三度户型绘制工具源码

package com.sandu.api.house.model;

import java.io.Serializable;

import java.math.BigDecimal;

import java.util.Date;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonInclude;

import lombok.Data;

/\*\*

\* Description:

\* 户型实体

\* @author 何文

\* @version 1.0

\* Company:Sandu

\* Copyright:Copyright(c)2017

\* @date 2017/12/28

\*/

public class DrawBaseHouse implements Serializable {

/\*\*

\* base\_house.id

\*/

private Long id;

/\*\*

\* 用户id

\*/

private Long userId;

/\*\*

\* 系统编码<p>

\* base\_house.sys\_code

\*/

private String sysCode;

/\*\*

\* 创建者<p>

\* base\_house.creator

\*/

private String creator;

/\*\*

\* base\_house.gmt\_create

\*/

private Date gmtCreate;

/\*\*

\* 修改人<p>

\* base\_house.modifier

\*/

private String modifier;

/\*\*

\* 修改时间<p>

\* base\_house.gmt\_modified

\*/

private Date gmtModified;

/\*\*

\* 是否删除<p>

\* base\_house.is\_deleted

\*/

private Integer isDeleted;

/\*\*

\* 户型编码<p>

\* base\_house.house\_code

\*/

private String houseCode;

/\*\*

\* 户型简称<p>

\* base\_house.house\_short\_name

\*/

private String houseShortName;

/\*\*

\* 户型全称<p>

\* base\_house.house\_name

\*/

private String houseName;

/\*\*

\* 户型图<p>

\* base\_house.pic\_res1\_id

\*/

@JsonIgnore

private Integer picRes1Id;

/\*\*

\* CAD图<p>

\* base\_house.pic\_res2\_id

\*/

private Integer picRes2Id;

/\*\*

\* 被处理可以分析的户型图<p>

\* base\_house.pic\_res3\_id

\*/

private Integer picRes3Id;

/\*\*

\* 被查询去掉分析数据的户型图<p>

\* base\_house.pic\_res4\_id

\*/

private Integer picRes4Id;

/\*\*

\* 户型描述<p>

\* base\_house.house\_desc

\*/

private String houseDesc;

/\*\*

\* 所属小区<p>

\* base\_house.living\_id

\*/

private Integer livingId;

/\*\*

\* 总面积<p>

\* base\_house.total\_area

\*/

private String totalArea;

/\*\*

\* 总楼层<p>

\* base\_house.total\_floors

\*/

private String totalFloors;

/\*\*

\* 销售期数<p>

\* base\_house.current\_period

\*/

private String currentPeriod;

/\*\*

\* 户型高度<p>

\* base\_house.house\_high

\*/

private String houseHigh;

/\*\*

\* 适用楼层<p>

\* base\_house.apply\_floors

\*/

private String applyFloors;

/\*\*

\* 户型结构<p>

\* base\_house.house\_layout

\*/

private String houseLayout;

/\*\*

\* 室结构类型<p>

\* base\_house.room\_layout

\*/

private String roomLayout;

/\*\*

\* 厅结构类型<p>

\* base\_house.office\_layout

\*/

private String officeLayout;

/\*\*

\* 过道结构类型<p>

\* base\_house.way\_layout

\*/

private String wayLayout;

/\*\*

\* 是否存在对称户型<p>

\* base\_house.is\_exist\_symmetry

\*/

private String isExistSymmetry;

/\*\*

\* 对称户型组ids<p>

\* base\_house.symmetry\_ids

\*/

private String symmetryIds;

/\*\*

\* 是否合并<p>

\* base\_house.is\_merge

\*/

private String isMerge;

/\*\*

\* 合并户型组ids<p>

\* base\_house.merge\_ids

\*/

private String mergeIds;

/\*\*

\* 合并后新的户型id<p>

\* base\_house.merge\_new\_id

\*/

private Integer mergeNewId;

/\*\*

\* 生成模拟图<p>

\* base\_house.pic\_res\_id

\*/

private Integer picResId;

/\*\*

\* 图形是否规则<p>

\* base\_house.is\_all

\*/

private Integer isAll;

/\*\*

\* 户型类型<p>

\* base\_house.house\_common\_code

\*/

private String houseCommonCode;

/\*\*

\* 户型标示<p>

\* base\_house.house\_type\_code

\*/

private String houseTypeCode;

/\*\*

\* 户型标签<p>

\* base\_house.house\_tag

\*/

private String houseTag;

/\*\*

\* 户型状态<p>

\* base\_house.house\_status

\*/

private String houseStatus;

/\*\*

\* 处理状态<p>

\* base\_house.deal\_status

\*/

private String dealStatus;

/\*\*

\* 是否审核<p>

\* base\_house.is\_review

\*/

private String isReview;

/\*\*

\* 字符备用1<p>

\* base\_house.att1

\*/

private String att1;

/\*\*

\* base\_house.area\_long\_code

\*/

private String areaLongCode;

/\*\*

\* base\_house.house\_kind

\*/

private String houseKind;

/\*\*

\* 字符备用4<p>

\* base\_house.att4

\*/

private String att4;

/\*\*

\* 字符备用5<p>

\* base\_house.att5

\*/

private String att5;

/\*\*

\* 字符备用6<p>

\* base\_house.att6

\*/

private String att6;

/\*\*

\* 时间备用1<p>

\* base\_house.date\_att1

\*/

private Date dateAtt1;

/\*\*

\* 时间备用2<p>

\* base\_house.date\_att2

\*/

private Date dateAtt2;

/\*\*

\* base\_house.reset\_state

\*/

private Integer resetState;

/\*\*

\* 整数备用2<p>

\* base\_house.num\_att2

\*/

private Integer numAtt2;

/\*\*

\* 数字备用1<p>

\* base\_house.num\_att3

\*/

private BigDecimal numAtt3;

/\*\*

\* 数字备用2<p>

\* base\_house.num\_att4

\*/

private BigDecimal numAtt4;

/\*\*

\* 备注<p>

\* base\_house.remark

\*/

private String remark;

/\*\*

\* base\_house.area\_code

\*/

private String areaCode;

/\*\*

\* 户型状态(1:公开,0:不公开)<p>

\* base\_house.is\_public

\*/

private Integer isPublic;

/\*\*

\* 户型编码<p>

\* base\_house.house\_door\_code

\*/

private String houseDoorCode;

/\*\*

\* 户型下的房间数量

\*/

private Integer houseSpaceNum;

/\*\*

\* 转换数据后对应base\_house表的id

\*/

@JsonInclude(JsonInclude.Include.ALWAYS)

private Long baseHouseId;

/\*\*

\* 户型快照图片（截图）

\*/

@JsonIgnore

private Long snapPicId;

/\*\*

\* 数据来源

\* 1、手动绘制

\* 2、临摹老数据

\*/

@JsonInclude(JsonInclude.Include.ALWAYS)

private Integer origin;

/\*\*

\* 还原文件

\*/

@JsonIgnore

private Long restoreFileId;

@JsonIgnore

private Long perfectUserId;

/\*\*

\* 0、户型绘制员绘制的平台数据；1、用户个人数据

\*/

@JsonIgnore

private Integer dataType;

@JsonIgnore

private Integer isChangedUser;

/\*\*

\* 审核人

\*/

@JsonIgnore

private Long auditorId;

/\*\*

\* 审核时间

\*/

@JsonIgnore

private Date auditTime;

/\*\*

\* 驳回原因

\*/

private String rejectReason;

private Integer platformType;

/\*\*

\* 0、待修复、1、修复异常；2、修复成功

\*/

private Integer bugfixStatus;

private static final long serialVersionUID = 1L;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public Long getUserId() {

return userId;

}

public void setUserId(Long userId) {

this.userId = userId;

}

public String getSysCode() {

return sysCode;

}

public void setSysCode(String sysCode) {

this.sysCode = sysCode;

}

public String getCreator() {

return creator;

}

public void setCreator(String creator) {

this.creator = creator;

}

public Date getGmtCreate() {

return gmtCreate;

}

public void setGmtCreate(Date gmtCreate) {

this.gmtCreate = gmtCreate;

}

public String getModifier() {

return modifier;

}

public void setModifier(String modifier) {

this.modifier = modifier;

}

public Date getGmtModified() {

return gmtModified;

}

public void setGmtModified(Date gmtModified) {

this.gmtModified = gmtModified;

}

public Integer getIsDeleted() {

return isDeleted;

}

public void setIsDeleted(Integer isDeleted) {

this.isDeleted = isDeleted;

}

public String getHouseCode() {

return houseCode;

}

public void setHouseCode(String houseCode) {

this.houseCode = houseCode;

}

public String getHouseShortName() {

return houseShortName;

}

public void setHouseShortName(String houseShortName) {

this.houseShortName = houseShortName;

}

public String getHouseName() {

return houseName;

}

public void setHouseName(String houseName) {

this.houseName = houseName;

}

public Integer getPicRes1Id() {

return picRes1Id;

}

public void setPicRes1Id(Integer picRes1Id) {

this.picRes1Id = picRes1Id;

}

public Integer getPicRes2Id() {

return picRes2Id;

}

public void setPicRes2Id(Integer picRes2Id) {

this.picRes2Id = picRes2Id;

}

public Integer getPicRes3Id() {

return picRes3Id;

}

public void setPicRes3Id(Integer picRes3Id) {

this.picRes3Id = picRes3Id;

}

public Integer getPicRes4Id() {

return picRes4Id;

}

public void setPicRes4Id(Integer picRes4Id) {

this.picRes4Id = picRes4Id;

}

public String getHouseDesc() {

return houseDesc;

}

public void setHouseDesc(String houseDesc) {

this.houseDesc = houseDesc;

}

public Integer getLivingId() {

return livingId;

}

public void setLivingId(Integer livingId) {

this.livingId = livingId;

}

public String getTotalArea() {

return totalArea;

}

public void setTotalArea(String totalArea) {

this.totalArea = totalArea;

}

public String getTotalFloors() {

return totalFloors;

}

public void setTotalFloors(String totalFloors) {

this.totalFloors = totalFloors;

}

public String getCurrentPeriod() {

return currentPeriod;

}

public void setCurrentPeriod(String currentPeriod) {

this.currentPeriod = currentPeriod;

}

public String getHouseHigh() {

return houseHigh;

}

public void setHouseHigh(String houseHigh) {

this.houseHigh = houseHigh;

}

public String getApplyFloors() {

return applyFloors;

}

public void setApplyFloors(String applyFloors) {

this.applyFloors = applyFloors;

}

public String getHouseLayout() {

return houseLayout;

}

public void setHouseLayout(String houseLayout) {

this.houseLayout = houseLayout;

}

public String getRoomLayout() {

return roomLayout;

}

public void setRoomLayout(String roomLayout) {

this.roomLayout = roomLayout;

}

public String getOfficeLayout() {

return officeLayout;

}

public void setOfficeLayout(String officeLayout) {

this.officeLayout = officeLayout;

}

public String getWayLayout() {

return wayLayout;

}

public void setWayLayout(String wayLayout) {

this.wayLayout = wayLayout;

}

public String getIsExistSymmetry() {

return isExistSymmetry;

}

public void setIsExistSymmetry(String isExistSymmetry) {

this.isExistSymmetry = isExistSymmetry;

}

public String getSymmetryIds() {

return symmetryIds;

}

public void setSymmetryIds(String symmetryIds) {

this.symmetryIds = symmetryIds;

}

public String getIsMerge() {

return isMerge;

}

public void setIsMerge(String isMerge) {

this.isMerge = isMerge;

}

public String getMergeIds() {

return mergeIds;

}

public void setMergeIds(String mergeIds) {

this.mergeIds = mergeIds;

}

public Integer getMergeNewId() {

return mergeNewId;

}

public void setMergeNewId(Integer mergeNewId) {

this.mergeNewId = mergeNewId;

}

public Integer getPicResId() {

return picResId;

}

public void setPicResId(Integer picResId) {

this.picResId = picResId;

}

public Integer getIsAll() {

return isAll;

}

public void setIsAll(Integer isAll) {

this.isAll = isAll;

}

public String getHouseCommonCode() {

return houseCommonCode;

}

public void setHouseCommonCode(String houseCommonCode) {

this.houseCommonCode = houseCommonCode;

}

public String getHouseTypeCode() {

return houseTypeCode;

}

public void setHouseTypeCode(String houseTypeCode) {

this.houseTypeCode = houseTypeCode;

}

public String getHouseTag() {

return houseTag;

}

public void setHouseTag(String houseTag) {

this.houseTag = houseTag;

}

public String getHouseStatus() {

return houseStatus;

}

public void setHouseStatus(String houseStatus) {

this.houseStatus = houseStatus;

}

public String getDealStatus() {

return dealStatus;

}

public void setDealStatus(String dealStatus) {

this.dealStatus = dealStatus;

}

public String getIsReview() {

return isReview;

}

public void setIsReview(String isReview) {

this.isReview = isReview;

}

public String getAtt1() {

return att1;

}

public void setAtt1(String att1) {

this.att1 = att1;

}

public String getAreaLongCode() {

return areaLongCode;

}

public void setAreaLongCode(String areaLongCode) {

this.areaLongCode = areaLongCode;

}

public String getHouseKind() {

return houseKind;

}

public void setHouseKind(String houseKind) {

this.houseKind = houseKind;

}

public String getAtt4() {

return att4;

}

public void setAtt4(String att4) {

this.att4 = att4;

}

public String getAtt5() {

return att5;

}

public void setAtt5(String att5) {

this.att5 = att5;

}

public String getAtt6() {

return att6;

}

public void setAtt6(String att6) {

this.att6 = att6;

}

public Date getDateAtt1() {

return dateAtt1;

}

public void setDateAtt1(Date dateAtt1) {

this.dateAtt1 = dateAtt1;

}

public Date getDateAtt2() {

return dateAtt2;

}

public void setDateAtt2(Date dateAtt2) {

this.dateAtt2 = dateAtt2;

}

public Integer getResetState() {

return resetState;

}

public void setResetState(Integer resetState) {

this.resetState = resetState;

}

public Integer getNumAtt2() {

return numAtt2;

}

public void setNumAtt2(Integer numAtt2) {

this.numAtt2 = numAtt2;

}

public BigDecimal getNumAtt3() {

return numAtt3;

}

public void setNumAtt3(BigDecimal numAtt3) {

this.numAtt3 = numAtt3;

}

public BigDecimal getNumAtt4() {

return numAtt4;

}

public void setNumAtt4(BigDecimal numAtt4) {

this.numAtt4 = numAtt4;

}

public String getRemark() {

return remark;

}

public void setRemark(String remark) {

this.remark = remark;

}

public String getAreaCode() {

return areaCode;

}

public void setAreaCode(String areaCode) {

this.areaCode = areaCode;

}

public Integer getIsPublic() {

return isPublic;

}

public void setIsPublic(Integer isPublic) {

this.isPublic = isPublic;

}

public String getHouseDoorCode() {

return houseDoorCode;

}

public void setHouseDoorCode(String houseDoorCode) {

this.houseDoorCode = houseDoorCode;

}

public Integer getHouseSpaceNum() {

return houseSpaceNum;

}

public void setHouseSpaceNum(Integer houseSpaceNum) {

this.houseSpaceNum = houseSpaceNum;

}

public Long getBaseHouseId() {

return baseHouseId;

}

public void setBaseHouseId(Long baseHouseId) {

this.baseHouseId = baseHouseId;

}

public Long getSnapPicId() {

return snapPicId;

}

public void setSnapPicId(Long snapPicId) {

this.snapPicId = snapPicId;

}

public Integer getOrigin() {

return origin;

}

public void setOrigin(Integer origin) {

this.origin = origin;

}

public Long getRestoreFileId() {

return restoreFileId;

}

public void setRestoreFileId(Long restoreFileId) {

this.restoreFileId = restoreFileId;

}

public Long getPerfectUserId() {

return perfectUserId;

}

public void setPerfectUserId(Long perfectUserId) {

this.perfectUserId = perfectUserId;

}

public Integer getDataType() {

return dataType;

}

public void setDataType(Integer dataType) {

this.dataType = dataType;

}

public Integer getIsChangedUser() {

return isChangedUser;

}

public void setIsChangedUser(Integer isChangedUser) {

this.isChangedUser = isChangedUser;

}

public Long getAuditorId() {

return auditorId;

}

public void setAuditorId(Long auditorId) {

this.auditorId = auditorId;

}

public Date getAuditTime() {

return auditTime;

}

public void setAuditTime(Date auditTime) {

this.auditTime = auditTime;

}

public String getRejectReason() {

return rejectReason;

}

public void setRejectReason(String rejectReason) {

this.rejectReason = rejectReason;

}

public Integer getPlatformType() {

return platformType;

}

public void setPlatformType(Integer platformType) {

this.platformType = platformType;

}

public Integer getBugfixStatus() {

return bugfixStatus;

}

public void setBugfixStatus(Integer bugfixStatus) {

this.bugfixStatus = bugfixStatus;

}

public static long getSerialVersionUID() {

return serialVersionUID;

}

}

package com.sandu.api.house.output;

import java.io.Serializable;

import java.util.Date;

import java.util.List;

import com.fasterxml.jackson.annotation.JsonInclude;

import io.swagger.annotations.ApiModel;

import io.swagger.annotations.ApiModelProperty;

import lombok.Data;

/\*\*

\* Description: 户型输出类

\*

\* @author 何文

\* @version 1.0 Company:Sandu Copyright:Copyright(c)2017

\* @date 2017/12/21

\*/

@ApiModel

public class DrawBaseHouseVO implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

@ApiModelProperty("户型id")

private Long houseId;

@ApiModelProperty("户型编码")

private String houseCode;

@ApiModelProperty("户型名称")

private String houseName;

@ApiModelProperty("小区详细地址")

private String detailAddress;

@ApiModelProperty("小区名字")

private String livingName;

@ApiModelProperty("小区Id")

private Long livingId;

@ApiModelProperty("最后修改时间")

private Date lastUpdateTime;

@ApiModelProperty("户型图片")

@JsonInclude(JsonInclude.Include.ALWAYS)

private String housePic;

@ApiModelProperty("还原文件")

@JsonInclude(JsonInclude.Include.ALWAYS)

private String restoreFile;

@ApiModelProperty("户型下的空间")

private List<DrawSpaceCommonDataVO> spaceCommon;

@ApiModelProperty("用户Id")

private Long userId;

@ApiModelProperty("用于用户是否绘制过改户型") // 0 表示没有 1 表示有

private Integer checkOwnHouse;

@ApiModelProperty("正式表中判断户型是否公开")

private Integer IsPublic;

@ApiModelProperty("户型面积")

private String totalArea;

@ApiModelProperty("户型简称")

private String houseShortName;

@ApiModelProperty("draw\_base\_house表中的id")

private Long relDrawId;

@ApiModelProperty("户型处理状态")

private String dealStatus;

@ApiModelProperty("是否标准")

private Integer isStandard;

@ApiModelProperty("快照截图")

private String snapshotPic;

private String userName;

private String userMobile;

private String houseStatus;

private String rejectReason;

private Integer platformType;

public static long getSerialVersionUID() {

return serialVersionUID;

}

public Long getHouseId() {

return houseId;

}

public void setHouseId(Long houseId) {

this.houseId = houseId;

}

public String getHouseCode() {

return houseCode;

}

public void setHouseCode(String houseCode) {

this.houseCode = houseCode;

}

public String getHouseName() {

return houseName;

}

public void setHouseName(String houseName) {

this.houseName = houseName;

}

public String getDetailAddress() {

return detailAddress;

}

public void setDetailAddress(String detailAddress) {

this.detailAddress = detailAddress;

}

public String getLivingName() {

return livingName;

}

public void setLivingName(String livingName) {

this.livingName = livingName;

}

public Long getLivingId() {

return livingId;

}

public void setLivingId(Long livingId) {

this.livingId = livingId;

}

public Date getLastUpdateTime() {

return lastUpdateTime;

}

public void setLastUpdateTime(Date lastUpdateTime) {

this.lastUpdateTime = lastUpdateTime;

}

public String getHousePic() {

return housePic;

}

public void setHousePic(String housePic) {

this.housePic = housePic;

}

public String getRestoreFile() {

return restoreFile;

}

public void setRestoreFile(String restoreFile) {

this.restoreFile = restoreFile;

}

public List<DrawSpaceCommonDataVO> getSpaceCommon() {

return spaceCommon;

}

public void setSpaceCommon(List<DrawSpaceCommonDataVO> spaceCommon) {

this.spaceCommon = spaceCommon;

}

public Long getUserId() {

return userId;

}

public void setUserId(Long userId) {

this.userId = userId;

}

public Integer getCheckOwnHouse() {

return checkOwnHouse;

}

public void setCheckOwnHouse(Integer checkOwnHouse) {

this.checkOwnHouse = checkOwnHouse;

}

public Integer getIsPublic() {

return IsPublic;

}

public void setIsPublic(Integer isPublic) {

IsPublic = isPublic;

}

public String getTotalArea() {

return totalArea;

}

public void setTotalArea(String totalArea) {

this.totalArea = totalArea;

}

public String getHouseShortName() {

return houseShortName;

}

public void setHouseShortName(String houseShortName) {

this.houseShortName = houseShortName;

}

public Long getRelDrawId() {

return relDrawId;

}

public void setRelDrawId(Long relDrawId) {

this.relDrawId = relDrawId;

}

public String getDealStatus() {

return dealStatus;

}

public void setDealStatus(String dealStatus) {

this.dealStatus = dealStatus;

}

public Integer getIsStandard() {

return isStandard;

}

public void setIsStandard(Integer isStandard) {

this.isStandard = isStandard;

}

public String getSnapshotPic() {

return snapshotPic;

}

public void setSnapshotPic(String snapshotPic) {

this.snapshotPic = snapshotPic;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public String getUserMobile() {

return userMobile;

}

public void setUserMobile(String userMobile) {

this.userMobile = userMobile;

}

public String getHouseStatus() {

return houseStatus;

}

public void setHouseStatus(String houseStatus) {

this.houseStatus = houseStatus;

}

public String getRejectReason() {

return rejectReason;

}

public void setRejectReason(String rejectReason) {

this.rejectReason = rejectReason;

}

public Integer getPlatformType() {

return platformType;

}

public void setPlatformType(Integer platformType) {

this.platformType = platformType;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

DrawBaseHouseVO that = (DrawBaseHouseVO) o;

if (houseId != null ? !houseId.equals(that.houseId) : that.houseId != null) return false;

if (houseCode != null ? !houseCode.equals(that.houseCode) : that.houseCode != null) return false;

if (houseName != null ? !houseName.equals(that.houseName) : that.houseName != null) return false;

if (detailAddress != null ? !detailAddress.equals(that.detailAddress) : that.detailAddress != null)

return false;

if (livingName != null ? !livingName.equals(that.livingName) : that.livingName != null) return false;

if (livingId != null ? !livingId.equals(that.livingId) : that.livingId != null) return false;

if (lastUpdateTime != null ? !lastUpdateTime.equals(that.lastUpdateTime) : that.lastUpdateTime != null)

return false;

if (housePic != null ? !housePic.equals(that.housePic) : that.housePic != null) return false;

if (restoreFile != null ? !restoreFile.equals(that.restoreFile) : that.restoreFile != null) return false;

if (spaceCommon != null ? !spaceCommon.equals(that.spaceCommon) : that.spaceCommon != null) return false;

if (userId != null ? !userId.equals(that.userId) : that.userId != null) return false;

if (checkOwnHouse != null ? !checkOwnHouse.equals(that.checkOwnHouse) : that.checkOwnHouse != null)

return false;

if (IsPublic != null ? !IsPublic.equals(that.IsPublic) : that.IsPublic != null) return false;

if (totalArea != null ? !totalArea.equals(that.totalArea) : that.totalArea != null) return false;

if (houseShortName != null ? !houseShortName.equals(that.houseShortName) : that.houseShortName != null)

return false;

if (relDrawId != null ? !relDrawId.equals(that.relDrawId) : that.relDrawId != null) return false;

if (dealStatus != null ? !dealStatus.equals(that.dealStatus) : that.dealStatus != null) return false;

if (isStandard != null ? !isStandard.equals(that.isStandard) : that.isStandard != null) return false;

if (snapshotPic != null ? !snapshotPic.equals(that.snapshotPic) : that.snapshotPic != null) return false;

if (userName != null ? !userName.equals(that.userName) : that.userName != null) return false;

if (userMobile != null ? !userMobile.equals(that.userMobile) : that.userMobile != null) return false;

if (houseStatus != null ? !houseStatus.equals(that.houseStatus) : that.houseStatus != null) return false;

if (rejectReason != null ? !rejectReason.equals(that.rejectReason) : that.rejectReason != null) return false;

return platformType != null ? platformType.equals(that.platformType) : that.platformType == null;

}

@Override

public int hashCode() {

int result = houseId != null ? houseId.hashCode() : 0;

result = 31 \* result + (houseCode != null ? houseCode.hashCode() : 0);

result = 31 \* result + (houseName != null ? houseName.hashCode() : 0);

result = 31 \* result + (detailAddress != null ? detailAddress.hashCode() : 0);

result = 31 \* result + (livingName != null ? livingName.hashCode() : 0);

result = 31 \* result + (livingId != null ? livingId.hashCode() : 0);

result = 31 \* result + (lastUpdateTime != null ? lastUpdateTime.hashCode() : 0);

result = 31 \* result + (housePic != null ? housePic.hashCode() : 0);

result = 31 \* result + (restoreFile != null ? restoreFile.hashCode() : 0);

result = 31 \* result + (spaceCommon != null ? spaceCommon.hashCode() : 0);

result = 31 \* result + (userId != null ? userId.hashCode() : 0);

result = 31 \* result + (checkOwnHouse != null ? checkOwnHouse.hashCode() : 0);

result = 31 \* result + (IsPublic != null ? IsPublic.hashCode() : 0);

result = 31 \* result + (totalArea != null ? totalArea.hashCode() : 0);

result = 31 \* result + (houseShortName != null ? houseShortName.hashCode() : 0);

result = 31 \* result + (relDrawId != null ? relDrawId.hashCode() : 0);

result = 31 \* result + (dealStatus != null ? dealStatus.hashCode() : 0);

result = 31 \* result + (isStandard != null ? isStandard.hashCode() : 0);

result = 31 \* result + (snapshotPic != null ? snapshotPic.hashCode() : 0);

result = 31 \* result + (userName != null ? userName.hashCode() : 0);

result = 31 \* result + (userMobile != null ? userMobile.hashCode() : 0);

result = 31 \* result + (houseStatus != null ? houseStatus.hashCode() : 0);

result = 31 \* result + (rejectReason != null ? rejectReason.hashCode() : 0);

result = 31 \* result + (platformType != null ? platformType.hashCode() : 0);

return result;

}

}

package com.sandu.web.house.controller;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.Objects;

import java.util.stream.Collectors;

import javax.servlet.http.HttpServletRequest;

import com.google.common.collect.Lists;

import com.sandu.api.v2.house.dto.DrawBaseHouseCallbackDTO;

import com.sandu.api.v2.house.service.DrawBaseHouseServiceV2;

import com.sandu.exception.BizException;

import com.sandu.exception.DrawBusinessException;

import com.sandu.util.JSON;

import org.apache.commons.lang3.StringUtils;

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

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

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 com.fasterxml.jackson.databind.JavaType;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.sandu.api.house.bo.DrawBakeTaskBO;

import com.sandu.api.house.bo.DrawBaseHouseBO;

import com.sandu.api.house.bo.DrawBaseProductBO;

import com.sandu.api.house.bo.UserLoginBO;

import com.sandu.api.house.input.DrawBaseHouseCheck;

import com.sandu.api.house.input.DrawBaseHouseSearch;

import com.sandu.api.house.input.PublicProductInfoDTO;

import com.sandu.api.house.model.DrawBakeTaskDetail;

import com.sandu.api.house.model.DrawDesignTemplet;

import com.sandu.api.house.model.DrawResFile;

import com.sandu.api.house.output.DrawBakeTaskDetailVO;

import com.sandu.api.house.output.DrawBakeTaskVO;

import com.sandu.api.house.output.DrawBaseHouseSubmitVO;

import com.sandu.api.house.output.DrawBaseHouseVO;

import com.sandu.api.house.output.DrawHouseAndAreaVO;

import com.sandu.api.house.output.DrawProductDataVO;

import com.sandu.api.house.service.DrawBakeTaskService;

import com.sandu.api.house.service.DrawBaseHouseService;

import com.sandu.api.house.service.DrawDesignTempletService;

import com.sandu.api.house.service.DrawResFileService;

import com.sandu.common.BaseController;

import com.sandu.common.ReturnData;

import com.sandu.common.constant.ResponseEnum;

import com.sandu.common.constant.house.DrawBaseProductConstant;

import com.sandu.exception.AppException;

import com.sandu.exception.BusinessException;

import com.sandu.util.ExceptionUtils;

import com.sandu.util.Regex;

import com.sandu.util.auth.HouseAuthUtils;

import io.swagger.annotations.Api;

import io.swagger.annotations.ApiOperation;

import lombok.extern.slf4j.Slf4j;

/\*\*

\* </p>

\* Company:Sandu Copyright:Copyright(c)2017 Description:

\*

\* @author 何文

\* @version 1.0

\*

\* @date 2017/12/28

\*/

@Slf4j

@RestController

@RequestMapping("/v1/draw/house")

@Api(tags = "绘制户型", description = "DrawBaseHouseController")

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

public class DrawBaseHouseController extends BaseController {

@Autowired

private DrawBaseHouseService drawBaseHouseService;

@Autowired

private DrawBaseHouseServiceV2 drawBaseHouseServiceV2;

@Autowired

private DrawBakeTaskService drawBakeTaskService;

@Autowired

private DrawDesignTempletService drawDesignTempletService;

@Autowired

private DrawResFileService drawResFileService;

@Autowired

ObjectMapper objectMapper;

@Value("${resource.domain}")

private String resourceDomain;

/\*\*

\* 请求示例：http://127.0.0.1:9898/v1/draw/house/list

\* </p>

\* 户型列表及户型审核列表

\*

\* @param query

\* @param request

\* @return

\*/

@PostMapping("/list")

@ApiOperation(value = "查询户型列表及户型审核列表", response = DrawBaseHouseVO.class)

public ReturnData listDrawHouse(DrawBaseHouseSearch query, HttpServletRequest request) {

ReturnData response = setMsgId(request);

Objects.requireNonNull(query, "参数不能为空");

query.setPageSize(super.getLimit(query.getPageSize()));

query.setPageNum(super.getPage(query.getPageNum(), query.getPageSize()));

Map<String, Object> map = drawBaseHouseService.listDrawHouse(query);

// 转换vo返回

List<DrawBaseHouseVO> drawBaseHouseVOS = adapterDrawBaseHouseVOS(((List<DrawBaseHouseBO>) map.get("list")));

return response.list(drawBaseHouseVOS).total((long) map.get("total"));

}

private List<DrawBaseHouseVO> adapterDrawBaseHouseVOS(List<DrawBaseHouseBO> list) {

List<DrawBaseHouseVO> drawBaseHouseVOS = new ArrayList<>();

for (DrawBaseHouseBO bo : list) {

DrawBaseHouseVO vo = new DrawBaseHouseVO();

vo.setHouseCode(bo.getHouseCode());

vo.setHouseId(bo.getId());

vo.setHouseName(bo.getHouseName());

vo.setHousePic(bo.getPicPath());

vo.setLastUpdateTime(bo.getGmtModified());

vo.setLivingName(bo.getLivingName());

vo.setRestoreFile(bo.getFilePath());

vo.setDetailAddress(bo.getDetailAddress());

vo.setHouseShortName(bo.getHouseShortName());

vo.setUserId(bo.getUserId());

vo.setTotalArea(bo.getTotalArea());

vo.setIsPublic(bo.getIsPublic());

vo.setCheckOwnHouse(bo.getCheckOwnHouse());

vo.setDealStatus(bo.getDealStatus());

vo.setIsStandard(bo.getIsStandard());

vo.setLivingId(bo.getLivingId());

vo.setSnapshotPic(bo.getSnapshotPic());

vo.setUserName(bo.getUserName());

vo.setUserMobile(bo.getUserMobile());

vo.setHouseStatus(bo.getHouseStatus());

vo.setRejectReason(bo.getRejectReason());

vo.setPlatformType(bo.getPlatformType());

drawBaseHouseVOS.add(vo);

}

return drawBaseHouseVOS;

}

/\*\*

\* 请求示例：http://127.0.0.1:9898/v1/draw/house/save

\* </p>

\* 户型保存接口

\*

\* @param request

\* @param data

\* @return

\*/

@PostMapping("/save")

@ApiOperation(value = "户型保存")

public ReturnData saveDrawBaseHouse(HttpServletRequest request, String userId, String data) {

ReturnData response = setMsgId(request);

// Map<String, Object> map = drawBaseHouseService.saveDrawBaseHouse(data, userId);

// v2版本的保存户型，添加了异常时，删除资源文件处理功能

Map<String, Object> map = drawBaseHouseService.saveDrawHouseV2(data, userId);

return response.data(map);

}

/\*\*

\* 请求示例：http://127.0.0.1:9898/v1/draw/house/delete

\* </p>

\* 户型删除接口

\*

\* @param request

\* @param houseId

\* @return

\*/

@PostMapping("/delete")

@ApiOperation(value = "户型删除")

public ReturnData deleteDrawBaseHouse(HttpServletRequest request, Long houseId, Long userId) {

// TODO 获取用户

ReturnData response = setMsgId(request);

drawBaseHouseService.deleteDrawBaseHouse(houseId, userId);

return response;

}

/\*\*

\* 请求示例：http://127.0.0.1:9898/v1/draw/house/check

\* <p>

\* 户型审核接口

\*

\* @param check

\* @param request

\* @return

\*/

@PostMapping("/check")

@ApiOperation(value = "户型审核接口")

public ReturnData checkBaseHouse(DrawBaseHouseCheck check, HttpServletRequest request) {

ReturnData response = setMsgId(request);

drawBaseHouseService.checkDrawBaseHouse(check);

return response;

}

/\*\*

\* 请求示例：http://127.0.0.1:9898/v1/draw/house/executeBakeTask

\* <p>

\* 烘焙任务列表查询

\*

\* @param queueName

\* @param taskId

\* @param request

\* @return

\*/

@PostMapping("/executeBakeTask")

@ApiOperation(value = "执行烘焙任务", response = DrawBakeTaskVO.class)

public ReturnData executeBakeTask(String queueName, Long taskId, HttpServletRequest request) {

ReturnData returnData = setMsgId(request);

List<DrawBakeTaskBO> list = drawBakeTaskService.listBakeTask(queueName, taskId);

returnData.setResourceUrl(resourceDomain);

return returnData.data(getDrawBakeTaskVO(list));

}

@PostMapping("/getBakeTaskId")

@ApiOperation("获取烘焙任务id")

public ReturnData getBakeTaskId(String queueName, String msgId) {

ReturnData returnData = ReturnData.builder();

returnData.msgId(msgId);

if (StringUtils.isEmpty(queueName)) {

returnData.setStatus(false);

returnData.message("队列名称不能为空!!!");

return returnData;

}

Long bakeTaskId = drawBakeTaskService.getBakeTaskId(queueName);

if (bakeTaskId != null) {

returnData.setStatus(true);

DrawBakeTaskVO b = new DrawBakeTaskVO();

b.setTaskId(bakeTaskId);

returnData.setData(b);

return returnData;

} else {

returnData.status(true);

returnData.setData(null);

returnData.setMessage("暂时没有烘焙任务");

return returnData;

}

}

private DrawBakeTaskVO getDrawBakeTaskVO(List<DrawBakeTaskBO> list) throws BusinessException {

// 返回页面数据对象

DrawBakeTaskVO taskVO = new DrawBakeTaskVO();

// json

ObjectMapper objectMapper = new ObjectMapper();

JavaType javaType = objectMapper.getTypeFactory().constructParametricType(ArrayList.class, PublicProductInfoDTO.class);

list.parallelStream().forEach(task -> {

// 任务数据

taskVO.setTaskId(task.getId());

taskVO.setHouseId(task.getHouseId());

// 任务详情数据

List<DrawBakeTaskDetail> taskDetails = task.getDrawBakeTaskDetail();

if (taskDetails != null && !taskDetails.isEmpty()) {

DrawBakeTaskDetail detail = taskDetails.get(0);

DrawBakeTaskDetailVO detailVO = new DrawBakeTaskDetailVO();

detailVO.setTaskDetailId(detail.getId());

// 空间

detailVO.setSpaceId(detail.getSpaceId());

detailVO.setSpaceCode(detail.getSpaceCode());

detailVO.setSpaceFileId(Long.parseLong(detail.getSpaceFileIds()));

detailVO.setSpaceFilePath(detail.getSpaceFilePath());

// 产品view对象

List<DrawProductDataVO> drawProductDataVOS = getDrawProductDataVOS(detail);

// public产品对象(栏杆/窗户等) ->start

List<DrawDesignTemplet> drawDesignTempletList = drawDesignTempletService.findAllBySpaceCommonId2(detail.getSpaceId());

if (drawDesignTempletList == null || drawDesignTempletList.size() == 0) {

log.error("draw\_样板房信息没有找到,drawSpaceCommonId = {}", detail.getSpaceId());

} else {

DrawDesignTemplet drawDesignTemplet = drawDesignTempletList.get(0);

if (StringUtils.isNotEmpty(drawDesignTemplet.getPublicProductInfo())) {

List<PublicProductInfoDTO> publicProductInfoDTOList = JSON.parse(drawDesignTemplet.getPublicProductInfo(), javaType);

if (publicProductInfoDTOList != null && publicProductInfoDTOList.size() > 0) {

for (PublicProductInfoDTO publicProductInfoDTO : publicProductInfoDTOList) {

DrawResFile drawResFile = drawResFileService.get(publicProductInfoDTO.getFileId());

if (drawResFile == null) {

log.error("draw\_res\_file数据没有找到,id = {}", publicProductInfoDTO.getFileId());

continue;

}

DrawProductDataVO drawProductDataVO = new DrawProductDataVO();

drawProductDataVO.setFileId(publicProductInfoDTO.getFileId());

drawProductDataVO.setFileName(drawResFile.getFileName());

drawProductDataVO.setFilePath(drawResFile.getFilePath());

drawProductDataVO.setUniqueId(publicProductInfoDTO.getUniqueId());

drawProductDataVO.setModelType(DrawBaseProductConstant.MODEL\_TYPE\_PUBLIC);

drawProductDataVOS.add(drawProductDataVO);

}

}

}

}

// public产品对象(栏杆/窗户等) ->end

detailVO.setProductData(drawProductDataVOS);

taskVO.setTaskDetail(detailVO);

}

});

return taskVO;

}

private List<DrawProductDataVO> getDrawProductDataVOS(DrawBakeTaskDetail detail) {

ObjectMapper objectMapper = new ObjectMapper();

String data = detail.getProductData();

JavaType javaType = objectMapper.getTypeFactory().constructParametricType(List.class, DrawBaseProductBO.class);

List<DrawBaseProductBO> drawBaseProductBOS;

try {

drawBaseProductBOS = objectMapper.readValue(data, javaType);

} catch (IOException e) {

throw new BizException(e);

}

List<DrawProductDataVO> drawProductDataVOS = new ArrayList<>();

if (drawBaseProductBOS != null && !drawBaseProductBOS.isEmpty()) {

for (DrawBaseProductBO drawBaseProductBO : drawBaseProductBOS) {

DrawProductDataVO drawProductDataVO = new DrawProductDataVO();

drawProductDataVO.setProductId(drawBaseProductBO.getId());

drawProductDataVO.setProductCode(drawBaseProductBO.getProductCode());

drawProductDataVO.setFileId(drawBaseProductBO.getFileId());

drawProductDataVO.setFileName(drawBaseProductBO.getFileName());

drawProductDataVO.setFilePath(drawBaseProductBO.getFilePath());

drawProductDataVO.setUniqueId(drawBaseProductBO.getUniqueId());

drawProductDataVO.setModelType(DrawBaseProductConstant.MODEL\_TYPE\_HARD);

drawProductDataVOS.add(drawProductDataVO);

}

}

return drawProductDataVOS;

}

/\*\*

\* 烘培任务回调

\*

\* @author huangsongbo

\* @return

\*/

// @PostMapping("/callback")

// public ReturnData callback(String jsonStr, HttpServletRequest request) {

// ReturnData<DrawBaseHouseSubmitVO> returnData = this.setMsgId(request);

//

// // 参数验证 ->start

// if (StringUtils.isEmpty(jsonStr)) {

// return returnData.status(false).code(ResponseEnum.PARAM\_ERROR).message("参数jsonStr不能为空");

// }

// // 参数验证 ->end

// try {

// Map<String, Object> msgMap = drawBaseHouseService.callback(jsonStr);

// returnData.message(Objects.toString(msgMap.get("msg")));

// } catch (Exception e) {

// log.error("处理烘焙任务异常, JsonStr => {}", jsonStr, e);

// return handlerBakeFail(returnData, jsonStr, e);

// }

//

// return returnData.status(true);

// }

@PostMapping("/callback")

@ApiOperation(value = "v2-烘焙回调")

public ReturnData callback(String msgId, @RequestParam("jsonStr") String data) {

long startup = System.currentTimeMillis();

ReturnData response = setMsgId(msgId);

// log.error("处理烘焙参数:"+jsonStr);

if (StringUtils.isEmpty(data)) {

return response.status(false).code(ResponseEnum.PARAM\_ERROR);

}

DrawBaseHouseCallbackDTO dtoNew = null;

try {

dtoNew = objectMapper.readValue(data, DrawBaseHouseCallbackDTO.class);

if (dtoNew == null) {

throw new DrawBusinessException(false, ResponseEnum.PARAM\_ERROR);

}

// handler bake task

String message = drawBaseHouseService.callbackV2(dtoNew);

return response.message(Objects.toString(message, "未知的错误!"));

} catch (Exception e) {

log.error("处理烘焙任务异常, data => {}", data, e);

// handler error task

return handlerCallbackError(response, dtoNew, data, e);

} finally {

log.info("#################### 处理烘焙任务耗时：{}(ms) ####################", System.currentTimeMillis() - startup);

}

}

/\*\*

\* 处理烘焙失败的任务

\*

\* @param response

\* @param dtoNew

\* @param data

\* @param e

\* @return

\*/

private ReturnData handlerCallbackError(ReturnData response, DrawBaseHouseCallbackDTO dtoNew,

String data, Exception e) {

try {

if (dtoNew == null) {

dtoNew = objectMapper.readValue(data, DrawBaseHouseCallbackDTO.class);

}

dtoNew.setMessage(ExceptionUtils.getStackTrace(e));

// handler callback error

drawBaseHouseServiceV2.handlerCallbackError(dtoNew, data);

} catch (Exception ex) {

log.error("ERROR--ERROR--ERROR--回滚烘焙任务异常--ERROR--ERROR--ERROR", ex);

}

// handler app exception

if (e instanceof AppException) {

AppException ex = (AppException) e;

return response.status(false).code(ex.getResponseEnum());

}

// handler exception

return response.status(false).message(Objects.toString(e.getMessage(), "未知的错误!"));

}

/\*\*

\* 处理烘焙失败

\*

\* @param returnData

\* @param data

\* @param e

\* @return

\*/

private ReturnData handlerBakeFail(ReturnData returnData, String data, Exception e) {

String errorMessage = getErrorMessage(e);

try {

// 回滚处理烘焙任务失败

drawBaseHouseService.handlerBakeTaskFailed(data, errorMessage);

if (e instanceof AppException) {

AppException ex = (AppException) e;

return returnData.status(false).code(ex.getResponseEnum());

}

} catch (IOException ex) {

log.error("回滚处理失败的烘焙任务异常-ERROR-ERROR", e);

}

return returnData.status(false).message(Objects.toString(e.getMessage(), ResponseEnum.ERROR.getMessage()));

}

/\*\*

\* 获取异常信息

\*

\* @param ex

\* @return

\*/

private String getErrorMessage(Exception ex) {

String stackTrace = ExceptionUtils.getStackTrace(ex);

if (ex instanceof AppException) {

AppException exception = (AppException) ex;

String message = exception.getResponseEnum().getMessage();

return StringUtils.isEmpty(message) ? stackTrace : message + "\n" + stackTrace;

}

return stackTrace;

}

// /\*\*

// \* 保存户型绘制所有数据 涉及到的流程:

// \* <pre>

// \* 1.新建空间

// \* 2.新建产品数据

// \* 3.新建样板房

// \* 4.新建样板房产品表

// \* </pre>

// \*

// \* @param jsonStr

// \* @param request

// \* @return

// \* @author huangsongbo

// \*/

// @PostMapping("/submit")

// public ReturnData<DrawBaseHouseSubmitVO> submit(String jsonStr, HttpServletRequest request) {

// ReturnData<DrawBaseHouseSubmitVO> returnData = ReturnData.builder();

//

// // 参数验证 ->start

// if (StringUtils.isEmpty(jsonStr)) {

// return returnData.status(false).code(ResponseEnum.PARAM\_ERROR).message("参数jsonStr不能为空");

// }

// // 参数验证 ->end

//

// returnData = setMsgId(request);

// UserLoginBO userLoginBO = HouseAuthUtils.getUserLoginBO();

// DrawBaseHouseSubmitDTO data = drawBaseHouseService.submit(jsonStr, userLoginBO);

//

// return returnData.status(true).data(data);

// }

/\*\*

\* 保存户型绘制所有数据 涉及到的流程:

\* <pre>

\* 1.新建空间

\* 2.新建产品数据

\* 3.新建样板房

\* 4.新建样板房产品表

\* </pre>

\*

\* @param jsonStr

\* @param request

\* @return

\* @author huangsongbo

\*/

@PostMapping("/submit")

public ReturnData<DrawBaseHouseSubmitVO> submit(String jsonStr, HttpServletRequest request) {

long startup = System.currentTimeMillis();

ReturnData returnData = ReturnData.builder();

if (StringUtils.isEmpty(jsonStr)) {

return returnData.status(false).code(ResponseEnum.PARAM\_ERROR).message("参数jsonStr不能为空");

}

returnData = setMsgId(request);

drawBaseHouseService.submitV2(jsonStr);

log.info("#################### 处理提交户型耗时：{}(ms) ####################", System.currentTimeMillis() - startup);

return returnData;

}

/\*\*

\* 普通绘制者 上传户型

\* @author songjianming@sanduspace.com

\* @param houseId

\* @param request

\* @return

\*/

@PostMapping("/uploadHouse")

public ReturnData uploadHouse(Long houseId, HttpServletRequest request) {

ReturnData response = setMsgId(request);

if (null == houseId) {

return response.status(false).code(ResponseEnum.PARAM\_ERROR).message("houseId参数为空");

}

drawBaseHouseService.uploadHouse(houseId);

return response;

}

/\*\*

\* 完善户型

\* @author songjianming@sanduspace.com

\* @param houseId

\* @param request

\* @return

\*/

@PostMapping("/perfectHouse")

public ReturnData perfectHouse(Long houseId, HttpServletRequest request) {

ReturnData response = ReturnData.builder();

if (null == houseId) {

return response.status(false).code(ResponseEnum.PARAM\_ERROR).message("houseId参数为空");

}

response = setMsgId(request);

DrawHouseAndAreaVO drawHouse = drawBaseHouseService.perfectHouse(houseId);

return response.data(drawHouse);

}

@PostMapping("/searchList")

@ApiOperation(value = "用户绘制时查询户型可供临摹", response = DrawBaseHouseVO.class)

public ReturnData searchHouse(DrawBaseHouseSearch query, HttpServletRequest request) {

ReturnData response = ReturnData.builder();

query = setUserId(query);

response = setMsgId(request);

Map<String, Object> map = drawBaseHouseService.getHouses(query);

if (map.size() > 0) {

// 转换vo返回

List<DrawBaseHouseVO> drawBaseHouseVOS = adapterDrawBaseHouseVOS(((List<DrawBaseHouseBO>) map.get("list")));

// 没有数据提示

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

return response.status(false).code(ResponseEnum.NOT\_HOUSE\_CONTENT);

}

response.setResourceUrl(resourceDomain);

response.status(true).code(ResponseEnum.SUCCESS).list(drawBaseHouseVOS)

.total((Integer) map.get("total")).message("请求成功").msgId(request.getParameter("msgId"));

}

return response;

}

@PostMapping("/facsimileHouse")

@ApiOperation(value = "用户搜索户型临摹或者修改", response = DrawBaseHouseVO.class)

public ReturnData copyOrEditHouse(Long houseId, String type, HttpServletRequest request) {

if (StringUtils.isEmpty(type) || houseId == null) {

throw new BusinessException(false, ResponseEnum.PARAM\_ERROR);

}

ReturnData response = ReturnData.builder();

response = setMsgId(request);

DrawHouseAndAreaVO drawHouseAndAreaVO = drawBaseHouseService.facsimileHouse(houseId, type);

response.setResourceUrl(resourceDomain);

return response.status(true).code(ResponseEnum.SUCCESS).data(drawHouseAndAreaVO).message("请求成功").msgId(request.getParameter("msgId"));

}

// public ReturnData checkOutParameters(DrawBaseHouseSearch query, ReturnData response) {

// if (query.getLivingName() == null || query.getAreaCode() == null) {

// if (query.getLivingId() == null) {

// return response.status(false).message("livingId为空").code(ResponseEnum.PARAM\_ERROR);

// }

// }

// if (query.getLivingId() == null) {

// if (query.getLivingName() == null || query.getAreaCode() == null) {

// return response.status(false).message("参数为空").code(ResponseEnum.PARAM\_ERROR);

// }

// }

// response.status(true);

// return response;

// }

public DrawBaseHouseSearch setUserId(DrawBaseHouseSearch query) {

UserLoginBO userLoginBO = HouseAuthUtils.getUserLoginBO();

Long userId = userLoginBO.getId();

log.debug("户型绘制:获取用户Id ==>:" + userId);

if (userId == null) {

throw new BusinessException(false, ResponseEnum.PARAM\_ERROR);

}

query.setUserId(userId);

return query;

}

@PostMapping("/transformHouse")

public ReturnData transformHouse(String phone, Long houseId, HttpServletRequest request) {

ReturnData response = setMsgId(request);

if (!StringUtils.isNoneBlank(phone)) {

return response.status(false).message("手机号不能为空！");

}

if (!phone.matches(Regex.MOBILE\_PHONE.getValue())) {

return response.status(false).message(phone.concat("不是手机号码！"));

}

if (houseId == null || houseId <= 0) {

return response.status(false).message("参数houseId不能为空！");

}

drawBaseHouseService.transformHouse(phone, houseId);

return response;

}

@PostMapping("/reset")

public ReturnData resetDrawHouse(String data) {

if (StringUtils.isEmpty(data)) {

return ReturnData.builder().status(false).code(ResponseEnum.PARAM\_ERROR);

}

List<String> strings = Lists.newArrayList(data.split(","));

List<Integer> houseIds = strings.stream().filter(StringUtils::isNoneBlank).distinct().map(Integer::parseInt).collect(Collectors.toList());

drawBaseHouseService.resetDrawHouse(houseIds);

return ReturnData.builder().code(ResponseEnum.SUCCESS).status(true);

}

}

package com.sandu.service.house.impl;

import java.io.IOException;

import java.lang.reflect.InvocationTargetException;

import java.util.\*;

import java.util.stream.Collectors;

import com.fasterxml.jackson.databind.DeserializationFeature;

import com.fasterxml.jackson.databind.SerializationFeature;

import com.google.common.base.Strings;

import com.google.common.collect.Lists;

import com.sandu.api.area.bo.BaseAreaLivingBO;

import com.sandu.api.house.service.\*;

import com.sandu.api.res.service.ResHandlerErrorService;

import com.sandu.api.v2.house.dto.DrawBaseHouseCallbackDTO;

import com.sandu.api.v2.house.service.DrawBaseHouseServiceV2;

import com.sandu.api.mail.model.PerfectHouseMailMessage;

import com.sandu.api.volume.room.model.VolumeRoomHouseBO;

import com.sandu.api.volume.room.service.DrawVolumeRoomService;

import com.sandu.common.constant.bake.BakeCallbackStatus;

import com.sandu.common.constant.bake.BakeTaskQueue;

import com.sandu.common.constant.kechuang.PlatformType;

import com.sandu.common.constant.kechuang.VolumeRoomConstant;

import com.sandu.exception.BizException;

import com.sandu.mq.queue.SyncMessage;

import com.sandu.mq.queue.service.QueueService;

import com.sandu.service.volume.room.dao.DrawVolumeRoomMapper;

import com.sandu.util.http.HttpContextUtils;

import lombok.AllArgsConstructor;

import lombok.Getter;

import org.apache.commons.beanutils.BeanUtils;

import org.apache.commons.lang3.StringUtils;

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

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import com.fasterxml.jackson.annotation.JsonInclude;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.sandu.api.house.bo.BaseAreaBO;

import com.sandu.api.house.bo.DrawBaseHouseBO;

import com.sandu.api.house.bo.DrawBaseProductBO;

import com.sandu.api.house.bo.DrawHouseResouceBO;

import com.sandu.api.house.bo.DrawSpaceCommonBO;

import com.sandu.api.house.bo.LivingCodeBO;

import com.sandu.api.house.bo.UserLoginBO;

import com.sandu.api.house.input.DrawBaseHouseCheck;

import com.sandu.api.house.input.DrawBaseHouseControllerCallBackDTO;

import com.sandu.api.house.input.DrawBaseHouseSubmitDTO;

import com.sandu.api.house.input.DrawBaseHouseNew;

import com.sandu.api.house.input.DrawBaseHouseSearch;

import com.sandu.api.house.input.DrawBaseProductDTO;

import com.sandu.api.house.input.DrawDesignTempletDTO;

import com.sandu.api.house.input.DrawFileDataNew;

import com.sandu.api.house.model.BaseHouse;

import com.sandu.api.house.model.BaseLiving;

import com.sandu.api.house.model.DrawBakeTask;

import com.sandu.api.house.model.DrawBakeTaskDetail;

import com.sandu.api.house.model.DrawBaseHouse;

import com.sandu.api.house.model.DrawBaseProduct;

import com.sandu.api.house.model.DrawDesignTemplet;

import com.sandu.api.house.model.DrawDesignTempletProduct;

import com.sandu.api.house.model.DrawResFile;

import com.sandu.api.house.model.DrawResHousePic;

import com.sandu.api.house.model.DrawSpaceCommon;

import com.sandu.api.res.model.ResFile;

import com.sandu.api.res.model.ResHousePic;

import com.sandu.api.house.output.DrawHouseAndAreaVO;

import com.sandu.api.house.search.DrawDesignTempletProductSearch;

import com.sandu.api.product.service.DrawBaseProductService;

import com.sandu.api.mail.service.TemplateMailService;

import com.sandu.common.DrawType;

import com.sandu.common.HouseRole;

import com.sandu.common.constant.FileKey;

import com.sandu.common.constant.ResponseEnum;

import com.sandu.common.constant.SystemConfigEnum;

import com.sandu.common.constant.house.DrawBaseHouseConstant;

import com.sandu.common.constant.house.DrawDesignTempletProductConstant;

import com.sandu.exception.BusinessException;

import com.sandu.exception.DrawBusinessException;

import com.sandu.service.file.dao.DrawResFileMapper;

import com.sandu.service.file.dao.ResFileMapper;

import com.sandu.service.house.dao.BaseHouseMapper;

import com.sandu.service.house.dao.DrawBaseHouseMapper;

import com.sandu.service.living.dao.BaseLivingMapper;

import com.sandu.service.pic.dao.DrawResHousePicMapper;

import com.sandu.service.pic.dao.ResHousePicMapper;

import com.sandu.util.Utils;

import com.sandu.util.auth.HouseAuthUtils;

import lombok.extern.slf4j.Slf4j;

/\*\*

\* Description:

\*

\* @author 何文

\* @version 1.0 Company:Sandu Copyright:Copyright(c)2017

\* @date 2017/12/28

\*/

@Slf4j

@Service("drawBaseHouseService")

public class DrawBaseHouseServiceImpl implements DrawBaseHouseService {

public final static Map<Integer, Integer[]> ARGS\_MAP = new HashMap<>();

static {

ARGS\_MAP.put(1, null);

ARGS\_MAP.put(0, new Integer[]{DrawBaseHouseConstant.CHECK\_APPLY});

ARGS\_MAP.put(2, new Integer[]{DrawBaseHouseConstant.HOUSE\_WAIT\_PERFECT, DrawBaseHouseConstant.HOUSE\_PERFECT\_PROCESSING});

}

@Autowired

private DrawBaseHouseMapper drawBaseHouseMapper;

@Autowired

private DrawResHousePicMapper drawResHousePicMapper;

@Autowired

private DrawResFileMapper drawResFileMapper;

@Autowired

private DrawSpaceCommonService drawSpaceCommonService;

@Autowired

private DrawBaseProductService drawBaseProductService;

@Autowired

private BaseLivingService baseLivingService;

@Autowired

private DrawDesignTempletService drawDesignTempletService;

@Autowired

private DrawDesignTempletProductService drawDesignTempletProductService;

@Autowired

private DrawBakeTaskService drawBakeTaskService;

@Autowired

private DrawBakeTaskDetailService drawBakeTaskDetailService;

@Autowired

private BaseLivingMapper baseLivingMapper;

@Autowired

private DrawBaseHouseService drawBaseHouseService;

@Autowired

private BaseHouseMapper baseHouseMapper;

@Autowired

private ResHousePicMapper resHousePicMapper;

@Autowired

private ResFileMapper resFileMapper;

@Autowired

private TemplateMailService templateMailService;

@Autowired

private QueueService queueService;

@Autowired

private DesignTempletJumpPositionRelService designTempletJumpPositionRelService;

@Autowired

private DrawBaseHouseServiceV2 drawBaseHouseServiceV2;

@Autowired

private DrawVolumeRoomMapper drawVolumeRoomMapper;

@Autowired

@Qualifier("KCDrawVolumeRoomServiceImpl")

private DrawVolumeRoomService drawVolumeRoomService;

@Autowired

DrawResFileService drawResFileService;

@Autowired

ResHandlerErrorService resHandlerErrorService;

@Override

public Map<String, Object> listDrawHouse(DrawBaseHouseSearch query) {

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

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

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

if (query.getUserId() == null) {

query.setUserId(HouseAuthUtils.getRequestUserId());

}

// 检查权限

this.wrapCheckArgs(query);

Long total = drawBaseHouseMapper.countDrawHouse(query);

if (total > 0) {

list = drawBaseHouseMapper.listDrawHouse(query);

}

if (list != null && !list.isEmpty()) {

// 图片资源

Map<Long, String> mapResource = this.mapResource(list);

// 量房资源

Map<Long, VolumeRoomHouseBO> mapVolumeRoom = this.mapVolumeRoomHouse(list);

StringBuilder buf = new StringBuilder();

for (DrawBaseHouseBO house : list) {

if (StringUtils.isNoneBlank(house.getAreaLongCode())) {

String detailAddress = getDetailAddress(localAreaCache, house.getAreaLongCode());

buf.append(detailAddress).append(house.getLivingName());

house.setDetailAddress(buf.toString());

// clear

buf.setLength(0);

}

// 图片资源优先级 ： 截图 > 户型图

house.setPicPath(mapResource.get(house.getId()));

// 量房 lf 文件处理

VolumeRoomHouseBO volumeRoom = mapVolumeRoom.get(house.getId());

if (volumeRoom != null && !PlatformType.SANDU.getType().equals(house.getPlatformType())) {

house.setFilePath(volumeRoom.getLfFilePath());

}

}

}

map.put("total", total);

map.put("list", list);

return map;

}

/\*\*

\* 加入本地临时缓存，缓解数据库压力

\* @param localAreaCache

\* @param areaLongCode

\* @return

\*/

private String getDetailAddress(Map<String, String> localAreaCache, String areaLongCode) {

// 本地缓存

String localCacheValue;

if (localAreaCache.containsKey(areaLongCode) && (localCacheValue = localAreaCache.get(areaLongCode)) != null) {

return localCacheValue;

}

// TODO redis缓存

// db

String[] split = areaLongCode.substring(1, areaLongCode.length() - 1).split("\\.");

String detailAddress = baseLivingService.getDetailAddress(split);

if (!Strings.isNullOrEmpty(detailAddress)) {

localAreaCache.put(areaLongCode, detailAddress);

}

return detailAddress == null ? Utils.VOID\_VALUE : detailAddress;

}

/\*\*

\* 审核列表(0) => 审核 => house\_status = 2

\* </p>

\* 我的绘制(1) => 草稿、驳回 => house\_status = all

\* </p>

\* 完善列表(2) => 完善绘制员可见 => hosue\_status = 8

\* </p>

\* 默认我的绘制

\*

\* @param query

\*/

private void wrapCheckArgs(DrawBaseHouseSearch query) {

Integer checkStatus = query.getCheckStatus();

// 我的绘制列表

if (checkStatus == null || checkStatus == HouseListStatus.DRAW.getStatus()) {

return;

}

// 审核列表，审核户型权限

if (checkStatus == HouseListStatus.CHECK.getStatus() && HouseAuthUtils.hasApprovedPermission()) {

throw new BusinessException(false, ResponseEnum.FORBIDDEN);

}

// 完善列表，完善户型权限

else if (checkStatus == HouseListStatus.IMPROVER.getStatus() && !HouseAuthUtils.hasImproverPermission()) {

throw new BusinessException(false, ResponseEnum.FORBIDDEN);

}

// 特别注意，只有完善员和审核员角色才可以userId = null

query.setUserId(null);

// 默认我的绘制 => 草稿、驳回

query.setCheckArgs(ARGS\_MAP.get(checkStatus));

}

@Getter

@AllArgsConstructor

enum HouseListStatus {

// 审核

CHECK(0),

// 绘制

DRAW(1),

// 完善

IMPROVER(2);

private int status;

}

/\*\*

\* 户型资源

\*

\* @param drawHouses

\* @return

\*/

@Override

public Map<Long, String> mapResource(List<DrawBaseHouseBO> drawHouses) {

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

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

return map;

}

// filter null

List<DrawBaseHouseBO> collect = drawHouses.stream().filter(h -> !(h.getSnapPicId() == null

|| h.getSnapPicId() <= 0)).collect(Collectors.toList());

// 查询snap pic

// 图片资源优先级 ： 截图 > 户型图

List<DrawBaseHouseBO> listResource = drawBaseHouseMapper.listDrawHouseResource(1, collect);

// 查询res\_pic

if (listResource.size() < drawHouses.size()) {

List<DrawBaseHouseBO> snap = drawHouses.stream().filter(p -> (p.getSnapPicId() == null || p.getSnapPicId() <= 0)

&& (p.getPicRes1Id() != null && p.getPicRes1Id() > 0)).collect(Collectors.toList());

// Fuck NullPointerException

if (snap != null && !snap.isEmpty()) {

snap = drawBaseHouseMapper.listDrawHouseResource(null, snap);

listResource.addAll(snap);

}

}

if (!listResource.isEmpty()) {

// (prev, next) -> next 解决重复key抛出异常问题，取后一个的值为准

map = listResource.stream().collect(Collectors.toMap(DrawBaseHouseBO::getId, DrawBaseHouseBO::getPicPath, (prev, next) -> next));

}

return Optional.of(map).orElse(new HashMap<>());

}

/\*\*

\* 量房资源

\*

\* @param drawHouses

\* @return

\*/

@Override

public Map<Long, VolumeRoomHouseBO> mapVolumeRoomHouse(List<DrawBaseHouseBO> drawHouses) {

Map<Long, VolumeRoomHouseBO> map = new HashMap<>();

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

return map;

}

List<DrawBaseHouseBO> volumeRooms = drawHouses.stream().filter(p -> !PlatformType.SANDU.getType().equals(p.getPlatformType())

&& StringUtils.isEmpty(p.getFilePath())).collect(Collectors.toList());

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

return map;

}

List<VolumeRoomHouseBO> listVolumeRooms = drawVolumeRoomMapper.listVolumeRoomByHouseId(volumeRooms, VolumeRoomConstant.DEFAULTS.getStatus());

if (!listVolumeRooms.isEmpty()) {

// (prev, next) -> next 解决重复key抛出异常问题，取后一个的值为准

map = listVolumeRooms.stream().collect(Collectors.toMap(VolumeRoomHouseBO::getDrawHouseId, v -> v, (p, n) -> n));

}

return Optional.of(map).orElse(new HashMap<>());

}

/\*\*

\* v2版本的保存户型，添加了异常时，删除资源文件处理功能

\* @param data

\* @param userId

\* @return

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED)

public Map<String, Object> saveDrawHouseV2(String data, String userId) {

try {

return this.saveDrawBaseHouse(data, userId);

} catch (Exception e) {

// 保存异常，删除已上传过的文件资源

// 保存户型的主要有截图、户型图片(.png)、户型的还原文件(.sdkj)三个资源文件，异常时删除这三个文件即可

resHandlerErrorService.handlerErrorWithSave(data);

throw e;

}

}

/\*\*

\* 保存户型

\* @param data

\* @param userId

\* @return

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)

public Map<String, Object> saveDrawBaseHouse(String data, String userId) {

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

// 解析请求参数

DrawBaseHouseNew houseNew = getDrawBaseHouseNew(data, userId);

// 户型处理

DrawBaseHouse house = handlerDrawBaseHouse(houseNew);

map.put("house", house);

DrawResHousePic pic = new DrawResHousePic();

DrawResHousePic snapPic = new DrawResHousePic();

DrawResFile file = new DrawResFile();

// 处理户型和文件的关系

for (DrawFileDataNew drawFileDataNew : houseNew.getDatas()) {

if (SystemConfigEnum.HOUSE\_PIC\_UPLOAD.equals(SystemConfigEnum.valueOf(drawFileDataNew.getFileType()))) {

// 户型图

pic = adapterResHousePic(house, drawFileDataNew);

drawResHousePicMapper.saveDrawResHousePic(pic);

// 户型图

house.setPicRes1Id(pic.getId());

} else if (SystemConfigEnum.HOUSE\_SNAP\_PIC\_UPLOAD.equals(SystemConfigEnum.valueOf(drawFileDataNew.getFileType()))) {

// 户型快照图片（截图）

snapPic = adapterResHousePic(house, drawFileDataNew);

drawResHousePicMapper.saveDrawResHousePic(snapPic);

// 户型快照图片（截图）

house.setSnapPicId(snapPic.getId().longValue());

} else {

// 还原文件

file = getDrawResFile(house, drawFileDataNew);

drawResFileMapper.saveDrawResFile(file);

// 还原文件

house.setRestoreFileId(file.getId());

}

}

// 更新户型图、截图、原因文件、修改时间

house.setGmtModified(new Date());

house.setModifier(HouseAuthUtils.getUserLoginBO().getLoginName());

// 审核时编辑不需要转移户型

if (!Objects.equals(houseNew.getDrawType(), DrawType.DRAW\_AUDITOR\_EDIT)

&& !Objects.equals(houseNew.getDrawType(), DrawType.DRAW\_ROUGH\_HOUSE)) {

// 搜索去修改，户型归属转移给修改的人，需要平台数据才可以

if (DrawBaseHouseConstant.DATA\_PLATFORM.equals(house.getDataType()) && HouseAuthUtils.hasDrawPermission()) {

house.setIsChangedUser(1);

house.setUserId(HouseAuthUtils.getRequestUserId());

log.info("{}(houseId)户型归属转交给 {}(userId)用户", house.getId(), house.getUserId());

}

}

int updateBaseHouse = drawBaseHouseMapper.updateDrawBaseHouse(house);

if (updateBaseHouse <= 0) {

throw new BizException("更新资源文件异常");

}

map.put("pic", pic);

map.put("snapPic", snapPic);

map.put("file", file);

return map;

}

private DrawBaseHouseNew getDrawBaseHouseNew(String dataJson, String userId) {

if (!StringUtils.isNotBlank(dataJson)) {

throw new BusinessException(false, ResponseEnum.PARAM\_ERROR);

}

DrawBaseHouseNew drawHouseNew;

try {

drawHouseNew = new ObjectMapper().readValue(dataJson, DrawBaseHouseNew.class);

} catch (IOException e) {

throw new BusinessException(false, ResponseEnum.JSON\_FORMAT\_ERROR, e);

}

if (drawHouseNew == null) {

throw new BizException("保存的户型为空");

}

if (!StringUtils.isNotBlank(drawHouseNew.getHouseName())) {

throw new BizException("户型名不能为空");

}

// userId valid

if (!StringUtils.isNotBlank(userId)) {

Long requestUserId = HouseAuthUtils.getRequestUserId();

if (requestUserId == null) {

throw new BusinessException(false, ResponseEnum.USER\_INFO\_IS\_NULL);

}

userId = requestUserId.toString();

}

drawHouseNew.setUserId(Long.valueOf(userId));

if (drawHouseNew.getDatas().isEmpty()) {

throw new BusinessException(false, ResponseEnum.PARAM\_ERROR);

}

return drawHouseNew;

}

private DrawBaseHouse handlerDrawBaseHouse(DrawBaseHouseNew data) throws BusinessException {

// 过滤

DrawBaseHouse drawHouse = drawFilter(data);

// 新增绘制、绘制通用PC户型老数据

if (null == data.getHouseId()) {

// 小区

BaseLiving living = getBaseLiving(data);

// 产生0-10000000的整数随机数

// int randomNum = (int) (Math.random() \* 10000000);

String randomNum = Utils.random();

drawHouse.setHouseCode(null == living ? ("0" + randomNum) : (living.getLivingCode() + randomNum));

drawHouse.setUserId(data.getUserId());

drawHouse.setHouseName(data.getHouseName());

drawHouse.setSysCode(Utils.getCurrentDateTime(Utils.DATETIMESSS) + "\_" + Utils.generateRandomDigitString(6));

drawHouse.setIsDeleted(DrawBaseHouseConstant.IS\_DELETE\_INIT);

drawHouse.setLivingId(null == living ? 0 : living.getId().intValue());

drawHouse.setTotalArea(data.getTotalArea());

drawHouse.setAreaLongCode(null == living ? "" : living.getAreaLongCode());

drawHouse.setHouseStatus(String.valueOf(DrawBaseHouseConstant.CHECK\_INIT));

drawHouse.setGmtCreate(new Date());

drawHouse.setCreator(HouseAuthUtils.getLoginName());

// 数据类型 0、非公开 1、公开

drawHouse.setIsPublic(Utils.getHouseIsPublic());

// 数据类型 0、平台数据 1、用户个人数据

drawHouse.setDataType(Utils.getUserDataType());

// 保存户型数据

drawBaseHouseMapper.insertSelective(drawHouse);

} else {

drawHouse = drawBaseHouseMapper.getBaseHouseById(data.getHouseId());

if (null == drawHouse) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

if (Integer.valueOf(drawHouse.getHouseStatus()).intValue() == DrawBaseHouseConstant.HOUSE\_BAKE\_WAIT) {

throw new BusinessException(false, ResponseEnum.HOUSE\_ALREADY\_AUDIT\_PASS);

}

drawHouse.setHouseName(data.getHouseName());

drawHouse.setTotalArea(data.getTotalArea());

// 草稿状态

// 非完善户型和审核时编辑

String houseStatus = drawHouse.getHouseStatus();

if (!Objects.equals(houseStatus, DrawBaseHouseConstant.HOUSE\_WAIT\_PERFECT.toString())

&& !Objects.equals(houseStatus, DrawBaseHouseConstant.HOUSE\_PERFECT\_PROCESSING.toString())

&& !Objects.equals(data.getDrawType(), DrawType.DRAW\_AUDITOR\_EDIT)) {

drawHouse.setHouseStatus(DrawBaseHouseConstant.CHECK\_INIT.toString());

}

// 删除资源处理

this.deletedFile(drawHouse, data.getDatas());

}

// 处理量房status

drawVolumeRoomService.handlerVolumeRoom(drawHouse, VolumeRoomConstant.PARSING\_SUCCESS.getStatus(), VolumeRoomConstant.DEFAULTS.getStatus());

return drawHouse;

}

/\*\*

\* @param drawHouse

\* @param files

\*/

public void deletedFile(DrawBaseHouse drawHouse, List<DrawFileDataNew> files) {

List<Integer> deletedFile = new ArrayList<>();

for (DrawFileDataNew file : files) {

if (SystemConfigEnum.HOUSE\_PIC\_UPLOAD.equals(SystemConfigEnum.valueOf(file.getFileType()))) {

if (drawHouse.getPicRes1Id() != null) {

deletedFile.add(drawHouse.getPicRes1Id());

}

}

if (SystemConfigEnum.HOUSE\_SNAP\_PIC\_UPLOAD.equals(SystemConfigEnum.valueOf(file.getFileType()))) {

if (drawHouse.getSnapPicId() != null) {

deletedFile.add(drawHouse.getSnapPicId().intValue());

}

}

}

// 删除户型图片、截图

if (!deletedFile.isEmpty()) {

drawResHousePicMapper.deleteDrawResHousePic(deletedFile.toArray(new Integer[]{}));

}

// 删除户型还原文件

drawResFileMapper.deleteDrawResFile(drawHouse.getRestoreFileId(), DrawBaseHouseConstant.RESTORE\_FILE\_TYPE);

}

private DrawBaseHouse drawFilter(DrawBaseHouseNew houseNew) {

if (houseNew.getDrawType() == null || houseNew.getDrawType() == DrawType.DRAW\_NEW) {

return drawNewHouseFilter(houseNew);

} else if (houseNew.getDrawType() == DrawType.DRAW\_OLD) {

// 临摹老数据的户型

return drawOldHouseFilter(houseNew);

} else if (houseNew.getDrawType() == DrawType.DRAW\_AUDITOR\_EDIT) {

// 审核人编辑

return drawCheckHouseFilter(houseNew);

} else if (houseNew.getDrawType() == DrawType.DRAW\_ROUGH\_HOUSE) {

// 毛坯房

DrawBaseHouse drawBaseHouse = drawRoughHouseFilter(houseNew);

drawBaseHouse.setOrigin(DrawBaseHouseConstant.ORIGIN\_ROUGH\_HOUSE);

return drawBaseHouse;

} else {

throw new BizException("未知的绘制类型, drawType=" + houseNew.getDrawType());

}

}

/\*\*

\* 绘制工具绘制的新户型过滤处理

\* <pre>

\* 1、待审核的户型不能绘制

\* 2、烘焙中的户型不能绘制

\* 3、烘焙中的户型不能绘制

\* 4、一个户型不能有多个人绘制

\* </pre>

\* {@link com.sandu.common.constant.house.DrawBaseHouseConstant}

\*

\* @param houseNew

\* @return

\*/

private DrawBaseHouse drawNewHouseFilter(DrawBaseHouseNew houseNew) {

// 通过绘制工具绘制的户型

if (null != houseNew.getHouseId()) {

DrawBaseHouse drawBaseHouse = drawBaseHouseMapper.selectByPrimaryKey(houseNew.getHouseId());

if (drawBaseHouse != null) {

if (drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.CHECK\_APPLY.toString())) {

// 待审核

throw new BizException("待审核的户型不能绘制");

} else if (drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_WAIT.toString())

|| drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_PROCESSING.toString())

|| drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_FAIL.toString())) {

// 待烘焙、烘焙中、烘焙失败

throw new BizException("烘焙中的户型不能绘制");

}

if (drawBaseHouse.getBaseHouseId() != null && drawBaseHouse.getBaseHouseId() != 0) {

// 老数据重新绘制

BaseHouse baseHouse = baseHouseMapper.selectByPrimaryKey(drawBaseHouse.getBaseHouseId());

if (baseHouse != null) {

if (StringUtils.isNotBlank(baseHouse.getDealStatus())) {

// 是否正在烘焙中

if (DrawBaseHouseConstant.DEAL\_STATUS\_BAKE.equals(baseHouse.getDealStatus())) {

throw new BizException("烘焙中的户型不能绘制");

}

// 是否有人正在绘制, 没有人绘制就锁住当前绘制的户型

if (!DrawBaseHouseConstant.DEAL\_STATUS\_RUNNING.equals(baseHouse.getDealStatus())

&& !DrawBaseHouseConstant.DEAL\_STATUS\_RERUNNING.equals(baseHouse.getDealStatus())) {

// 锁住当前绘制的户型

this.lockHouse(baseHouse.getDealStatus(), houseNew.getUserId(), baseHouse.getId());

} else if ((DrawBaseHouseConstant.DEAL\_STATUS\_RUNNING.equals(baseHouse.getDealStatus())

|| DrawBaseHouseConstant.DEAL\_STATUS\_RERUNNING.equals(baseHouse.getDealStatus()))

&& !houseNew.getUserId().equals(baseHouse.getUserId())) {

// 绘制的人是否是我

throw new BizException("一个户型不能有多个人绘制");

}

} else {

// 第一次处理 dealStatus = null

this.lockHouse(baseHouse.getDealStatus(), houseNew.getUserId(), houseNew.getHouseId());

}

}

}

} else {

throw new BizException("绘制的户型[" + houseNew.getHouseId() + "(houseId), " + houseNew.getHouseName() + "(houseName)]不存在");

}

}

return new DrawBaseHouse();

}

/\*\*

\* 锁住当前绘制的户型

\*

\* @param dealStatus

\* @param userId

\* @param houseId

\*/

private void lockHouse(String dealStatus, Long userId, Long houseId) {

String status = DrawBaseHouseConstant.DEAL\_STATUS\_EDIT.equals(dealStatus) ?

DrawBaseHouseConstant.DEAL\_STATUS\_RERUNNING : DrawBaseHouseConstant.DEAL\_STATUS\_RUNNING;

int updateHouseDealStatus = baseHouseMapper.updateHouseDealStatusForLock(status, userId, houseId);

if (updateHouseDealStatus <= 0) {

throw new BizException("数据出现异常");

}

}

/\*\*

\* 临摹通用app老数据的户型

\* <pre>

\* 1、烘焙中的户型不能绘制

\* 2、一个户型不能有多个人绘制

\* </pre>

\* {@link com.sandu.common.constant.house.DrawBaseHouseConstant}

\*

\* @param houseNew

\* @return

\*/

private DrawBaseHouse drawOldHouseFilter(DrawBaseHouseNew houseNew) {

DrawBaseHouse drawBaseHouse = new DrawBaseHouse();

if (houseNew.getHouseId() != null && houseNew.getBaseHouseId() == null) {

BaseHouse baseHouse = baseHouseMapper.selectByPrimaryKey(houseNew.getHouseId());

if (baseHouse != null) {

// 是否正在烘焙中

if (DrawBaseHouseConstant.DEAL\_STATUS\_BAKE.equals(baseHouse.getDealStatus())) {

throw new BizException("烘焙中的户型不能绘制");

}

// 是否有人正在绘制

if ((StringUtils.isEmpty(baseHouse.getDealStatus()) || baseHouse.getUserId().equals(0L))

|| (!DrawBaseHouseConstant.DEAL\_STATUS\_RUNNING.equals(baseHouse.getDealStatus())

&& !DrawBaseHouseConstant.DEAL\_STATUS\_RERUNNING.equals(baseHouse.getDealStatus()))) {

// 将户型变成绘制中

this.lockHouse(baseHouse.getDealStatus(), houseNew.getUserId(), baseHouse.getId());

} else if ((DrawBaseHouseConstant.DEAL\_STATUS\_RUNNING.equals(baseHouse.getDealStatus())

|| DrawBaseHouseConstant.DEAL\_STATUS\_RERUNNING.equals(baseHouse.getDealStatus()))

&& !baseHouse.getUserId().equals(houseNew.getUserId())) {

// 绘制的人是否是我

throw new BizException("一个户型不能有多个人绘制");

}

drawBaseHouse.setBaseHouseId(houseNew.getHouseId());

drawBaseHouse.setOrigin(DrawBaseHouseConstant.ORIGIN\_OLD\_COPY); // 老数据临摹标识

// drawBaseHouse.setHouseCode(baseHouse.getHouseCode()); // 使用老数据的house\_code

// 是否关联过绘制数据

DrawBaseHouse drawHouse = drawBaseHouseMapper.getDrawHouseByBaseHouseId(drawBaseHouse.getBaseHouseId());

// 是否新增一条绘制数据和户型老数据关联

houseNew.setHouseId(drawHouse != null ? drawHouse.getId() : null);

} else {

throw new BizException("绘制的户型[" + houseNew.getHouseId() + "(baseHouseId), " + houseNew.getHouseName() + "(houseName)]不存在");

}

} else if (houseNew.getHouseId() != null && houseNew.getBaseHouseId() != null) {

// 重绘制

return drawNewHouseFilter(houseNew);

} else {

throw new BizException("绘制的户型[" + houseNew.getHouseId() + "(baseHouseId), " + houseNew.getHouseName() + "(houseName)]数据异常");

}

return drawBaseHouse;

}

/\*\*

\* 审核时进行编辑

\*

\* @param houseNew

\* @return

\*/

private DrawBaseHouse drawCheckHouseFilter(DrawBaseHouseNew houseNew) {

if (null == houseNew.getHouseId()) {

throw new BizException("绘制的户型[" + houseNew.getHouseId() + "(houseId), " + houseNew.getHouseName() + "(houseName)]数据异常");

}

DrawBaseHouse drawBaseHouse = drawBaseHouseMapper.selectByPrimaryKey(houseNew.getHouseId());

if (drawBaseHouse != null) {

if (drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_WAIT.toString())

|| drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_PROCESSING.toString())

|| drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_FAIL.toString())) {

// 待烘焙、烘焙中、烘焙失败

throw new BizException("烘焙中的户型不能绘制");

}

} else {

throw new BizException("绘制的户型[" + houseNew.getHouseId() + "(houseId), " + houseNew.getHouseName() + "(houseName)]不存在");

}

return drawBaseHouse;

}

/\*\*

\* 毛坯房

\*

\* @param houseNew

\* @return

\*/

private DrawBaseHouse drawRoughHouseFilter(DrawBaseHouseNew houseNew) {

DrawBaseHouse drawBaseHouse = drawBaseHouseMapper.selectByPrimaryKey(houseNew.getHouseId());

if (drawBaseHouse != null) {

if (drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_WAIT.toString())

|| drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_PROCESSING.toString())

|| drawBaseHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_FAIL.toString())) {

// 待烘焙、烘焙中、烘焙失败

throw new BizException("烘焙中的户型不能绘制");

}

}

return drawBaseHouse == null ? new DrawBaseHouse() : drawBaseHouse;

}

private DrawResFile getDrawResFile(DrawBaseHouse house, DrawFileDataNew drawFileDataNew) {

DrawResFile drawResFile = new DrawResFile();

drawResFile.setIsDeleted(DrawBaseHouseConstant.IS\_DELETE\_INIT);

drawResFile.setBusinessId(String.valueOf(house.getId()));

drawResFile.setSysCode(house.getHouseCode());

drawResFile.setFileOriginalName(drawFileDataNew.getFileOriginalName());

drawResFile.setFileName(drawFileDataNew.getFileName());

drawResFile.setFilePath(drawFileDataNew.getFilePath());

drawResFile.setFileSize(drawFileDataNew.getFileSize());

drawResFile.setFileSuffix(drawFileDataNew.getFileSuffix());

drawResFile.setFileType(drawFileDataNew.getFileType());

drawResFile.setFileCode(house.getHouseCode());

drawResFile.setCreator(house.getUserId() + "");

drawResFile.setModifier(house.getUserId() + "");

drawResFile.setGmtCreate(new Date());

drawResFile.setFileKey(FileKey.HOUSE\_DRAW\_FILE.getFileKey());

return drawResFile;

}

/\*\*

\* 获取保存户型的小区信息

\*

\* @param data

\* @return

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED)

public BaseLiving getBaseLiving(DrawBaseHouseNew data) {

BaseLiving living = new BaseLiving();

StringBuilder buf = new StringBuilder();

if (data.getLivingId() == null || data.getLivingId() == 0) {

String firstAreaCode = data.getFirstAreaCode();

String secondAreaCode = data.getSecondAreaCode();

if (StringUtils.isNotBlank(firstAreaCode)) {

buf.append(".").append(firstAreaCode).append(".");

}

if (StringUtils.isNotBlank(secondAreaCode)) {

buf.append(secondAreaCode).append(".");

}

if (StringUtils.isNotBlank(data.getThirdAreaCode())) {

buf.append(data.getThirdAreaCode()).append(".");

living.setAreaId(data.getThirdAreaCode());

} else {

living.setAreaId(StringUtils.isNotBlank(secondAreaCode) ? secondAreaCode : firstAreaCode);

}

/\*

\* v1.0.2

\* 查询areaId下相同小区名的小区, 如果有多条时limit 1

\*/

if (StringUtils.isNoneBlank(data.getLivingName())) {

BaseLiving queryLiving = baseLivingService.getLivingByName(living.getAreaId(), data.getLivingName());

if (queryLiving != null) {

return queryLiving;

}

}

living.setLivingName(data.getLivingName());

living.setAreaLongCode(buf.toString());

living.setCreator(data.getUserId() + "");

living.setGmtCreate(new Date());

living.setModifier(data.getUserId() + "");

living.setGmtModified(new Date());

living.setIsDeleted(DrawBaseHouseConstant.IS\_DELETE\_INIT);

living.setSysCode(Utils.getCurrentDateTime(Utils.DATETIMESSS) + "\_" + Utils.generateRandomDigitString(6));

// livingCode

LivingCodeBO livingCode = baseLivingService.getLivingCode(living.getAreaId());

if (livingCode != null) {

Integer maxCount = livingCode.getMaxCount();

if (maxCount != null && livingCode.getZipCode() != null) {

living.setLivingCode(livingCode.getZipCode() + "A" + (maxCount + 1));

}

}

baseLivingService.insertSelective(living);

} else {

// 小区

living = baseLivingService.getLiving(data.getLivingId());

}

return living;

}

private DrawResHousePic adapterResHousePic(DrawBaseHouse house, DrawFileDataNew drawFileDataNew) {

DrawResHousePic pic = new DrawResHousePic();

pic.setBusinessId(house.getId().intValue());

pic.setPicHigh(drawFileDataNew.getHeight());

pic.setPicWeight(drawFileDataNew.getWidth());

pic.setPicSuffix(drawFileDataNew.getFileSuffix());

pic.setPicSize(drawFileDataNew.getFileSize());

pic.setPicPath(drawFileDataNew.getFilePath());

pic.setPicName(drawFileDataNew.getFileName());

pic.setPicFormat("");

pic.setSysCode(house.getHouseCode());

pic.setPicCode(house.getHouseCode());

pic.setIsDeleted(DrawBaseHouseConstant.IS\_DELETE\_INIT);

pic.setPicType(drawFileDataNew.getFileType());

pic.setFileKey(FileKey.BASE\_HOUSE\_PIC.getFileKey());

pic.setGmtCreate(new Date());

return pic;

}

@Override

@Transactional(propagation = Propagation.REQUIRED)

public void deleteDrawBaseHouse(Long houseId, Long userId) throws BusinessException {

DrawBaseHouse drawHouse = validateHouse(houseId, userId);

// v1.0.10 删除的户型执行删除动作默认为成功

if (drawHouse.getIsDeleted() == DrawBaseHouseConstant.IS\_DELETED) {

log.warn("{}(houseId), {}", houseId, "户型已被删除过");

return;

}

// 解锁绘制中状态

if (drawHouse.getBaseHouseId() != null && drawHouse.getBaseHouseId() != 0) {

if (drawHouse.getHouseStatus().equals(DrawBaseHouseConstant.CHECK\_APPLY.toString())) {

throw new BusinessException(false, ResponseEnum.DELETE\_FAIL);

}

if (drawHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_WAIT.toString())

|| drawHouse.getHouseStatus().equals(DrawBaseHouseConstant.HOUSE\_BAKE\_PROCESSING.toString())) {

throw new BusinessException(false, ResponseEnum.DELETE\_FAIL);

}

// 解锁户型绘制状态

log.info("户型{}(baseHouseId)解锁", drawHouse.getBaseHouseId());

baseHouseMapper.updateHouseDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_FACSIMILE, houseId);

}

// 删除 draw\_base\_house

log.info("删除户型{}(houseId)", houseId);

drawBaseHouseMapper.deleteById(houseId);

}

/\*\*

\* 验证户型

\* @param houseId

\* @param userId

\* @return

\* @throws BusinessException

\*/

private DrawBaseHouse validateHouse(Long houseId, Long userId) throws BusinessException {

if (null == houseId) {

throw new BusinessException(false, ResponseEnum.PARAM\_ERROR);

}

// 登录USERID

if (userId == null) {

userId = HouseAuthUtils.getRequestUserId();

}

// 检测数据是否存在

DrawBaseHouse drawHouse = drawBaseHouseMapper.get(houseId);

if (null == drawHouse) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

// 检测当前登陆者是否有权限操作

if (!drawHouse.getUserId().equals(userId)) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_BELONG\_USER);

}

return drawHouse;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)

public void checkDrawBaseHouse(DrawBaseHouseCheck check) throws BusinessException {

// 校验部分参数

DrawBaseHouse drawHouse = validateAndResponseDrawBaseHouse(check.getUserId(), check.getHouseId(), check.getCheckStatus());

// 处理户型

this.handlerCheck(check, drawHouse);

}

@Override

@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)

public void handlerCheck(DrawBaseHouseCheck check, DrawBaseHouse drawHouse) {

log.info("户型 {}(houseId)进行审核处理", drawHouse.getId());

// 修改户型审核状态

DrawBaseHouse checkHouse = new DrawBaseHouse();

checkHouse.setHouseStatus(DrawBaseHouseConstant.CHECK\_APPLY.toString());

checkHouse.setRejectReason(check.getRejectReason());

checkHouse.setId(check.getHouseId());

// 审核人

checkHouse.setAuditorId(HouseAuthUtils.getRequestUserId());

// 审核时间

checkHouse.setAuditTime(new Date());

int updateCount = drawBaseHouseMapper.updateDrawHouseStatus3(checkHouse, check.getCheckStatus());

if (updateCount == 1) {

// 判断前端需要处理的类型, 通过就生成任务

if (check.getCheckStatus().intValue() == DrawBaseHouseConstant.HOUSE\_BAKE\_WAIT) {

// 烘焙任务

DrawBakeTask task = getDrawBakeTask(check.getUserId(), drawHouse);

drawBakeTaskService.saveDrawBakeTask(task);

// 烘焙任务明细

List<DrawBakeTaskDetail> drawBakeTaskDetails = getDrawBakeTaskDetails(check.getHouseId(), task);

// 批量保存烘焙任务明细

drawBakeTaskService.batchSaveDrawBakeTaskDetail(drawBakeTaskDetails);

// 已被烘焙数据更改状态 ==> 烘焙中(base\_house => 4)

baseHouseMapper.updateHouseDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_BAKE, check.getHouseId());

log.info("{}(houseId)户型添加到烘焙队列中", drawHouse.getId());

}

}

}

@Override

public DrawBaseHouse getBaseHouse(Long houseId) {

return drawBaseHouseMapper.getBaseHouseById(houseId);

}

@Override

public DrawBaseHouseBO loadDrawBaseHouse(Long houseId, Long userId) throws BusinessException {

// 户型数据

DrawBaseHouse house = validateHouse(houseId, userId);

// 户型空间数据

List<DrawSpaceCommonBO> spaceCommons = drawSpaceCommonService.listSpaceCommon(houseId);

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

for (DrawSpaceCommonBO spaceCommonBO : spaceCommons) {

spaceIds.add(spaceCommonBO.getSpaceId());

}

// 空间产品数据

Map<Long, List<DrawBaseProductBO>> productMap = getProductDataMap(spaceIds);

// 组装返回数据对象

return getDrawBaseHouseBO(house, spaceCommons, productMap);

}

private DrawBaseHouseBO getDrawBaseHouseBO(DrawBaseHouse house, List<DrawSpaceCommonBO> spaceCommons, Map<Long, List<DrawBaseProductBO>> productMap) {

DrawBaseHouseBO response = new DrawBaseHouseBO();

response.setHouseCode(house.getHouseCode());

response.setId(house.getId());

response.setHouseName(house.getHouseName());

for (DrawSpaceCommonBO space : spaceCommons) {

List<DrawBaseProductBO> drawBaseProductBOList = productMap.get(space.getSpaceId());

space.setProductData(drawBaseProductBOList);

}

response.setDrawSpaceCommonBOList(spaceCommons);

return response;

}

private DrawBaseHouse validateAndResponseDrawBaseHouse(Long userId, Long houseId, Integer checkStatus) throws BusinessException {

// 是否有审核的权限

if (HouseAuthUtils.hasApprovedPermission()) {

throw new BusinessException(false, ResponseEnum.FORBIDDEN);

}

// 简单校验数据

userId = (userId == null) ? HouseAuthUtils.getRequestUserId() : userId;

if (null == userId || null == houseId || null == checkStatus) {

throw new BusinessException(false, ResponseEnum.PARAM\_ERROR);

}

// 检验审核操作数据合法

if (checkStatus.intValue() != DrawBaseHouseConstant.HOUSE\_BAKE\_WAIT && checkStatus.intValue() != DrawBaseHouseConstant.CHECK\_REJECT) {

throw new BusinessException(false, ResponseEnum.CHECK\_STATUS\_UNKNOW);

}

// 检验户型数据是否存在

DrawBaseHouse house = drawBaseHouseMapper.getBaseHouseById(houseId);

if (null == house) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

return house;

}

private DrawBakeTask getDrawBakeTask(Long userId, DrawBaseHouse house) {

DrawBakeTask task = new DrawBakeTask();

task.setHouseCode(house.getHouseCode());

task.setHouseId(house.getId());

task.setStatus(DrawBaseHouseConstant.BAKE\_TASK\_STATUS\_INIT);

task.setUserId(userId);

task.setGmtCreate(new Date());

task.setQueueName(getQueueName(house.getId()));

task.setPriority(5);

return task;

}

/\*\*

\* 获取队列名

\* 1、内部用户绘制的户型，进入内部队列 INTERNAL,

\* 2、外部的队列分成两种

\* 1)、普通的外部户型，进入外部队列(EXTERNAL)

\* 2)、外部用户绘制的毛坯房，进入外部的毛坯房队列(ROUGH\_HOUSE)

\* @param houseId

\* @return

\*/

private String getQueueName(Long houseId) {

// 数据类型

Map<String, Integer> map = drawBaseHouseMapper.getDataType(houseId);

Integer origin = map.get("origin");

Integer dataType = map.get("dataType");

log.debug("获取烘焙的队列名，{}(origin), {}(dataType)", origin, dataType);

// 内部户型队列

if (Objects.isNull(dataType) || dataType.equals(DrawBaseHouseConstant.DATA\_PLATFORM)) {

return BakeTaskQueue.INTERNAL.toString();

}

// 毛坯房

if (DrawBaseHouseConstant.ORIGIN\_ROUGH\_HOUSE.equals(origin)) {

return BakeTaskQueue.ROUGH\_HOUSE.toString();

}

// 普通外部队列

return BakeTaskQueue.EXTERNAL.toString();

}

/\*\*

\* 获取烘焙任务的内容

\*

\* @param houseId

\* @param task

\* @return

\* @throws BusinessException

\*/

private List<DrawBakeTaskDetail> getDrawBakeTaskDetails(Long houseId, DrawBakeTask task) throws BusinessException {

// 户型下的空间集合

List<DrawSpaceCommonBO> spaceCommons = drawSpaceCommonService.listSpaceCommon(houseId);

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

throw new BusinessException(false, ResponseEnum.SPACE\_NOT\_FOUND);

}

// 空间主键集合

List<Long> spaceIds = spaceCommons.stream().map(DrawSpaceCommonBO::getSpaceId).collect(Collectors.toList());

// 获取空间下的产品信息

Map<Long, List<DrawBaseProductBO>> productMap = getProductDataMap(spaceIds);

// objectMapper null值不参与序列化

ObjectMapper objectMapper = new ObjectMapper();

objectMapper.setSerializationInclusion(JsonInclude.Include.NON\_NULL);

// 烘焙任务详情集合

List<DrawBakeTaskDetail> drawBakeTaskDetails = new ArrayList<>();

for (DrawSpaceCommonBO spaceCommonBO : spaceCommons) {

DrawBakeTaskDetail taskDetail = new DrawBakeTaskDetail();

taskDetail.setSpaceCode(spaceCommonBO.getSpaceCode());

taskDetail.setSpaceFileIds(String.valueOf(spaceCommonBO.getFileId()));

taskDetail.setSpaceFilePath(spaceCommonBO.getFilePath());

taskDetail.setSpaceId(spaceCommonBO.getSpaceId());

taskDetail.setTaskId(task.getId());

taskDetail.setStatus(DrawBaseHouseConstant.BAKE\_TASK\_STATUS\_INIT);

// 空间下的产品集合, 产品json串信息

String jsonProductInfo = getJsonProductInfo(objectMapper, productMap.get(spaceCommonBO.getSpaceId()));

taskDetail.setProductData(jsonProductInfo);

// add

drawBakeTaskDetails.add(taskDetail);

}

return drawBakeTaskDetails;

}

/\*\*

\* 空间下的产品集合, 产品json串信息

\*

\* @param objectMapper

\* @param product

\* @return

\*/

private String getJsonProductInfo(ObjectMapper objectMapper, List<DrawBaseProductBO> product) {

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

return null;

}

try {

return objectMapper.writeValueAsString(product);

} catch (JsonProcessingException e) {

throw new BusinessException(false, ResponseEnum.JSON\_FORMAT\_ERROR, e);

}

}

/\*\*

\* 获取空间产品信息

\*

\* @param spaceIds

\* @return

\*/

private Map<Long, List<DrawBaseProductBO>> getProductDataMap(List<Long> spaceIds) {

// 空间id为key，空间下的产品信息集合为value

Map<Long, List<DrawBaseProductBO>> productMap = new HashMap<>();

// 空间下的所有的商品信息

List<DrawBaseProductBO> products = drawBaseProductService.listBaseProductBo(spaceIds, 1);

// List<DrawBaseProductBO> products = drawBaseProductService.listBaseProductBo(spaceIds, null);

if (products != null && !products.isEmpty()) {

productMap = products.stream().collect(Collectors.groupingBy(DrawBaseProductBO::getSpaceId));

}

return productMap;

}

@Override

@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)

public DrawBaseHouseSubmitDTO submit(String data, UserLoginBO loginBO) throws DrawBusinessException {

// 验证数据

DrawBaseHouseSubmitDTO dtoNew = validateSubmitDTO(data);

if (loginBO == null) {

log.warn("用户信息为空");

loginBO = HouseAuthUtils.getUnknownUser();

}

// 清空旧数据处理

this.handlerSubmitHouse(dtoNew.getHouseId().longValue());

// 保存所有信息(空间/样板房/产品)

dtoNew = this.saveDrawBaseHouseBySubmit(dtoNew, loginBO);

// 提交后相当于发起审核操作，保存未提交都不能再审核列表中展示

drawBaseHouseMapper.updateDrawHouseStatus(DrawBaseHouseConstant.CHECK\_APPLY, null, dtoNew.getHouseId().longValue());

// 用户个人户型数据(普通绘制员、内部演示员绘制的数据)直接跳过审核权限，进入烘焙阶段

this.handlerJumpHouseCheck(dtoNew, loginBO);

return dtoNew;

}

@Override

@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)

public void submitV2(String data) throws DrawBusinessException {

DrawBaseHouseSubmitDTO dtoNew = null;

try {

UserLoginBO loginBO = HouseAuthUtils.getUserLoginBO();

// 验证数据

dtoNew = validateSubmitDTO(data);

// 清空旧数据处理

this.handlerSubmitHouse(dtoNew.getHouseId().longValue());

/\*\*

\* v2 submit 处理

\*/

drawBaseHouseServiceV2.submit(dtoNew, loginBO);

// 提交后相当于发起审核操作，保存未提交都不能再审核列表中展示

drawBaseHouseMapper.updateDrawHouseStatus(DrawBaseHouseConstant.CHECK\_APPLY, null, dtoNew.getHouseId().longValue());

// 用户个人户型数据(普通绘制员、内部演示员绘制的数据)直接跳过审核权限，进入烘焙阶段

this.handlerJumpHouseCheck(dtoNew, loginBO);

// 处理量房status

drawVolumeRoomService.handlerVolumeRoom(dtoNew.getDrawHouse(), VolumeRoomConstant.PERFECTED.getStatus(), VolumeRoomConstant.PARSING\_SUCCESS.getStatus());

} catch (Exception e) {

// 异常时，删除上传的资源文件

resHandlerErrorService.handlerErrorWithSubmit(dtoNew, data);

throw e;

}

}

/\*\*

\* 验证提交户型的DTO

\*

\* @param data

\* @return

\*/

private DrawBaseHouseSubmitDTO validateSubmitDTO(String data) {

if (StringUtils.isEmpty(data)) {

throw new DrawBusinessException(false, ResponseEnum.PARAM\_ERROR\_JSONSTR\_EMPTY);

}

DrawBaseHouseSubmitDTO dtoNew;

try {

dtoNew = new ObjectMapper().readValue(data, DrawBaseHouseSubmitDTO.class);

} catch (Exception e) {

throw new BusinessException(false, ResponseEnum.JSON\_FORMAT\_ERROR, e);

}

if (dtoNew == null || dtoNew.getHouseId() == null) {

throw new DrawBusinessException(false, ResponseEnum.PARAM\_ERROR);

}

// 验证户型是否存在

DrawBaseHouse drawHouse = drawBaseHouseMapper.getBaseHouseById(dtoNew.getHouseId().longValue());

if (drawHouse == null) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

// 绘制户型

dtoNew.setDrawHouse(drawHouse);

dtoNew.setDataType(Objects.isNull(drawHouse.getDataType()) ? DrawBaseHouseConstant.DATA\_PLATFORM : drawHouse.getDataType());

return dtoNew;

}

/\*\*

\* 保存所有数据(空间/样板房/产品/样板房产品表)

\*

\* @param loginBO

\* @param dto

\* @return

\* @throws DrawBusinessException

\* @author huangsongbo

\*/

private DrawBaseHouseSubmitDTO saveDrawBaseHouseBySubmit(DrawBaseHouseSubmitDTO dto, UserLoginBO loginBO) throws DrawBusinessException {

// 创建空间数据(同时更新对应文件信息)

dto = drawSpaceCommonService.saveDrawSpaceCommonBySubmit(dto, loginBO);

// 创建样板房数据(同时更新对应文件信息)

dto = drawDesignTempletService.saveDrawDesignTempletBySubmit(dto, loginBO);

// 创建基本产品数据(同时更新对应文件信息)

dto = drawBaseProductService.saveDrawBaseProductBySubmit(dto, loginBO);

// 创建样板房产品表(同时更新对应文件信息)

dto = drawDesignTempletProductService.saveDesignTempletProductBySubmit(dto, loginBO);

return dto;

}

/\*\*

\* 普通绘制员、内部演示员直接跳过审核权限，进入烘焙阶段

\*

\* @param dtoNew

\*/

private void handlerJumpHouseCheck(DrawBaseHouseSubmitDTO dtoNew, UserLoginBO loginUser) {

if (!DrawBaseHouseConstant.DATA\_PERSONAL.equals(dtoNew.getDataType())) {

log.info("{}(houseId)是内部员绘制户型，需要审核户型流程", dtoNew.getHouseId());

return;

}

if (!HouseAuthUtils.hasJumpHouseCheckPermission()) {

log.info("{}(userId)用户权限不足，无法跳过审核户型流程", loginUser.getId());

return;

}

DrawBaseHouse drawHouse = drawBaseHouseMapper.selectByPrimaryKey(dtoNew.getHouseId().longValue());

if (drawHouse == null) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

log.info("{}(houseId)户型跳过审核户型流程，直接进入烘焙队列", drawHouse.getId());

DrawBaseHouseCheck drawCheck = new DrawBaseHouseCheck();

drawCheck.setHouseId(drawHouse.getId());

drawCheck.setCheckStatus(DrawBaseHouseConstant.HOUSE\_BAKE\_WAIT);

drawCheck.setUserId(HouseAuthUtils.getRequestUserId());

// 审核户型流程处理

this.handlerCheck(drawCheck, drawHouse);

}

@Override

public Map<String, Object> getHouses(DrawBaseHouseSearch query) {

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

// 判断用户的户型绘制角色

boolean checkUser = false;

String role = checkUserDrawRole();

if (HouseRole.COMMON\_HOUSE\_DRAW.equals(role)) {

checkUser = true;

}

if (checkUser) {

query.setIsPublic(1);

}

query.setPageNum((query.getPageNum() - 1) \* query.getPageSize());

int count;

List<DrawBaseHouseBO> list;

if (StringUtils.isEmpty(query.getAreaCode()) && query.getLivingId() == null) {

count = drawBaseHouseMapper.selectCountAllHouses();

} else {

count = drawBaseHouseMapper.selectConutHouses(query);

}

if (count > 0) {

list = drawBaseHouseMapper.getHouses(query);

list.stream().filter(l -> l.getPicPath() == null).forEach(r -> {

String picPath = drawBaseHouseMapper.getSnapPicPathById(r.getId());

r.setPicPath(picPath);

});

for (DrawBaseHouseBO drawBaseHouseBO : list) {

if (StringUtils.isEmpty(drawBaseHouseBO.getDealStatus())) {

drawBaseHouseBO.setDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_FACSIMILE);

}

if (checkUser) {

if (query.getUserId().equals(drawBaseHouseBO.getUserId())) {

drawBaseHouseBO.setDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_EDIT);

} else {

drawBaseHouseBO.setDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_FACSIMILE);

}

} else {

// 自己继续修改

if (DrawBaseHouseConstant.DEAL\_STATUS\_RERUNNING.equals(drawBaseHouseBO.getDealStatus())

&& query.getUserId().equals(drawBaseHouseBO.getUserId())) {

drawBaseHouseBO.setDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_EDIT);

}

}

// // v1.1.2 检验编辑权限

// // add by songjianming@sanduspace.cn on 2018/7/31

// if (!HouseAuthUtils.hasHouseModification() &&

// DrawBaseHouseConstant.DEAL\_STATUS\_EDIT.equals(drawBaseHouseBO.getDealStatus())) {

// // 禁止编辑

// drawBaseHouseBO.setDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_DISABLE);

// }

}

dataMap.put("list", list);

dataMap.put("total", count);

}

return dataMap;

}

private String checkUserDrawRole() {

UserLoginBO userLoginBO = HouseAuthUtils.getUserLoginBO();

Set<String> roles = userLoginBO.getRoles();

if (roles != null) {

if (!roles.contains(HouseRole.COMMON\_HOUSE\_DRAW) && !roles.contains(HouseRole.INTERNAL\_HOUSE\_DRAW)) {

throw new BizException("您没有户型绘制员的角色");

}

if ((roles.contains(HouseRole.COMMON\_HOUSE\_DRAW) && roles.contains(HouseRole.INTERNAL\_HOUSE\_DRAW))

|| roles.contains(HouseRole.INTERNAL\_HOUSE\_DRAW)) {

return HouseRole.INTERNAL\_HOUSE\_DRAW;

} else {

return HouseRole.COMMON\_HOUSE\_DRAW;

}

}

return HouseRole.COMMON\_HOUSE\_DRAW;

}

public DrawHouseAndAreaVO facsimileHouse(Long houseId, String type) {

DrawBaseHouseBO drawBaseHouseBO = null;

switch (type) {

case "facsimile":

drawBaseHouseBO = drawBaseHouseMapper.selectHouseAndPicPathById(houseId);

break;

case "continueDraw":

drawBaseHouseBO = drawBaseHouseMapper.selectDrawHouseRestoreFile(houseId);

// 量房数据

if (drawBaseHouseBO != null && !PlatformType.SANDU.getType().equals(drawBaseHouseBO.getPlatformType())) {

List<VolumeRoomHouseBO> volumeRoom = drawVolumeRoomMapper.listVolumeRoomByHouseId(Lists.newArrayList(drawBaseHouseBO), VolumeRoomConstant.DEFAULTS.getStatus());

if (volumeRoom != null && !volumeRoom.isEmpty()) {

drawBaseHouseBO.setFilePath(volumeRoom.get(0).getLfFilePath());

}

}

break;

case "edit":

drawBaseHouseBO = drawBaseHouseMapper.selectHouseAndRestoreFile(houseId);

// v1.1.1 重置的户型，没有还原文件，取绘制表里的还原文件

if (drawBaseHouseBO != null && drawBaseHouseBO.getFilePath() == null) {

DrawHouseResouceBO houseFileResource = drawBaseHouseMapper.getHouseFileResource(houseId);

drawBaseHouseBO.setFilePath(houseFileResource == null ? null : houseFileResource.getRestoreFilePath());

}

break;

default:

break;

}

DrawHouseAndAreaVO drawHouseAndAreaVO = new DrawHouseAndAreaVO();

drawHouseAndAreaVO.setDrawBaseHouseBO(drawBaseHouseBO);

drawHouseAndAreaVO.setBaseAreaBO(handlerAreaByLivingId(drawBaseHouseBO));

return drawHouseAndAreaVO;

}

/\*\*

\* v1.1.1

\* 1、优化省市区查询，根据小区检索其所在的省市区（以后版本需要加入redis缓存）

\* 、而不检索出所有的省市区进行对比，这样非常低效；

\* 2、修复临摹时，省市区某个一个值引起 NullPointerException 问题

\*

\* @param drawBaseHouseBO

\* @return

\*/

private BaseAreaBO handlerAreaByLivingId(DrawBaseHouseBO drawBaseHouseBO) {

if (drawBaseHouseBO == null || drawBaseHouseBO.getAreaId() == null) {

return new BaseAreaBO();

}

BaseAreaLivingBO livingBO = baseLivingMapper.listAreaByLivingId(drawBaseHouseBO.getAreaId());

if (livingBO == null) {

return new BaseAreaBO();

}

// 区

BaseAreaBO distArea = new BaseAreaBO();

distArea.setAreaCode(livingBO.getDistCode());

distArea.setAreaName(livingBO.getDistName());

// 市

BaseAreaBO cityArea = new BaseAreaBO();

cityArea.setAreaCode(livingBO.getCityCode());

cityArea.setAreaName(livingBO.getCityName());

cityArea.setChildAreas(Lists.newArrayList(distArea));

// 省

BaseAreaBO proArea = new BaseAreaBO();

proArea.setAreaCode(livingBO.getProCode());

proArea.setAreaName(livingBO.getProName());

proArea.setChildAreas(Lists.newArrayList(cityArea));

return proArea;

}

@Override

@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)

public Map<String, Object> callback(String jsonStr) throws DrawBusinessException {

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

// 转化json

DrawBaseHouseControllerCallBackDTO dto;

try {

ObjectMapper objectMapper = new ObjectMapper();

// 如果是空对象的时候,不抛异常

objectMapper.configure(SerializationFeature.FAIL\_ON\_EMPTY\_BEANS, false);

// 反序列化的时候如果多了其他属性,不抛出异常

objectMapper.configure(DeserializationFeature.FAIL\_ON\_UNKNOWN\_PROPERTIES, false);

dto = objectMapper.readValue(jsonStr, DrawBaseHouseControllerCallBackDTO.class);

// dto = new ObjectMapper().readValue(jsonStr, DrawBaseHouseControllerCallBackDTO.class);

} catch (Exception e) {

throw new DrawBusinessException(false, ResponseEnum.JSON\_FORMAT\_ERROR, e);

}

if (dto == null) {

throw new DrawBusinessException(false, ResponseEnum.PARAM\_ERROR);

}

log.info("正在处理烘焙任务 {}(taskDetailId)", dto.getTaskDetailId());

// 烘焙失败处理

if (Objects.equals(dto.getStatus(), BakeCallbackStatus.FAILURE.getStatus())) {

this.handlerBakeTaskFailed(dto);

String msg = Utils.replace(ResponseEnum.BAKE\_TASK\_FIAL.getMessage(), Objects.toString(dto.getTaskDetailId()));

map.put("msg", msg);

} else {

// 以下是烘焙成功的处理

// 更新挂节点/模型文件等信息

dto = this.updateAllData(dto);

// 转换数据

Map<Long, Long> transformAllData = this.transformAllData(dto.getTaskDetailId());

map.put("msg", "烘焙成功");

// 推送搜索服务产品

if (!transformAllData.isEmpty()) {

Collection<Long> values = transformAllData.values();

this.doSyncToIndex(SyncMessage.ACTION\_ADD, new ArrayList<>(values));

log.info("推送搜索产品，push data => {}", Arrays.toString(values.toArray()));

}

}

log.info("{}(taskDetailId)烘焙任务处理完成", dto.getTaskDetailId());

return map;

}

@Override

@Transactional(propagation = Propagation.REQUIRED)

public String callbackV2(DrawBaseHouseCallbackDTO dtoNew) {

return drawBaseHouseServiceV2.callback(dtoNew);

}

@Override

@Transactional(propagation = Propagation.NOT\_SUPPORTED)

public void doSyncToIndex(Integer messageAction, List<Long> ids) {

try {

List<Map> content = ids.stream().map(item -> {

HashMap<String, Long> tmp = new HashMap<>(1);

tmp.put("id", item);

return tmp;

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

SyncMessage message = new SyncMessage();

message.setAction(messageAction);

message.setMessageId("house-draw-" + System.currentTimeMillis());

message.setModule(SyncMessage.MODULE\_PRODUCT);

message.setPlatformType(SyncMessage.PlatFormType.pcHouseDraw.toString());

message.setObject(content);

queueService.send(message);

} catch (Exception e) {

// 推送服务不参与事务处理，不抛异常

log.error("推送搜索异常 => {}", e.getMessage(), e);

}

}

/\*\*

\* 转换数据

\*

\* @param taskDetailId

\* @throws DrawBusinessException

\*/

@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)

public Map<Long, Long> transformAllData(Long taskDetailId) throws DrawBusinessException {

String logMes = "function:DrawBaseHouseServiceImpl.transformAllData->";

// 参数验证 ->start

if (taskDetailId == null) {

log.error(logMes + "taskDetailId = null");

throw new DrawBusinessException(false, ResponseEnum.DRAWBASEHOUSESERVICEIMPL\_TRANSFORMALLDATA\_FAILED);

}

DrawBakeTaskDetail drawBakeTaskDetail = drawBakeTaskDetailService.get(taskDetailId);

if (drawBakeTaskDetail == null) {

log.error(logMes + "drawBakeTaskDetail = null; taskDetailId = " + taskDetailId);

throw new DrawBusinessException(false, ResponseEnum.DRAWBASEHOUSESERVICEIMPL\_TRANSFORMALLDATA\_FAILED);

}

DrawSpaceCommon drawSpaceCommon = drawSpaceCommonService.get(drawBakeTaskDetail.getSpaceId());

if (drawSpaceCommon == null) {

log.error(logMes + "drawSpaceCommon = null; drawSpaceCommonId = " + drawBakeTaskDetail.getSpaceId());

throw new DrawBusinessException(false, ResponseEnum.DRAWBASEHOUSESERVICEIMPL\_TRANSFORMALLDATA\_FAILED);

}

List<DrawDesignTemplet> drawDesignTempletList = drawDesignTempletService.findAllBySpaceCommonId(drawBakeTaskDetail.getSpaceId());

if (drawDesignTempletList == null || drawDesignTempletList.size() == 0) {

log.error(logMes + "drawDesignTempletList = null or size = 0; drawSpaceCommonId = " + drawBakeTaskDetail.getSpaceId());

throw new DrawBusinessException(false, ResponseEnum.DRAWBASEHOUSESERVICEIMPL\_TRANSFORMALLDATA\_FAILED);

}

DrawBaseHouse drawBaseHouse = drawBaseHouseService.get(drawSpaceCommon.getDrawBaseHouseId());

if (drawBaseHouse == null) {

log.error(logMes + "drawBaseHouse = null; drawBaseHouseId = " + drawSpaceCommon.getDrawBaseHouseId());

throw new DrawBusinessException(false, ResponseEnum.DRAWBASEHOUSESERVICEIMPL\_TRANSFORMALLDATA\_FAILED);

}

// 参数验证 ->end

// TODO 清空正式表数据

this.handlerCallbackHouse(drawBaseHouse.getId(), drawBakeTaskDetail.getTaskId());

// 转换户型数据

Map<Long, Long> drawBaseHouseTransformMap = new HashMap<>();

if (drawBaseHouse.getBaseHouseId() != null && drawBaseHouse.getBaseHouseId() > 0) {

// 标识该drawBaseHouse已经被转换过

drawBaseHouseTransformMap.put(drawBaseHouse.getId(), drawBaseHouse.getBaseHouseId());

log.info("户型 {}(drawBaseHouseId)已经被转换过, 不需要再转化", drawBaseHouse.getId());

} else {

drawBaseHouseTransformMap = this.transformToBaseHouse(drawBaseHouse);

}

// 同步户型资源

this.transformHouseResource(drawBaseHouse, drawBakeTaskDetail.getTaskId());

// 转换产品数据

Map<Long, Long> drawBaseProductTransformMap = new HashMap<>();

for (DrawDesignTemplet drawDesignTemplet : drawDesignTempletList) {

List<DrawBaseProduct> drawBaseProductList = drawBaseProductService.findAllByDrawDesignTempletId(drawDesignTemplet.getId());

// 剔除软装白膜(因为正是表中已经存在了软装白膜产品数据,所以不需要重复新增) ->start

if (drawBaseProductList != null && drawBaseProductList.size() > 0) {

for (int index = 0; index < drawBaseProductList.size(); index++) {

DrawBaseProduct drawBaseProduct = drawBaseProductList.get(index);

if (DrawDesignTempletProductConstant.SOFT\_PRODUCT.equals(drawBaseProduct.getCreateProductStatus())) {

log.debug("跳过软装产品 {}(productId)处理", drawBaseProduct.getId());

drawBaseProductList.remove(index);

index--;

}

}

}

// 提出软装白膜(因为正是表中已经存在了软装白膜产品数据,所以不需要重复新增) ->end

Map<Long, Long> drawBaseProductTransformMapItem = drawBaseProductService.transformToBaseProduct(drawBaseProductList);

drawBaseProductTransformMap.putAll(drawBaseProductTransformMapItem);

}

// 转换空间数据

Map<Long, Long> drawSpaceCommonTransformMap = drawSpaceCommonService.transformToSpaceCommon(drawSpaceCommon,

drawBaseHouseTransformMap.get(drawBaseHouse.getId()));

// 转换样板房数据

Map<Long, Long> drawDesignTempletTransformMap = drawDesignTempletService.transformToDesignTemplet(drawDesignTempletList,

drawSpaceCommonTransformMap.get(drawSpaceCommon.getId()));

// 转换样板房产品数据

for (DrawDesignTemplet drawDesignTemplet : drawDesignTempletList) {

List<DrawDesignTempletProduct> drawDesignTempletProductList = drawDesignTempletProductService.findAllByDesignTempletId(drawDesignTemplet.getId());

drawDesignTempletProductService.transformToDesignTempletProduct(drawDesignTempletProductList, drawDesignTempletTransformMap.get(drawDesignTemplet.getId()),

drawBaseProductTransformMap);

}

// v1.0.7处理相邻样板房的跳跃点信息(埋点)

// created by songjianming@sanduspace.cn on 2018/5/24

designTempletJumpPositionRelService.transformJumpPositionRel(drawBaseHouse);

// 处理户型的状态和资源

// created by songjianming@sanduspace.cn on 2018/2/6

this.handlerHouseStatus(drawBaseHouse, taskDetailId);

return drawBaseProductTransformMap;

}

/\*\*

\* 处理户型状态

\*

\* @param drawHouse

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED)

public void handlerHouseStatus(DrawBaseHouse drawHouse, Long taskDetailId) {

if (drawHouse != null && taskDetailId != null) {

// 剩余需要烘焙的空间

int taskCount = drawBakeTaskService.getTaskDetailCount(taskDetailId);

if (taskCount <= 1) {

// draw\_base\_house ==> house\_status(HOUSE\_BAKE\_SUCCESS)

drawBaseHouseMapper.updateDrawHouseStatus(DrawBaseHouseConstant.HOUSE\_BAKE\_SUCCESS, null, drawHouse.getId());

// 主任务

drawBakeTaskService.updateTaskStatusByTaskDetailId(DrawBaseHouseConstant.BAKE\_TASK\_STATUS\_SUCCESS, taskDetailId);

// base\_house的所有的空间烘焙完成 deal\_status(DEAL\_STATUS\_EDIT)

baseHouseMapper.updateHouseDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_EDIT, drawHouse.getId());

log.info("{}(houseId)户型下的所有空间烘焙完成", drawHouse.getId());

}

// 更新任务详情

drawBakeTaskDetailService.updateTaskDetailStatus(DrawBaseHouseConstant.BAKE\_TASK\_STATUS\_SUCCESS, taskDetailId);

} else {

log.warn("烘焙的户型状态处理异常, drawHouse => {}, taskDetailId => {}", drawHouse, taskDetailId);

}

}

/\*\*

\* 转化户型资源

\*

\* @param drawHouse

\* @param taskId

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED)

public void transformHouseResource(DrawBaseHouse drawHouse, Long taskId) {

if (drawHouse == null) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

// 是否第一个空间烘焙

int taskCount = drawBakeTaskService.getBakeSuccessTask(taskId);

if (taskCount != DrawBaseHouseConstant.EMPTY\_DATA\_INIT\_FLAG) {

log.info("户型 {}(houseId)的资源已被同步过，不需要同步", drawHouse.getId());

return;

}

this.updateHouseResource(drawHouse);

}

/\*\*

\* 截图、还原文件

\*

\* @param drawHouse

\*/

private void updateHouseResource(DrawBaseHouse drawHouse) {

if (drawHouse == null) {

log.warn("户型的不存在");

return;

}

log.info("开始同步绘制户型{}(houseId)图片资源", drawHouse.getId());

if (drawHouse.getBaseHouseId() == null || drawHouse.getBaseHouseId() <= 0) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

// base\_house

BaseHouse house = baseHouseMapper.selectByPrimaryKey(drawHouse.getBaseHouseId());

if (house == null) {

log.error("户型 {}(baseHouseId)不存在或被删除", drawHouse.getBaseHouseId());

// 户型被删除不处理

// throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

return;

}

// 绘制户型的资源 => pic 、snap\_pic、restore\_file

DrawHouseResouceBO drawResource = drawBaseHouseMapper.getDrawHouseFileResource(drawHouse.getId());

if (drawResource == null) {

log.warn("户型 {}(houseId)的资源不存在", drawHouse.getId());

return;

}

// update snap\_pic\_id

Long snapPicId = house.getSnapPicId();

if (snapPicId != null && snapPicId > 0) {

if (!StringUtils.isEmpty(drawResource.getSnapPicPath())) {

ResHousePic snap = new ResHousePic();

snap.setId(snapPicId);

snap.setPicPath(drawResource.getSnapPicPath());

resHousePicMapper.updateByPrimaryKeySelective(snap);

}

} else {

// 同步 snap\_pic\_id

copySnapPicById(drawResource.getSnapPicId(), house);

}

// origin => 1 是手动绘制、导入图临摹需要同步户型图，老数据不需要同步户型图

if (drawHouse.getOrigin() != null && !drawHouse.getOrigin().equals(DrawBaseHouseConstant.ORIGIN\_OLD\_COPY)) {

// pic\_res\_1id

Integer picRes1Id = house.getPicRes1Id();

if (picRes1Id != null && picRes1Id > 0) {

if (!StringUtils.isEmpty(drawResource.getPicRes1Path())) {

ResHousePic pic = new ResHousePic();

pic.setId(picRes1Id.longValue());

pic.setPicPath(drawResource.getPicRes1Path());

resHousePicMapper.updateByPrimaryKeySelective(pic);

}

} else {

// 同步 pic\_res1\_id

copyPicRes1ById(Long.valueOf(Objects.toString(drawResource.getPicRes1Id(), "0")), house, drawHouse.getOrigin());

}

}

// restore\_file\_id

Long restoreFileId = house.getRestoreFileId();

if (restoreFileId != null && restoreFileId > 0) {

if (!StringUtils.isEmpty(drawResource.getRestoreFilePath())) {

ResFile restoreFile = new ResFile();

restoreFile.setId(restoreFileId);

restoreFile.setFilePath(drawResource.getRestoreFilePath());

resFileMapper.updateByPrimaryKeySelective(restoreFile);

}

} else {

// 同步 restore file

copyRestoreFileById(drawResource.getRestoreFileId(), house);

}

// 更新资源

if ((house.getPicRes1Id() != null && house.getPicRes1Id() > 0)

|| (house.getSnapPicId() != null && house.getSnapPicId() > 0)

|| (house.getRestoreFileId() != null && house.getRestoreFileId() > 0)) {

baseHouseMapper.updateHouseResouce(house);

log.info("update sql户型{}(houseId)的资源picResId = {}, snapPicId = {}, restoreFileId = {}", house.getId(),

house.getPicRes1Id(), house.getSnapPicId(), house.getRestoreFileId());

}

log.info("绘制户型{}(houseId)同步图片资源结束", drawHouse.getId());

}

/\*\*

\* copy restore file

\*

\* @param restoreFileId

\*/

private void copyRestoreFileById(Long restoreFileId, BaseHouse house) {

// restore\_file\_id;

if (restoreFileId != null && restoreFileId != 0) {

DrawResFile drawFile = drawResFileMapper.selectByPrimaryKey(restoreFileId);

if (drawFile != null) {

ResFile file = new ResFile();

try {

drawFile.setId(null);

BeanUtils.copyProperties(file, drawFile);

} catch (IllegalAccessException | InvocationTargetException e) {

log.error("同步户型 {}(houseId)还原文件资源异常", house.getId(), e);

throw new BusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("同步户型" + house.getId() + "(houseId)还原文件资源异常"));

}

resFileMapper.insertSelective(file);

house.setRestoreFileId(file.getId());

}

}

}

/\*\*

\* copy snap pic

\*

\* @param snapPicId

\* @param house

\*/

private void copySnapPicById(Long snapPicId, BaseHouse house) {

if (snapPicId != null && snapPicId != 0) {

DrawResHousePic drawSnap = drawResHousePicMapper.getDrawResHousePic(snapPicId);

if (drawSnap != null) {

ResHousePic snap = new ResHousePic();

try {

drawSnap.setId(null);

BeanUtils.copyProperties(snap, drawSnap);

} catch (IllegalAccessException | InvocationTargetException e) {

log.error("同步户型 {}(houseId)截图资源异常", house.getId(), e);

throw new BusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("同步户型" + house.getId() + "(houseId)截图资源异常"));

}

resHousePicMapper.insertSelective(snap);

house.setSnapPicId(snap.getId());

}

}

}

/\*\*

\* copy pic res1

\*

\* @param picRes1Id

\*/

private void copyPicRes1ById(Long picRes1Id, BaseHouse house, Integer origin) {

// pic\_res1\_id

// origin => 1 是手动绘制、导入图临摹需要同步户型图，老数据不需要同步户型图

if (origin != null && !origin.equals(DrawBaseHouseConstant.ORIGIN\_OLD\_COPY)) {

if (picRes1Id != null && picRes1Id != 0) {

DrawResHousePic drawPic = drawResHousePicMapper.getDrawResHousePic(picRes1Id);

if (drawPic == null) {

log.warn("未找到 {}(picRes1Id)的户型图资源", picRes1Id);

return;

}

ResHousePic pic = new ResHousePic();

try {

drawPic.setId(null);

BeanUtils.copyProperties(pic, drawPic);

} catch (IllegalAccessException | InvocationTargetException e) {

log.error("同步户型 {}(houseId)图资源异常", house.getId(), e);

throw new BusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("同步户型" + house.getId() + "(houseId)图资源异常"));

}

resHousePicMapper.insertSelective(pic);

house.setPicRes1Id(pic.getId().intValue());

}

}

}

/\*\*

\* 更新模型文件/挂节点信息

\*

\* @param dtoUpdate

\* @throws DrawBusinessException

\* @author huangsongbo

\*/

@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)

public DrawBaseHouseControllerCallBackDTO updateAllData(DrawBaseHouseControllerCallBackDTO dtoUpdate) throws DrawBusinessException {

if (dtoUpdate == null) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("参数为空"));

}

Long taskDetailId = dtoUpdate.getTaskDetailId();

if (taskDetailId == null) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("烘焙任务ID为空"));

}

DrawBakeTaskDetail drawBakeTaskDetail = drawBakeTaskDetailService.get(taskDetailId);

if (drawBakeTaskDetail == null) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("不存在" + taskDetailId + "烘焙任务"));

}

Long drawSpaceCommonId = drawBakeTaskDetail.getSpaceId();

if (drawSpaceCommonId == null) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("烘焙的空间不存在taskDetailId=" + taskDetailId));

}

DrawDesignTempletDTO drawDesignTempletDTO = dtoUpdate.getDrawDesignTempletDTO();

if (drawDesignTempletDTO == null) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("参数drawDesignTempletDTO错误！"));

}

// 处理资源

DrawDesignTemplet drawDesignTemplet = this.updateResource(drawDesignTempletDTO, drawSpaceCommonId);

// 处理产品信息

this.updateProduct(drawDesignTempletDTO, drawDesignTemplet);

return dtoUpdate;

}

/\*\*

\* 处理资源

\*

\* @param drawDesignTempletDTO

\* @param drawSpaceCommonId

\* @return

\*/

@Override

public DrawDesignTemplet updateResource(DrawDesignTempletDTO drawDesignTempletDTO, Long drawSpaceCommonId) {

DrawSpaceCommon drawSpaceCommon = drawSpaceCommonService.getSpaceCommon(drawSpaceCommonId);

if (drawSpaceCommon == null) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("更新数据失败"));

}

// 处理空间灯光文件

Utils.requireGreaterZero(drawDesignTempletDTO.getDaylightPcU3dModelId(), ResponseEnum.SPACE\_DAYLIGHT\_EMPTY.getMessage());

Utils.requireGreaterZero(drawDesignTempletDTO.getDusklightPcU3dModelId(), ResponseEnum.SPACE\_DUSKLIGHT\_EMPTY.getMessage());

Utils.requireGreaterZero(drawDesignTempletDTO.getNightlightPcU3dModelId(), ResponseEnum.SPACE\_NIGHTLIGHT\_EMPTY.getMessage());

// 处理空间

drawSpaceCommon = new DrawSpaceCommon();

drawSpaceCommon.setId(drawSpaceCommonId);

// 白天

drawSpaceCommon.setDaylightU3dModelId(Integer.valueOf(drawDesignTempletDTO.getDaylightPcU3dModelId() + ""));

// 黄昏

drawSpaceCommon.setDusklightU3dModelId(Integer.valueOf(drawDesignTempletDTO.getDusklightPcU3dModelId() + ""));

// 晚上

drawSpaceCommon.setNightlightU3dModelId(Integer.valueOf(drawDesignTempletDTO.getNightlightPcU3dModelId() + ""));

drawSpaceCommonService.update(drawSpaceCommon);

// 更新样板房数模型、配置文件数据

List<DrawDesignTemplet> drawDesignTempletList = drawDesignTempletService.findAllBySpaceCommonId(drawSpaceCommonId);

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

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("户型绘制样板房信息没有找到: drawSpaceCommonId = " + drawSpaceCommonId));

}

Utils.requireGreaterZero(drawDesignTempletDTO.getDesignTempletPcModelU3dId(), ResponseEnum.DESIGN\_TEMPLET\_U3DMODLE\_EMPTY.getMessage());

Utils.requireGreaterZero(drawDesignTempletDTO.getDesignTempletConfigFileId(), ResponseEnum.DESIGN\_TEMPLET\_CONFIG\_EMPTY.getMessage());

// update draw design templet

DrawDesignTemplet drawDesignTemplet = new DrawDesignTemplet();

drawDesignTemplet.setId(drawDesignTempletList.get(0).getId());

drawDesignTemplet.setPcModelU3dId(Integer.valueOf(drawDesignTempletDTO.getDesignTempletPcModelU3dId() + ""));

drawDesignTemplet.setConfigFileId(Integer.valueOf(drawDesignTempletDTO.getDesignTempletConfigFileId() + ""));

// update

drawDesignTempletService.update(drawDesignTemplet);

return drawDesignTemplet;

}

/\*\*

\* 处理产品信息

\*

\* @param drawDesignTempletDTO

\* @param drawDesignTemplet

\*/

@Override

public void updateProduct(DrawDesignTempletDTO drawDesignTempletDTO, DrawDesignTemplet drawDesignTemplet) {

// 更新样板房产品表/基本产品表

DrawDesignTempletProductSearch drawDesignTempletProductSearch = new DrawDesignTempletProductSearch();

drawDesignTempletProductSearch.setDesignTempletId(Integer.valueOf(drawDesignTemplet.getId() + ""));

List<DrawDesignTempletProduct> drawDesignTempletProductList = drawDesignTempletProductService.findAllBySearch(drawDesignTempletProductSearch);

if (drawDesignTempletProductList == null || drawDesignTempletProductList.size() == 0) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("户型绘制样板房产品明细没有找到: drawDesignTempletId = " + drawDesignTemplet.getId()));

}

// 验证产品数据

Map<String, DrawDesignTempletProduct> drawDesignTempletProductMap = new HashMap<>();

for (DrawDesignTempletProduct drawDesignTempletProduct : drawDesignTempletProductList) {

if (StringUtils.isEmpty(drawDesignTempletProduct.getUniqueId())) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("draw样板房产品表,uniqueId为空: drawDesignTempletProductId = " + drawDesignTempletProduct.getId()));

}

drawDesignTempletProductMap.put(drawDesignTempletProduct.getUniqueId(), drawDesignTempletProduct);

}

// base product

List<DrawBaseProduct> drawBaseProducts = new ArrayList<>();

// draw design templet product

List<DrawDesignTempletProduct> drawDesignTempletProducts = new ArrayList<>();

// 产品

Map<Long, DrawBaseProduct> mapPorducts = this.mapDrawBaseProduct(new ArrayList<>(drawDesignTempletProductMap.values()));

for (DrawBaseProductDTO drawBaseProductDTO : drawDesignTempletDTO.getDrawBaseProductDTOList()) {

if (StringUtils.isEmpty(drawBaseProductDTO.getUniqueId())) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("jsonStr中,某个产品信息uniqueId为空"));

}

DrawDesignTempletProduct drawDesignTempletProduct = drawDesignTempletProductMap.get(drawBaseProductDTO.getUniqueId());

if (drawDesignTempletProduct == null) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("jsonStr中,某个产品信息uniqueId和数据库对应不上,uniqueId = " + drawBaseProductDTO.getUniqueId()));

}

// 挂节点信息更新

drawDesignTempletProduct.setPosIndexPath(drawBaseProductDTO.getPosIndexPath());

drawDesignTempletProduct.setPosName(drawBaseProductDTO.getPosName());

drawDesignTempletProduct.setProductSequence(Utils.replaceString(drawBaseProductDTO.getPosIndexPath(), "/"));

// 产品空间坐标，烘焙完成后获取

drawDesignTempletProduct.setProPosition(drawBaseProductDTO.getProPosition());

// add

drawDesignTempletProducts.add(drawDesignTempletProduct);

// 更新基本产品信息

// update by huangsongbo 软装不需要更新产品信息(因为软装在base\_product表中已经存在记录)

if (drawDesignTempletProduct.getCreateProductStatus() != null

&& DrawDesignTempletProductConstant.SOFT\_PRODUCT.intValue() == drawDesignTempletProduct.getCreateProductStatus().intValue()) {

log.debug("drawDesignTempletProduct.getCreateProductStatus() = {}", drawDesignTempletProduct.getCreateProductStatus());

} else {

DrawBaseProduct drawBaseProduct = mapPorducts.get(drawDesignTempletProduct.getProductId().longValue());

if (drawBaseProduct == null) {

throw new DrawBusinessException(false, ResponseEnum.CUSTOM\_RESPONSEENUM.setMessage("样板房对应的某个产品not found"));

}

// 硬装产品部分

drawBaseProduct.setWindowsU3dmodelId(Integer.valueOf(drawBaseProductDTO.getWindowsU3dmodelId() + ""));

// add

drawBaseProducts.add(drawBaseProduct);

}

}

// 批量更新挂节点、埋点信息

drawDesignTempletProductService.batchUpdateDesignTempletProduct(drawDesignTempletProducts);

// 批量更新windows u3d model 资源

drawBaseProductService.batchUpdateDrawBaseProduct(drawBaseProducts);

}

/\*\*

\* 获取空间的产品

\*

\* @param drawDesignTempletProducts

\* @return

\*/

Map<Long, DrawBaseProduct> mapDrawBaseProduct(List<DrawDesignTempletProduct> drawDesignTempletProducts) {

List<Long> ids = drawDesignTempletProducts.stream().filter(dtp -> dtp != null && dtp.getProductId() != null)

.map(dtp -> dtp.getProductId().longValue()).collect(Collectors.toList());

List<DrawBaseProduct> drawBaseProducts = drawBaseProductService.listDrawBaseProductByIds(ids);

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

return new HashMap<>();

}

Map<Long, DrawBaseProduct> map = drawBaseProducts.stream().collect(Collectors.toMap(DrawBaseProduct::getId, dp -> dp, (p, n) -> n));

return Optional.of(map).orElse(new HashMap<>());

}

@Override

public DrawBaseHouse get(Long id) {

if (id == null) {

return null;

}

return drawBaseHouseMapper.selectByPrimaryKey(id);

}

@Override

@Transactional(propagation = Propagation.REQUIRED)

public Map<Long, Long> transformToBaseHouse(DrawBaseHouse drawBaseHouse) {

Map<Long, Long> returnMap = new HashMap<>();

// 参数验证 ->start

if (drawBaseHouse == null) {

return returnMap;

}

// 参数验证 ->end

// 清空资源

drawBaseHouse.setPicRes1Id(null);

drawBaseHouse.setSnapPicId(null);

drawBaseHouse.setRestoreFileId(null);

drawBaseHouseMapper.transformToBaseHouse(drawBaseHouse);

// update base\_house\_id

this.update(drawBaseHouse);

returnMap.put(drawBaseHouse.getId(), drawBaseHouse.getBaseHouseId());

return returnMap;

}

@Override

@Transactional(propagation = Propagation.REQUIRED)

public void update(DrawBaseHouse drawBaseHouse) {

// 参数验证 ->start

if (drawBaseHouse == null) {

return;

}

// 参数验证 ->end

drawBaseHouseMapper.updateByPrimaryKeySelective(drawBaseHouse);

}

@Override

@Transactional(propagation = Propagation.REQUIRED)

public void uploadHouse(Long houseId) throws DrawBusinessException {

DrawBaseHouse drawHouse = new DrawBaseHouse();

drawHouse.setId(houseId);

drawHouse.setHouseStatus(DrawBaseHouseConstant.HOUSE\_WAIT\_PERFECT.toString());

int update = drawBaseHouseMapper.updateByPrimaryKeySelective(drawHouse);

if (update == 1) {

Map<String, Object> house = drawBaseHouseMapper.getBaseHouseInfo(houseId);

if (house == null) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

String baseHouseId = Objects.toString(house.get("baseHouseId"), "0");

if (baseHouseId != null && !baseHouseId.equals("0")) {

// 已被烘焙数据更改状态 ==> 烘焙中(base\_house => 4)

baseHouseMapper.updateHouseDealStatus(DrawBaseHouseConstant.DEAL\_STATUS\_FACSIMILE, Long.valueOf(baseHouseId));

}

// 邮件推送

this.handlerMailPush(house);

}

}

/\*\*

\* 邮件推送

\*

\* @param house

\*/

@Transactional(propagation = Propagation.NOT\_SUPPORTED)

void handlerMailPush(Map<String, Object> house) {

// 邮件通知完善员

UserLoginBO login = HouseAuthUtils.getUserLoginBO();

String userName = Objects.isNull(login.getName()) ? login.getLoginName() : login.getName() + "(" + login.getLoginName() + ")";

// 地址

String areaLongCode = Objects.toString(house.get("areaLongCode"), "0");

String[] split = areaLongCode.substring(1, areaLongCode.length() - 1).split("\\.");

String detailAddress = baseLivingService.getDetailAddress(split);

// 户型名

String houseName = Objects.toString(house.get("houseName"), "");

// 小区名

String livingName = Objects.toString(house.get("livingName"), "");

String args = detailAddress + livingName + "-" + houseName;

PerfectHouseMailMessage msg = new PerfectHouseMailMessage(new String[]{userName, args}, args);

templateMailService.push(msg);

}

@Override

@Transactional(propagation = Propagation.REQUIRED)

public DrawHouseAndAreaVO perfectHouse(Long houseId) {

final String type = "edit";

if (null == houseId) {

throw new BusinessException(false, ResponseEnum.PARAM\_ERROR);

}

// 完善户型员

if (!HouseAuthUtils.hasImproverPermission()) {

throw new BusinessException(false, ResponseEnum.FORBIDDEN);

}

// 户型完善

Long userId = HouseAuthUtils.getRequestUserId();

int lockPerfectHouse = drawBaseHouseMapper.lockPerfectHouse(houseId, userId,

DrawBaseHouseConstant.HOUSE\_PERFECT\_PROCESSING, DrawBaseHouseConstant.HOUSE\_WAIT\_PERFECT);

if (lockPerfectHouse != 1) {

log.warn("有人正在完善户型 {}(houseId)", houseId);

// 有人正在完善户型...

DrawBaseHouse house = drawBaseHouseMapper.getBaseHouseById(houseId);

if (house != null && !house.getPerfectUserId().equals(userId)) {

throw new BusinessException(false, ResponseEnum.HOUSE\_PERFECT\_PROCESSING\_ERROR);

}

}

return facsimileHouse(houseId, type);

}

/\*\*

\* 清空原先的户型数据

\*

\* @param houseId

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED)

public void handlerSubmitHouse(Long houseId) {

if (houseId == null) {

return;

}

DrawBaseHouse drawHouse = drawBaseHouseMapper.getBaseHouseById(houseId);

if (drawHouse == null) {

log.warn("绘制的户型{}(houseId)不存在或被删除", houseId);

return;

}

log.info("开始删除老数据绘制户型数据，{}(houseId)", houseId);

List<Long> emptySpaces = drawSpaceCommonService.getDealDrawSpaceCommon(DrawBaseHouseConstant.IS\_DELETE\_INIT, houseId);

if (emptySpaces != null && !emptySpaces.isEmpty()) {

log.info("需要删除绘制户型老数据, emptySpaces ==> {}", emptySpaces);

final int status = 1;

// 空间

drawSpaceCommonService.deleteDrawSpaceCommon(status, emptySpaces);

// 样板房

drawDesignTempletService.deleteDrawDesignTemplet(status, emptySpaces);

// 样板房硬装产品和软装产品

List<Long> emptyTemplets = drawDesignTempletService.getEmptyDrawDesignTemplet(emptySpaces);

drawDesignTempletProductService.deleteDrawDesignTempletProduct(status, emptyTemplets);

// 硬装产品

List<Long> baseProducts = drawBaseProductService.getDeleteDrawBaseProductId(emptySpaces);

if (baseProducts != null && !baseProducts.isEmpty()) {

drawBaseProductService.deleteDrawBaseProduct(status, baseProducts);

}

// 清除空间的旧数据资源

// drawResFileService.clearDrawResFileResource(emptySpaces, null);

}

log.info("删除提交的绘制户型数据完成，{}(houseId)", houseId);

}

/\*\*

\* 清空原先的正式表的数据户型数据

\*

\* @param houseId

\* @param taskId

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED)

public void handlerCallbackHouse(Long houseId, Long taskId) {

if (houseId == null || taskId == null) {

log.warn("参数 houseId 为空");

return;

}

// 是否第一个空间烘焙

int taskCount = drawBakeTaskService.getBakeSuccessTask(taskId);

if (taskCount != DrawBaseHouseConstant.EMPTY\_DATA\_INIT\_FLAG) {

log.debug("已清空过正式表数据，不需要清空数据");

return;

}

log.info("开始删除正式老数据户型数据，{}(houseId)，{}(taskId)", taskId);

DrawBaseHouse drawHouse = drawBaseHouseMapper.getBaseHouseById(houseId);

if (drawHouse != null && drawHouse.getBaseHouseId() != null && drawHouse.getBaseHouseId() != 0) {

log.info("需要删除的正式户型老数据, {}(baseHouseId)", drawHouse.getBaseHouseId());

List<Long> emptySpaces = drawSpaceCommonService.getEmptyDealSpaceCommon(drawHouse.getBaseHouseId());

if (emptySpaces != null && !emptySpaces.isEmpty()) {

log.info("需要删除正式户型老数据, emptySpaces ==> {}", emptySpaces);

final int status = 1;

// 空间

log.info("删除正式户型空间老数据信息");

drawSpaceCommonService.emptySpaceCommon(status, emptySpaces);

// 户型、空间关系

log.info("删除正式户型户型、空间关系老数据信息");

drawSpaceCommonService.emptyHouseSpaceRelation(status, emptySpaces);

// 这个查询要在删除design\_templet之前执行

List<Long> designTemplets = drawDesignTempletService.getEmptyDesignTemplet(emptySpaces);

// 样板房硬装产品和软装产品

if (designTemplets != null && !designTemplets.isEmpty()) {

log.info("删除正式样板房硬装产品和软装产品老数据信息");

drawDesignTempletProductService.emptyDesignTempletProduct(status, designTemplets);

}

// 样板房

log.info("删除正式户型样板房老数据信息");

drawDesignTempletService.emptyDesignTemplet(status, emptySpaces);

// 硬装产品

List<Long> baseProducts = drawBaseProductService.getEmptyBaseProduct(emptySpaces);

if (baseProducts != null && !baseProducts.isEmpty()) {

log.info("删除正式户型硬装产品老数据信息, baseProducts ==> {}", baseProducts);

drawBaseProductService.emptyBaseProduct(status, baseProducts);

}

// 清除assetbundle资源文件、draw\_res\_file表相关的assetbundle的文件记录

List<Long> emptyDrawSpaces = drawSpaceCommonService.getDealDrawSpaceCommon(DrawBaseHouseConstant.IS\_DELETED, houseId);

if (emptyDrawSpaces != null && !emptyDrawSpaces.isEmpty()) {

drawResFileService.clearDrawResFileResource(emptyDrawSpaces, 1);

}

}

}

log.info("删除正式表的老数据户型数据完成，{}(houseId)", houseId);

}

/\*\*

\* 处理烘焙任务失败

\*/

@Transactional(propagation = Propagation.REQUIRED)

void handlerBakeTaskFailed(DrawBaseHouseControllerCallBackDTO dto) {

// 烘焙失败, update status = 4

if (dto != null) {

log.warn("回滚处理失败的烘焙任务 {}(taskDetailId), message ==> {}", dto.getTaskDetailId(), dto.getMessage());

DrawBakeTaskDetail record = new DrawBakeTaskDetail();

record.setId(dto.getTaskDetailId());

// ip 记录

record.setMessage("Bake Client IP => " + HttpContextUtils.getIpAddress() + "\n" + dto.getMessage());

record.setGmtModified(new Date());

record.setStatus(DrawBaseHouseConstant.BAKE\_TASK\_STATUS\_FAIL);

record.setBakeFailCount(1);

drawBakeTaskDetailService.updateByPrimaryKeySelective(record);

}

}

/\*\*

\* 处理烘焙任务失败

\*/

@Override

@Transactional(propagation = Propagation.REQUIRED)

public Long handlerBakeTaskFailed(String data, String message) throws IOException {

DrawBaseHouseControllerCallBackDTO dto = null;

if (StringUtils.isNoneBlank(data)) {

ObjectMapper objectMapper = new ObjectMapper();

// 如果是空对象的时候,不抛异常

objectMapper.configure(SerializationFeature.FAIL\_ON\_EMPTY\_BEANS, false);

// 反序列化的时候如果多了其他属性,不抛出异常

objectMapper.configure(DeserializationFeature.FAIL\_ON\_UNKNOWN\_PROPERTIES, false);

dto = objectMapper.readValue(data, DrawBaseHouseControllerCallBackDTO.class);

}

if (dto == null) {

return -1L;

}

dto.setMessage(message);

this.handlerBakeTaskFailed(dto);

return dto.getTaskDetailId();

}

@Override

@Transactional(propagation = Propagation.REQUIRED)

public void transformHouse(String phone, Long houseId) {

// 外部全流程绘制(内部演示员)

if (!HouseAuthUtils.hasInternalDemoPermission()) {

throw new BusinessException(false, ResponseEnum.FORBIDDEN);

}

// 验证户型

DrawBaseHouse drawHouse = drawBaseHouseMapper.getBaseHouseById(houseId);

if (drawHouse == null) {

throw new BusinessException(false, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

// 不是自己的户型不能转换

if (!drawHouse.getUserId().equals(HouseAuthUtils.getRequestUserId())) {

throw new BusinessException(false, ResponseEnum.FORBIDDEN);

}

// 验证用户

Map<String, Object> user = drawBaseHouseMapper.getUserByPhone(phone);

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

throw new BusinessException(false, ResponseEnum.USER\_INFO\_IS\_NULL);

}

String userId = Objects.toString(user.get("userId"));

if (StringUtils.isEmpty(userId)) {

throw new BusinessException(false, ResponseEnum.USER\_INFO\_IS\_NULL);

}

// draw base house

DrawBaseHouse updateDrawHouse = new DrawBaseHouse();

updateDrawHouse.setId(houseId);

// 设置公开数据

updateDrawHouse.setIsPublic(DrawBaseHouseConstant.HOUSE\_IS\_PUBLIC);

updateDrawHouse.setUserId(Long.parseLong(userId));

drawBaseHouseMapper.updateByPrimaryKeySelective(updateDrawHouse);

// base house

if (drawHouse.getBaseHouseId() != null && drawHouse.getBaseHouseId() > 0) {

BaseHouse baseHouse = new BaseHouse();

baseHouse.setId(drawHouse.getBaseHouseId());

baseHouse.setUserId(Long.parseLong(userId));

baseHouseMapper.updateByPrimaryKeySelective(baseHouse);

}

}

@Override

public void resetDrawHouse(List<Integer> houseIds) {

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

throw new BusinessException(false, ResponseEnum.PARAM\_ERROR);

}

// draw house

drawBaseHouseMapper.resetDrawHouse(DrawBaseHouseConstant.HOUSE\_BAKE\_SUCCESS, houseIds);

// base house

baseHouseMapper.resetDrawHouse(Integer.valueOf(DrawBaseHouseConstant.DEAL\_STATUS\_EDIT), houseIds);

// task、task detail

drawBakeTaskService.resetDrawHouse(DrawBaseHouseConstant.IS\_DELETED, houseIds);

}

}

package com.sandu.api.house.service;

import java.io.IOException;

import java.util.List;

import java.util.Map;

import com.sandu.api.house.bo.DrawBaseHouseBO;

import com.sandu.api.house.bo.UserLoginBO;

import com.sandu.api.house.input.\*;

import com.sandu.api.house.model.BaseLiving;

import com.sandu.api.house.model.DrawBaseHouse;

import com.sandu.api.house.model.DrawDesignTemplet;

import com.sandu.api.house.output.DrawHouseAndAreaVO;

import com.sandu.api.v2.house.dto.DrawBaseHouseCallbackDTO;

import com.sandu.api.volume.room.model.VolumeRoomHouseBO;

import com.sandu.exception.BusinessException;

import com.sandu.exception.DrawBusinessException;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

/\*\*

\* Description:

\* 户型绘制接口类

\*

\* @author 何文

\* @version 1.0

\* Company:Sandu

\* Copyright:Copyright(c)2017

\* @date 2017/12/19

\*/

public interface DrawBaseHouseService {

/\*\*

\* 查询绘制户型列表

\*

\* @param drawBaseHouseSearch 户型查询对象

\* @return

\*/

Map<String, Object> listDrawHouse(DrawBaseHouseSearch drawBaseHouseSearch);

Map<Long, String> mapResource(List<DrawBaseHouseBO> drawHouses);

Map<Long, VolumeRoomHouseBO> mapVolumeRoomHouse(List<DrawBaseHouseBO> drawHouses);

/\*\*

\* 保存户型绘制

\*

\* @param data

\* @return

\* @throws BusinessException

\*/

Map<String, Object> saveDrawBaseHouse(String data, String userId);

Map<String, Object> saveDrawHouseV2(String data, String userId);

/\*\*

\* 删除户型绘制

\*

\* @param houseId

\* @param userId

\* @throws BusinessException

\*/

void deleteDrawBaseHouse(Long houseId, Long userId) throws BusinessException;

/\*\*

\* 审核户型绘制

\*

\* @param check

\* @throws BusinessException

\*/

void checkDrawBaseHouse(DrawBaseHouseCheck check) throws BusinessException;

/\*\*

\* 获取户型数据

\*

\* @param houseId

\* @return

\*/

DrawBaseHouse getBaseHouse(Long houseId);

/\*\*

\* 加载户型详情数据

\*

\* @param houseId 户型id

\* @param userId 用户id

\* @return

\* @throws BusinessException

\*/

DrawBaseHouseBO loadDrawBaseHouse(Long houseId, Long userId) throws BusinessException;

/\*\*

\* 户型绘制提交数据

\* 1.新建空间

\* 2.新建产品数据

\* 3.新建样板房

\* 4.新建样板房产品表

\*

\* @param jsonStr

\* @param loginBO 用户信息

\* @return

\* @throws DrawBusinessException

\* @author huangsongbo

\*/

DrawBaseHouseSubmitDTO submit(String jsonStr, UserLoginBO loginBO) throws DrawBusinessException;

void submitV2(String data) throws DrawBusinessException;

/\*\*

\* 普通绘制员上传户型

\* 1.新建空间

\* 2.新建产品数据

\* 3.新建样板房

\* 4.新建样板房产品表

\*

\* @param houseId

\* @throws DrawBusinessException

\* @author songjianming@sandusapace.com

\*/

void uploadHouse(Long houseId) throws DrawBusinessException;

/\*\*

\* 户型绘制获取小区下的户型，包括用户已绘制户型和未绘制户型

\*

\* @param query

\* @return

\*/

Map<String, Object> getHouses(DrawBaseHouseSearch query);

DrawHouseAndAreaVO facsimileHouse(Long houseId, String type);

/\*\*

\* 烘培回调

\* 主要任务:

\* 1.更新样板房挂节点等信息

\* 2.draw表数据转换到正式表数据

\*

\* @param jsonStr

\* @throws DrawBusinessException

\* @author huangsongbo

\*/

Map<String, Object> callback(String jsonStr) throws DrawBusinessException;

String callbackV2(DrawBaseHouseCallbackDTO dtoNew);

void handlerHouseStatus(DrawBaseHouse drawHouse, Long taskDetailId);

void transformHouseResource(DrawBaseHouse drawHouse, Long taskId);

DrawDesignTemplet updateResource(DrawDesignTempletDTO drawDesignTempletDTO, Long drawSpaceCommonId);

void updateProduct(DrawDesignTempletDTO drawDesignTempletDTO, DrawDesignTemplet drawDesignTemplet);

DrawBaseHouse get(Long drawBaseHouseId);

/\*\*

\* drawBaseHouse ->baseHouse

\*

\* @param drawBaseHouse

\* @return

\* @author huangsongbo

\*/

Map<Long, Long> transformToBaseHouse(DrawBaseHouse drawBaseHouse);

void update(DrawBaseHouse drawBaseHouse);

DrawHouseAndAreaVO perfectHouse(Long houseId);

void handlerSubmitHouse(Long houseId);

void handlerCallbackHouse(Long houseId, Long taskId);

Long handlerBakeTaskFailed(String data, String message) throws IOException;

void handlerCheck(DrawBaseHouseCheck check, DrawBaseHouse drawHouse);

void transformHouse(String phone, Long houseId);

void doSyncToIndex(Integer messageAction, List<Long> ids);

BaseLiving getBaseLiving(DrawBaseHouseNew data);

void resetDrawHouse(List<Integer> taskIds);

}

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.sandu.service.house.dao.BaseHouseMapper">

<select id="queryBaseHouse" resultType="com.sandu.api.house.bo.DrawBaseHouseBO">

SELECT

dh.id,

dh.house\_code,

dh.house\_name,

dh.gmt\_modified,

l.living\_name,

dh.area\_long\_code,

pic.pic\_path,

dh.is\_review

FROM

draw\_base\_house dh,

draw\_res\_house\_pic pic,

base\_living l

WHERE dh.pic\_res1\_id = pic.id

AND dh.living\_id = l.id

AND dh.is\_deleted = 0

AND dh.user\_id = #{userId}

AND dh.is\_review IN

<foreach collection="checkArgs" item="arg" open="(" separator="," close=")" >

#{arg}

</foreach>

AND dh.id IN

(SELECT

h.rel\_draw\_id

FROM

base\_house h

WHERE h.is\_deleted = 0

AND h.is\_public = 1

AND h.rel\_draw\_id != 0

ORDER BY h.gmt\_modified DESC)

<if test="keyword != null and keyword !=''">

AND l.living\_name like CONCAT('%', #{keyword}, '%')

<!-- AND (l.living\_name like CONCAT('%', #{keyword}, '%') OR dh.house\_name like concat('%', #{keyword},'%')) -->

</if>

LIMIT #{pageNum}, #{pageSize}

</select>

<select id="countBaseHouse" resultType="long">

SELECT

COUNT(\*)

FROM

draw\_base\_house dh,

draw\_res\_house\_pic pic,

base\_living l

WHERE dh.pic\_res1\_id = pic.id

AND dh.living\_id = l.id

AND dh.is\_deleted = 0

AND dh.user\_id = #{userId}

AND dh.is\_review IN

<foreach collection="checkArgs" item="arg" open="(" separator="," close=")" >

#{arg}

</foreach>

AND dh.id IN

(SELECT

h.rel\_draw\_id

FROM

base\_house h

WHERE h.is\_deleted = 0

AND h.is\_public = 1

AND h.rel\_draw\_id != 0

ORDER BY h.gmt\_modified DESC)

<if test="keyword != null and keyword !=''">

AND l.living\_name like CONCAT('%', #{keyword}, '%')

<!-- AND (l.living\_name like CONCAT('%', #{keyword}, '%') OR dh.house\_name like concat('%', #{keyword},'%')) -->

</if>

</select>

<update id="updateHouseDealStatus">

UPDATE

base\_house

SET

deal\_status = #{dealStatus}

WHERE id =

(SELECT

dh.base\_house\_id

FROM

draw\_base\_house dh

WHERE is\_deleted = 0

and dh.id = #{id})

</update>

<update id="updateHouseDealStatusForLock">

UPDATE

base\_house

SET

deal\_status = #{dealStatus}

<if test="userId != null"> , user\_id = #{userId} </if>

WHERE id = #{id}

</update>

<select id="selectByPrimaryKey" resultType="com.sandu.api.house.model.BaseHouse">

select \* from base\_house h where h.id = #{id} and h.is\_deleted = 0

</select>

<update id="updateHouseResouce">

UPDATE

base\_house h

<set>

<if test="picRes1Id != null"> h.pic\_res1\_id = #{picRes1Id}, </if>

<if test="snapPicId != null"> h.snap\_pic\_id = #{snapPicId}, </if>

<if test="restoreFileId != null"> h.restore\_file\_id = #{restoreFileId}, </if>

</set>

WHERE h.id = #{id}

</update>

<update id="restFailTask">

UPDATE

base\_house

SET

deal\_status = #{status}

WHERE deal\_status = 4

AND id IN

<foreach collection="args" item="item" open="(" separator="," close=")">

#{item}

</foreach>

</update>

<update id="updateByPrimaryKeySelective" parameterType="com.sandu.api.house.model.BaseHouse">

update base\_house

<set>

<if test="userId != null">

user\_id = #{userId},

</if>

</set>

where id = #{id,jdbcType=BIGINT}

</update>

<update id="fix2\_3Task">

UPDATE

base\_house h

SET

h.deal\_status = 4

WHERE h.id IN

(SELECT

dh.base\_house\_id

FROM

draw\_base\_house dh

WHERE dh.id IN

<foreach collection="failTasks" item="item" open="(" separator="," close=")">

#{item.houseId}

</foreach>)

</update>

<update id="resetDrawHouse">

UPDATE base\_house h SET h.deal\_status = #{dealStatus} WHERE h.id IN (

SELECT h.base\_house\_id FROM draw\_base\_house h WHERE h.id IN

<foreach collection="houseIds" item="item" open="(" separator="," close=")">

#{item}

</foreach>

)

</update>

<select id="countPutwayDesignTemplet" resultType="int">

SELECT

COUNT(\*)

FROM

house\_space\_new sn,

design\_templet dt

WHERE sn.space\_id = dt.space\_common\_id

AND sn.is\_deleted = 0

AND dt.is\_deleted = 0

AND dt.origin = 1

AND dt.putaway\_state = 3

AND sn.house\_id = #{houseId}

</select>

<select id="getPutawayState" resultType="int">

SELECT

dt.putaway\_state

FROM

house\_space\_new sn,

design\_templet dt

WHERE sn.space\_id = dt.space\_common\_id

AND sn.is\_deleted = 0

AND dt.is\_deleted = 0

AND dt.origin = 1

AND sn.house\_id = #{houseId}

LIMIT 1

</select>

</mapper>