<String, Double> consumedPowerValueMap = new HashMap<>();

if (!CollectionUtil.isEmpty(consumedPowerList)) {

Map<String, List<ConsumedMaterialDayData>> consumedPowerMap = consumedPowerList.stream().collect(Collectors.groupingBy(ConsumedMaterialDayData::getDate));

consumedPowerMap.forEach((k, v) -> {

Double powerSum = v.stream().mapToDouble(ConsumedMaterialDayData::getTotal).sum();

consumedPowerValueMap.put(k, powerSum);

});

}

int monthOfDays = LocalDate.now().lengthOfMonth();

LocalDate firstDayOfMonth = LocalDate.now().with(TemporalAdjusters.firstDayOfMonth());

DateTimeFormatter dateTimeFormatter2 = DateTimeFormatter.ofPattern("yyyy-MM-dd");

List<Map<Integer, Object>> consumedMaterialResultList = new ArrayList<>();

List<Map<Integer, Object>> consumedMaterialComparePowerList = new ArrayList<>();

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

String date = firstDayOfMonth.plusDays(i).format(dateTimeFormatter2);

ConsumedMaterialDayData consumedMaterialDayData = consumedMaterialMap.get(date);

Map<Integer, Object> consumedMaterialObjectMap = new HashMap<>();

if (consumedMaterialDayData != null) {

consumedMaterialObjectMap.put(i + 1, consumedMaterialDayData.getTotal());

} else {

consumedMaterialObjectMap.put(i + 1, "");

}

consumedMaterialResultList.add(consumedMaterialObjectMap);

Double powerValue = consumedPowerValueMap.get(date);

Map<Integer, Object> consumedMaterialComparePowerObjectMap = new HashMap<>();

if (powerValue != null && consumedMaterialDayData != null) {

consumedMaterialComparePowerObjectMap.put(i + 1, NumberUtil.div(consumedMaterialDayData.getTotal(), powerValue, 2));

} else {

consumedMaterialComparePowerObjectMap.put(i + 1, "");

}

consumedMaterialComparePowerList.add(consumedMaterialComparePowerObjectMap);

}

consumedMaterialComparePower.setConsumedMaterialList(consumedMaterialResultList);

consumedMaterialComparePower.setConsumedMaterialComparePowerList(consumedMaterialComparePowerList);

return CommonResult.success("查询成功", consumedMaterialComparePower);

}

@ApiOperation("计量设备基本信息展示")

@GetMapping("/getMeteingInfo")

public CommonResult getMeteingInfo(@RequestParam("deviceNo") String deviceNo) {

MeasuringEquipment mes = lowCarbonService.getMeteingInfo(deviceNo);

// if (Objects.nonNull(mes) && StringUtils.isNotBlank(deviceNo)) {

// List<Equipment> equipmentList = equipmentService.getEquipmentByCode(mes.getCustomerId(), mes.getEquipmentNo());

// if (CollectionUtils.isNotEmpty(equipmentList)) {

// Equipment equipment = equipmentList.get(0);

// mes.setLocation(equipment.getBranchFactory() + "-" + mes.getMeasuringEquipmentName());

// }

// }

return CommonResult.success("查询成功", mes);

}

@ApiOperation("碳排放类型分项数据统计展示")

@GetMapping("/getCarbonStatistic")

public CommonResult getCarbonStatistic(@RequestParam("customerId") Long customerId, @RequestParam(value = "processId", required = false) Long processId) {

// 1-重量 2-碳排放量

List<Map<String, Object>> list = lowCarbonStatisticalService.getCarbonStatistic(customerId, 2, processId);

list.forEach(item -> {

item.put("count", new BigDecimal(item.get("count").toString()).setScale(2, BigDecimal.ROUND\_HALF\_UP));

int type = Integer.parseInt(item.get("type").toString());

Optional<Carbon> optional = Arrays.stream(Carbon.values()).filter(p -> p.getValue() == type).findFirst();

if (optional.isPresent()) {

Carbon carbon = optional.get();

item.put("type", carbon.getName());

} else {

item.put("type", "未知类型");

}

});

//钢坯产量

String startTime = LocalDate.now().getYear() < 2022 ? "2021-10-26" : LocalDate.now().with(TemporalAdjusters.firstDayOfYear()).toString();

String endTime = LocalDate.now().toString();

BigDecimal sumSteelYield = lowCarbonService.getSumSteelYield(customerId, CarbonConstants.CRUDE\_STEEL\_NAME, startTime, endTime);

BigDecimal r = sumSteelYield.multiply(CarbonConstants.R\_CARBON);

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

map.put("count", r);

map.put("type", Carbon.CARBON\_SEQUESTRATION.getName());

list.add(map);

return CommonResult.success("查询成功", list);

}

@ApiOperation("碳足迹展示")

@GetMapping("/getCarbonFoo")

public CommonResult getCarbonFoo(@RequestParam("customerId") Long customerId) {

//查询碳排放

Map<String, Object> list = lowCarbonStatisticalService.getCarbonFoo(customerId, 2);

return CommonResult.success("查询成功", list);

}

@ApiOperation("工序碳排放量分项展示")

@GetMapping("/getProcedureTypeData")

public CommonResult getProcedureTypeDataV2(@RequestParam("customerId") Long customerId) {

List<Map<String, Object>> procedureTypeData = carbonEmissionService.getProcedureTypeData(customerId);

return CommonResult.success("查询成功", procedureTypeData);

}

@ApiOperation("工序碳排放量分项展示")

@GetMapping("/getProcedureTypeData1")

public CommonResult getProcedureTypeData(@RequestParam("customerId") Long customerId) {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd 00:00:00");

//目前此需求是同比 同比时间则定义为 今年的上个月 和去年上个月 同比

Date date = DateUtils.addDays(new Date(), -1);

Date dateRing = DateUtils.addDays(new Date(), -7);

//获取当年上个月份时间的数据

List<Map<String, Object>> thisYearList = lowCarbonStatisticalService.getProcedureTypeData(customerId, sdf.format(date), sdf.format(date));

//获取去年上个月份的数据

List<Map<String, Object>> lastYearList = lowCarbonStatisticalService.getProcedureTypeData(customerId, sdf.format(dateRing), sdf.format(dateRing));

//先把去年的集合转为map以今年参数为主

Map<String, List<Map<String, Object>>> lastYearMap = lastYearList.stream().collect(Collectors.groupingBy(e -> String.valueOf(e.get("process") + "" + e.get("process\_id"))));

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

thisYearList.stream().forEach(item -> {

Double sum = Double.parseDouble(item.get("sum").toString());

String process = item.get("process").toString();

String process\_id = item.get("process\_id").toString();

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

if (lastYearMap.get(process + "" + process\_id) != null) {

//获取同比去年的数据

Double lastSum = Double.parseDouble(lastYearList.get(0).get("sum").toString());

Double rate = (sum - lastSum) / lastSum \* 100;

newMap.put("process", process);

newMap.put("percentage", (double) Math.round(rate \* 100) / 100);

newMap.put("total", sum);

} else {

newMap.put("process", process);

newMap.put("percentage", 100);

newMap.put("total", sum);

}

listResult.add(newMap);

});

return CommonResult.success("查询成功", listResult);

}

public Map<String, Object> findTime() {

String startTime = "";

String endTime = "";

String startYearTime = "";

String endYearTime = "";

//获取前月的第一天

Calendar cal\_1 = Calendar.getInstance();//获取当前日期

cal\_1.add(Calendar.MONTH, -1);

cal\_1.set(Calendar.DAY\_OF\_MONTH, 1);//设置为1号,当前日期既为本月第一天

startTime = format.format(cal\_1.getTime());

//获取前月的最后一天

Calendar cale = Calendar.getInstance();

cale.set(Calendar.DAY\_OF\_MONTH, 0);//设置为1号,当前日期既为本月第一天

endTime = format.format(cale.getTime());

//获取去年前月的第一天

Calendar cal\_11 = Calendar.getInstance();//获取当前日期

cal\_11.add(Calendar.YEAR, -1);

cal\_11.add(Calendar.MONTH, -1);

cal\_11.set(Calendar.DAY\_OF\_MONTH, 1);//设置为1号,当前日期既为本月第一天

startYearTime = format.format(cal\_11.getTime());

//获取去年前月的最后一天

Calendar calee = Calendar.getInstance();

calee.add(Calendar.YEAR, -1);

calee.set(Calendar.DAY\_OF\_MONTH, 0);//设置为1号,当前日期既为本月第一天

endYearTime = format.format(calee.getTime());

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

map.put("startTime", startTime);

map.put("endTime", endTime);

map.put("startYearTime", startYearTime);

map.put("endYearTime", endYearTime);

return map;

}

@ApiOperation("计量设备实时重量数据展示")

@GetMapping("/getMeteingData")

public CommonResult getMeteingData(

@RequestParam("deviceNo") String deviceNo) throws Exception {

MeasuringEquipmentData data = new MeasuringEquipmentData();

LocalDateTime nowTime = LocalDateTime.now();

;

long startTime = nowTime.minusHours(6).withMinute(0).withSecond(0).withNano(0).toEpochSecond(ZoneOffset.of("+8"));

long endTime = nowTime.withMinute(0).withSecond(0).withNano(0).toEpochSecond(ZoneOffset.of("+8")) - 1;

List<Map<String, Object>> longNumberLinkedHashMap = historyService.getDayOrHourData("carbon", deviceNo, startTime, endTime, "HH:00", "1h-sum");

data.setMeasuringEquipmentCode(deviceNo);

data.setList(longNumberLinkedHashMap);

return CommonResult.success(data);

}

@ApiOperation("工序日、月、季的碳排放量")

@GetMapping("/emissionByProcess")

public CommonResult emissionByProcess(@RequestParam("customerId") Long customerId, @RequestParam("dimension") Integer dimension, @RequestParam("processId") Long processId) {

//dimension 时间维度，1-日，2-月，3-季

//获取当年上个月份时间的数据

List<Map<String, Object>> thisYearList = lowCarbonStatisticalService.emissionByProcess(customerId, findSunMoon(dimension).get("startTime").toString(), findSunMoon(dimension).get("endTime").toString(), dimension, processId);

return CommonResult.success("查询成功", thisYearList);

}

public Map<String, Object> findSunMoon(Integer dimension) {

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

String startTime = "";

String endTime = "";

if (dimension == 1) {

//日 当前时间往前推7天

endTime = format.format(new Date());

Calendar cal\_1 = Calendar.getInstance();//获取当前日期

cal\_1.add(Calendar.DATE, -7);

startTime = format.format(cal\_1.getTime());

map.put("startTime", startTime);

map.put("endTime", endTime);

} else if (dimension == 2) {

SimpleDateFormat format1 = new SimpleDateFormat("yyyy-MM-dd");

//日 当前时间往前推24小时

endTime = format.format(new Date());

Calendar cal\_1 = Calendar.getInstance();//获取当前日期

cal\_1.add(Calendar.YEAR, -1);

startTime = format1.format(cal\_1.getTime());

map.put("startTime", startTime);

map.put("endTime", endTime);

} else {

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

//查询季度时间

Calendar c = Calendar.getInstance();

int m = c.get(Calendar.MONTH) + 1; //月份

int s = (int) Math.ceil((m - 1) / 3) + 1; //季度

System.out.println(s);

int em = (s - 1) \* 3; //上一个季度最后一个月

Calendar sc = Calendar.getInstance();

sc.setTimeInMillis(c.getTimeInMillis());

sc.set(Calendar.MONTH, em);

sc.set(Calendar.DAY\_OF\_MONTH, 1);

sc.set(Calendar.HOUR\_OF\_DAY, 0);

sc.set(Calendar.MINUTE, 0);

sc.set(Calendar.SECOND, 0);

sc.set(Calendar.MILLISECOND, 0);

sc.add(Calendar.MILLISECOND, -1);

endTime = sdf.format(sc.getTime());

Calendar ec = Calendar.getInstance();

ec.setTimeInMillis(c.getTimeInMillis());

ec.set(Calendar.MONTH, em - 1); //month 基于0,一月的值是0

ec.set(Calendar.DAY\_OF\_MONTH, 1);

ec.set(Calendar.HOUR\_OF\_DAY, 0);

ec.set(Calendar.MINUTE, 0);

ec.set(Calendar.SECOND, 0);

ec.set(Calendar.MILLISECOND, 0);

ec.add(Calendar.MONTH, -11); //向前12个月，但是-11

startTime = sdf.format(ec.getTime());

map.put("startTime", startTime);

map.put("endTime", format.format(new Date()));

}

return map;

}

@ApiOperation("工序碳排放量分项展示")

@GetMapping("/findProcedureTypeData")

public CommonResult findProcedureTypeData(@RequestParam("customerId") Long customerId, @RequestParam("processId") Long processId) {

List<Map<String, Object>> list = lowCarbonStatisticalService.findProcedureTypeData(customerId, 2, processId);

list.forEach(item -> {

int type = Integer.parseInt(item.get("type").toString());

Optional<Carbon> optional = Arrays.stream(Carbon.values()).filter(p -> p.getValue() == type).findFirst();

if (optional.isPresent()) {

Carbon carbon = optional.get();

item.put("type", carbon.getName());

} else {

item.put("type", "未知类型");

}

});

return CommonResult.success("查询成功", list);

}

/\*\*

\* 碳生产率

\*/

@ApiOperation("碳生产率展示")

@GetMapping("/findCarbonProductivity")

public CommonResult findCarbonProductivity(@RequestParam("customerId") Long customerId) {

List<CarbonView> carbonProductivity = carbonService.findCarbonProductivity(customerId);

return CommonResult.success("查询成功", carbonProductivity);

}

}

package com.bme.screen.module.carbon.service;

import com.bme.cloud.common.constant.CarbonConstants;

import com.bme.cloud.common.enums.CarbonTypeEnum;

import com.bme.cloud.common.model.Tuple3;

import com.bme.cloud.common.model.carbon.Equipment;

import com.bme.screen.module.carbon.feign.LowCarbonService;

import com.bme.screen.module.carbon.feign.LowCarbonStatisticalService;

import com.bme.screen.module.carbon.feign.MeterDeviceFeign;

import com.bme.screen.module.carbon.feign.StatisticalService;

import com.bme.screen.module.carbon.model.PercentResult;

import lombok.extern.slf4j.Slf4j;

import org.apache.commons.collections4.CollectionUtils;

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

import org.springframework.stereotype.Service;

import javax.annotation.Resource;

import java.math.BigDecimal;

import java.math.RoundingMode;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.ZoneOffset;

import java.time.temporal.TemporalAdjusters;

import java.util.\*;

import java.util.stream.Collectors;

@Service

@Slf4j

public class CarbonEmissionService {

@Resource

MeterDeviceFeign meterDeviceFeign;

@Autowired

private LowCarbonStatisticalService lowCarbonStatisticalService;

@Autowired

private HistoryService historyService;

@Autowired

private LowCarbonService lowCarbonService;

@Resource

private StatisticalService statisticalService;

public Map<String, Object> realTimeWeightV2(String measuringEquipmentCode, Integer materialId) {

LocalDateTime nowTime = LocalDateTime.now();

long startTime = nowTime.minusHours(6).withMinute(0).withSecond(0).withNano(0).toEpochSecond(ZoneOffset.of("+8"));

long endTime = nowTime.withMinute(0).withSecond(0).withNano(0).toEpochSecond(ZoneOffset.of("+8")) - 1;

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

// TODO: 2021/9/8 电力

if (materialId == 26) {

List<Map<String, Object>> power = historyService.getOriginalData("forward\_power", measuringEquipmentCode, startTime, endTime, "HH:mm:ss");

result.put("unit", "电量/KWh");

result.put("list", power);

return result;

}

List<Map<String, Object>> weight = historyService.getOriginalData("weight", measuringEquipmentCode, startTime, endTime, "HH:mm:ss");

result.put("unit", "重量/t");

result.put("list", weight);

return result;

}

public List<Map<String, Object>> realTimeWeight(String measuringEquipmentCode, Integer materialId) {

LocalDateTime nowTime = LocalDateTime.now();

long startTime = nowTime.minusHours(6).withMinute(0).withSecond(0).withNano(0).toEpochSecond(ZoneOffset.of("+8"));

long endTime = nowTime.withMinute(0).withSecond(0).withNano(0).toEpochSecond(ZoneOffset.of("+8")) - 1;

// TODO: 2021/9/8 电力

if (materialId == 26) {

return historyService.getOriginalData("forward\_power", measuringEquipmentCode, startTime, endTime, "HH:mm:ss");

}

return historyService.getOriginalData("weight", measuringEquipmentCode, startTime, endTime, "HH:mm:ss");

}

/\*\*

\* 首页工序碳排放

\*/

public List<Map<String, Object>> getProcedureTypeData(Long customerId) {

List<Map<String, Object>> processList = meterDeviceFeign.getProcessList(customerId);

if (CollectionUtils.isEmpty(processList)) {

return Collections.emptyList();

}

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

//总的排放量

long processId = Long.parseLong(map.get("processId").toString());

BigDecimal total = getSumCarbonByProcess(customerId, processId);

map.put("total", total == null ? null : total.setScale(2, BigDecimal.ROUND\_HALF\_UP));

//todo 同比

map.put("percentage", 100);

}

//过滤排放量为空的

List<Map<String, Object>> result = processList.stream().filter(map -> map.get("total") != null).sorted(Comparator.comparing(e -> new BigDecimal(e.get("total").toString()))).collect(Collectors.toList());

Collections.reverse(result);

return result;

}

public List<Map<String, Object>> getProcessList(Long customerId) {

List<Map<String, Object>> processList = meterDeviceFeign.getProcessList(customerId);

if (CollectionUtils.isEmpty(processList)) {

return Collections.emptyList();

}

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

//总的排放量

long processId = Long.parseLong(map.get("processId").toString());

map.put("total", getSumCarbonByProcess(customerId, processId));

List<Equipment> list = meterDeviceFeign.selectEquipmentByProcess(customerId, processId);

LocalDateTime start = LocalDateTime.now().with(TemporalAdjusters.firstDayOfYear()).withHour(0).withMinute(0).withSecond(0).withNano(0);

LocalDateTime now = LocalDateTime.now();

for (Equipment equipment : list) {

//计算每个设备排放量

Double emission = historyService.getDataDayAndHourSum("carbon", start, now, equipment.getEquipmentNo());

equipment.setCumulativeEmission(new BigDecimal(Double.toString(emission)));

}

List<Equipment> data = list.stream().sorted(Comparator.comparing(Equipment::getCumulativeEmission).reversed()).limit(3).collect(Collectors.toList());

map.put("list", data);

}

//过滤排放量为空的

return processList.stream().filter(map -> map.get("total") != null).collect(Collectors.toList());

}

public BigDecimal getSumCarbonByProcess(Long customerId, Long processId) {

int year = LocalDate.now().getYear();

return lowCarbonStatisticalService.getSumCarbonByProcess(customerId, processId, String.valueOf(year));

}

public List<Map<String, Object>> getProcessEmissions(Long customerId) {

List<Map<String, Object>> processList = meterDeviceFeign.getProcessList(customerId);

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

List<Map<String, Object>> list = lowCarbonStatisticalService.getCarbonStatistic(customerId, 2, Long.parseLong(map.get("processId").toString()));

if (CollectionUtils.isEmpty(list)) {

continue;

}

Map<Integer, List<Map<String, Object>>> contains = list.stream().collect(Collectors.groupingBy(e -> Integer.parseInt(e.get("type").toString())));

for (CarbonTypeEnum typeEnum : CarbonTypeEnum.values()) {

if (typeEnum.getCarbonType() == 0 || typeEnum.getCarbonType() == 4 || typeEnum.getCarbonType() == 5) {

continue;

}

if (contains.containsKey(typeEnum.getCarbonType())) {

continue;

}

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

def.put("type", typeEnum.getCarbonType());

def.put("count", 0);

list.add(def);

}

List<Map<String, Object>> data = list.stream().sorted(Comparator.comparing(e -> e.get("type").toString())).collect(Collectors.toList());

map.put("data", data);

}

//过滤排放量为空的

return processList.stream().filter(map -> map.get("data") != null).collect(Collectors.toList());

}

public Map<String,Object> calculate(Long customerId) {

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

// 计算碳排放吨钢排放和同比(从10-23后开始算)

String startTime = LocalDate.now().getYear() < 2022 ? "2021-10-26" : LocalDate.now().with(TemporalAdjusters.firstDayOfYear()).toString();

String endTime = LocalDate.now().toString();

String startTimeLast = LocalDate.now().getYear() - 1 < 2021 ? "2020-10-26" : LocalDate.now().plusYears(-1).with(TemporalAdjusters.firstDayOfYear()).toString();

String endTimeLast = LocalDate.now().plusYears(-1).toString();

List<PercentResult> carbonResultList = new ArrayList<>();

PercentResult co2 = perCarbonOfSteel(customerId, startTime, endTime);

PercentResult co2Last = perCarbonOfSteel(customerId, startTimeLast, endTimeLast);

if (co2.getValue().compareTo(BigDecimal.ZERO) == 0) {

co2.setPercentage("0");

} else {

co2.setPercentage(co2.getValue().subtract(co2Last.getValue()).divide(co2.getValue(), 2, BigDecimal.ROUND\_HALF\_UP).multiply(new BigDecimal("100")).toString());

}

carbonResultList.add(co2);

result.put("Carbon",carbonResultList);

List<PercentResult> pollutionResultList = new ArrayList<>();

Tuple3<PercentResult, PercentResult, PercentResult> percentResults = perPollutionOfSteel(customerId, startTime, endTime);

Tuple3<PercentResult, PercentResult, PercentResult> percentResultsLast = perPollutionOfSteel(customerId, startTimeLast, endTimeLast);

// 污染物吨钢排放和同比 - tdc

PercentResult tdc = percentResults.getFirst();

PercentResult tdcLast = percentResultsLast.getFirst();

if (tdc.getValue().compareTo(BigDecimal.ZERO) == 0) {

tdc.setPercentage("0");

} else {

tdc.setPercentage(tdc.getValue().subtract(tdcLast.getValue()).divide(tdc.getValue(), 2, BigDecimal.ROUND\_HALF\_UP).multiply(new BigDecimal("100")).toString());

}

pollutionResultList.add(tdc);

// 污染物吨钢排放和同比 - so2

PercentResult so2 = percentResults.getSecond();

PercentResult so2Last = percentResultsLast.getSecond();

if (so2.getValue().compareTo(BigDecimal.ZERO) == 0) {

so2.setPercentage("0");

} else {

so2.setPercentage(so2.getValue().subtract(so2Last.getValue()).divide(so2.getValue(), 2, BigDecimal.ROUND\_HALF\_UP).multiply(new BigDecimal("100")).toString());

}

pollutionResultList.add(so2);

// 污染物吨钢排放和同比 - nox

PercentResult nox = percentResults.getThird();

PercentResult noxLast = percentResultsLast.getThird();

if (nox.getValue().compareTo(BigDecimal.ZERO) == 0) {

nox.setPercentage("0");

} else {

nox.setPercentage(nox.getValue().subtract(noxLast.getValue()).divide(nox.getValue(), 2, BigDecimal.ROUND\_HALF\_UP).multiply(new BigDecimal("100")).toString());

}

pollutionResultList.add(nox);

result.put("Pollution",pollutionResultList);

return result;

}

/\*\*

\* CO2吨钢排放

\*/

private PercentResult perCarbonOfSteel(Long customerId, String startTime, String endTime) {

//今年总的碳排放量

BigDecimal sumTotalCarbon = null;

Map<String, Object> carbonSum = lowCarbonStatisticalService.getCarbon(customerId, startTime, endTime);

if (Objects.nonNull(carbonSum)) {

sumTotalCarbon = new BigDecimal(carbonSum.get("sumTotal") == null ? "0" : carbonSum.get("sumTotal").toString());

}

log.info("碳排放总量：{}，{}", sumTotalCarbon,carbonSum);

//今年总的钢坯产量

BigDecimal sumSteelYield = lowCarbonService.getSumSteelYield(customerId, CarbonConstants.CRUDE\_STEEL\_NAME, startTime, endTime);

log.info("钢坯产量：{}，{}", sumSteelYield,carbonSum);

//吨钢排放

PercentResult co2 = new PercentResult();

if (Objects.isNull(sumSteelYield) || Objects.isNull(sumTotalCarbon)) {

co2.setType("CO2");

co2.setValue(BigDecimal.ZERO);

} else {

co2.setType("CO2");

co2.setValue(sumTotalCarbon.divide(sumSteelYield, 2, BigDecimal.ROUND\_HALF\_UP));

}

log.info("吨钢排放：{}，{}", co2,carbonSum);

return co2;

}

/\*\*

\* 污染物吨钢排放

\*/

private Tuple3<PercentResult, PercentResult, PercentResult> perPollutionOfSteel(Long customerId, String startTime, String endTime) {

BigDecimal sumTotalTdc = null;

BigDecimal sumTotalSo2 = null;

BigDecimal sumTotalNox = null;

Map<String, Object> pollupSum = statisticalService.findPollup(customerId, startTime, endTime);

log.info("污染物总量：{}，{}", pollupSum,startTime);

if (Objects.nonNull(pollupSum)) {

sumTotalTdc = new BigDecimal(pollupSum.get("tdc") == null ? "0" : pollupSum.get("tdc").toString());

sumTotalSo2 = new BigDecimal(pollupSum.get("so2") == null ? "0" : pollupSum.get("so2").toString());

sumTotalNox = new BigDecimal(pollupSum.get("nox") == null ? "0" : pollupSum.get("nox").toString());

}

//总的钢坯产量

BigDecimal sumSteelYield = lowCarbonService.getSumSteelYield(customerId, CarbonConstants.CRUDE\_STEEL\_NAME, startTime, endTime);

log.info("钢坯产量：{}，{}", sumSteelYield,pollupSum);

//吨钢排放

PercentResult tdc = new PercentResult();

if (Objects.isNull(sumSteelYield) || Objects.isNull(sumTotalTdc)) {

tdc.setType("颗粒物");

tdc.setValue(BigDecimal.ZERO);

} else {

tdc.setType("颗粒物");

tdc.setValue(sumTotalTdc.divide(sumSteelYield, 2, BigDecimal.ROUND\_HALF\_UP));

}

PercentResult so2 = new PercentResult();

if (Objects.isNull(sumSteelYield) || Objects.isNull(sumTotalSo2)) {

so2.setType("SO2");

so2.setValue(BigDecimal.ZERO);

} else {

so2.setType("SO2");

so2.setValue(sumTotalSo2.divide(sumSteelYield, 2, BigDecimal.ROUND\_HALF\_UP));

}

PercentResult nox = new PercentResult();

if (Objects.isNull(sumSteelYield) || Objects.isNull(sumTotalNox)) {

nox.setType("NOx");

nox.setValue(BigDecimal.ZERO);

} else {

nox.setType("NOx");

nox.setValue(sumTotalNox.divide(sumSteelYield, 2, BigDecimal.ROUND\_HALF\_UP));

}

log.info("吨钢排放：{}，{}, {}, {}", tdc, so2, nox, pollupSum);

return Tuple3.build().of(tdc, so2, nox);

}

}

import com.bme.cloud.common.model.basic.Page;

import com.bme.cloud.common.model.carbon.CarbonView;

import com.bme.cloud.common.model.carbon.Equipment;

import com.bme.cloud.common.model.carbon.EquipmentQuery;

import com.bme.cloud.common.model.carbon.MeasuringEquipment;

import com.bme.cloud.common.support.CommonResult;

import com.bme.screen.module.carbon.model.DeviceDetail;

import com.bme.screen.module.carbon.service.EquipmentService;

import io.swagger.annotations.ApiOperation;

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

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.math.BigDecimal;

import java.math.RoundingMode;

import java.time.LocalDateTime;

import java.time.temporal.TemporalAdjusters;

import java.util.List;

import java.util.Map;

/\*\*

\* description:

\* <p></p>

\*

\* @author huanghao

\* @since 2021/8/27 11:13

\* Copyright: 2021, BME (Shanghai) Co., Ltd. All Rights Reserved.

\*/

@RestController

@RequestMapping("carbon")

public class EquipmentController {

@Autowired

private EquipmentService equipmentService;

@GetMapping("/deviceInfo/query")

public CommonResult queryEquipmentDetail(@RequestParam("customerId") long customerId, @RequestParam("deviceNo") String deviceNo){

DeviceDetail deivceDetail = equipmentService.getDeivceDetail(customerId, deviceNo);

return CommonResult.success("查询成功", deivceDetail);

}

@GetMapping("/emissionByDevice")

public CommonResult queryEmissionByDevice(@RequestParam("deviceNo") String deviceNo, @RequestParam("dimension") Integer dimension){

List<Map<String, Object>> emissionByDevice = equipmentService.queryEmissionByDevice(deviceNo, dimension);

return CommonResult.success("查询成功!", emissionByDevice);

}

/\*\*

\* 设施列表

\*/

@ApiOperation("设施列表")

@GetMapping("/equipment/list")

public CommonResult findEquipmentList(@RequestParam("customerId") Long customerId) {

List<Equipment> equipmentList = equipmentService.findEquipmentList(customerId);

return CommonResult.success("查询成功",equipmentList);

}

/\*\*

\* 计量设备列表

\*/

@ApiOperation("计量设备列表")

@GetMapping("/measuringEquipment/page")

public CommonResult findMeasuringEquipmentList(EquipmentQuery equipmentQuery) {

Page<MeasuringEquipment> equipmentPage = equipmentService.findMeasuringEquipmentList(equipmentQuery);

return CommonResult.success(equipmentPage);

}

}

package com.bme.screen.module.carbon.service;

import cn.hutool.core.date.DateUtil;

import cn.hutool.core.util.NumberUtil;

import com.bme.cloud.common.constant.CarbonConstants;

import com.bme.cloud.common.model.basic.Page;

import com.bme.cloud.common.model.carbon.\*;

import com.bme.cloud.common.util.DateUtils;

import com.bme.screen.module.carbon.feign.EquipmentServiceFeign;

import com.bme.screen.module.carbon.feign.IndustryFacilityConfigFeign;

import com.bme.screen.module.carbon.feign.LowCarbonService;

import com.bme.screen.module.carbon.feign.MeasuringEquipmentFeign;

import com.bme.screen.module.carbon.model.DeviceDetail;

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

import org.springframework.stereotype.Service;

import org.springframework.util.CollectionUtils;

import java.time.DayOfWeek;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.ZoneOffset;

import java.time.temporal.TemporalAdjusters;

import java.util.\*;

/\*\*

\* description:

\* <p></p>

\*

\* @author huanghao

\* @since 2021/8/27 13:52

\* Copyright: 2021, BME (Shanghai) Co., Ltd. All Rights Reserved.

\*/

@Service

public class EquipmentService {

@Autowired

private MeasuringEquipmentFeign measuringEquipmentFeign;

@Autowired

private EquipmentServiceFeign equipmentServiceFeign;

@Autowired

private IndustryFacilityConfigFeign industryFacilityConfigFeign;

@Autowired

private HistoryService historyService;

@Autowired

private LowCarbonService lowCarbonService;

public DeviceDetail getDeivceDetail(long customerId, String deviceNo) {

//获取计量设备信息

MeasuringEquipment measuringEquipment = measuringEquipmentFeign.queryEquipmentDetail(customerId, deviceNo);

if (measuringEquipment == null) {

return new DeviceDetail();

}

//获取设备信息

List<Equipment> processByEquipmentNo = equipmentServiceFeign.getProcessByEquipmentNo(customerId, Arrays.asList(measuringEquipment.getEquipmentNo()));

DeviceDetail deviceDetail = new DeviceDetail();

deviceDetail.setDeviceNo(measuringEquipment.getEquipmentNo());

if (!CollectionUtils.isEmpty(processByEquipmentNo)) {

Equipment equipment = processByEquipmentNo.get(0);

deviceDetail.setDeviceName(equipment.getEquipmentName());

deviceDetail.setFacilitiesSpec(equipment.getFacilitySpecs());

}

//查询设备的行业排放水平

IndustryFacilityConfig industryFacilityConfig = industryFacilityConfigFeign.getIndustryFacilityConfigByFactility(measuringEquipment.getEquipmentNo());

if (industryFacilityConfig != null) {

deviceDetail.setEmissionHighest(industryFacilityConfig.getMaxCarbonEmission());

deviceDetail.setEmissionLowest(industryFacilityConfig.getMinCarbonEmission());

}

// 查询当前设备最近一周碳排放量，求这个日期上一周的周一、周日

Date currentDate = new Date();

Date yearFirst = getYearFirst(currentDate);

long dayStartTime = yearFirst.getTime();

long dayEndTime = currentDate.getTime();

Double carbonEmission = historyService.getDataSum("carbon", dayStartTime, dayEndTime, measuringEquipment.getEquipmentNo(), "1h-sum");

//获取上周的粗钢产量

List<SteelYieldData> steelYieldList = lowCarbonService.getSteelYieldByParams(customerId, CarbonConstants.CRUDE\_STEEL\_NAME, measuringEquipment.getEquipmentNo(),

DateUtils.format(yearFirst, DateUtils.DATE\_FORMAT\_SECOND), DateUtils.format(currentDate, DateUtils.DATE\_FORMAT\_SECOND));

if (Objects.nonNull(carbonEmission) && !CollectionUtils.isEmpty(steelYieldList)) {

Double sum = steelYieldList.stream().mapToDouble(SteelYieldData::getSteelYield).sum();

if (sum > 0) {

double level = NumberUtil.div(carbonEmission, sum, 2);

deviceDetail.setEmissionLevel(level);

}

}

return deviceDetail;

}

public List<Map<String, Object>> queryEmissionByDevice(String deviceNo, Integer dimension) {

LocalDateTime nowTime = LocalDateTime.now();

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

if (dimension == 1) { // 天变化

// 当月天数：从1号0点到最后一天12；59：59

// 20211014改成最近7天

long start = LocalDate.now().atStartOfDay().plusDays(-7).toEpochSecond(ZoneOffset.of("+8")) - 1;

long end = LocalDate.now().atStartOfDay().toEpochSecond(ZoneOffset.of("+8"));

resultMap = historyService.getDayOrHourData("carbon", deviceNo, start, end, "MM-dd", "1d-sum");

} else if (dimension == 2) { // 月变化

// 当年月份统计变化

LocalDateTime loopStartTime = nowTime.with(TemporalAdjusters.firstDayOfYear()).withHour(0).withMinute(0).withSecond(0).withNano(0);

LocalDateTime loopEndTime = nowTime.with(TemporalAdjusters.firstDayOfYear()).withHour(0).withMinute(0).withSecond(0).withNano(0).plusYears(1);

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

while (loopStartTime.isBefore(loopEndTime)) {

long startTime = loopStartTime.toEpochSecond(ZoneOffset.of("+8"));

LocalDateTime nextMonth = loopStartTime.plusMonths(1);

long endTime = nextMonth.toEpochSecond(ZoneOffset.of("+8")) - 1;

Double carbonSum = historyService.getDataSum("carbon", startTime, endTime, deviceNo, "1d-sum");

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

dataMap.put("date", loopStartTime.getMonthValue() + "月");

dataMap.put("sunTotal", carbonSum);

dataList.add(dataMap);

loopStartTime = nextMonth;

}

resultMap = dataList;

} else if (dimension == 3) { // 季变化

LocalDateTime loopStartTime = nowTime.with(TemporalAdjusters.firstDayOfYear()).withHour(0).withMinute(0).withSecond(0).withNano(0);

LocalDateTime loopEndTime = nowTime.with(TemporalAdjusters.firstDayOfYear()).withHour(0).withMinute(0).withSecond(0).withNano(0).plusYears(1);

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

while (loopStartTime.isBefore(loopEndTime)) {

long startTime = loopStartTime.toEpochSecond(ZoneOffset.of("+8"));

LocalDateTime nextMonth = loopStartTime.plusMonths(3);

long endTime = nextMonth.toEpochSecond(ZoneOffset.of("+8")) - 1;

Double carbonSum = historyService.getDataSum("carbon", startTime, endTime, deviceNo, "1d-sum");

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

dataMap.put("date", getQuarter(loopStartTime.getMonthValue()));

dataMap.put("sunTotal", carbonSum);

dataList.add(dataMap);

loopStartTime = nextMonth;

}

resultMap = dataList;

}

return resultMap;

}

private static String getQuarter(int month) {

if (month == 1 || month == 2 || month == 3) {

return "第一季度";

} else if (month == 4 || month == 5 || month == 6) {

return "第二季度";

} else if (month == 7 || month == 8 || month == 9) {

return "第三季度";

} else {

return "第四季度";

}

}

public List<Equipment> findEquipmentList(Long customerIdId) {

return equipmentServiceFeign.getEquipmentList(customerIdId);

}

public Page<MeasuringEquipment> findMeasuringEquipmentList(EquipmentQuery equipmentQuery) {

return measuringEquipmentFeign.getMeasuringEquipmentList(equipmentQuery);

}

public List<Equipment> getEquipmentByCode(Long customerId, String equipmentCode) {

return equipmentServiceFeign.getProcessByEquipmentNo(customerId, Collections.singletonList(equipmentCode));

}

/\*\*

\* 获取年初时间

\*

\* @param date 指定日期

\* @return 年初时间

\*/

public static Date getYearFirst(Date date) {

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

calendar.clear(Calendar.MONTH);

calendar.set(Calendar.DAY\_OF\_MONTH, 1);

//将小时至0

calendar.set(Calendar.HOUR\_OF\_DAY, 0);

//将分钟至0

calendar.set(Calendar.MINUTE, 0);

//将秒至0

calendar.set(Calendar.SECOND, 0);

//将毫秒至0

calendar.set(Calendar.MILLISECOND, 0);

return calendar.getTime();

}

}