package com.sandu;

import com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceAutoConfigure;

import lombok.extern.slf4j.Slf4j;

import org.mybatis.spring.boot.autoconfigure.MybatisAutoConfiguration;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.context.annotation.ImportResource;

import java.util.concurrent.CountDownLatch;

/\*\*

\* CopyRight (c) 2018 Sandu Technology Inc.

\*

\* @author songjianming@sanduspace.cn

\* @date 2018/09/06

\*/

@Slf4j

@ImportResource("classpath:/dubbo/spring-dubbo.xml")

@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class, DruidDataSourceAutoConfigure.class,

MybatisAutoConfiguration.class})

public class HouseDrawBakeService {

public static void main(String[] args) {

new SpringApplicationBuilder()

.sources(HouseDrawBakeService.class)

.web(false)

.run(args);

try {

new CountDownLatch(1).await();

} catch (InterruptedException e) {

log.error("{} 项目启动异常", HouseDrawBakeService.class, e);

}

}

}

package com.sandu.bake.api.task.model;

import lombok.Data;

import java.io.Serializable;

import java.util.Date;

/\*\*

\* CopyRight (c) 2018 Sandu Technology Inc.

\*

\* @author songjianming@sanduspace.cn

\* @date 2018/09/06

\*/

@Data

public class DrawBakeTask implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

/\*\*

\* 主键id

\*/

private Long id;

/\*\*

\* 用户id

\*/

private Long userId;

/\*\*

\* 房型id

\*/

private Long houseId;

/\*\*

\* 房型编码

\*/

private String houseCode;

private Integer isDeleted;

/\*\*

\* 烘焙状态 （0：未烘焙 1：成功 2：失败）

\*/

private Integer status;

/\*\*

\* 烘焙任务结果描述

\*/

private String message;

/\*\*

\* 创建人

\*/

private String creator;

/\*\*

\* 创建时间

\*/

private Date gmtCreate;

/\*\*

\* 修改人

\*/

private String modifier;

/\*\*

\* 修改时间

\*/

private Date gmtModified;

/\*\*

\* 队列名称:不同渲染机对应不同队列.</p>

\* internal : 内部 external : 外部

\*/

private String queueName;

/\*\*

\* 优先级:请参考java线程优先级,1为最小 5为一般 10为最大

\*/

private Integer priority;

}

package com.sandu.bake.api.task.model;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

import java.io.Serializable;

import java.util.Date;

@Getter

@Setter

@ToString

public class DrawBakeRecord implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private Long id;

private Long taskId;

private String ipAddress;

private Date gmtCreate;

private Integer isDeleted;

private String remark;

}

package com.sandu.bake.api.task.model;

import lombok.Data;

import java.io.Serializable;

import java.util.Date;

/\*\*

\* Description: 户型绘制烘焙任务详情表

\*

\* @author 何文

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

\* @date 2017/12/27

\*/

@Data

public class DrawBakeTaskDetail implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

/\*\*

\* 主键id

\*/

private Long id;

/\*\*

\* 任务id

\*/

private Long taskId;

/\*\*

\* 空间id

\*/

private Long spaceId;

/\*\*

\* 空间编码

\*/

private String spaceCode;

/\*\*

\* 产品信息

\*/

private String productData;

/\*\*

\* 空间文件id串

\*/

private String spaceFileIds;

/\*\*

\* 空间文件地址多文件逗号隔开

\*/

private String spaceFilePath;

/\*\*

\* 空间烘焙状态（0：未烘焙、1：烘焙成功、2：烘焙失败；3、烘焙中；4回退）<p>

\* draw\_bake\_task\_detail.status

\*/

private Integer status;

private String message;

private Date gmtModified;

/\*\*

\* 烘焙失败的次数

\*/

private Integer bakeFailCount;

private Integer isDeleted;

}

package com.sandu.general.api.product.model;

import lombok.Data;

import java.io.Serializable;

import java.math.BigDecimal;

import java.util.Date;

/\*\*

\* CopyRight (c) 2017 Sandu Technology Inc.<p>

\*

\* 产品日报表

\*

\* @author songjianming@sanduspace.cn <p>

\* 2017-12-22 09:55:31.981

\*/

@Data

public class BaseProduct implements Serializable {

/\*\*

\* base\_product.id

\*/

private Long id;

/\*\*

\* 产品编码<p>

\* base\_product.product\_code

\*/

private String productCode;

/\*\*

\* 产品名称<p>

\* base\_product.product\_name

\*/

private String productName;

/\*\*

\* 产品品牌<p>

\* base\_product.brand\_id

\*/

private Integer brandId;

/\*\*

\* 产品风格<p>

\* base\_product.pro\_style\_id

\*/

private Integer proStyleId;

/\*\*

\* 产品规格<p>

\* base\_product.product\_spec

\*/

private String productSpec;

/\*\*

\* 产品颜色<p>

\* base\_product.color\_id

\*/

private Integer colorId;

/\*\*

\* 产品长度<p>

\* base\_product.product\_length

\*/

private String productLength;

/\*\*

\* 产品宽度<p>

\* base\_product.product\_width

\*/

private String productWidth;

/\*\*

\* 产品高度<p>

\* base\_product.product\_height

\*/

private String productHeight;

/\*\*

\* 销售价格<p>

\* base\_product.sale\_price

\*/

private BigDecimal salePrice;

/\*\*

\* 建议售价价格<p>

\* base\_product.sale\_price\_advice

\*/

private BigDecimal salePriceAdvice;

/\*\*

\* 图片缩略图id<p>

\* base\_product.pic\_id

\*/

private Integer picId;

/\*\*

\* 模型id<p>

\* base\_product.model\_id

\*/

private Integer modelId;

/\*\*

\* 产品描述<p>

\* base\_product.product\_desc

\*/

private String productDesc;

/\*\*

\* 系统编码<p>

\* base\_product.sys\_code

\*/

private String sysCode;

/\*\*

\* 创建者<p>

\* base\_product.creator

\*/

private String creator;

/\*\*

\* 创建时间<p>

\* base\_product.gmt\_create

\*/

private Date gmtCreate;

/\*\*

\* 修改人<p>

\* base\_product.modifier

\*/

private String modifier;

/\*\*

\* 修改时间<p>

\* base\_product.gmt\_modified

\*/

private Date gmtModified;

/\*\*

\* 是否删除<p>

\* base\_product.is\_deleted

\*/

private Integer isDeleted;

/\*\*

\* PC端产品图片列表<p>

\* base\_product.pic\_ids

\*/

private String picIds;

/\*\*

\* 产品材质<p>

\* base\_product.material\_pic\_ids

\*/

private String materialPicIds;

/\*\*

\* 产品房间归属<p>

\* base\_product.house\_type\_values

\*/

private String houseTypeValues;

/\*\*

\* 产品类型<p>

\* base\_product.product\_type\_value

\*/

private String productTypeValue;

/\*\*

\* u3d模型<p>

\* base\_product.u3dmodel\_Id

\*/

private String u3dmodelId;

/\*\*

\* 同类型合并（合并产品ID）<p>

\* base\_product.merge\_product\_ids

\*/

private String mergeProductIds;

/\*\*

\* 上架操作修改时间<p>

\* base\_product.putaway\_modified

\*/

private Date putawayModified;

/\*\*

\* 时间备用2<p>

\* base\_product.date\_att2

\*/

private Date dateAtt2;

/\*\*

\* 产品类型（小类）<p>

\* base\_product.product\_small\_type\_value

\*/

private Integer productSmallTypeValue;

/\*\*

\* 上架状态<p>

\* base\_product.putaway\_state

\*/

private Integer putawayState;

/\*\*

\* 数字备用1<p>

\* base\_product.num\_att3

\*/

private BigDecimal numAtt3;

/\*\*

\* 数字备用2<p>

\* base\_product.num\_att4

\*/

private BigDecimal numAtt4;

/\*\*

\* 备注<p>

\* base\_product.remark

\*/

private String remark;

/\*\*

\* ipad使用u3d模型<p>

\* base\_product.ipad\_u3dmodel\_id

\*/

private Integer ipadU3dmodelId;

/\*\*

\* ios使用u3d模型<p>

\* base\_product.ios\_u3dmodel\_id

\*/

private Integer iosU3dmodelId;

/\*\*

\* android使用u3d模型<p>

\* base\_product.android\_u3dmodel\_id

\*/

private Integer androidU3dmodelId;

/\*\*

\* macBook使用u3d模型<p>

\* base\_product.macbook\_u3dmodel\_id

\*/

private Integer macbookU3dmodelId;

/\*\*

\* windows使用u3d模型<p>

\* base\_product.windows\_u3dmodel\_id

\*/

private Integer windowsU3dmodelId;

/\*\*

\* base\_product.web\_u3dmodel\_id

\*/

private Integer webU3dmodelId;

/\*\*

\* 饰品id<p>

\* base\_product.decoration\_id

\*/

private String decorationId;

/\*\*

\* 设计师id<p>

\* base\_product.designer\_id

\*/

private Integer designerId;

/\*\*

\* 建模师id<p>

\* base\_product.modeling\_id

\*/

private Integer modelingId;

/\*\*

\* 渲染师id<p>

\* base\_product.rendering\_id

\*/

private Integer renderingId;

/\*\*

\* 产品大分类标示<p>

\* base\_product.product\_type\_mark

\*/

private String productTypeMark;

/\*\*

\* 产品小分类标示<p>

\* base\_product.product\_small\_type\_mark

\*/

private String productSmallTypeMark;

/\*\*

\* 原始产品编码<p>

\* base\_product.original\_product\_code

\*/

private String originalProductCode;

/\*\*

\* 关联空间Id<p>

\* base\_product.spaceComon\_id

\*/

private Integer spacecomonId;

/\*\*

\* 产品短编码<p>

\* base\_product.product\_short\_code

\*/

private String productShortCode;

/\*\*

\* 硬装产品新增存的白模id<p>

\* base\_product.bm\_ids

\*/

private String bmIds;

/\*\*

\* 编码序号<p>

\* base\_product.code\_number

\*/

private Integer codeNumber;

/\*\*

\* 同类型产品(标识同一类)<p>

\* base\_product.parent\_id

\*/

private Integer parentId;

/\*\*

\* 关联样板房id<p>

\* base\_product.designTemplet\_id

\*/

private Integer designtempletId;

/\*\*

\* 材质球配置文件id(关联res\_file)<p>

\* base\_product.material\_file\_id

\*/

private Integer materialFileId;

/\*\*

\* 3dMax材质图片<p>

\* base\_product.material3d\_pic\_ids

\*/

private String material3dPicIds;

/\*\*

\* base\_product.sync\_status

\*/

private String syncStatus;

/\*\*

\* 该code使用状态 ， 1:已使用 2未使用<p>

\* base\_product.code\_is\_employ

\*/

private Integer codeIsEmploy;

/\*\*

\* 批次<p>

\* base\_product.code\_batch

\*/

private String codeBatch;

/\*\*

\* 产品型号<p>

\* base\_product.product\_model\_number

\*/

private String productModelNumber;

/\*\*

\* 背景墙全铺长度<p>

\* base\_product.full\_pave\_length

\*/

private String fullPaveLength;

/\*\*

\* 记录测试操作修改时间<p>

\* base\_product.test\_modified

\*/

private Date testModified;

/\*\*

\* 产品价格单位<p>

\* base\_product.sale\_price\_value

\*/

private Integer salePriceValue;

/\*\*

\* 最小高度<p>

\* base\_product.min\_height

\*/

private String minHeight;

/\*\*

\* 产品材质描述<p>

\* base\_product.product\_material\_desc

\*/

private String productMaterialDesc;

/\*\*

\* 产品色系编码<p>

\* base\_product.colors\_long\_code

\*/

private String colorsLongCode;

/\*\*

\* 产品配置文件<p>

\* base\_product.config\_id

\*/

private Integer configId;

/\*\*

\* FBX配置文件<p>

\* base\_product.fbx\_file\_id

\*/

private Integer fbxFileId;

/\*\*

\* FBX贴图<p>

\* base\_product.fbx\_texture

\*/

private String fbxTexture;

/\*\*

\* FBX处理状态 0未处理 1已处理<p>

\* base\_product.fbx\_state

\*/

private Integer fbxState;

/\*\*

\* 选中风格信息<p>

\* base\_product.product\_style\_id\_info

\*/

private String productStyleIdInfo;

/\*\*

\* 款式ID<p>

\* base\_product.style\_id

\*/

private Long styleId;

/\*\*

\* 区域标识<p>

\* base\_product.region\_mark

\*/

private String regionMark;

/\*\*

\* 尺寸代码<p>

\* base\_product.measure\_code

\*/

private String measureCode;

/\*\*

\* 墙体分类<p>

\* base\_product.wall\_type

\*/

private String wallType;

/\*\*

\* 天花布局标识<p>

\* base\_product.product\_smallpox\_identify

\*/

private Integer productSmallpoxIdentify;

/\*\*

\* 地面布局标识<p>

\* base\_product.product\_ground\_identify

\*/

private Integer productGroundIdentify;

/\*\*

\* 系列ID<p>

\* base\_product.series\_id

\*/

private Integer seriesId;

/\*\*

\* 拼花信息文件id <p>

\* base\_product.spelling\_flower\_file\_id

\*/

private Integer spellingFlowerFileId;

/\*\*

\* 移动端材质替换图片列表<p>

\* base\_product.app\_pic\_texture\_ids

\*/

private String appPicTextureIds;

/\*\*

\* 布局标识<p>

\* base\_product.product\_smallpox\_identify\_str

\*/

private String productSmallpoxIdentifyStr;

/\*\*

\* 是否是复杂拼花 0否1是<p>

\* base\_product.is\_complex\_parquet

\*/

private Integer isComplexParquet;

/\*\*

\* 拼花材质ids<p>

\* base\_product.parquet\_texture\_ids

\*/

private String parquetTextureIds;

/\*\*

\* 关联的draw\_base\_product=>id<p>

\* base\_product.rel\_draw\_id

\*/

private Long relDrawId;

/\*\*

\* 软装白膜产品是否拉伸过;1、是; null或2、否

\*/

private Integer isStretched;

/\*\*

\* 拆分材质信息<p>

\* base\_product.split\_textures\_info

\*/

private String splitTexturesInfo;

private Long originProductId;

private static final long serialVersionUID = 1L;

}

package com.sandu.general.api.product.model;

import lombok.Data;

import java.io.Serializable;

import java.util.Date;

/\*\*

\* CopyRight (c) 2017 Sandu Technology Inc.<p>

\*

\* 产品属性

\*

\* @author songjianming@sanduspace.cn <p>

\* 2018-04-25 14:21:51.127

\*/

@Data

public class ProductAttribute implements Serializable {

/\*\*

\* product\_attribute.id

\*/

private Long id;

/\*\*

\* 产品小分类（产品属性父级）<p>

\* product\_attribute.attribute\_type\_value

\*/

private String attributeTypeValue;

/\*\*

\* 属性值<p>

\* product\_attribute.attribute\_id

\*/

private Integer attributeId;

/\*\*

\* 属性key<p>

\* product\_attribute.attribute\_key

\*/

private String attributeKey;

/\*\*

\* 属性名称<p>

\* product\_attribute.attribute\_name

\*/

private String attributeName;

/\*\*

\* 属性值value<p>

\* product\_attribute.attribute\_value\_id

\*/

private Integer attributeValueId;

/\*\*

\* 属性值key<p>

\* product\_attribute.attribute\_value\_key

\*/

private String attributeValueKey;

/\*\*

\* 属性值名称<p>

\* product\_attribute.attribute\_value\_name

\*/

private String attributeValueName;

/\*\*

\* 属性值图片<p>

\* product\_attribute.attribute\_value\_pic\_ids

\*/

private String attributeValuePicIds;

/\*\*

\* 产品ID<p>

\* product\_attribute.product\_id

\*/

private Integer productId;

/\*\*

\* 系统编码<p>

\* product\_attribute.sys\_code

\*/

private String sysCode;

/\*\*

\* 创建者<p>

\* product\_attribute.creator

\*/

private String creator;

/\*\*

\* product\_attribute.gmt\_create

\*/

private Date gmtCreate;

/\*\*

\* 修改人<p>

\* product\_attribute.modifier

\*/

private String modifier;

/\*\*

\* 修改时间<p>

\* product\_attribute.gmt\_modified

\*/

private Date gmtModified;

/\*\*

\* 是否删除<p>

\* product\_attribute.is\_deleted

\*/

private Integer isDeleted;

/\*\*

\* product\_attribute.product\_code

\*/

private String productCode;

/\*\*

\* 字符备用2<p>

\* product\_attribute.att2

\*/

private String att2;

/\*\*

\* 整数备用1<p>

\* product\_attribute.numa1

\*/

private Integer numa1;

/\*\*

\* 整数备用2<p>

\* product\_attribute.numa2

\*/

private Integer numa2;

/\*\*

\* 备注<p>

\* product\_attribute.remark

\*/

private String remark;

private static final long serialVersionUID = 1L;

}

package com.sandu.general.api.product.model;

import lombok.Data;

import java.io.Serializable;

import java.util.Date;

/\*\*

\* CopyRight (c) 2017 Sandu Technology Inc.<p>

\*

\* 产品属性表

\*

\* @author songjianming@sanduspace.cn <p>

\* 2018-04-25 14:21:51.144

\*/

@Data

public class ProductProps implements Serializable {

/\*\*

\* product\_props.id

\*/

private Long id;

/\*\*

\* 属性CODE<p>

\* product\_props.code

\*/

private String code;

/\*\*

\* 属性长CODE<p>

\* product\_props.long\_code

\*/

private String longCode;

/\*\*

\* 属性名称<p>

\* product\_props.name

\*/

private String name;

/\*\*

\* 属性值<p>

\* product\_props.prop\_value

\*/

private String propValue;

/\*\*

\* 图片<p>

\* product\_props.pic\_path

\*/

private Integer picPath;

/\*\*

\* 父级ID<p>

\* product\_props.pid

\*/

private Integer pid;

/\*\*

\* 类型<p>

\* product\_props.type

\*/

private Integer type;

/\*\*

\* 等级<p>

\* product\_props.level

\*/

private Integer level;

/\*\*

\* 是否子级<p>

\* product\_props.is\_leaf

\*/

private Integer isLeaf;

/\*\*

\* 排序<p>

\* product\_props.ordering

\*/

private Integer ordering;

/\*\*

\* 系统编码<p>

\* product\_props.sys\_code

\*/

private String sysCode;

/\*\*

\* 创建者<p>

\* product\_props.creator

\*/

private String creator;

/\*\*

\* product\_props.gmt\_create

\*/

private Date gmtCreate;

/\*\*

\* 修改人<p>

\* product\_props.modifier

\*/

private String modifier;

/\*\*

\* 修改时间<p>

\* product\_props.gmt\_modified

\*/

private Date gmtModified;

/\*\*

\* 是否删除<p>

\* product\_props.is\_deleted

\*/

private Integer isDeleted;

/\*\*

\* 字符备用1<p>

\* product\_props.att1

\*/

private String att1;

/\*\*

\* 字符备用2<p>

\* product\_props.att2

\*/

private String att2;

/\*\*

\* 整数备用1<p>

\* product\_props.numa1

\*/

private Integer numa1;

/\*\*

\* 整数备用2<p>

\* product\_props.numa2

\*/

private Integer numa2;

/\*\*

\* 备注<p>

\* product\_props.remark

\*/

private String remark;

/\*\*

\* 用途：（排序、过滤）<p>

\* product\_props.filter\_order

\*/

private String filterOrder;

/\*\*

\* 属性序号<p>

\* product\_props.sequence\_number

\*/

private Integer sequenceNumber;

/\*\*

\* 属性长序号<p>

\* product\_props.long\_numbers

\*/

private String longNumbers;

private static final long serialVersionUID = 1L;

}

package com.sandu.general.api.product.bo;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonInclude;

import lombok.Data;

import java.io.Serializable;

/\*\*

\* CopyRight (c) 2017 Sandu Technology Inc.

\* <p>

\*

\* house-draw

\*

\* @author songjianming@sanduspace.cn

\* <p>

\* 2018年1月21日

\*/

@Data

public class BaseProductBO implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private Long id;

/\*\*

\* 产品名

\*/

private String productName;

/\*\*

\* 产品编码

\*/

private String productCode;

/\*\*

\* 大分类名称

\*/

private String productTypeName;

/\*\*

\* 大分类value

\*/

private Integer productTypeValue;

/\*\*

\* 大分类valuekey

\*/

private String productTypeCode;

/\*\*

\* 小分类名称

\*/

private String productSmallTypeName;

/\*\*

\* 小分类

\*/

private Integer productSmallTypeValue;

/\*\*

\* 小分类valuekey

\*/

private String productSmallTypeCode;

/\*\*

\* 图片ID

\*/

@JsonInclude(JsonInclude.Include.ALWAYS)

private Long picId;

@JsonInclude(JsonInclude.Include.ALWAYS)

private String picName;

/\*\*

\* 图片路径

\*/

@JsonInclude(JsonInclude.Include.ALWAYS)

private String picPath;

@JsonInclude(JsonInclude.Include.ALWAYS)

private Long modelId;

@JsonInclude(JsonInclude.Include.ALWAYS)

private String modelName;

/\*\*

\* 模型路径

\*/

@JsonInclude(JsonInclude.Include.ALWAYS)

private String modelPath;

/\*\*

\* 长

\*/

private Double productLength;

/\*\*

\* 宽

\*/

private Double productWidth;

/\*\*

\* 高

\*/

private Double productHeight;

@JsonIgnore

private Long groupId;

/\*\*

\* 是否主产品

\* 0、否

\* 1、是

\*/

private Integer isMainProduct;

}

package com.sandu.core.api.house.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonInclude;

import lombok.Data;

import java.io.Serializable;

import java.math.BigDecimal;

import java.util.Date;

/\*\*

\* CopyRight (c) 2018 Sandu Technology Inc.

\*

\* @author songjianming@sanduspace.cn

\* @date 2018/09/06

\*/

@Data

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;

private String livingName;

/\*\*

\* 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 Long companyId;

private Long fullHouseObjId;

private static final long serialVersionUID = 1L;

}

package com.sandu.core.api.house.dto;

import io.swagger.annotations.ApiModelProperty;

import lombok.Data;

import java.io.Serializable;

/\*\*

\* 户型绘制,保存数据DTO->产品数据(基本产品表/样板房产品表数据都在此) 备注问题:

\*

\* @author huangsongbo

\*/

@Data

public class DrawBaseProductDTO implements Serializable {

private static final long serialVersionUID = 4351281303161050399L;

// draw\_base\_Product产品表数据 ->start

@ApiModelProperty("前端生成的产品唯一值(烘培之后作为key更新产品)")

private String uniqueId;

@ApiModelProperty("产品大类valuekey")

private String bigTypeValueKey;

@ApiModelProperty("产品小类valuekey")

private String smallTypeValueKey;

@ApiModelProperty("产品长度/全铺长度")

private String productLength;

@ApiModelProperty("产品宽度")

private String productWidth;

@ApiModelProperty("产品高度")

private String productHeight;

@ApiModelProperty("产品还原/烘培文件id")

private Long productFileId;

@ApiModelProperty("产品模型文件id")

private Long windowsU3dmodelId;

@ApiModelProperty("产品id")

private Long productId;

@ApiModelProperty("产品类型:1:硬装;2:软装;3:public(窗户,栏杆等等)")

private Integer modelType;

/\*\*

\* 是否是软装,户型绘制的话,软装是系统中已经存在的产品白膜,不需要新建

\*/

private boolean isSoft = false;

// draw\_base\_Product产品表数据 ->end

// draw\_design\_templet\_product样板房产品表数据 ->start

// 放在烘焙之后给我 ->start

@ApiModelProperty("产品顺序")

private String productSequence;

@ApiModelProperty("挂节点")

private String posIndexPath;

@ApiModelProperty("挂节点")

private String posName;

// 放在烘焙之后给我 ->end

@ApiModelProperty("墙体方位")

private String wallOrientation;

@ApiModelProperty("墙体类型")

private String wallType;

@ApiModelProperty("样板房产品表id")

private Long designTempletProductId;

@ApiModelProperty("绑定的主墙产品的uniqueId;eg:如果该产品是背景墙,该字段记录的是其对应的主墙的uniqueId")

private String bindUniqueId;

// 组合产品ID

private Integer groupProductId;

// 是否主产品 0、否；1、是

private Integer isMainProduct;

// 是组合还是结构 0:组合,1:结构

private Integer groupType;

// 组合产品唯一标识，判断存在两个相同的组合产品的唯一标识

private String groupUniqueId;

/\*\*

\* 软装白膜产品是否拉伸过</p>

\* 1、是拉伸过; null或2、没有拉伸过

\*/

private Integer isChanged;

// ## 子母门1、单开门2、双开门3、推拉门5

private Integer proAttrType;

// 产品空间坐标，烘焙完成后获取

private String proPosition;

// draw\_design\_templet\_product样板房产品表数据 ->end

// 拉伸过的产品生成的新产品编码，u3d前端处理

// 规则：产品大分类\_小分类\_时间戳\_6位随机码

private String newProductCode;

}

package com.sandu.general.service.bake.impl;

import com.sandu.common.exception.dubbo.GeneralDubboException;

import com.sandu.common.monitor.Monitor;

import com.sandu.general.api.bake.dto.BakeSyncDTO;

import com.sandu.general.api.bake.service.BakeSyncCallback;

import com.sandu.general.api.bake.service.BasicBakeSyncService;

import com.sandu.general.api.bake.service.CleanupHouseHandler;

import com.sandu.general.api.bake.service.CleanupSyncResourceService;

import com.sandu.general.api.design.templet.service.DesignTemplateProductService;

import com.sandu.general.api.design.templet.service.DesignTemplateService;

import com.sandu.general.api.house.service.BaseHouseGuidePicInfoService;

import com.sandu.general.api.house.service.BaseHouseService;

import com.sandu.general.api.product.service.BaseProductService;

import com.sandu.general.api.space.service.SpaceCommonService;

import lombok.extern.slf4j.Slf4j;

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

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.Arrays;

/\*\*

\* CopyRight (c) 2018 Sandu Technology Inc.

\*

\* @author songjianming@sanduspace.cn

\* @date 2018/09/18

\*/

@Slf4j

@Service("defaultBasicBakeSyncService")

public class DefaultBasicBakeSyncServiceImpl implements BasicBakeSyncService {

@Autowired

BaseHouseService baseHouseService;

@Autowired

SpaceCommonService spaceCommonService;

@Autowired

DesignTemplateService designTemplateService;

@Autowired

DesignTemplateProductService designTemplateProductService;

@Autowired

BaseProductService baseProductService;

@Autowired

@Qualifier("defaultCleanupHouseHandler")

CleanupHouseHandler defaultCleanupHouseHandler;

@Autowired

@Qualifier("defaultBakeSyncCallback")

BakeSyncCallback defaultBakeSyncCallback;

@Autowired

CleanupSyncResourceService cleanupSyncResourceService;

@Autowired

BaseHouseGuidePicInfoService baseHouseGuidePicInfoService;

@Override

@Transactional

public BakeSyncDTO doSyncService(BakeSyncDTO dtoSync) {

return doSyncService(dtoSync, defaultCleanupHouseHandler, defaultBakeSyncCallback);

}

@Override

@Transactional

@Monitor(message = "烘焙回调：烘焙转化处理耗时：")

public BakeSyncDTO doSyncService(BakeSyncDTO dtoSync, CleanupHouseHandler handler, BakeSyncCallback... callbacks) {

try {

return doSyncAllData(dtoSync, handler, callbacks);

} catch (Exception ex) {

// 处理异常，把生成过的文件删除

cleanupSyncResourceService.cleanupSyncResource(dtoSync.getGcFiles());

throw ex;

}

}

/\*\*

\* 转化数据

\*

\* @param dtoSync

\* @param handler

\* @return

\*/

private BakeSyncDTO doSyncAllData(BakeSyncDTO dtoSync, CleanupHouseHandler handler, BakeSyncCallback... callbacks) {

if (dtoSync == null) {

log.error("Argument dtoSync cannot be empty.");

return null;

}

// cleanup

handler.doCleanup(dtoSync.isFirstSync(), dtoSync.getHouseSync().getDrawHouse());

// 转移户型数据(draw数据/文件->正式表/文件)

baseHouseService.doTransformData(dtoSync);

// 转移空间信息(draw数据/文件->正式表/文件)

spaceCommonService.doTransformData(dtoSync);

// 转移样板房信息(draw数据/文件->正式表/文件)

designTemplateService.doTransformData(dtoSync);

// 转移产品信息(draw数据/文件->正式表/文件)

baseProductService.doTransformData(dtoSync);

// 转移样板房产品信息(draw数据/文件->正式表/文件)

designTemplateProductService.doTransformData(dtoSync);

// 图片定位信息

baseHouseGuidePicInfoService.doTransformData(dtoSync);

// 回调函数，处理v720等

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

Arrays.stream(callbacks).forEach(call -> call.apply(dtoSync));

}

return dtoSync;

}

}

package com.sandu.core.service.house.impl;

import com.google.common.base.Strings;

import com.google.common.collect.Lists;

import com.sandu.bake.api.task.model.DrawBakeTask;

import com.sandu.common.constants.auth.Authorizes;

import com.sandu.common.constants.file.FileKey;

import com.sandu.common.constants.house.\*;

import com.sandu.common.constants.http.message.ResponseEnum;

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

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

import com.sandu.common.constants.mysql.DataType;

import com.sandu.common.constants.system.SystemConfigEnum;

import com.sandu.common.constants.system.SystemConstant;

import com.sandu.common.entity.file.DrawSpaceFileBO;

import com.sandu.common.entity.login.UserLoginBO;

import com.sandu.common.exception.BusinessException;

import com.sandu.common.exception.biz.BizException;

import com.sandu.common.monitor.Monitor;

import com.sandu.common.util.Utils;

import com.sandu.common.util.array.Arrays;

import com.sandu.common.util.file.FileUtils;

import com.sandu.common.util.http.ReturnData;

import com.sandu.common.util.json.JSON;

import com.sandu.common.util.number.Numbers;

import com.sandu.core.api.design.templet.service.DrawDesignTempletProductService;

import com.sandu.core.api.design.templet.service.DrawDesignTempletService;

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

import com.sandu.core.api.house.dto.DrawBaseHouseSubmitDTO;

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

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

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

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

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

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

import com.sandu.core.api.house.service.DrawFilter;

import com.sandu.core.api.house.vo.DrawHouseAreaVO;

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

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

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

import com.sandu.core.api.res.bo.DrawHouseResouceBO;

import com.sandu.core.api.res.input.DrawFileDataNew;

import com.sandu.core.api.res.model.DrawResFile;

import com.sandu.core.api.res.model.DrawResHousePic;

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

import com.sandu.core.api.space.service.DrawSpaceCommonService;

import com.sandu.core.api.task.service.GeneralBakeTaskService;

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

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

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

import com.sandu.core.service.res.dao.DrawResFileMapper;

import com.sandu.core.service.res.dao.DrawResHousePicMapper;

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

import com.sandu.general.api.area.bo.BaseAreaBO;

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

import com.sandu.general.api.area.input.BaseLivingNew;

import com.sandu.general.api.area.model.BaseLiving;

import com.sandu.general.api.area.service.BaseAreaService;

import com.sandu.general.api.area.service.BaseLivingService;

import com.sandu.general.api.house.bo.DrawHouseAreaBO;

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

import com.sandu.general.api.house.service.BaseHouseService;

import com.sandu.general.api.user.bo.DrawHouseUserBO;

import com.sandu.general.api.user.model.SysUser;

import com.sandu.general.api.user.service.SysUserService;

import lombok.extern.slf4j.Slf4j;

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.Transactional;

import java.util.\*;

import java.util.stream.Collectors;

/\*\*

\* CopyRight (c) 2018 Sandu Technology Inc.

\*

\* @author songjianming@sanduspace.cn

\* @date 2018/09/10

\*/

@Slf4j

@Service("drawBaseHouseService")

public class DrawBaseHouseServiceImpl implements DrawBaseHouseService {

@Autowired

DrawResFileMapper drawResFileMapper;

@Autowired

DrawBaseHouseMapper drawBaseHouseMapper;

@Autowired

DrawVolumeRoomMapper drawVolumeRoomMapper;

@Autowired

DrawResHousePicMapper drawResHousePicMapper;

@Autowired

BaseLivingService baseLivingService;

@Autowired

BaseAreaService baseAreaService;

@Autowired

DrawVolumeRoomService drawVolumeRoomService;

@Autowired

ResHandlerErrorService resHandlerErrorService;

@Autowired

DrawFilter drawSaveFilter;

@Autowired

DrawBaseHouseServiceV2 drawBaseHouseServiceV2;

@Autowired

BaseHouseService baseHouseService;

@Autowired

DrawSpaceCommonService drawSpaceCommonService;

@Autowired

DrawBaseProductService drawBaseProductService;

@Autowired

DrawDesignTempletService drawDesignTempletService;

@Autowired

DrawDesignTempletProductService drawDesignTempletProductService;

@Autowired

GeneralBakeTaskService generalBakeTaskService;

@Autowired

@Qualifier("perfectHouseTemplateMailService")

TemplateMailService perfectHouseTemplateMailService;

@Autowired

SysUserService sysUserService;

@Override

@Monitor(message = "绘制列表: 处理绘制列表耗时：")

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

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

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

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

query.setUserId(loginBO.getId());

// 检查权限

this.wrapCheckArgs(query, loginBO);

Long total = drawBaseHouseMapper.countDrawHouse(query);

if (total > 0) {

list = drawBaseHouseMapper.listDrawHouse(query);

}

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

// 图片资源

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

// 还原文件

// v1.1.4 解决因为draw\_res\_file数据太多，连表查询count()太耗时问题

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

// 户型的烘焙状态问题

Map<Long, DrawBakeTask> mapTaskStatus = this.mapTaskStatus(list);

// 量房资源

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

// 绘制户型的用户信息

Map<Long, DrawHouseUserBO> mapDrawHouseUser = this.mapDrawHouseUser(list);

// 处理我的绘制列表数据

this.handlerListDrawHouse(localAreaCache, list, mapPicResource, mapFileResource, mapDrawHouseUser, mapTaskStatus, mapVolumeRoom);

}

map.put("total", total);

map.put("list", list);

return map;

}

/\*\*

\* 处理我的绘制列表数据

\*

\* @param localAreaCache

\* @param list

\* @param mapPicResource

\* @param mapFileResource

\* @param mapTaskStatus

\* @param mapVolumeRoom

\*/

private void handlerListDrawHouse(Map<String, String> localAreaCache, List<DrawBaseHouseBO> list,

Map<Long, String> mapPicResource, Map<Long, String> mapFileResource,

Map<Long, DrawHouseUserBO> mapDrawHouseUser, Map<Long, DrawBakeTask> mapTaskStatus,

Map<Long, VolumeRoomHouseBO> mapVolumeRoom) {

StringBuilder buf = new StringBuilder();

for (DrawBaseHouseBO house : list) {

// 小区名

house.setLivingName(house.getLivingName() == null ? "" : house.getLivingName());

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(mapPicResource.get(house.getId()));

// 还原文件

house.setFilePath(mapFileResource.get(house.getRestoreFileId()));

// v1.1.4 失败次数 <= 3 显示为烘培失败，重新烘培中.

// http://wiki.3du-inc.net/pages/viewpage.action?pageId=18350724

if (Objects.equals(DrawBaseHouseConstant.HOUSE\_BAKE\_FAIL.toString(), house.getHouseStatus())

&& mapTaskStatus.get(house.getId()) != null) {

house.setHouseStatus(DrawBaseHouseConstant.FAILED\_CONTINUE\_BAKE\_PROCESSING.toString());

}

// 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());

}

// 用户信息

DrawHouseUserBO drawHouseUser = mapDrawHouseUser.get(house.getUserId());

if (drawHouseUser != null) {

house.setUserName(drawHouseUser.getUserName());

house.setUserMobile(drawHouseUser.getMobile());

}

}

}

/\*\*

\* 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, UserLoginBO loginBO) {

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() && Authorizes.hasApprovedPermission(loginBO)) {

throw new BusinessException(false, ResponseEnum.FORBIDDEN);

}

// 完善列表，完善户型权限

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

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));

query.setCheckArgs(NavItemType.getStatus(checkStatus));

}

/\*\*

\* 户型资源

\*

\* @param drawHouses

\* @return

\*/

@Override

public Map<Long, String> mapPicResource(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<>());

}

/\*\*

\* 获取还原文件

\*

\* @param drawHouses

\* @return

\*/

private Map<Long, String> mapFileResource(List<DrawBaseHouseBO> drawHouses) {

List<Long> files = drawHouses.stream().filter(f -> f.getRestoreFileId() != null)

.map(DrawBaseHouseBO::getRestoreFileId).collect(Collectors.toList());

if (files.isEmpty()) {

return new HashMap<>();

}

List<DrawSpaceFileBO> drawSpaceFileBOS = drawResFileMapper.listDrawResFileByFileIds(files);

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

return new HashMap<>();

}

return drawSpaceFileBOS.stream().collect(Collectors.toMap(DrawSpaceFileBO::getFileId, DrawSpaceFileBO::getFilePath, (p, n) -> n));

}

/\*\*

\* 户型的烘焙状态

\*

\* @param drawHouses

\* @return

\*/

private Map<Long, DrawBakeTask> mapTaskStatus(List<DrawBaseHouseBO> drawHouses) {

List<DrawBaseHouseBO> failedHouses = drawHouses.stream()

.filter(h -> Objects.equals(DrawBaseHouseConstant.HOUSE\_BAKE\_FAIL.toString(), h.getHouseStatus()))

.collect(Collectors.toList());

List<DrawBakeTask> tasks = generalBakeTaskService.listTaskStatusByHouseIds(failedHouses);

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

return new HashMap<>();

}

return tasks.stream().collect(Collectors.toMap(DrawBakeTask::getHouseId, t -> t, (p, n) -> n));

}

/\*\*

\* 绘制的用户信息

\*

\* @param drawHouses

\* @return

\*/

private Map<Long, DrawHouseUserBO> mapDrawHouseUser(List<DrawBaseHouseBO> drawHouses) {

List<DrawHouseUserBO> drawHouseUserBOS = sysUserService.listDrawHouseUser(drawHouses);

if (Arrays.isEmpty(drawHouseUserBOS)) {

return new HashMap<>();

}

return drawHouseUserBOS.stream().collect(Collectors.toMap(DrawHouseUserBO::getId, u -> u, (p, n) -> n));

}

/\*\*

\* 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<>());

}

@Override

@Transactional

@Monitor(message = "保存户型：处理保存户型耗时：")

public Map<String, Object> saveDrawHouse(String data, UserLoginBO loginBO) {

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

try {

return this.saveDrawBaseHouse(data, loginBO, fileRecovers);

} 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);

if (!fileRecovers.isEmpty()) {

fileRecovers.clear();

}

throw e;

} finally {

// 删除一次的旧资源文件数据

resHandlerErrorService.handlerErrorWithSave(fileRecovers.get("pics"), fileRecovers.get("files"));

}

}

@Override

@Transactional

public Map<String, Object> saveDrawBaseHouse(String data, UserLoginBO loginBO,

Map<String, List<Long>> fileRecovers) {

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

// 解析请求参数

DrawBaseHouseNew houseNew = getDrawBaseHouseNew(data, loginBO);

// 户型处理

DrawBaseHouse house = handlerDrawBaseHouse(houseNew, loginBO, fileRecovers);

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.insertSelective(pic);

house.setPicRes1Id(pic.getId() != null ? pic.getId().intValue() : null);

} 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.insertSelective(snapPic);

house.setSnapPicId(snapPic.getId());

} else {

// 还原文件

file = getDrawResFile(house, drawFileDataNew);

drawResFileMapper.insertSelective(file);

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(loginBO.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())

&& Authorizes.hasDrawPermission(loginBO)) {

house.setIsChangedUser(1);

house.setUserId(loginBO.getId());

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, UserLoginBO loginBO) {

if (!StringUtils.isNotBlank(dataJson)) {

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

}

DrawBaseHouseNew drawHouseNew = JSON.parse(dataJson, DrawBaseHouseNew.class);

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

drawHouseNew.setUserId(drawHouseNew.getUserId() == null ? loginBO.getId() : drawHouseNew.getUserId());

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

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

}

return drawHouseNew;

}

/\*\*

\* 处理户型保存

\*

\* @param drawHouseNew

\* @param loginBO

\* @param fileRecovers

\* @return

\* @throws BusinessException

\*/

private DrawBaseHouse handlerDrawBaseHouse(DrawBaseHouseNew drawHouseNew, UserLoginBO loginBO,

Map<String, List<Long>> fileRecovers) throws BusinessException {

// 过滤

DrawBaseHouse drawHouse = drawSaveFilter.filter(drawHouseNew, loginBO);

// 新增绘制、绘制通用PC户型老数据

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

// 小区

BaseLiving living = this.getBaseLiving(drawHouseNew);

// 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).replace(" ", ""));

drawHouse.setUserId(drawHouseNew.getUserId());

drawHouse.setHouseName(drawHouseNew.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.setLivingName(living == null ? "" : living.getLivingName());

drawHouse.setTotalArea(drawHouseNew.getTotalArea());

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

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

drawHouse.setGmtCreate(new Date());

drawHouse.setCreator(loginBO.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(getUserDataType(loginBO));

// 设置公司管理

SysUser user = sysUserService.getUser(loginBO.getId());

drawHouse.setCompanyId(user == null ? null : user.getCompanyId());

// 保存户型数据

drawBaseHouseMapper.insertSelective(drawHouse);

} else {

drawHouse = drawBaseHouseMapper.getBaseHouseById(drawHouseNew.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(drawHouseNew.getHouseName());

drawHouse.setTotalArea(drawHouseNew.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(drawHouseNew.getDrawType(), DrawType.DRAW\_AUDITOR\_EDIT)) {

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

}

// 删除资源处理

// this.deletedFile(drawHouse, drawHouseNew.getDatas());

// v1.1.4 物理删除，删除数据记录和资源数据

fileRecovers.put("pics", Lists.newArrayList(drawHouse.getSnapPicId()));

fileRecovers.put("files", Lists.newArrayList(drawHouse.getRestoreFileId()));

}

// 处理量房status

drawVolumeRoomService.updateVolumeRoomByLock(drawHouse, VolumeRoomConstant.PARSING\_SUCCESS.getStatus(),

VolumeRoomConstant.DEFAULTS.getStatus());

return drawHouse;

}

/\*\*

\* 获取保存的绘制数据类型

\*

\* @param loginBO

\* @return

\*/

public Integer getUserDataType(UserLoginBO loginBO) {

return Authorizes.hasDataPermission(loginBO) ?

DrawBaseHouseConstant.DATA\_PERSONAL :

DrawBaseHouseConstant.DATA\_PLATFORM;

}

/\*\*

\* 获取小区处理

\*

\* @param drawHouseNew

\* @return

\*/

private BaseLiving getBaseLiving(DrawBaseHouseNew drawHouseNew) {

BaseLivingNew livingNew = BaseLivingNew.builder().livingId(drawHouseNew.getLivingId())

.livingName(drawHouseNew.getLivingName())

.firstAreaCode(drawHouseNew.getFirstAreaCode())

.secondAreaCode(drawHouseNew.getSecondAreaCode())

.thirdAreaCode(drawHouseNew.getThirdAreaCode()).build();

return baseLivingService.getBaseLiving(livingNew);

}

private DrawResFile getDrawResFile(DrawBaseHouse house, DrawFileDataNew drawFileDataNew) {

DrawResFile drawResFile = new DrawResFile();

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

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

drawResFile.setFileName(drawFileDataNew.getFileName());

drawResFile.setFilePath(drawFileDataNew.getFilePath());

drawResFile.setFileSize(drawFileDataNew.getFileSize());

drawResFile.setFileSuffix(drawFileDataNew.getFileSuffix());

drawResFile.setFileType(drawFileDataNew.getFileType());

drawResFile.setFileCode(Utils.getSysCode(6));

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

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

drawResFile.setGmtCreate(new Date());

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

return drawResFile;

}

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(Integer.valueOf(Objects.toString(drawFileDataNew.getFileSize(), "0")));

pic.setPicPath(drawFileDataNew.getFilePath());

pic.setPicName(drawFileDataNew.getFileName());

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

@Monitor(message = "删除户型：处理删除户型耗时：")

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

ReturnData response = ReturnData.builder();

// 检测数据是否存在

DrawBaseHouse drawHouse = drawBaseHouseMapper.get(houseId);

if (null == drawHouse) {

log.warn("删除户型：{}(houseId) {}", houseId, ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

return response.status(false).code(ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

// 检测当前登陆者是否有权限操作

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

log.error("删除户型：{}(houseId) {}", houseId, ResponseEnum.HOUSE\_IS\_NOT\_BELONG\_USER);

return response.status(false).code(ResponseEnum.HOUSE\_IS\_NOT\_BELONG\_USER);

}

// 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.error("删除户型：{}(houseId) {}", houseId, ResponseEnum.DATA\_DELETED);

return response.status(false).code(ResponseEnum.DATA\_DELETED);

}

// 解锁绘制中状态

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

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

log.error("删除户型：{}(houseId) {}", houseId, ResponseEnum.DATA\_ERROR);

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

}

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

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

log.error("删除户型：{}(houseId) {}", houseId, ResponseEnum.DATA\_ERROR);

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

}

// 解锁户型绘制状态

log.info("删除户型：{}(baseHouseId)户型解锁成功", drawHouse.getBaseHouseId());

baseHouseService.updateHouseDealStatus(drawHouse.getBaseHouseId(), DrawBaseHouseConstant.DEAL\_STATUS\_FACSIMILE);

}

// 删除 draw\_base\_house

log.info("删除户型：{}(houseId)删除成功", houseId);

drawBaseHouseMapper.deleteById(houseId);

return response.status(true).code(ResponseEnum.SUCCESS);

}

/\*\*

\* 绘制户型审核

\*

\* @param drawHouseCheck

\* @param loginBO

\*/

@Override

@Transactional

@Monitor(message = "审核户型：处理审核户型耗时：")

public ReturnData drawHouseReview(DrawBaseHouseCheck drawHouseCheck, UserLoginBO loginBO) {

ReturnData response = ReturnData.builder();

// 检验户型数据是否存在

DrawBaseHouse drawHouse = drawBaseHouseMapper.getBaseHouseById(drawHouseCheck.getHouseId());

if (null == drawHouse) {

log.error("审核户型：{}(houseId) 进行审核处理", ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

return response.status(false).code(ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

return this.drawHouseReview(drawHouseCheck, drawHouse, loginBO);

}

@Override

@Transactional

@Monitor(message = "审核户型：处理审核户型耗时：")

public ReturnData drawHouseReview(DrawBaseHouseCheck drawHouseCheck, DrawBaseHouse drawHouse, UserLoginBO loginBO) {

ReturnData response = ReturnData.builder();

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

// 修改户型审核状态

DrawBaseHouse checkHouse = new DrawBaseHouse();

checkHouse.setAuditTime(new Date());

checkHouse.setAuditorId(loginBO.getId());

checkHouse.setId(drawHouseCheck.getHouseId());

checkHouse.setRejectReason(drawHouseCheck.getRejectReason());

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

// 更改户型审核状态

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

if (updateCount <= 0) {

log.error("户型审核：{}(houseId)户型审核失败", drawHouse.getId());

return response.status(false).code(ResponseEnum.CHECK\_FAIL);

}

log.info("户型审核：{}(houseId)户型审核, checkStatus => {}", drawHouse.getId(), drawHouseCheck.getCheckStatus());

// 判断前端需要处理的类型, 通过就生成任务

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

generalBakeTaskService.addBakeTask(drawHouse, loginBO);

// 更新户型状态

baseHouseService.updateHouseDealStatus(drawHouse.getBaseHouseId(), DrawBaseHouseConstant.DEAL\_STATUS\_BAKE);

log.info("户型审核：{}(houseId)户型审核通过", drawHouse.getId());

}

return response.status(true).code(ResponseEnum.SUCCESS);

}

@Override

@Transactional

@Monitor(message = "提交户型：处理提交户型耗时：")

public ReturnData submit(DrawBaseHouseSubmitDTO submitNew, UserLoginBO loginBO) {

ReturnData response = ReturnData.builder();

try {

DrawBaseHouse drawHouse = drawBaseHouseMapper.getBaseHouseById(Numbers.longValue(submitNew.getHouseId()));

if (drawHouse == null) {

return response.status(false).code(ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

// 绘制户型

submitNew.setDrawHouse(drawHouse);

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

// 清空旧数据处理

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

drawBaseHouseServiceV2.submit(submitNew, 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, submitNew.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(submitNew, loginBO);

// 提交户型时删除app\_online\_30.base\_house的响应的户型数据

this.handlerBaseHouse(drawHouse);

// 处理量房status

drawVolumeRoomService.updateVolumeRoomByLock(submitNew.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(submitNew, null);

throw e;

}

return response.status(true).code(ResponseEnum.SUCCESS);

}

/\*\*

\* 提交户型时删除app\_online\_30.base\_house的响应的户型数据

\*

\* @param drawHouse

\*/

private void handlerBaseHouse(DrawBaseHouse drawHouse) {

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

BaseHouse updateHouse = new BaseHouse();

updateHouse.setGmtModified(new Date());

updateHouse.setIsDeleted(DataType.IS\_DELETED);

updateHouse.setId(drawHouse.getBaseHouseId());

baseHouseService.updateHouse(updateHouse);

log.info("提交户型：删除户型{}(baseHouseId)", drawHouse.getBaseHouseId());

}

}

/\*\*

\* 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 loginBO) {

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

log.info("提交户型：{}(houseId)为内部员绘制的户型，不能自动审核通过", dtoNew.getHouseId());

return;

}

if (!Authorizes.hasJumpHouseCheckPermission(loginBO)) {

log.info("提交户型：{}(userId)权限不足, {}(houseId)无法自动审核通过", loginBO.getId(), dtoNew.getHouseId());

return;

}

if (dtoNew.getDrawHouse() == null) {

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

}

// 审核户型流程处理

DrawBaseHouseCheck drawCheck = new DrawBaseHouseCheck();

drawCheck.setUserId(loginBO.getId());

drawCheck.setHouseId(Numbers.longValue(dtoNew.getHouseId()));

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

this.drawHouseReview(drawCheck, dtoNew.getDrawHouse(), loginBO);

log.info("提交户型：{}(houseId)自动审核户型", dtoNew.getHouseId());

}

@Override

@Transactional

public void uploadHouse(Long houseId, UserLoginBO loginBO) {

DrawBaseHouse drawHouse = new DrawBaseHouse();

drawHouse.setId(houseId);

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

int update = drawBaseHouseMapper.updateByPrimaryKeySelective(drawHouse);

if (update == 1) {

DrawBaseHouse drawBaseHouse = drawBaseHouseMapper.selectByPrimaryKey(houseId);

if (drawBaseHouse == null) {

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

}

BaseLiving living = baseLivingService.getLiving(Numbers.longValue(drawBaseHouse.getLivingId()));

Long baseHouseId = drawBaseHouse.getBaseHouseId();

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

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

baseHouseService.updateHouseDealStatus(baseHouseId, DrawBaseHouseConstant.DEAL\_STATUS\_FACSIMILE);

}

this.handlerMailPush2(drawBaseHouse, living, loginBO);

}

}

/\*\*

\* 发送短信

\*

\* @param drawHouse

\* @param living

\* @param loginBO

\*/

private void handlerMailPush2(DrawBaseHouse drawHouse, BaseLiving living, UserLoginBO loginBO) {

try {

// 邮件通知完善员

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

// 地址

String areaLongCode = Objects.toString(living.getAreaLongCode(), "0");

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

String detailAddress = baseLivingService.getDetailAddress(split);

// 户型名

String houseName = Objects.toString(drawHouse.getHouseName(), "");

// 小区名

String livingName = Objects.toString(living.getLivingName(), "");

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

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

perfectHouseTemplateMailService.push(msg);

} catch (Exception e) {

log.error("上传户型：邮件推送异常", e);

}

}

@Override

public DrawBaseHouse get(Long drawBaseHouseId) {

return null;

}

@Override

@Transactional

@Monitor(message = "完善户型：处理完善户型耗时：")

public ReturnData perfectHouse(Long houseId, UserLoginBO loginBO) {

final String type = "edit";

ReturnData response = ReturnData.builder();

// 完善户型员

if (!Authorizes.hasImproverPermission(loginBO)) {

log.error("完善户型：{}", ResponseEnum.FORBIDDEN);

return response.status(true).code(ResponseEnum.FORBIDDEN);

}

// 户型完善

Long userId = loginBO.getId();

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 response.status(true).code(ResponseEnum.SUCCESS).data(getDrawHouse(houseId, type));

}

@Override

@Transactional

public void handlerSubmitHouse(Long houseId) {

if (houseId == null) {

return;

}

DrawBaseHouse drawHouse = drawBaseHouseMapper.getBaseHouseById(houseId);

if (drawHouse == null) {

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

return;

}

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

final int status = 1;

// 空间

drawSpaceCommonService.cleanupDrawSpaceCommon(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("提交户型：删除提交的绘制户型数据完成，{}(houseId)", houseId);

}

@Override

@Transactional

@Monitor(message = "转化户型：处理转化户型耗时：")

public ReturnData transformHouse(String phone, Long houseId, UserLoginBO loginBO) {

ReturnData response = ReturnData.builder();

// 外部全流程绘制(内部演示员)

if (!Authorizes.hasInternalDemoPermission(loginBO)) {

log.error("转换户型：{}", ResponseEnum.FORBIDDEN);

return response.status(false).code(ResponseEnum.FORBIDDEN);

}

// 验证户型

DrawBaseHouse drawHouse = drawBaseHouseMapper.getBaseHouseById(houseId);

if (drawHouse == null) {

log.error("转换户型：{}", ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

return response.status(false).code(ResponseEnum.HOUSE\_IS\_NOT\_FOUND);

}

// 不是自己的户型不能转换

if (!drawHouse.getUserId().equals(loginBO.getId())) {

log.error("转换户型：{}", ResponseEnum.FORBIDDEN);

return response.status(false).code(ResponseEnum.FORBIDDEN);

}

// 验证用户

SysUser sysUser = sysUserService.getSysUserByMobile(phone);

if (sysUser == null || sysUser.getId() == null) {

log.error("转换户型：{}", ResponseEnum.USER\_INFO\_IS\_NULL);

return response.status(false).code(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(sysUser.getId());

drawBaseHouseMapper.updateByPrimaryKeySelective(updateDrawHouse);

// base house

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

BaseHouse baseHouse = new BaseHouse();

baseHouse.setId(drawHouse.getBaseHouseId());

baseHouse.setUserId(sysUser.getId());

baseHouseService.updateHouse(baseHouse);

}

return response.status(true).code(ResponseEnum.SUCCESS);

}

@Override

public void resetDrawHouse(List<Integer> taskIds) {

}

@Override

public DrawBaseHouse getDrawHouse(Long houseId) {

if (houseId == null) {

log.warn("Arguments houseId cannot be empty.");

return null;

}

return drawBaseHouseMapper.selectByPrimaryKey(houseId);

}

@Override

@Monitor(message = "获取户型：处理获取户型耗时：")

public DrawHouseAreaVO getDrawHouse(Long houseId, String type) {

DrawBaseHouseBO drawHouseBO = new DrawBaseHouseBO();

if (SearchHouseType.CONTINUE\_DRAW.getType().equals(type)) {

drawHouseBO = drawBaseHouseMapper.getDrawHouse(houseId);

if (drawHouseBO != null) {

BaseLiving living = baseLivingService.getLiving(drawHouseBO.getLivingId());

if (living != null) {

drawHouseBO.setLivingId(living.getId());

drawHouseBO.setAreaId(living.getAreaId());

drawHouseBO.setLivingName(living.getLivingName());

}

// v1.1.0 量房处理数据，如果户型有量房数据，返回lf格式量房文件

if (!PlatformType.SANDU.getType().equals(drawHouseBO.getPlatformType())) {

List<VolumeRoomHouseBO> volumeRoom = drawVolumeRoomMapper.listVolumeRoomByHouseId(

Lists.newArrayList(drawHouseBO), VolumeRoomConstant.DEFAULTS.getStatus());

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

log.info("获取户型：{}检索到量房文件数据", houseId);

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

}

}

}

} else if (SearchHouseType.EDIT.getType().equals(type)

|| SearchHouseType.FACSIMILE.getType().equals(type)) {

// house-draw-general-service 的 dubbo 服务

DrawHouseAreaBO drawHouseAreaBO = baseHouseService.getDrawHouse(houseId);

try {

BeanUtils.copyProperties(drawHouseBO, drawHouseAreaBO);

} catch (Exception e) {

throw new BusinessException(false, ResponseEnum.ERROR, e);

}

// 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 (SearchHouseType.EDIT.getType().equals(type)) {

DrawHouseResouceBO houseFileResource = null;

if (!FileUtils.exists(drawHouseBO.getFilePath(), SystemConstant.UPLOAD\_ROOT\_FORMAL, SystemConstant.UPLOAD\_ROOT)) {

log.warn("获取户型：{} 没有还原文件，检索绘制数据还原文件", houseId);

houseFileResource = drawBaseHouseMapper.getHouseFileResource(houseId);

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

}

// v1.1.10 重置的户型，没有底图，取绘制表里的底图

if (!FileUtils.exists(drawHouseBO.getPicPath(), SystemConstant.UPLOAD\_ROOT\_FORMAL, SystemConstant.UPLOAD\_ROOT)) {

log.warn("获取户型：{} 没有还原文件，检索绘制数据还原文件", houseId);

if (houseFileResource == null) {

houseFileResource = drawBaseHouseMapper.getHouseFileResource(houseId);

drawHouseBO.setPicPath(houseFileResource == null ? null : Objects.toString(houseFileResource.getPicRes1Path(),

houseFileResource.getSnapPicPath()));

} else {

drawHouseBO.setPicPath(Objects.toString(houseFileResource.getPicRes1Path(), houseFileResource.getSnapPicPath()));

}

}

}

} else {

throw new BizException("获取户型：type参数错误");

}

DrawHouseAreaVO drawHouseVO = new DrawHouseAreaVO();

drawHouseVO.setDrawBaseHouseBO(drawHouseBO);

drawHouseVO.setBaseAreaBO(handlerArea(drawHouseBO));

return drawHouseVO;

}

@Override

public Integer updateHouse(DrawBaseHouse drawHouse) {

if (drawHouse == null) {

return null;

}

return drawBaseHouseMapper.updateByPrimaryKeySelective(drawHouse);

}

/\*\*

\* 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 handlerArea(DrawBaseHouseBO drawBaseHouseBO) {

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

return new BaseAreaBO();

}

BaseAreaLivingBO livingBO = baseLivingService.listAreaByLivingId(drawBaseHouseBO.getLivingId());

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;

}

}

package com.sandu.core.service.house.impl;

import com.sandu.bake.api.task.model.DrawBakeTaskDetail;

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

import com.sandu.common.constants.house.bugfix.FixType;

import com.sandu.common.constants.http.message.ResponseEnum;

import com.sandu.common.entity.login.UserLoginBO;

import com.sandu.common.exception.BusinessException;

import com.sandu.common.exception.DrawBusinessException;

import com.sandu.common.util.json.JSON;

import com.sandu.common.util.number.Numbers;

import com.sandu.common.util.regex.Regexs;

import com.sandu.core.api.design.templet.model.DrawDesignTemplet;

import com.sandu.core.api.design.templet.service.DrawDesignTempletProductService;

import com.sandu.core.api.design.templet.service.DrawDesignTempletService;

import com.sandu.core.api.fix.cupboard.bo.FixCupboardSubmitBO;

import com.sandu.core.api.house.bo.DrawHouseSubmitBO;

import com.sandu.core.api.house.bo.DrawSpaceBO;

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

import com.sandu.core.api.house.dto.DrawBaseHouseSubmitDTO;

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

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

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

import com.sandu.core.api.res.service.DrawResHousePicService;

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

import com.sandu.core.api.space.dto.DrawSpaceCommonDTO;

import com.sandu.core.api.space.model.DrawSpaceCommon;

import com.sandu.core.api.space.service.DrawSpaceCommonService;

import com.sandu.core.api.task.service.GeneralBakeTaskService;

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

import lombok.extern.slf4j.Slf4j;

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.Transactional;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Date;

import java.util.List;

@Slf4j

@Service("drawBaseHouseServiceV2")

public class DrawBaseHouseServiceV2Impl implements DrawBaseHouseServiceV2 {

@Autowired

DrawFlowGeneralService drawSpaceCommonServiceV2;

@Autowired

DrawFlowGeneralService drawDesignTempletServiceV2;

@Autowired

DrawFlowGeneralService drawBaseProductServiceV2;

@Autowired

DrawFlowGeneralService drawDesignTempletProductServiceV2;

@Autowired

DrawBaseHouseMapper drawBaseHouseMapper;

@Autowired

DrawBaseHouseService drawBaseHouseService;

@Autowired

DrawSpaceCommonService drawSpaceCommonService;

@Autowired

DrawDesignTempletService drawDesignTempletService;

@Autowired

DrawBaseProductService drawBaseProductService;

@Autowired

DrawDesignTempletProductService drawDesignTempletProductService;

@Autowired

ResHandlerErrorService resHandlerErrorService;

@Autowired

DrawResHousePicService drawResHousePicService;

@Autowired

GeneralBakeTaskService generalBakeTaskService;

@Autowired

@Qualifier("defaultSubmitHouseCallback")

SubmitHouseCallback defaultSubmitHouseCallback;

@Override

@Transactional

public void submit(DrawBaseHouseSubmitDTO dtoNew, UserLoginBO loginBO) {

this.submit(dtoNew, loginBO, defaultSubmitHouseCallback);

}

@Override

@Transactional

public void submit(DrawBaseHouseSubmitDTO dtoNew, UserLoginBO loginBO, SubmitHouseCallback... callbacks) {

// handler draw space

List<DrawSpaceBO> drawSpaceBOS = this.getDrawSpaceBOS(dtoNew, loginBO);

this.handlerSave(drawSpaceBOS, dtoNew);

// 回调函数处理

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

Arrays.stream(callbacks).forEach(call -> call.apply(dtoNew, loginBO, drawSpaceBOS));

}

}

@Override

@Transactional

public List<DrawSpaceBO> submitByFixCupboard(UserLoginBO loginBO,

FixCupboardSubmitBO fixCupboard,

DrawSpaceSubmitFilter filter) {

DrawBaseHouseSubmitDTO newSubmit = new DrawBaseHouseSubmitDTO();

newSubmit.setHouseId(fixCupboard.getHouseId().intValue());

newSubmit.setSpaceCommonDTOList(fixCupboard.getSpaceCommonDTOList());

// handler draw space

List<DrawSpaceBO> drawSpaceBOS = this.getDrawSpaceBOS(newSubmit, loginBO, fixCupboard.getDrawHouse(), filter);

// 标记修复类型，1、修复橱柜

for (DrawSpaceBO drawSpaceBO : drawSpaceBOS) {

DrawSpaceCommon drawSpaceCommon = drawSpaceBO.getDrawSpaceCommon();

if (drawSpaceCommon != null) {

drawSpaceCommon.setFixType(FixType.FIX\_CUPBOARD.getStatus());

}

DrawDesignTemplet drawDesignTemplet = drawSpaceBO.getDrawDesignTemplet();

if (drawDesignTemplet != null) {

drawDesignTemplet.setFixType(FixType.FIX\_CUPBOARD.getStatus());

}

}

// save

this.handlerSave(drawSpaceBOS);

return drawSpaceBOS;

}

/\*\*

\* 处理提交的空间、样板房、产品

\*

\* @param dtoNew

\* @param loginBO

\* @return

\*/

private List<DrawSpaceBO> getDrawSpaceBOS(DrawBaseHouseSubmitDTO dtoNew, UserLoginBO loginBO) {

return this.getDrawSpaceBOS(dtoNew, loginBO, dtoNew.getDrawHouse(), null);

}

/\*\*

\* 处理提交的空间、样板房、产品

\*

\* @param dtoNew

\* @param loginBO

\* @param drawHouse

\* @param drawSpaceFilter

\* @return

\*/

private List<DrawSpaceBO> getDrawSpaceBOS(DrawBaseHouseSubmitDTO dtoNew,

UserLoginBO loginBO, DrawBaseHouse drawHouse,

DrawSpaceSubmitFilter drawSpaceFilter) {

DrawHouseSubmitBO houseSubmitBO = new DrawHouseSubmitBO();

houseSubmitBO.setLoginBO(loginBO);

houseSubmitBO.setDrawHouse(drawHouse);

List<DrawSpaceBO> drawSpaceBOS = new ArrayList<>();

for (DrawSpaceCommonDTO drawSpace : dtoNew.getSpaceCommonDTOList()) {

DrawSpaceBO drawSpaceBO = new DrawSpaceBO();

drawSpaceBO.setDrawSpaceCommonDTO(drawSpace);

houseSubmitBO.setDrawSpaceBO(drawSpaceBO);

// 处理空间

DrawSpaceCommon drawSpaceCommon = drawSpaceCommonServiceV2.handlerHouseSubmit(houseSubmitBO);

drawSpaceBO.setDrawSpaceCommon(drawSpaceCommon);

// filter space

if (drawSpaceFilter != null && drawSpaceFilter.test(drawSpaceCommon)) {

continue;

}

// 处理样板房

DrawDesignTemplet drawDesignTemplet = drawDesignTempletServiceV2.handlerHouseSubmit(houseSubmitBO);

drawSpaceBO.setDrawDesignTemplet(drawDesignTemplet);

// 处理 base product

// 处理 design templet product

drawBaseProductServiceV2.handlerHouseSubmit(houseSubmitBO);

// add draw space bo

drawSpaceBOS.add(drawSpaceBO);

}

return drawSpaceBOS;

}

/\*\*

\* 保存处理

\*

\* @param drawSpaceBOS

\* @param dtoNew

\* @return

\*/

private Integer handlerSave(List<DrawSpaceBO> drawSpaceBOS, DrawBaseHouseSubmitDTO dtoNew) {

// 更新全屋文件

if (dtoNew.getFullHouseObjId() != null && dtoNew.getFullHouseObjId() > 0) {

DrawBaseHouse drawHouse = new DrawBaseHouse();

drawHouse.setId(dtoNew.getHouseId().longValue());

drawHouse.setFullHouseObjId(dtoNew.getFullHouseObjId());

drawBaseHouseService.updateHouse(drawHouse);

}

int updateCount = this.handlerSave(drawSpaceBOS);

// 户型的总面积处理

updateCount += handlerHouseArea(drawSpaceBOS, Numbers.longValue(dtoNew.getHouseId()));

return updateCount;

}

/\*\*

\* 修复橱柜save by fix cupboard submit

\*

\* @param drawSpaceBOS

\* @return

\*/

private Integer handlerSave(List<DrawSpaceBO> drawSpaceBOS) {

// 空间

int updateCount = drawSpaceCommonServiceV2.batchSaveByHouseSubmit(drawSpaceBOS);

// 样板房

updateCount += drawDesignTempletServiceV2.batchSaveByHouseSubmit(drawSpaceBOS);

// 产品

updateCount += drawBaseProductServiceV2.batchSaveByHouseSubmit(drawSpaceBOS);

// 样板房产品

updateCount += drawDesignTempletProductServiceV2.batchSaveByHouseSubmit(drawSpaceBOS);

return updateCount;

}

/\*\*

\* 户型的总面积处理

\*

\* @param drawSpaceBOS

\* @param houseId

\* @return

\*/

private int handlerHouseArea(List<DrawSpaceBO> drawSpaceBOS, Long houseId) {

// 户型的总面积处理

BigDecimal totalArea = new BigDecimal(0);

for (DrawSpaceBO drawSpaceBO : drawSpaceBOS) {

// 累计空间面积

totalArea = this.getSpaceTotalArea(totalArea, drawSpaceBO.getDrawSpaceCommonDTO().getSpaceCommonArea());

}

// 更新户型的总面积

DrawBaseHouse drawHouse = new DrawBaseHouse();

drawHouse.setTotalArea(Math.round(totalArea.doubleValue()) + "");

drawHouse.setId(houseId);

return drawBaseHouseMapper.updateDrawBaseHouse(drawHouse);

}

/\*\*

\* 累计空间面积

\*

\* @param totalArea

\* @param spaceTotalArea

\* @return

\*/

private BigDecimal getSpaceTotalArea(BigDecimal totalArea, String spaceTotalArea) {

return (StringUtils.isNoneBlank(spaceTotalArea) && spaceTotalArea.matches(Regexs.DOUBLE\_NUMBER.getValue()))

? totalArea.add(new BigDecimal(spaceTotalArea)) : totalArea;

}

@Override

@Transactional

public void handlerCallbackError(DrawBaseHouseCallbackDTO dtoNew, String data, String ipAddress) {

if (dtoNew == null) {

if (StringUtils.isEmpty(data)) {

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

}

dtoNew = JSON.parse(data, DrawBaseHouseCallbackDTO.class);

if (dtoNew == null) {

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

}

}

DrawBakeTaskDetail record = new DrawBakeTaskDetail();

record.setBakeFailCount(1);

record.setGmtModified(new Date());

record.setId(dtoNew.getTaskDetailId());

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

record.setMessage("Bake Client IP => " + ipAddress + "\n" + dtoNew.getMessage());

log.warn("烘焙任务失败处理 {}(taskDetailId), {}(bakeIpAddress), message ==> {}", dtoNew.getTaskDetailId(), ipAddress, dtoNew.getMessage());

// add failed record

generalBakeTaskService.updateBakeTaskDetail(record);

// 更改户型状态为烘焙失败

drawBaseHouseMapper.updateHouseStatus(DrawBaseHouseConstant.HOUSE\_BAKE\_FAIL, dtoNew.getTaskDetailId());

}

}

package com.sandu.core.service.design.templet.impl;

import com.sandu.common.constants.house.design.DrawDesignTempletProductConstant;

import com.sandu.common.constants.house.product.DrawBaseProductConstant;

import com.sandu.common.constants.house.space.RegionMarkConstant;

import com.sandu.common.constants.http.message.ResponseEnum;

import com.sandu.common.entity.login.UserLoginBO;

import com.sandu.common.exception.BusinessException;

import com.sandu.common.exception.DrawBusinessException;

import com.sandu.common.exception.biz.BizException;

import com.sandu.common.util.Utils;

import com.sandu.core.api.design.templet.model.DrawDesignTempletProduct;

import com.sandu.core.api.house.bo.DrawHouseProductBO;

import com.sandu.core.api.house.bo.DrawHouseSubmitBO;

import com.sandu.core.api.house.bo.DrawSpaceBO;

import com.sandu.core.api.house.bo.RegionMark;

import com.sandu.core.api.house.dto.DrawBaseProductDTO;

import com.sandu.core.api.house.service.DrawFlowGeneralService;

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

import com.sandu.core.service.design.templet.dao.DrawDesignTempletProductMapper;

import com.sandu.general.api.product.service.BaseProductService;

import lombok.extern.slf4j.Slf4j;

import org.apache.commons.lang3.StringUtils;

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

import org.springframework.stereotype.Service;

import java.util.\*;

@Slf4j

@Service("drawDesignTempletProductServiceV2")

public class DrawDesignTempletProductServiceV2Impl implements DrawFlowGeneralService {

@Autowired

BaseProductService baseProductService;

@Autowired

DrawBaseProductService drawBaseProductService;

@Autowired

DrawDesignTempletProductMapper drawDesignTempletProductMapper;

@Override

public <T> T handlerHouseSubmit(DrawHouseSubmitBO houseSubmitBO) {

return null;

}

@Override

public Integer batchSaveByHouseSubmit(List<DrawSpaceBO> drawSpaceBOS) {

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

log.warn("参数 drawSpaceBOS 为空");

return -1;

}

int updateCount = 0;

// 拼装绑定点信息

Map<String, List<Long>> bingInfoMap = this.getBingInfoMap(drawSpaceBOS);

for (DrawSpaceBO drawSpaceBO : drawSpaceBOS) {

if (drawSpaceBO.getDrawHouseProductBOS() == null || drawSpaceBO.getDrawHouseProductBOS().isEmpty()) {

continue;

}

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

for (DrawHouseProductBO drawHouseProductBO : drawSpaceBO.getDrawHouseProductBOS()) {

if (drawHouseProductBO.getDrawDesignTempletProduct() != null) {

drawDesignTempletProducts.add(this.getDrawDesignTempletProduct(drawSpaceBO, bingInfoMap, drawHouseProductBO));

}

}

if (drawDesignTempletProducts.isEmpty()) {

continue;

}

updateCount += drawDesignTempletProductMapper.batchSaveDesignTempletProduct(drawDesignTempletProducts);

}

return updateCount;

}

/\*\*

\* @param drawSpaceBO

\* @param drawHouseProductBO

\* @return

\*/

private DrawDesignTempletProduct getDrawDesignTempletProduct(DrawSpaceBO drawSpaceBO,

Map<String, List<Long>> bingInfoMap,

DrawHouseProductBO drawHouseProductBO) {

if (drawHouseProductBO.getDrawBaseProduct() != null) {

if (drawHouseProductBO.getDrawBaseProduct().getId() == null) {

throw new BizException("保存产品信息失败，请重试！");

}

// product id

if (drawHouseProductBO.getDrawDesignTempletProduct().getProductId() == null

|| drawHouseProductBO.getDrawDesignTempletProduct().getProductId() <= 0) {

drawHouseProductBO.getDrawDesignTempletProduct().setProductId(drawHouseProductBO.getDrawBaseProduct().getId().intValue());

}

// 绑定产品处理

String bindInfoMapStr = Utils.listToStr(bingInfoMap.get(drawHouseProductBO.getDrawBaseProductDTO().getUniqueId()));

drawHouseProductBO.getDrawDesignTempletProduct().setBindParentProductid(bindInfoMapStr);

}

// design templet id

if (drawSpaceBO.getDrawDesignTemplet().getId() == null) {

throw new BizException("保存样板房信息失败，请重试！");

}

drawHouseProductBO.getDrawDesignTempletProduct().setDesignTempletId(drawSpaceBO.getDrawDesignTemplet().getId().intValue());

return drawHouseProductBO.getDrawDesignTempletProduct();

}

/\*\*

\* 样板房产品处理

\*

\* @param drawBaseProductDTO

\* @param houseSubmitBO

\* @return

\*/

public DrawDesignTempletProduct handlerHouseSubmit(DrawBaseProductDTO drawBaseProductDTO,

DrawHouseSubmitBO houseSubmitBO) {

// public文件不保存在样板房产品表

if (DrawBaseProductConstant.MODEL\_TYPE\_PUBLIC.intValue() == drawBaseProductDTO.getModelType().intValue()) {

log.debug("跳过不处理的公共部分{}(productId)", drawBaseProductDTO.getUniqueId());

return null;

}

// getDrawDesignTempletProduct

DrawDesignTempletProduct drawDesignTempletProduct = this.getDrawDesignTempletProduct(drawBaseProductDTO, houseSubmitBO.getLoginBO());

// 组合产品处理

this.handlerGroupProduct(drawDesignTempletProduct, drawBaseProductDTO, houseSubmitBO);

// 地面、天花区域标识处理

this.handlerRegionMark(drawDesignTempletProduct, drawBaseProductDTO, houseSubmitBO);

// 处理软装产品

return this.handlerSoftProduct(drawDesignTempletProduct, drawBaseProductDTO);

}

/\*\*

\* 处理软装白膜产品

\*

\* @param dtpProduct

\* @param drawProduct

\* @return

\*/

private DrawDesignTempletProduct handlerSoftProduct(DrawDesignTempletProduct dtpProduct, DrawBaseProductDTO drawProduct) {

// 处理未拉伸的软装白膜产品

// 0 软装，1、非软装

if (DrawBaseProductConstant.MODEL\_TYPE\_SOFT.intValue() != drawProduct.getModelType().intValue()) {

log.debug("提交户型：非软装产品, modelType => {}", drawProduct.getModelType());

dtpProduct.setCreateProductStatus(DrawDesignTempletProductConstant.NON\_SOFT\_PRODUCT);

return dtpProduct;

}

Utils.requireGreaterZero(drawProduct.getProductId(), "引用的软装产品不能为空");

// 标记从软装产品

dtpProduct.setCreateProductStatus(DrawDesignTempletProductConstant.SOFT\_PRODUCT);

// 处理拉伸的软装产品

if (!DrawBaseProductConstant.IS\_CHANGED\_SOFT.equals(drawProduct.getIsChanged())) {

log.debug("提交户型：白膜产品 {}(productId)没有拉伸，不处理", drawProduct.getProductId());

return dtpProduct;

}

log.info("提交户型：开始处理软装产品，drawBaseProductDTO = {}, isChanged = {}", drawProduct.getProductId(), drawProduct.getIsChanged());

return baseProductService.addChangedSoftProduct(dtpProduct, drawProduct);

}

/\*\*

\* @param drawBaseProduct

\* @param userLoginBO

\* @return

\*/

private DrawDesignTempletProduct getDrawDesignTempletProduct(DrawBaseProductDTO drawBaseProduct,

UserLoginBO userLoginBO) {

Date now = new Date();

DrawDesignTempletProduct drawDesignTempletProduct = new DrawDesignTempletProduct();

drawDesignTempletProduct.setGmtCreate(now);

drawDesignTempletProduct.setGmtModified(now);

drawDesignTempletProduct.setCreator(userLoginBO.getLoginName());

drawDesignTempletProduct.setModifier(userLoginBO.getLoginName());

drawDesignTempletProduct.setIsDeleted(DrawDesignTempletProductConstant.DEFAULTS\_IS\_DELETED);

drawDesignTempletProduct.setUniqueId(drawBaseProduct.getUniqueId());

drawDesignTempletProduct.setWallOrientation(drawBaseProduct.getWallOrientation());

drawDesignTempletProduct.setWallType(drawBaseProduct.getWallType());

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

// 软装产品ID, 硬装部分在保存之前处理

drawDesignTempletProduct.setProductId(drawBaseProduct.getProductId() != null ? drawBaseProduct.getProductId().intValue() : null);

return drawDesignTempletProduct;

}

/\*\*

\* 组合产品处理，

\* 通用app需要这样处理,用来区分多个相同的组合产品;plan\_group\_id.

\* 处理组合的产品，相同的组合产品的planGroupId = groupProductId\_后缀一样

\*

\* @param drawDesignTempletProduct

\* @param drawBaseProductDTO

\* @param houseSubmitBO

\*/

private void handlerGroupProduct(DrawDesignTempletProduct drawDesignTempletProduct,

DrawBaseProductDTO drawBaseProductDTO,

DrawHouseSubmitBO houseSubmitBO) {

if (drawBaseProductDTO.getGroupProductId() != null && drawBaseProductDTO.getGroupProductId() > 0) {

// 组合产品

drawDesignTempletProduct.setProductGroupId(drawBaseProductDTO.getGroupProductId());

// 是否主产品 ==> 0-> 否、1-> 是

drawDesignTempletProduct.setIsMainProduct(drawBaseProductDTO.getIsMainProduct());

// 组合类型 ==> 0-> 组合、1-> 结构

drawDesignTempletProduct.setGroupType(DrawDesignTempletProductConstant.GROUP\_PRODUCT\_TYPE);

// 组合产品唯一标识，判断存在两个相同的组合产品的唯一标识

drawDesignTempletProduct.setGroupUniqueId(drawBaseProductDTO.getGroupUniqueId());

// 通用app需要这样处理,用来区分多个相同的组合产品;plan\_group\_id.

// 处理组合的产品，相同的组合产品的planGroupId = groupProductId\_后缀一样

drawDesignTempletProduct.setPlanGroupId(drawBaseProductDTO.getGroupProductId() + "\_"

+ getGroupIndex(houseSubmitBO.getIdxMap(), drawBaseProductDTO.getGroupUniqueId()));

} else {

// defaults value

drawDesignTempletProduct.setProductGroupId(DrawDesignTempletProductConstant.DEFAULTS\_GROUP\_PRODUCT\_ID);

drawDesignTempletProduct.setIsMainProduct(DrawDesignTempletProductConstant.DEFAULTS\_IS\_MAIN\_PRODUCT);

drawDesignTempletProduct.setGroupType(DrawDesignTempletProductConstant.GROUP\_PRODUCT\_TYPE);

drawDesignTempletProduct.setPlanGroupId(DrawDesignTempletProductConstant.DEFAULTS\_PLAN\_GROUP\_ID.toString());

}

}

/\*\*

\* 地面、天花区域标识处理.

\* 相同的类型以基数累加.

\* eg: 同一个空间里有5块地面以基数10开始累加，regionMark ==> 10 11...15;

\* 不同类型基数不一样{@link RegionMarkConstant}

\*

\* @param drawDesignTempletProduct

\* @param drawBaseProductDTO

\*/

private void handlerRegionMark(DrawDesignTempletProduct drawDesignTempletProduct,

DrawBaseProductDTO drawBaseProductDTO,

DrawHouseSubmitBO houseSubmitBO) {

if (RegionMarkConstant.contains(drawBaseProductDTO.getSmallTypeValueKey())) {

// 第一次时，初始化区域标识

if (houseSubmitBO.getDrawSpaceBO().getRegionMarks() == null

|| houseSubmitBO.getDrawSpaceBO().getRegionMarks().isEmpty()) {

houseSubmitBO.getDrawSpaceBO().setRegionMarks(this.getInitRegionMarks());

}

// get region mark

int regionMark = getRegionMark(houseSubmitBO.getDrawSpaceBO().getRegionMarks(),

drawBaseProductDTO.getSmallTypeValueKey());

if (regionMark > 0) {

drawDesignTempletProduct.setRegionMark(Objects.toString(regionMark, Utils.\_ONE\_VALUE));

}

log.debug("地面、天花区域标识处理, regionMark ==> {}", regionMark);

}

}

/\*\*

\* 获取初始化区域标识

\*

\* @return

\*/

private List<RegionMark> getInitRegionMarks() {

List<RegionMark> regions = new ArrayList<>();

RegionMarkConstant[] regionMarks = RegionMarkConstant.values();

for (RegionMarkConstant regionMark : regionMarks) {

regions.add(new RegionMark(regionMark));

}

return regions;

}

/\*\*

\* 地面、天花区域标识处理.

\*

\* <pre>

\* 相同的类型以基数累加.

\* eg: 同一个空间里有5块地面以基数10开始累加，regionMark ==> 10 11...15;

\* 不同类型基数不一样{@link RegionMarkConstant}

\* </pre>

\*

\* @param regions

\* @param valueKey

\* @return -1 标识不是地面类型处理

\*/

private int getRegionMark(List<RegionMark> regions, String valueKey) {

// 地面区域标识

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

return -1;

}

for (RegionMark region : regions) {

int regionMark = region.getRegionMark(valueKey);

if (regionMark > 0) {

return regionMark;

}

}

return -1;

}

/\*\*

\* 处理组合的产品，相同的组合产品的planGroupId = groupProductId\_后缀一样

\*

\* @param idxMap

\* @param groupUniqueId

\* @return

\*/

private String getGroupIndex(Map<String, String> idxMap, String groupUniqueId) {

if (StringUtils.isEmpty(groupUniqueId)) {

throw new BusinessException(false, ResponseEnum.GROUP\_UNIQUEID\_EMPTY);

}

if (idxMap.containsKey(groupUniqueId)) {

return idxMap.get(groupUniqueId);

} else {

String groupIndex = Objects.toString(System.nanoTime());

idxMap.put(groupUniqueId, groupIndex);

return groupIndex;

}

}

@Override

public void beforeSaveHandler(List<DrawSpaceBO> drawSpaceBOS) {

}

/\*\*

\* 拼装绑定点信息

\* Map<String, List<String>> bingInfoMap

\* key:主墙uniqueId

\* value:背景墙draw\_base\_product -> id

\*

\* @param drawSpaceBOS

\* @return

\*/

private Map<String, List<Long>> getBingInfoMap(List<DrawSpaceBO> drawSpaceBOS) {

// 先整理出一份uniqueId对应drawBaseProductId的map

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

for (DrawSpaceBO drawSpaceBO : drawSpaceBOS) {

for (DrawHouseProductBO drawHouseProductBO : drawSpaceBO.getDrawHouseProductBOS()) {

if (StringUtils.isEmpty(drawHouseProductBO.getDrawBaseProductDTO().getUniqueId())) {

throw new DrawBusinessException(false, ResponseEnum.DRAWBASEHOUSE\_SUBMIT\_FAILED);

}

uniqueIdToDrawBaseProductIdMap.put(drawHouseProductBO.getDrawBaseProductDTO().getUniqueId(),

drawHouseProductBO.getDrawBaseProduct() != null ? drawHouseProductBO.getDrawBaseProduct().getId() : null);

}

}

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

for (DrawSpaceBO drawSpaceBO : drawSpaceBOS) {

for (DrawHouseProductBO drawHouseProductBO : drawSpaceBO.getDrawHouseProductBOS()) {

String bindUniqueId = drawHouseProductBO.getDrawBaseProductDTO().getBindUniqueId();

String uniqueId = drawHouseProductBO.getDrawBaseProductDTO().getUniqueId();

if (StringUtils.isEmpty(bindUniqueId)) {

continue;

}

if (StringUtils.isEmpty(uniqueId)) {

throw new DrawBusinessException(false, ResponseEnum.DRAWBASEHOUSE\_SUBMIT\_FAILED);

}

if (!uniqueIdToDrawBaseProductIdMap.containsKey(uniqueId)) {

log.error("cannot find value by key:{}; map:{}", bindUniqueId, uniqueIdToDrawBaseProductIdMap.toString());

}

List<Long> drawProductIdList;

if (bingInfoMap.containsKey(bindUniqueId) && (drawProductIdList = bingInfoMap.get(bindUniqueId)) != null) {

drawProductIdList.add(uniqueIdToDrawBaseProductIdMap.get(uniqueId));

} else {

drawProductIdList = new ArrayList<>();

drawProductIdList.add(uniqueIdToDrawBaseProductIdMap.get(uniqueId));

bingInfoMap.put(bindUniqueId, drawProductIdList);

}

}

}

return bingInfoMap;

}

}

package com.sandu.general.service.product.impl;

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

import com.fasterxml.jackson.databind.ObjectMapper;

import com.sandu.common.constants.redis.RedisKeyConstant;

import com.sandu.common.util.Utils;

import com.sandu.common.util.json.JSON;

import com.sandu.general.api.dictionary.manager.redis.RedisManager;

import com.sandu.general.api.product.bo.BaseProductBO;

import com.sandu.general.api.product.bo.GroupProductBO;

import com.sandu.general.api.product.bo.ProductTypeBO;

import com.sandu.general.api.product.input.GroupProductQuery;

import com.sandu.general.api.product.service.BaseProductService;

import com.sandu.general.api.product.service.GroupProductService;

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

import com.sandu.general.api.res.model.ResModel;

import com.sandu.general.api.res.model.ResPic;

import com.sandu.general.api.res.service.ResFileService;

import com.sandu.general.api.res.service.ResModelService;

import com.sandu.general.api.res.service.ResPicService;

import com.sandu.general.service.product.dao.GroupProductMapper;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.extern.slf4j.Slf4j;

import org.apache.commons.lang3.StringUtils;

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

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

import org.springframework.stereotype.Service;

import java.util.\*;

import java.util.concurrent.TimeUnit;

import java.util.stream.Collectors;

/\*\*

\* CopyRight (c) 2017 Sandu Technology Inc.

\* <p>

\* house-draw

\*

\* @author songjianming@sanduspace.cn

\* <p>

\* 2018年1月21日

\*/

@Slf4j

@Service("groupProductService")

public class GroupProductServiceImpl implements GroupProductService {

@Autowired

GroupProductMapper groupProductMapper;

@Autowired

ResModelService resModelService;

@Autowired

ResPicService resPicService;

@Autowired

ResFileService resFileService;

@Autowired

BaseProductService baseProductService;

@Autowired

RedisManager redisManager;

@Autowired

ObjectMapper objectMapper;

@Autowired

GroupProductService groupProductService;

/\*\*

\* defaults(4320 min) 3天

\*/

@Value("${redis.soft.group.product.expire:4320}")

private Long redisSoftGroupProductExpire;

@Override

public Long countGroupProduct(GroupProductQuery query) {

if (query == null) {

return 0L;

}

return groupProductMapper.countGroupProduct(query);

}

@Override

public Map<String, Object> listGroupProduct(GroupProductQuery query, boolean useCache) {

if (useCache) {

String hashKey = getHashKey(query);

String key = RedisKeyConstant.HOUSE\_DRAW\_GROUP\_SOFT\_PRODUCT.toString();

String value = redisManager.getCacheByHash(key, hashKey, null,

TimeUnit.MINUTES, () -> JSON.stringify(groupProductService.listGroupProduct(query)));

// get mapper value

if (!StringUtils.isEmpty(value)) {

log.debug("listGroupProduct get cache by hashKey => {}", hashKey);

return JSON.parse(value, new TypeReference<Map<String, Object>>() {

});

}

}

return this.listGroupProduct(query);

}

/\*\*

\* 生成redis的缓存key

\* @param query

\* @return

\*/

private String getHashKey(GroupProductQuery query) {

StringBuilder buf = new StringBuilder(":listGroupProduct");

if (query.getGroupType() != null) {

buf.append(":").append(query.getGroupType());

}

if (query.getPageNum() != null) {

buf.append(":").append(query.getPageNum());

}

if (query.getPageSize() != null) {

buf.append(":").append(query.getPageSize());

}

if (query.getPutawayState() != null && query.getPutawayState().length > 0) {

for (Integer state : query.getPutawayState()) {

buf.append(":").append(state);

}

}

return buf.toString().replaceFirst(":", "");

}

@Override

public Map<String, Object> listGroupProduct(GroupProductQuery query) {

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

// List<GroupProductBO> groupProducts = new ArrayList<>();

//

// Long count = this.countGroupProduct(query);

// if (count != null && count > 0) {

// groupProducts = groupProductMapper.listGroupProduct(query);

// }

List<GroupProductBO> groupProducts = groupProductMapper.listGroupProduct(query);

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

// 图片、文件资源

Map<String, Set<Long>> mapRes = handlerResource(groupProducts);

Map<String, ResPic> resPicMap = resPicService.getResPicMap(mapRes.get("pics"));

Map<String, ResFile> resFileMap = resFileService.getResFileMap(mapRes.get("files"));

// 获取组合里的产品

Map<Long, List<BaseProductBO>> proMap = getProductForMap(groupProducts);

for (GroupProductBO bo : groupProducts) {

// 图片

ResPic resPic = resPicMap.get(Objects.toString(bo.getGroupPicId(), Utils.\_ONE\_VALUE));

if (resPic != null) {

bo.setGroupPicPath(resPic.getPicPath());

}

// 组合产品位置关系

ResFile resFile = resFileMap.get(Objects.toString(bo.getConfigId(), Utils.\_ONE\_VALUE));

if (resFile != null) {

bo.setConfigFile(resFile.getFilePath());

}

// C# null 处理(Fuck C#)

bo.setConfigId(bo.getConfigId() == null ? 0 : bo.getConfigId());

bo.setGroupPicId(bo.getGroupPicId() == null ? 0 : bo.getGroupPicId());

bo.setGroupType(bo.getGroupType() == null ? -1L : bo.getGroupType());

// products

bo.setProducts(proMap.get(bo.getGroupId()));

}

}

// map.put("total", count);

map.put("list", groupProducts);

return map;

}

/\*\*

\* 获取组合里的产品

\*

\* @param list

\* @return

\*/

private Map<Long, List<BaseProductBO>> getProductForMap(List<GroupProductBO> list) {

Map<Long, List<BaseProductBO>> proMap = new HashMap<>();

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

return proMap;

}

List<BaseProductBO> products = groupProductMapper.listBaseProduct(list);

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

log.warn("组合产品里的产品数据可能出现异常, groupProducts => {}", Arrays.toString(list.stream().map(GroupProductBO::getGroupId).toArray()));

return proMap;

}

// 处理产品 model资源、分类信息

this.handlerProduct(products);

proMap = products.stream().collect(Collectors.groupingBy(BaseProductBO::getGroupId));

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

}

/\*\*

\* 处理产品 model资源、分类信息

\*

\* @param products

\*/

private void handlerProduct(List<BaseProductBO> products) {

// 模型资源

Map<String, ResModel> modelMap = getModelMap(products);

// 大小分类

Map<Long, ProductTypeBO> productTypes = baseProductService.listProductType(products);

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

log.warn("产品的分类数据可能出现异常, products => {}", Arrays.toString(products.stream().map(BaseProductBO::getId).toArray()));

return;

}

for (BaseProductBO p : products) {

// 模型

ResModel resModel = modelMap.get(Objects.toString(p.getModelId(), Utils.\_ONE\_VALUE));

if (resModel != null) {

p.setModelName(resModel.getModelName());

p.setModelPath(resModel.getModelPath());

}

// C# null 处理

p.setModelId(p.getModelId() == null ? 0 : p.getModelId());

// 大小分类

baseProductService.buildProductType(productTypes.get(p.getId()), p);

}

}

/\*\*

\* 处理模型

\*

\* @param products

\* @return

\*/

private Map<String, ResModel> getModelMap(List<BaseProductBO> products) {

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

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

for (BaseProductBO p : products) {

// 清除Null Object (Fuck NullPointerException)

if (p.getModelId() != null && p.getModelId() > 0) {

models.add(p.getModelId());

}

}

}

return resModelService.getResModelMap(models);

}

/\*\*

\* 处理图片、文件

\*

\* @param gps

\* @return

\*/

private Map<String, Set<Long>> handlerResource(List<GroupProductBO> gps) {

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

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

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

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

for (GroupProductBO gp : gps) {

// 清除Null Object (Fuck NullPointerException)

if (gp.getGroupPicId() != null && gp.getGroupPicId() > 0) {

pics.add(gp.getGroupPicId());

}

if (gp.getConfigId() != null && gp.getConfigId() > 0) {

files.add(gp.getConfigId());

}

}

}

map.put("pics", pics);

map.put("files", files);

return map;

}

@Getter

@AllArgsConstructor

enum GroupType {

SOFA("sofa", "沙发组"),

TV\_BENCH("TVbench", "电视柜"),

DININGTABLE("diningTable", "餐桌组"),

SIDEBOARD("sideboard", "餐柜组"),

SHOECABINET("shoeCabinet", "鞋柜组"),

CONSOLECABINET("consoleCabinet", "隔断柜"),

ZHUANGSHI("zhuangshi", "装饰柜"),

CHUANGZU("chuangzu", "床组合"),

YUSHIGUI("yushigui", "浴室柜"),

SHUZHUO("shuzhuo", "书桌组"),

CHUANGTOUGUI("chuangtougui", "床头柜"),

SHUZHUANG("shuzhuang", "梳妆台"),

SHUGUI("shugui", "书柜组"),

SHUYI("shuyi", "书椅组");

final String key;

final String name;

}

}