package com.xunlu;

import org.apache.ibatis.mapping.DatabaseIdProvider;

import org.apache.ibatis.mapping.VendorDatabaseIdProvider;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import springfox.documentation.swagger2.annotations.EnableSwagger2;

import tk.mybatis.spring.annotation.MapperScan;

import java.util.Properties;

/\*\*
\* 会议系统

\* function

\*/

@SpringBootApplication

@EnableSwagger2

@MapperScan("com.xunlu.meeting.dao.mapper")

@ComponentScan(basePackages = {"com.xunlu.sec.uap.util.config","com.xunlu.sec.uap.client.rest.cache.memory"

,"com.xunlu.meeting","com.xunlu.common","com.xunlu.msg","com.xunlu.config","com.xunlu.task"

,"com.xunlu.utils.fastdfs","com.xunlu.utils.redis","com.xunlu.rabbitmq","com.xunlu.component"})

public class AiMeetingWebApplication {

public static void main(String[] args) {

SpringApplication.run(AiMeetingWebApplication.class, args);

}

/\*\*

\* DatabaseIdProvider元素主要是为了支持不同的数据库

\* @return

\*/

@Bean

public DatabaseIdProvider getDatabaseIdProvider() {

DatabaseIdProvider databaseIdProvider = new VendorDatabaseIdProvider();

Properties properties = new Properties();

properties.setProperty("Oracle", "oracle");

properties.setProperty("MySQL", "mysql");

databaseIdProvider.setProperties(properties);

return databaseIdProvider;

}

}

package com.xunlu;

import org.springframework.stereotype.Component;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Component

public class WebMvcConfig implements WebMvcConfigurer {

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/static/\*\*").addResourceLocations("classpath:/static/");

}

}

package com.xunlu.meeting.atomService.inter;

import com.xunlu.common.msg.BaseResult;

import com.xunlu.common.msg.MultiResult;

import com.xunlu.common.orm.Page;

import java.util.List;

public interface IMtOfficeAddressService {

/\*\*

\* 新增地址

\*

\* @param name

\* @param type

\* @param pCode

\* @param sort

\* @param floorNum

\* @return

\*/

BaseResult insertAddress(String name, int type, String pCode, int sort, Integer floorNum, String accountNo);

/\*\*

\* 激活或禁用地址

\*

\* @param status 1激活 0禁用

\* @param list

\* @return

\*/

BaseResult activateOrForbiddenAddress(int status, List<String> list, String accountNo);

/\*\*

\* 逻辑删除会议室

\*

\* @param list

\* @param accountNo

\* @return

\*/

BaseResult deleteAddress(List<String> list, String accountNo);

/\*\*

\* 修改地址排序

\*

\* @param sort

\* @param id

\* @return

\*/

BaseResult updateAddressSort(Integer sort, String id, String accountNo);

/\*\*

\* 查询地址列表分页数据

\*

\* @param page

\* @param pCode

\* @return

\*/

MultiResult queryAddressByPage(Page page, String pCode, String accountNo);

/\*\*

\* 根据code 获取地址全称

\* @param address

\* @return

\*/

String getOfficeAddressByCode(String address);

}

package com.xunlu.meeting.atomService.inter;

import com.xunlu.common.msg.BaseResult;

public interface IMtOfficeManageService {

/\*\*

\* 查询数据地点表

\* @return

\*/

BaseResult getOfficeAddress();

/\*\*

\* 获取规则

\* @return

\*/

BaseResult getRuleInfoList();

/\*\*

\*根据多个code ,隔开

\*/

BaseResult getOfficeAddressByCodes(String codes);

/\*\*

\* 查询地址信息

\* @param code

\* @param spaceMarker 拼接符号

\* @return

\*/

String getMtOfficeAddressNameByCode(String code, String spaceMarker);

/\*\*

\* 获取动态访客URL

\* @param visitorId

\* @return

\*/

String getVisitorUrlById(String visitorId);

}

package com.xunlu.meeting.controller;

import com.xunlu.common.msg.BaseResult;

import com.xunlu.common.orm.Page;

import com.xunlu.common.utils.MessageUtil;

import com.xunlu.meeting.atomService.inter.IMtOfficeOrderService;

import com.xunlu.meeting.atomService.inter.IMtOrderSummaryService;

import com.xunlu.meeting.dto.PcCompletedOrderParamDto;

import com.xunlu.meeting.dto.UserInfo;

import com.xunlu.utils.UserUtil;

import io.swagger.annotations.ApiOperation;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

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

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

import javax.servlet.http.HttpServletRequest;

/\*\*

\* @description: <p>已完成会议</p>

\*/

@Controller

@RequestMapping("/officepc/completedorder")

public class CompletedOrderController {

private static Logger logger = LoggerFactory.getLogger(CompletedOrderController.class);

@Autowired

private IMtOfficeOrderService mtOfficeOrderService;

@Autowired

IMtOrderSummaryService mtOrderSummaryService;

/\*\*

\* 查询已完成会议列表接口

\* @param

\* @param

\* @return

\* @throws Exception

\*/

@PostMapping(value = "/getCompletedOrderList.do")

@ResponseBody

public BaseResult getCompletedOrderList(HttpServletRequest request, Page page, PcCompletedOrderParamDto params) throws Exception {

long startTime = System.currentTimeMillis();

logger.info("查询已完成会议列表接口开始，params="+params);

UserInfo userInfo = UserUtil.getUserInfo(request);

params.setAccountName(userInfo.getUsername());

if (params.getPageSize() == null || params.getPageSize() > 100) {

params.setPageSize(10);

}

if (params.getCurrentPageNo() == null) {

params.setCurrentPageNo(1);

}

BaseResult result = mtOfficeOrderService.getCompletedOrderList(params);

logger.info("查询已完成会议列表接口结束,消耗时间：" + (System.currentTimeMillis() - startTime));

return result;

}

@ApiOperation(value = "获取会议内容信息", notes = "")

@PostMapping(value = "/getOrderContentById.do")

@ResponseBody

public BaseResult getOrderContentById(HttpServletRequest request,@RequestParam(value = "orderId") String orderId ) {

try {

//操作处理开始

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOrderSummaryService.getOrderContentById(orderId, userInfo.getUsername());

return result;

} catch (Exception e) {

logger.error(e.getMessage(), e);

return new BaseResult(false, MessageUtil.ERROR, "获取会议内容信息出错");

}

}

}

package com.xunlu.meeting.controller;

import com.alibaba.fastjson.JSONObject;

import com.xunlu.common.msg.BaseResult;

import com.xunlu.common.msg.ObjectResult;

import com.xunlu.meeting.atomService.inter.IMtOfficeOrderService;

import com.xunlu.meeting.atomService.inter.IMtOrderParticipantService;

import com.xunlu.meeting.atomService.inter.IOfficeManageForWebService;

import com.xunlu.meeting.atomService.inter.IUserForEmpInfoService;

import com.xunlu.meeting.dto.\*;

import com.xunlu.utils.SecLibUtil;

import com.xunlu.utils.SystemCode;

import com.xunlu.utils.UserUtil;

import org.apache.commons.lang3.StringUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

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

import javax.servlet.http.HttpServletRequest;

/\*\*

\* 描述:

\* 会议预定首页

\*/

@Controller

@RequestMapping("/officepc/order")

public class OfficeOrderController {

private static Logger logger = LoggerFactory.getLogger(OfficeOrderController.class);

@Autowired

private IMtOfficeOrderService mtOfficeOrderService;

@Autowired

private IOfficeManageForWebService officeManageForWebService;

@Autowired

private IMtOrderParticipantService mtOrderParticipantService;

@Autowired

private IUserForEmpInfoService userForEmpInfoService;

/\*\*

\* 常用会议室列表

\*

\* @param request

\* @return

\*/

@RequestMapping(value = "/queryUsedOffice.do")

@ResponseBody

public BaseResult queryUsedOffice(HttpServletRequest request) throws Exception {

logger.info("常用会议室列表接口调用");

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.queryUsedOffice(userInfo.getUsername());

return result;

}

/\*\*

\* 所在区域会议室列表

\*

\* @param request

\* @return

\*/

@RequestMapping(value = "/queryAreaOffice.do")

@ResponseBody

public BaseResult queryAreaOffice(HttpServletRequest request) throws Exception {

logger.info("所在区域会议室列表接口调用");

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.queryAreaOffice(userInfo.getUsername());

return result;

}

/\*\*

\* 获取全部会议室列表

\*

\* @param request

\* @param parmDto

\* @return

\*/

@RequestMapping(value = "/queryAllOffice.do")

@ResponseBody

public BaseResult queryAllOffice(HttpServletRequest request, OfficeSearchParmDto parmDto) throws Exception {

UserInfo userInfo = UserUtil.getUserInfo(request);

parmDto.setEmpAccountNo(userInfo.getUsername());

String str = parmDto.getDevice();

if(StringUtils.isNotBlank(str)){

parmDto.setDeviceTypes(str.split(","));

}

if (parmDto.getPageSize() == null || parmDto.getPageSize() > 100) {

parmDto.setPageSize(10);

}

if (parmDto.getPageNum() == null) {

parmDto.setPageNum(1);

}

BaseResult result = mtOfficeOrderService.queryAllOfficeByPage(parmDto);

return result;

}

/\*\*

\* 会议室信息查询

\*

\* @param id

\* @return

\*/

@RequestMapping(value = "/viewOfficeDetailById.do")

@ResponseBody

public BaseResult viewOfficeDetailById(String id) throws Exception {

BaseResult result = officeManageForWebService.findOfficeById(id);

return result;

}

/\*\*

\* 获取最近联系人（内部、访客）

\*

\* @param request

\* @param type

\* @return

\*/

@RequestMapping(value = "/getRecentContact.do")

@ResponseBody

public BaseResult getRecentContact(HttpServletRequest request, Integer type) {

logger.info("getRecentContact接口开始调用，type={}", type);

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOrderParticipantService.getRecentContact(type, userInfo.getUsername());

return result;

}

/\*\*

\* 获取会议室已定会议信息列表

\*

\* @param paramDto

\* @return

\*/

@RequestMapping(value = "/appointmentMeetingList.do")

@ResponseBody

public BaseResult appointmentMeetingList(QueryOrderParamDto paramDto, HttpServletRequest request) {

logger.info("获取会议室已定会议信息列表接口调用开始，参数[paramDto={}]", JSONObject.toJSONString(paramDto));

UserInfo userInfo = UserUtil.getUserInfo(request);

paramDto.setAccountNo(userInfo.getUsername());

BaseResult result = mtOfficeOrderService.queryOfficeOrderByDateAndId(paramDto);

return result;

}

/\*\*

\* 新增会议订单（锁定会议室）

\*

\* @param request

\* @param lockOfficeDto

\* @return

\*/

@RequestMapping(value = "/addMeetingOrder.do")

@ResponseBody

public BaseResult addMeetingOrder(HttpServletRequest request, LockOfficeDto lockOfficeDto) {

try {

/\*\*

\* 主题验证 (暂不做验证)

\*/

// boolean validateParam = SecLibUtil.validateSecLibHtml(lockOfficeDto.getOfficeTitle());

// if (!validateParam) {

// return new BaseResult(false, "S10000001", "会议主题验证不通过！");

// }

// lockOfficeDto.setOfficeTitle("请填写会议主题");//默认值

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.lockOffice(lockOfficeDto,userInfo, SystemCode.FromType.WEB);

return result;

} catch (Exception e) {

logger.error("锁定会议失败", e);

return new BaseResult(false, "", "锁定会议失败");

}

}

/\*\*

\* 获取已预定会议室的详细信息

\*

\* @param request

\* @param id 会议id

\* @returnid

\*/

@PostMapping(value = "/getMeetingDetail.do")

@ResponseBody

public BaseResult getMeetingDetail(HttpServletRequest request, String id) throws Exception {

logger.info("获取已预定会议室的详细信息开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.getMeetingDetail(id, userInfo.getUsername());

return result;

}

/\*\*

\* 保存会议订单详情接口

\*

\* @param request

\* @param param 数据详情

\* @returnid

\*/

@PostMapping(value = "/saveOfficeApply.do")

@ResponseBody

public BaseResult saveOfficeApply(HttpServletRequest request, String param) throws Exception {

logger.info("保存会议订单详情接口开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

OrderSaveParamDto paramDto = JSONObject.parseObject(param, OrderSaveParamDto.class);

if (!SecLibUtil.validateSecLibHtml(paramDto.getTheme())) {

return new BaseResult(false, SystemCode.MEETING\_START, "会议主题不符合规范！");

}

BaseResult result = mtOfficeOrderService.saveOfficeApply(paramDto, userInfo.getUsername());

return result;

}

/\*\*

\* 根据搜索内容获取人员信息列表

\*

\* @param searchContent 搜索内容

\* @returnid

\*/

@PostMapping(value = "/getUserListBySearchContent.do")

@ResponseBody

public BaseResult getUserListBySearchContent(String searchContent) throws Exception {

logger.info("保根据搜索内容获取人员信息列表接口开始");

ObjectResult result = new ObjectResult();

result.setData(userForEmpInfoService.getEmpListByParm(searchContent));

return result;

}

}

package com.xunlu.meeting.controller;

import com.xunlu.common.msg.BaseResult;

import com.xunlu.meeting.atomService.inter.IMtOfficeOrderService;

import com.xunlu.meeting.dto.PcOrderParamDto;

import com.xunlu.meeting.dto.UserInfo;

import com.xunlu.utils.UserUtil;

import io.swagger.annotations.ApiImplicitParam;

import io.swagger.annotations.ApiImplicitParams;

import io.swagger.annotations.ApiOperation;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

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

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

import javax.servlet.http.HttpServletRequest;

import java.text.ParseException;

/\*\*

\* @description: <p>未完成会议模块</p>

\*/

@Controller

@RequestMapping("/officepc/pendingorder")

public class PendingOrderController {

private static Logger logger = LoggerFactory.getLogger(PendingOrderController.class);

@Autowired

IMtOfficeOrderService mtOfficeOrderService;

/\*\*

\* 未完成会议列表

\* @param request

\* @param

\* @return

\*/

@RequestMapping(value = "/pendingOrderList.do")

@ResponseBody

public BaseResult getPendingOrderList(HttpServletRequest request, PcOrderParamDto paramDto) throws Exception {

String paramStr = paramDto.toString();

logger.info("获取待参加会议列表信息详情接口开始调用，接收参数paramDto={}", paramStr);

UserInfo userInfo = UserUtil.getUserInfo(request);

paramDto.setAccountName(userInfo.getUsername());

if (paramDto.getPageSize() == null || paramDto.getPageSize() > 100) {

paramDto.setPageSize(10);

}

if (paramDto.getCurrentPageNo() == null) {

paramDto.setCurrentPageNo(1);

}

BaseResult result = mtOfficeOrderService.pendingOrderList(paramDto);

logger.info("获取待参加会议列表信息详情接口结束调用,返回结果为{}" , result);

return result;

}

/\*\*

\* 根据ID获取会议室的详细信息(已完成，未完成通用)

\*

\* @param

\* @returnid

\*/

@ApiOperation(value = "获取已预定会议室的详细信息")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "获取已预定会议室的详细信息", value = "id", required = true, dataType = "String")})

@PostMapping(value = "/getMeetingDetail.do")

@ResponseBody

public BaseResult getMeetingDetail(String id,HttpServletRequest request) {

logger.info("获取已预定会议室的详细信息开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

try{

long startTime = System.currentTimeMillis();

BaseResult result = mtOfficeOrderService.getMeetingDetail(id,userInfo.getUsername());

logger.info("获取已预定会议室的详细信息结束,消耗时间：" + (System.currentTimeMillis() - startTime));

return result;

}catch (Exception e){

logger.error("获取已预定会议室的详细信息失败", e);

return new BaseResult(false, "", "获取已预定会议室的详细信息失败");

}

}

/\*\*

\* 取消预定会议 物理删除 改变状态

\*

\* @param id 预定会议室申请单Id

\* @return

\*/

@ApiOperation(value = "用来取消预定会议", notes = "")

@ApiImplicitParams({@ApiImplicitParam(paramType = "update", name = "id", value = "会议申请单id", required = true, dataType = "String")})

@PostMapping(value = "/updateCheckStatus.do")

@ResponseBody

public BaseResult updateCheckStatus(@RequestParam(value = "id") String id,HttpServletRequest request) throws ParseException {

UserInfo userInfo = UserUtil.getUserInfo(request);

long startTime = System.currentTimeMillis();

logger.info("取消预定会议室接口开始");

BaseResult result = mtOfficeOrderService.updateCheckStatus(id, userInfo.getUsername());

logger.info("取消预定会议室接口结束,消耗时间：" + (System.currentTimeMillis() - startTime));

return result;

}

/\*\*

\* 提前开始当前会议

\*

\* @param id

\* @return

\* @throws Exception

\*/

@PostMapping(value = "/startInAdvanceOfficeOrderById.do")

@ResponseBody

public BaseResult startInAdvanceOfficeOrderById(String id, HttpServletRequest request) throws Exception {

long startTime = System.currentTimeMillis();

UserInfo userInfo = UserUtil.getUserInfo(request);

logger.info("提前开始当前会议接口调用开始，接收参数id=" + id + ";accountNo=" + userInfo.getUsername());

BaseResult result = mtOfficeOrderService.startInAdvanceOfficeOrderById(id, userInfo.getUsername());

logger.info("提前开始当前会议接口调用结束,消耗时间：" + (System.currentTimeMillis() - startTime));

return result;

}

/\*\*

\* 停止当前会议

\*

\* @param orderId

\* @return

\* @throws Exception

\*/

@ApiOperation(value = "停止当前会议", notes = "")

@ApiImplicitParams({@ApiImplicitParam(paramType = "update", name = "orderId", value = "会议申请单id", required = true, dataType = "String")})

@PostMapping(value = "/stopOfficeOrderById.do")

@ResponseBody

public BaseResult stopOfficeOrderById(String orderId,HttpServletRequest request) throws Exception {

long startTime = System.currentTimeMillis();

UserInfo userInfo = UserUtil.getUserInfo(request);

logger.info("停止当前会议开始，接收参数orderId=" + orderId + ";accountNo=" + userInfo.getUsername());

BaseResult result = mtOfficeOrderService.stopOfficeOrderById(orderId, userInfo.getUsername());

logger.info("停止当前会议接口结束,消耗时间：" + (System.currentTimeMillis() - startTime));

return result;

}

/\*\*

\* 会议续时

\*

\* @param id

\* @return

\* @throws Exception

\*/

@ApiOperation(value = "会议续时", notes = "")

@ApiImplicitParams({@ApiImplicitParam(paramType = "update", name = "orderId", value = "会议申请单id", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "update", name = "continueMinutes", value = "续时时长", required = true, dataType = "int")})

@PostMapping(value = "/continueTimeOfficeOrderById.do")

@ResponseBody

public BaseResult continueTimeOfficeOrderById(String id ,int continueMinutes, HttpServletRequest request) throws Exception {

long startTime = System.currentTimeMillis();

UserInfo userInfo = UserUtil.getUserInfo(request);

logger.info("会议续时接口调用开始，接收参数id="+id+";accountNo="+userInfo.getUsername());

BaseResult result = mtOfficeOrderService.continueTimeOfficeOrderById(id,userInfo.getUsername(), userInfo.getUserId(), userInfo.getName(),continueMinutes);

logger.info("会议续时接口调用结束,消耗时间：" + (System.currentTimeMillis() - startTime));

return result;

}

}

package com.xunlu.meeting.controller.manage;

import com.xunlu.common.msg.BaseResult;

import com.xunlu.common.utils.AES;

import com.xunlu.meeting.atomService.inter.IConferenceManagerService;

import com.xunlu.meeting.dao.po.MtConferenceManage;

import com.xunlu.meeting.dto.OfficeQueryDto;

import com.xunlu.utils.BaseUtil;

import com.xunlu.utils.FileTransFormUtil;

import com.xunlu.utils.fastdfs.FastDFSClient;

import io.swagger.annotations.ApiOperation;

import org.apache.commons.lang3.StringUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

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

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;

/\*\*

\* @Description: 会务管理

\*/

@Controller

@RequestMapping("/conference/manage")

public class PcConferenceManageController {

private static Logger logger = LoggerFactory.getLogger(PcConferenceManageController.class);

@Autowired

private IConferenceManagerService conferenceManagerService;

@Autowired

private FastDFSClient fastDFSClient;

/\*\*

\* 会务管理查询列表页

\* @param queryDto

\* @return

\*/

@ApiOperation(value = "会务管理列表页", notes = "")

@PostMapping(value = "/getConferenceManageForPage.do")

@ResponseBody

public BaseResult getConferenceManageForPage(OfficeQueryDto queryDto) {

Long startTime = System.currentTimeMillis();

try {

logger.info("会务管理列表页列表查询开始");

if (queryDto.getPageSize() > 100) {

queryDto.setPageSize(10);

}

BaseResult result = conferenceManagerService.getConferenceManageForPage(queryDto);

logger.info("会务管理列表页，耗时： " + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用会务管理列表页分页查询接口出错", t);

return new BaseResult(false, null, "web调用会务管理列表页分页查询接口出错");

}

}

/\*\*

\* 会务管理详情

\* @param id

\* @return

\*/

@ApiOperation(value = "会务管理详情", notes = "")

@PostMapping(value = "/getConferenceManageDetail.do")

@ResponseBody

public BaseResult getConferenceManageDetail(String id) {

Long startTime = System.currentTimeMillis();

try {

logger.info("会务管理详情情查询开始");

BaseResult result = conferenceManagerService.getConferenceManageDetail(id);

logger.info("会务管理详情查询完成，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.debug("根据id获取会务管理详情失败，耗时：" + (System.currentTimeMillis() - startTime) + " ms");

return new BaseResult(false, null, "web调用会务管理详情接口出错");

}

}

/\*\*

\* 保存会务管理详情

\* @return

\*/

@ApiOperation(value = "保存会务管理详情", notes = "")

@PostMapping(value = "/saveConferenceManageDetail.do")

@ResponseBody

public BaseResult saveConferenceManageDetail(HttpServletRequest request, MultipartFile imageFile, MtConferenceManage mtConferenceManage) {

Long startTime = System.currentTimeMillis();

try {

logger.info("保存会务管理详情开始");

if (imageFile != null) {

if (!FileTransFormUtil.checkFileSize(imageFile.getSize())) {

return new BaseResult(false, null, "图片不能超过1M");

}

imageFile = FileTransFormUtil.compressImg(imageFile, request);

String imageURL = fastDFSClient.upload(imageFile);

BaseUtil.delTmpFile(imageFile.getOriginalFilename());

mtConferenceManage.setImageUrl(imageURL);

}

BaseResult result = conferenceManagerService.saveConferenceManageDetail(mtConferenceManage);

logger.info("保存会务管理详情成功，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用保存会务管理详情保存接口出错", t);

return new BaseResult(false, null, "web调用保存会务管理详情保存接口出错");

}

}

/\*\*

\* 删除会务管理

\* @param ids

\* @return

\*/

@ApiOperation(value = "删除会务管理", notes = "")

@PostMapping(value = "/removeConferenceManage.do")

@ResponseBody

public BaseResult removeConferenceManage(String[] ids) {

Long startTime = System.currentTimeMillis();

try {

logger.info("删除会务管理接口开始");

BaseResult result = conferenceManagerService.removeConferenceManage(ids);

logger.info("调用删除会务管理接口成功，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用删除会务管理出错", t);

return new BaseResult(false, null, "web调用删除会务管理保存接口出错");

}

}

/\*\*

\* 编辑会务管理详情

\* @return

\*/

@ApiOperation(value = "修改会议室信息", notes = "")

@PostMapping(value = "/updateConferenceManageDetail.do")

@ResponseBody

public BaseResult updateConferenceManageDetail(HttpServletRequest request, MtConferenceManage mtConferenceManage, MultipartFile imageFile) {

Long startTime = System.currentTimeMillis();

try {

logger.info("调用修改会议室接口开始");

if (imageFile != null) {

if (!FileTransFormUtil.checkFileSize(imageFile.getSize())) {

return new BaseResult(false, null, "图片不能超过1M");

}

imageFile = FileTransFormUtil.compressImg(imageFile, request);

String imageURL = fastDFSClient.upload(imageFile);

BaseUtil.delTmpFile(imageFile.getOriginalFilename());

mtConferenceManage.setImageUrl(imageURL);

} else if (StringUtils.isNotBlank(mtConferenceManage.getImageUrl())

&& mtConferenceManage.getImageUrl().startsWith("officepc/common/transImageUrl.do?service=")) {

String imageUrl = mtConferenceManage.getImageUrl().replace("officepc/common/transImageUrl.do?service=", "");

imageUrl = AES.decrypt(imageUrl, AES.findKey());

mtConferenceManage.setImageUrl(imageUrl);

}

BaseResult result = conferenceManagerService.updateConferenceManageDetail(mtConferenceManage);

logger.info("调用修改会议室接口成功，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用会议室保存接口出错", t);

return new BaseResult(false, null, "修改会议室信息接口出错");

}

}

/\*\*

\* 会议室资源列表

\* @param isFree 0免费1收费

\* @return

\*/

@ApiOperation(value = "会议室资源列表", notes = "")

@PostMapping(value = "/getConferenceResourceList.do")

@ResponseBody

public BaseResult getConferenceResourceList(String isFree) {

Long startTime = System.currentTimeMillis();

try {

logger.info("会议室资源列表查询开始");

BaseResult result = conferenceManagerService.getConferenceResourceList(isFree);

logger.info("会议室资源列表，耗时： " + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用会议室资源列表查询接口出错", t);

return new BaseResult(false, null, "web调用会议室资源列表查询接口出错");

}

}

}

package com.xunlu.meeting.controller.manage;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONArray;

import com.xunlu.common.msg.BaseResult;

import com.xunlu.common.orm.Page;

import com.xunlu.common.utils.util.ExcelUtil;

import com.xunlu.meeting.atomService.inter.IStatisticsService;

import com.xunlu.meeting.dto.StatisticsDto;

import com.xunlu.meeting.dto.StatisticsParam;

import io.swagger.annotations.ApiOperation;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

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

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.UnsupportedEncodingException;

import java.util.List;

/\*\*

\* @Description: 会议统计控制层

\*/

@Controller

@RequestMapping("/officepc/manage")

public class PcStatisticsController {

private static Logger logger = LoggerFactory.getLogger(PcStatisticsController.class);

@Autowired

private IStatisticsService statisticsService;

/\*\*

\* 获取分页统计数据接口

\*

\* @param param

\* @param page

\* @return

\*/

@ApiOperation(value = "获取分页统计数据接口", notes = "")

@RequestMapping(value = "/statisticsByOfficeForPage.do")

@ResponseBody

public BaseResult getStatisticsByOfficeForPage(StatisticsParam param, Page page) {

Long startTime = System.currentTimeMillis();

try {

logger.info("获取分页统计数据接口调用开始.接收参数param={}", param.toString());

BaseResult result = statisticsService.statisticsByOfficeForPage(param, page);

logger.info("统计接口调用成功，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("获取分页统计数据接口调用失败", t);

return new BaseResult(false, null, "web调用获取分页统计数据接口出错");

}

}

/\*\*

\* 获取统计数据接口

\*

\* @param param

\* @return

\*/

@ApiOperation(value = "获取统计数据接口", notes = "")

@RequestMapping(value = "/statistics.do")

@ResponseBody

public BaseResult getStatisticsInfo(StatisticsParam param) {

Long startTime = System.currentTimeMillis();

try {

logger.info("获取统计数据接口调用开始，接收参数param={}", param.toString());

BaseResult result = statisticsService.statistics(param);

logger.info("获取统计数据接口，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("获取分页统计数据接口调用失败", t);

return new BaseResult(false, null, "web调用获取分页统计数据接口出错");

}

}

/\*\*

\* 获取统计数据接口（按会议室）

\*

\* @param param

\* @return

\*/

@ApiOperation(value = "获取统计数据接口（按会议室）", notes = "")

@RequestMapping(value = "/statisticsByOffice.do")

@ResponseBody

public BaseResult getStatisticsByOfficeInfo(StatisticsParam param) {

Long startTime = System.currentTimeMillis();

try {

logger.info("统计接口开始调用，接收参数paramDto={}", param.toString());

BaseResult result = statisticsService.statisticsByOffice(param);

logger.info("统计接口调用成功，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("获取分页统计数据接口调用失败", t);

return new BaseResult(false, null, "web调用获取分页统计数据接口出错");

}

}

/\*\*

\* 导出统计excel

\*

\* @param

\* @return

\*/

@ApiOperation(value = "导出统计excel", notes = "")

@RequestMapping(value = "/exportStatisticsByExl.do")

public void exportStatisticsByExl(HttpServletResponse response,StatisticsParam entity,HttpServletRequest request) throws UnsupportedEncodingException {

// String param = URLDecoder.decode(StringUtil.trim(request.getParameter("param")), "utf-8");

// StatisticsParam entity = JSONObject.parseObject(param, StatisticsParam.class);

List<StatisticsDto> statisticsDtos = statisticsService.queryAllStatistis(entity);

JSONArray data = JSONArray.parseArray(JSON.toJSONString(statisticsDtos));

String queryType = String.valueOf(entity.getQueryType());

String[] title = null;

String[] field = null;

String name = "";

//按部门

if ("0".equals(queryType)) {

title = new String[]{"一级部门", "二级部门", "三级部门", "预定时长(h)", "预定次数(次)"};

field = new String[]{"levelIDepartmentsName", "secondarySectorName", "levelIIIDepartmentsName", "value", "countNumber"};

name = "会议统计\_按部门\_" + entity.getStartDate() + "至" + entity.getEndDate() + ".xlsx";

} else if ("1".equals(queryType)) {

//按会议室

title = new String[]{"会议室名称", "会议室位置", "预订时长(h)", "预订次数(次)"};

field = new String[]{"name", "addressStr", "value", "countNumber"};

name = "会议统计\_按会议室\_" + entity.getStartDate() + "至" + entity.getEndDate() + ".xlsx";

} else if ("2".equals(queryType)) {

//按参会人

title = new String[]{"一级部门", "二级部门", "三级部门", "参会人", "参会时长(h)", "参会次数(次)"};

field = new String[]{"levelIDepartmentsName", "secondarySectorName", "levelIIIDepartmentsName", "name", "value", "countNumber"};

name = "会议统计\_按参会人\_" + entity.getStartDate() + "至" + entity.getEndDate() + ".xlsx";

}

if ("3".equals(queryType)) {

//按预定人

title = new String[]{"一级部门", "二级部门", "三级部门", "预定人", "预订时长(h)", "预订次数(次)"};

field = new String[]{"levelIDepartmentsName", "secondarySectorName", "levelIIIDepartmentsName", "name", "value", "countNumber"};

name = "会议统计\_按预定人\_" + entity.getStartDate() + "至" + entity.getEndDate() + ".xlsx";

}

ExcelUtil.exportExcel(data, name, title, field, request, response);

}

}

package com.xunlu.meeting.controller.manage;

import com.xunlu.common.msg.BaseResult;

import com.xunlu.common.utils.AES;

import com.xunlu.meeting.atomService.inter.IOfficeManageForWebService;

import com.xunlu.meeting.dao.po.MtOfficeManage;

import com.xunlu.meeting.dto.MtOfficeManageDto;

import com.xunlu.meeting.dto.OfficeQueryDto;

import com.xunlu.meeting.dto.UserInfo;

import com.xunlu.utils.BaseUtil;

import com.xunlu.utils.FileTransFormUtil;

import com.xunlu.utils.UserUtil;

import com.xunlu.utils.fastdfs.FastDFSClient;

import io.swagger.annotations.ApiOperation;

import org.apache.commons.lang3.StringUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

import org.springframework.validation.annotation.Validated;

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

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

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

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;

import javax.validation.Valid;

/\*\*

\* @Description: 会议室管理控制层

\*/

@Controller

@RequestMapping("/officepc/manage")

public class PcOfficeManageController {

private static Logger logger = LoggerFactory.getLogger(PcOfficeManageController.class);

@Autowired

private IOfficeManageForWebService officeManageForWebService;

@Autowired

private FastDFSClient fastDFSClient;

/\*\*

\* 分页查询

\*

\* @param queryDto

\* @return

\*/

@ApiOperation(value = "分页查询", notes = "")

@RequestMapping(value = "/findOfficeManageForPage.do")

@ResponseBody

public BaseResult findOfficeManageForPage(HttpServletRequest request,OfficeQueryDto queryDto) {

Long startTime = System.currentTimeMillis();

try {

logger.info("会议室分页列表查询开始");

if (queryDto.getPageSize() > 100) {

queryDto.setPageSize(10);

}

UserInfo userInfo = UserUtil.getUserInfo(request);

queryDto.setAccountNo(userInfo.getUsername());

BaseResult result = officeManageForWebService.findOfficeForPage(queryDto);

logger.info("会议室分页列表查询成功，耗时： " + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用会议室分页查询接口出错", t);

return new BaseResult(false, null, "web调用会议室分页查询接口出错");

}

}

/\*\*

\* 会议室信息查询

\*

\* @param ids

\* @return

\*/

@ApiOperation(value = "会议室信息查询", notes = "")

@RequestMapping(value = "/viewOfficeDetailById.do")

@ResponseBody

public BaseResult viewOfficeDetailById(HttpServletRequest request, String ids) {

Long startTime = System.currentTimeMillis();

try {

logger.info("会议室详情查询开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = officeManageForWebService.findOfficeById(ids);

logger.info("会议室详情查询完成，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.debug("根据id获取会议室详情失败，耗时：" + (System.currentTimeMillis() - startTime) + " ms");

return new BaseResult(false, null, "web调用会议室详情接口出错");

}

}

/\*\*

\* 保存会议室信息

\*

\* @param mtOfficeManage

\* @return

\*/

@ApiOperation(value = "保存会议室信息", notes = "")

@RequestMapping(value = "/saveOfficeInfo.do")

@ResponseBody

public BaseResult saveOfficeInfo(HttpServletRequest request, MultipartFile imageFile, MtOfficeManageDto mtOfficeManage) {

Long startTime = System.currentTimeMillis();

try {

logger.info("保存会议室信息开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

if (userInfo != null) {

mtOfficeManage.setCreateBy(userInfo.getUsername());

}

if (imageFile != null) {

if (!FileTransFormUtil.checkFileSize(imageFile.getSize())) {

return new BaseResult(false, null, "图片不能超过1M");

}

imageFile = FileTransFormUtil.compressImg(imageFile, request);

String imageURL = fastDFSClient.upload(imageFile);

BaseUtil.delTmpFile(imageFile.getOriginalFilename());

mtOfficeManage.setImageUrl(imageURL);

}

BaseResult result = officeManageForWebService.saveOfficeInfo(mtOfficeManage);

logger.info("保存会议室信息成功，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用会议室保存接口出错", t);

return new BaseResult(false, null, "web调用会议室保存接口出错");

}

}

/\*\*

\* 激活会议室信息

\*

\* @param ids

\* @return

\*/

@ApiOperation(value = "激活会议室信息", notes = "")

@RequestMapping(value = "/activateOffice.do")

@ResponseBody

public BaseResult activateOffice(HttpServletRequest request, String ids) {

Long startTime = System.currentTimeMillis();

try {

logger.info("激活会议室接口开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

String modifyBy = "";

if (userInfo != null) {

modifyBy = userInfo.getUsername();

}

BaseResult result = officeManageForWebService.activateOffice(ids, modifyBy);

logger.info("激活会议室接口完成，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用激活会议室接口出错", t);

return new BaseResult(false, null, "web调用激活会议室接口出错");

}

}

/\*\*

\* 禁用会议室信息

\*

\* @param ids

\* @return

\*/

@ApiOperation(value = "禁用会议室信息", notes = "")

@RequestMapping(value = "/forbiddenOffice.do")

@ResponseBody

public BaseResult forbiddenOffice(HttpServletRequest request, String ids) {

Long startTime = System.currentTimeMillis();

try {

logger.info("调用禁用会议室接口开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

String modifyBy = "";

if (userInfo != null) {

modifyBy = userInfo.getUsername();

}

BaseResult result = officeManageForWebService.forbiddenOffice(ids, modifyBy);

logger.info("调用禁用会议室接口完成，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用禁用会议室接口出错", t);

return new BaseResult(false, null, "web调用禁用会议室保存接口出错");

}

}

/\*\*

\* 删除会议室

\*

\* @param request

\* @param ids

\* @return

\*/

@ApiOperation(value = "删除会议室", notes = "")

@RequestMapping(value = "/removeOffice.do")

@ResponseBody

public BaseResult removeOffice(HttpServletRequest request, String ids) {

Long startTime = System.currentTimeMillis();

try {

logger.info("调用删除会议室接口开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

String modifyBy = "";

if (userInfo != null) {

modifyBy = userInfo.getUsername();

}

BaseResult result = officeManageForWebService.removeOffice(ids, modifyBy);

logger.info("调用删除会议室接口成功，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用删除会议室接口出错", t);

return new BaseResult(false, null, "web调用删除会议室保存接口出错");

}

}

/\*\*

\* 修改会议室信息

\*

\* @param mtOfficeManage

\* @return

\*/

@ApiOperation(value = "修改会议室信息", notes = "")

@RequestMapping(value = "/updateOfficeInfo.do")

@ResponseBody

public BaseResult updateOfficeInfo(HttpServletRequest request, MtOfficeManageDto mtOfficeManage, MultipartFile imageFile) {

Long startTime = System.currentTimeMillis();

try {

logger.info("调用修改会议室接口开始");

UserInfo userInfo = UserUtil.getUserInfo(request);

if (userInfo != null) {

mtOfficeManage.setModifyBy(userInfo.getUsername());

}

if (imageFile != null) {

if (!FileTransFormUtil.checkFileSize(imageFile.getSize())) {

return new BaseResult(false, null, "图片不能超过1M");

}

imageFile = FileTransFormUtil.compressImg(imageFile, request);

String imageURL = fastDFSClient.upload(imageFile);

BaseUtil.delTmpFile(imageFile.getOriginalFilename());

mtOfficeManage.setImageUrl(imageURL);

} else if (StringUtils.isNotBlank(mtOfficeManage.getImageUrl())

&& mtOfficeManage.getImageUrl().startsWith("officepc/common/transImageUrl.do?service=")) {

String imageUrl = mtOfficeManage.getImageUrl().replace("officepc/common/transImageUrl.do?service=", "");

imageUrl = AES.decrypt(imageUrl, AES.findKey());

mtOfficeManage.setImageUrl(imageUrl);

}

if ("undefined".equals(mtOfficeManage.getGroupId())){

mtOfficeManage.setGroupId(null);

}

BaseResult result = officeManageForWebService.updateOfficeInfo(mtOfficeManage);

logger.info("调用修改会议室接口成功，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

} catch (Exception t) {

logger.error("web调用会议室保存接口出错", t);

return new BaseResult(false, null, "修改会议室信息接口出错");

}

}

}

package com.xunlu.rabbitmq;

import com.xunlu.component.ApplicationSupport;

import com.xunlu.meeting.atomService.inter.IOfficeOrderTaskService;

import com.rabbitmq.client.Channel;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.amqp.core.Message;

import org.springframework.amqp.rabbit.listener.api.ChannelAwareMessageListener;

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

import java.io.IOException;

/\*\*

\* 消费者（mq会议室任务消费）

\*

\*/

public class TaskReciver implements ChannelAwareMessageListener {

private Logger logger = LoggerFactory.getLogger(TaskReciver.class);

@Autowired

private IOfficeOrderTaskService officeOrderTaskService;

@Override

public void onMessage(Message message, Channel channel) throws Exception {

byte[] body = message.getBody();

try {

//业务逻辑

String msgContent = new String(body);

logger.info("消费 receive msg : " + msgContent);

// 消息的标识，false只确认当前一个消息收到，true确认所有consumer获得的消息

officeOrderTaskService.executeTask(msgContent);

//手动确认确认消息成功消费

channel.basicAck(message.getMessageProperties().getDeliveryTag(), false);

} catch (Exception e) {

logger.error("消费失败,记录任务内容: " + new String(body));

// ack返回false，并重新回到队列，api里面解释得很清楚

try {

// channel.basicNack(message.getMessageProperties().getDeliveryTag(), false, true);

//防止任务在中间阶段导致任务回滚，所以此时直接手动确认任务关闭任务

channel.basicAck(message.getMessageProperties().getDeliveryTag(), false);

} catch (IOException e1) {

logger.error("重新回到队列异常");

}

}

}

}

package com.xunlu.meeting.controller;

import com.xunlu.common.msg.BaseResult;

import com.xunlu.meeting.atomService.inter.IMtOrderSummaryService;

import com.xunlu.meeting.dto.UserInfo;

import com.xunlu.utils.UserUtil;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

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

import javax.servlet.http.HttpServletRequest;

/\*\*

\* @PackageName:com.xunlu.meeting.controller

\* @ClassName:AppCompletedOrderController

\* @Description:app会议内容

\*/

@CrossOrigin

@Controller

@RequestMapping("/officeapp/completed")

public class AppCompletedOrderController {

private static Logger logger = LoggerFactory.getLogger(AppCompletedOrderController.class);

@Autowired

private IMtOrderSummaryService mtOrderSummaryService;

/\*\*

\* 已完成会议内容获取

\* @param orderId 会议id

\* @return

\*/

@RequestMapping("/getAppSummaryInfo.do")

@ResponseBody

public BaseResult getAppSummaryInfo(String orderId, HttpServletRequest request) {

logger.info("已完成会议内容获取接口开始调用，orderId={}", orderId);

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOrderSummaryService.getOrderContentById(orderId,userInfo.getUsername());

return result;

}

/\*\*

\* 查询会议纪要接收人员列表

\* @param orderId

\* @return

\*/

@RequestMapping("/getSumReceivePersonList.do")

@ResponseBody

public BaseResult querySumReceivePersonList(String orderId) {

try {

BaseResult result = mtOrderSummaryService.querySumReceivePersonList(orderId);

return result;

} catch (Exception e) {

logger.error("查询会议纪要接收人员列表接口调用失败", e);

return new BaseResult(false, "", "查询失败");

}

}

}

package com.xunlu.meeting.controller;

import com.alibaba.fastjson.JSONObject;

import com.xunlu.common.utils.util.StringUtils;

import com.xunlu.digitalpark.commons.vo.BaseResult;

import com.xunlu.meeting.atomService.inter.IMtOfficeOrderService;

import com.xunlu.meeting.atomService.inter.IOfficeManageForWebService;

import com.xunlu.meeting.dto.\*;

import com.xunlu.utils.SecLibUtil;

import com.xunlu.utils.SystemCode;

import com.xunlu.utils.UserUtil;

import io.swagger.annotations.ApiImplicitParam;

import io.swagger.annotations.ApiImplicitParams;

import io.swagger.annotations.ApiOperation;

import lombok.extern.slf4j.Slf4j;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

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

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

import javax.servlet.http.HttpServletRequest;

@CrossOrigin

@Controller

@Slf4j

@RequestMapping("/officeapp/order")

public class AppOfficeOrderController {

private static Logger logger = LoggerFactory.getLogger(AppOfficeOrderController.class);

@Autowired

private IMtOfficeOrderService mtOfficeOrderService;

@Autowired

private IOfficeManageForWebService officeManageForWebService;

/\*\*

\* 常用会议室列表

\*

\* @param request

\* @return

\*/

@RequestMapping(value = "/queryUsedOffice.do")

@ResponseBody

public BaseResult queryUsedOffice(HttpServletRequest request) throws Exception {

log.info("常用会议室列表接口调用");

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.queryUsedOffice(userInfo.getUsername());

return result;

}

/\*\*

\* 所在区域会议室列表

\*

\* @param request

\* @return

\*/

@RequestMapping(value = "/queryAreaOffice.do")

@ResponseBody

public BaseResult queryAreaOffice(HttpServletRequest request) throws Exception {

log.info("所在区域会议室列表接口调用");

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.queryAreaOffice(userInfo.getUsername());

return result;

}

/\*\*

\* APP新增会议订单（锁定会议室）

\*

\* @param request

\* @param lockOfficeDto

\* @return

\*/

@ApiOperation(value = "APP新增会议订单（锁定会议室）接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "officeId", value = "会议室ID", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "officeTitle", value = "会议主题", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "date", name = "orderDate", value = "日期（yyyy-MM-dd）", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "date", name = "startTime", value = "开始时间（HH:mm）", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "date", name = "endTime", value = "结束时间（HH:mm）", required = true, dataType = "String")

})

@PostMapping(value = "/addMeetingOrder.do")

@ResponseBody

public BaseResult addMeetingOrder(HttpServletRequest request, LockOfficeDto lockOfficeDto) {

log.info("APP端锁定会议室接口开始调用，参数[lockOfficeDto={}]", JSONObject.toJSONString(lockOfficeDto));

try {

/\*\*

\* 主题验证

\*/

boolean validateParam = SecLibUtil.validateSecLibHtml(lockOfficeDto.getOfficeTitle());

if (!validateParam) {

return new BaseResult(false, "S10000001", "会议主题验证不通过！");

}

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.lockOffice(lockOfficeDto,userInfo, SystemCode.FromType.APP);

log.info("APP端锁定会议室接口开始调用结束");

return result;

} catch (Exception e) {

log.error("APP端锁定会议失败", e);

return new BaseResult(false, "", "锁定会议失败");

}

}

/\*\*

\* APP端获取预约会议室列表

\*

\* @param parmDto

\* @param sourceType

\* @return

\* @throws Exception

\*/

@ApiOperation(value = "获取预约会议室列表接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "sourceType", value = "来源（初始加载：init\_loading，其他时候可不传）", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "date", value = "日期", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "zone", value = "地区", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "building", value = "楼号", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "minFloor", value = "最小楼层", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "maxFloor", value = "最大楼层", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "device", value = "设备", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "minNum", value = "最小人数", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "maxNum", value = "最大人数", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "startOrderDate", value = "预定开始时间", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "endOrderDate", value = "预定结束时间", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "startOrderTime", value = "会议开始时间", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "endOrderTime", value = "会议结束时间", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "pageSize", value = "分页大小", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "pageNum", value = "当前页", required = true, dataType = "String")})

@PostMapping(value = "/queryOfficeList.do")

@ResponseBody

public BaseResult queryOfficeList(HttpServletRequest request, OfficeSearchParmDto parmDto, String sourceType) throws Exception {

logger.info("获取预约会议室列表接口调用开始，接收参数[parmDto={},sourceType={}]",

JSONObject.toJSONString(parmDto), sourceType);

UserInfo info = UserUtil.getUserInfo(request);

parmDto.setEmpAccountNo(info.getUsername());

if(StringUtils.isNotBlank(parmDto.getDevice())){

parmDto.setDeviceTypes(parmDto.getDevice().split(","));

}

BaseResult result = mtOfficeOrderService.queryOfficeListBySourceType(parmDto, sourceType);

logger.info("获取预约会议室列表接口调用结束");

return result;

}

/\*\*

\* 会议室详情-基本信息

\* @param id

\* @return

\*/

@ApiOperation(value = "APP端会议室信息查询接口")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "id", value = "会议ID", required = true, dataType = "String")})

@PostMapping(value = "/viewOfficeDetailById.do")

@ResponseBody

public BaseResult viewOfficeDetailById(String id) throws Exception {

logger.info("APP端会议室信息查询调用开始，参数[id={}]", id);

BaseResult result = officeManageForWebService.findOfficeById(id);

logger.info("APP端会议室信息查询调用结束");

return result;

}

/\*\*

\* 会议室详情-预约信息

\* @param paramDto

\* @return

\*/

@ApiOperation(value = "APP端会议室信息查询接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "orderId", value = "会议ID", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "officeId", value = "会议室ID", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "date", name = "orderId", value = "日期", required = true, dataType = "String")})

@RequestMapping(value = "/getAppointmentMeetingList.do")

@ResponseBody

public BaseResult getAppointmentMeetingList(QueryOrderParamDto paramDto) {

logger.info("APP端获取会议室已定会议信息列表接口调用开始，参数[paramDto={}]", JSONObject.toJSONString(paramDto));

BaseResult result = mtOfficeOrderService.getAppointmentMeetingList(paramDto);

logger.info("APP端获取会议室已定会议信息列表接口调用结束");

return result;

}

/\*\*

\* APP端获取会议室已定会议信息列表

\*

\* @param paramDto

\* @return

\*/

@ApiOperation(value = "APP端会议室信息查询接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "orderId", value = "会议ID", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "officeId", value = "会议室ID", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "date", name = "orderId", value = "日期", required = true, dataType = "String")})

@PostMapping(value = "/appointmentMeetingList.do")

@ResponseBody

public BaseResult appointmentMeetingList(QueryOrderParamDto paramDto, HttpServletRequest request) {

logger.info("APP端获取会议室已定会议信息列表接口调用开始，参数[paramDto={}]", JSONObject.toJSONString(paramDto));

UserInfo userInfo = UserUtil.getUserInfo(request);

paramDto.setAccountNo(userInfo.getUsername());

BaseResult result = mtOfficeOrderService.queryOfficeOrderByDateAndId(paramDto);

logger.info("APP端获取会议室已定会议信息列表接口调用结束");

return result;

}

/\*\*

\* 获取当前会议的预定结果

\*

\* @param orderId

\* @return

\*/

@ApiOperation(value = "获取当前会议的预定结果接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "orderId", value = "会议ID", required = true, dataType = "String"),

})

@PostMapping(value = "/getOrderResultById.do")

@ResponseBody

public BaseResult getOrderResultById(String orderId, HttpServletRequest request) {

logger.info("获取当前会议的预定结果开始，参数[orderId={}]", orderId);

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.getOrderResultById(orderId, userInfo.getUsername());

logger.info("获取当前会议的预定结果结束");

return result;

}

/\*\*

\* APP端获取已预定会议室的详细信息

\*

\* @param request

\* @param id 会议id

\* @returnid

\*/

@ApiOperation(value = "获取当前会议的预定结果接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "id", value = "会议ID", required = true, dataType = "String"),

})

@PostMapping(value = "/getMeetingDetail.do")

@ResponseBody

public BaseResult getMeetingDetail(HttpServletRequest request, String id) throws Exception {

logger.info("APP端获取已预定会议室的详细信息开始，参数[id={}]", id);

UserInfo userInfo = UserUtil.getUserInfo(request);

BaseResult result = mtOfficeOrderService.getMeetingDetail(id, userInfo.getUsername());

logger.info("APP端获取已预定会议室的详细信息结束");

return result;

}

/\*\*

\* APP端保存会议订单详情接口

\*

\* @param request

\* @param param 数据详情

\* @returnid

\*/

@ApiOperation(value = "获取当前会议的预定结果接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "param", value = "会议详情数据", required = true, dataType = "String"),

})

@PostMapping(value = "/saveOfficeApply.do")

@ResponseBody

public BaseResult saveOfficeApply(HttpServletRequest request, String param) throws Exception {

logger.info("APP端保存会议订单详情接口开始，参数[param={}]", param);

UserInfo userInfo = UserUtil.getUserInfo(request);

OrderSaveParamDto paramDto = JSONObject.parseObject(param, OrderSaveParamDto.class);

if (!SecLibUtil.validateSecLibHtml(paramDto.getTheme())) {

return new BaseResult(false, SystemCode.MEETING\_START, "会议主题不符合规范！");

}

BaseResult result = mtOfficeOrderService.saveOfficeApply(paramDto, userInfo.getUsername(),request.getHeader("code"));

logger.info("APP端保存会议订单详情接口结束");

return result;

}

}

package com.xunlu.meeting.controller;

import com.xunlu.digitalpark.commons.vo.BaseResult;

import com.xunlu.meeting.atomService.inter.IMtOfficeOrderService;

import com.xunlu.meeting.atomService.inter.IOrderVerifyService;

import com.xunlu.meeting.dto.GetMyMeetingListParams;

import com.xunlu.meeting.dto.TodoParmDto;

import com.xunlu.meeting.dto.UserInfo;

import com.xunlu.meeting.dto.meetingEvalParams;

import com.xunlu.utils.UserUtil;

import io.swagger.annotations.ApiImplicitParam;

import io.swagger.annotations.ApiImplicitParams;

import io.swagger.annotations.ApiOperation;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

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

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

import javax.servlet.http.HttpServletRequest;

/\*\*

\* @PackageName:com.xunlu.meeting.controller

\* @ClassName:AppMyMeetingController

\* @Description: APP端我的会议接口类

\*/

@CrossOrigin

@Controller

@RequestMapping("/officeapp/myMeeting")

public class AppMyMeetingController {

private static Logger logger = LoggerFactory.getLogger(AppMyMeetingController.class);

@Autowired

private IMtOfficeOrderService mtOfficeOrderService;

@Autowired

private IOrderVerifyService orderVerifyService;

/\*\*

\* 根据tab类型获取我的会议列表

\*

\* @param request

\* @param params

\* @return

\*/

@ApiOperation(value = "根据tab类型获取我的会议列表接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "flag", value = "tab类型（未完成：wait，已完成：done，待审核：check）", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "date", value = "日期", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "title", value = "搜索内容", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "currIndex", value = "当前页", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "pageSize", value = "分页大小", required = true, dataType = "String")

})

@PostMapping(value = "/myMeetingList.do")

@ResponseBody

public BaseResult getMyMeetingListByTab(HttpServletRequest request,GetMyMeetingListParams params) {

try {

UserInfo userInfo = UserUtil.getUserInfo(request);

// params.setEmpAccountNo(userInfo.getUsername());

params.setEmpAccountNo(userInfo.getUsername());

BaseResult result = mtOfficeOrderService.getMyMeetingListByTab(params);

return result;

} catch (Exception e) {

logger.error("根据tab类型获取我的会议列表获取失败", e);

return new BaseResult(false, "", "我的会议列表获取失败");

}

}

@ApiOperation(value = "会议评价接口")

@PostMapping(value = "/meetingEval.do")

@ResponseBody

public BaseResult meetingEval(HttpServletRequest request, meetingEvalParams params){

try {

UserInfo userInfo = UserUtil.getUserInfo(request);

params.setEmpAccountNo(userInfo.getUsername());

mtOfficeOrderService.newMeetingEval(params);

return new BaseResult(true, "", "会议评价新增成功");

}catch (Exception e){

logger.error("会议评价接口请求失败", e);

return new BaseResult(false, "", "会议评价接口请求失败");

}

}

/\*\*

\* 会议室审核

\*

\* @return

\* @Param

\*\*/

@ApiOperation(value = "会议室审核接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "orderId", value = "会议ID", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "checkResultType", value = "审核结果", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "checkContent", value = "审核内容", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "serverPerson", value = "服务人员", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "itServer", value = "IT技术支持", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "checkType", value = "审核类型", required = true, dataType = "String")

})

@PostMapping(value = "/verifyOrderOperation.do")

@ResponseBody

public BaseResult verifyOrderOperation(HttpServletRequest request, TodoParmDto paramDto) {

logger.debug("APP端会议室审核接口调用开始！");

UserInfo userInfo = UserUtil.getUserInfo(request);

paramDto.setCheckBy(userInfo.getUsername());

BaseResult result = orderVerifyService.verifyOrderOperation(paramDto);

logger.debug("APP端会议室审核接口调用结束");

return result;

}

}

package com.xunlu.meeting.controller;

import cn.hutool.core.util.ObjectUtil;

import com.alibaba.fastjson.JSONObject;

import com.xunlu.common.langx.LangXinTokenApi;

import com.xunlu.common.langx.vo.UserInfo;

import com.xunlu.digitalpark.commons.vo.BaseResult;

import com.xunlu.digitalpark.commons.vo.ObjectResult;

import com.xunlu.sec.uap.util.Constant;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Controller;

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

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

import javax.servlet.http.HttpServletRequest;

@Controller

@RequestMapping("/officeapp/appinfo")

public class AppInfoController {

private static Logger logger = LoggerFactory.getLogger(AppInfoController.class);

/\*\*

\* 获取token

\* @param code

\* @return

\*/

@RequestMapping("/getAppUserInfo.do")

@ResponseBody

public BaseResult getAccessToken(String code, HttpServletRequest request) throws Exception {

//根据code获取当前用户的接口

logger.info("接口获取APP请求码[code={}]", code);

LangXinTokenApi tokenApi=new LangXinTokenApi();

//获取用户使用信息

UserInfo appUserInfo = tokenApi.getUserInfo(code);

if(ObjectUtil.isNull(appUserInfo)){

logger.warn("获取用户信息失败，code,{}", code);

return new BaseResult(false, "", "获取用户信息失败");

}

//拿到当前用户信息，放到session

request.getSession().setAttribute(Constant.SESSION\_USER,appUserInfo);

ObjectResult result = new ObjectResult();

result.setData(appUserInfo);

return result;

}

}

package com.xunlu.meeting.dto;

/\*\*

\* @PackageName:com.xunlu.meeting.dto

\* @ClassName:MeetingStatisDto

\* @Description:会议信息统计传输类

\*/

public class MeetingStatisDto {

//签到中，距离会议开始时间：10:00

//会议进行中 会议已开始：01:12 会议剩余时长：00:18

//会议已完成,参会人数：5人,签到率：90%,会议预约时长：1h,会议实际时长：1h

/\*\*

\* 距离会议开始时间：10:00（签到中）

\*/

private String beforeStartTime;

/\*\*

\* 会议已开始：01:12（会议进行中）

\*/

private String afterStartTime;

/\*\*

\* 会议剩余时长：00:18（会议进行中）

\*/

private String meetingLeftTime;

/\*\*

\* 参会人数

\*/

private Integer participantNum;

/\*\*

\* 签到率

\*/

private String signPercent;

/\*\*

\* 会议预约时长

\*/

private String appointTime;

/\*\*

\* 会议实际时长

\*/

private String meetingRealTime;

public MeetingStatisDto() {

}

public String getBeforeStartTime() {

return beforeStartTime;

}

public void setBeforeStartTime(String beforeStartTime) {

this.beforeStartTime = beforeStartTime;

}

public String getAfterStartTime() {

return afterStartTime;

}

public void setAfterStartTime(String afterStartTime) {

this.afterStartTime = afterStartTime;

}

public String getMeetingLeftTime() {

return meetingLeftTime;

}

public void setMeetingLeftTime(String meetingLeftTime) {

this.meetingLeftTime = meetingLeftTime;

}

public Integer getParticipantNum() {

return participantNum;

}

public void setParticipantNum(Integer participantNum) {

this.participantNum = participantNum;

}

public String getSignPercent() {

return signPercent;

}

public void setSignPercent(String signPercent) {

this.signPercent = signPercent;

}

public String getAppointTime() {

return appointTime;

}

public void setAppointTime(String appointTime) {

this.appointTime = appointTime;

}

public String getMeetingRealTime() {

return meetingRealTime;

}

public void setMeetingRealTime(String meetingRealTime) {

this.meetingRealTime = meetingRealTime;

}

}

package com.xunlu.meeting.dto;

import com.xunlu.common.utils.util.StringUtils;

import com.xunlu.utils.MD5Util;

import java.text.SimpleDateFormat;

import java.util.Date;

/\*\*

\* @PackageName:com.xunlu.meeting.dto

\* @ClassName:MtOfficeOrderDto

\* @Description:

\*/

public class MtOfficeOrderDto {

private String id;

private String officeId;

private String officeName;

private String officeCode;

private String officeTitle;

private String empNoOrder;

private String accountNoOrder;

private Date orderDate;

private String startTime;

private String endTime;

private String type;

private String officeNotice;

private String officeServer;

private Date createTime;

private String createTimeStr;

private String createBy;

private Date modifyTime;

private String modifyBy;

private String imageUrl;

private String fromType;

private String videoCode;

private String checkStatus;

private String checkStatusName;

private String status;

private String masterName;

private String titleHide;

private String empNameOrder;

private Integer totalNum;

private String orderDateStr;

private String serverStatus;

private String needCheck;

private String serverPerson;

private String serverPersonName;

private String itServerPerson;

private String itServerPersonName;

private String typeAboutMe;

private String officeManageServer;

private String recordingDeviceId;

private String serialNumber;

private String meetingAddress;

/\*\*

\* 信息填写状态 0-未完善 1-已完善（重点、普通会议室都用这个字段）

\*/

private Integer infoStatus;

/\*\*

\* 预定会议开始时间，辅助字段（预定时保存，提前开始会议、提前结束会议、会议续时都不修改这两个字段）

\*/

private String bookStartTime;

/\*\*

\* 预定会议结束时间

\*/

private String bookEndTime;

/\*\*

\* 会议预定订单编号，比如20190306001

\*/

private String orderNo;

/\*\*

\* 是否签到 会议

\*/

private int needSign;

/\*\*

\* 会议签到时间 15

\*/

private String signAdvanceMinutes;

/\*\*

\* 会议室规则 签到

\*/

private String ruleNeedSign;

/\*\*

\* 会议室规则 15,30 逗号分开

\*/

private String ruleInAdvanceMinutes;

/\*\*

\* orderId

\*/

private String minWOrderId;

private String webSocketToTransUrl;

private String webSocketToPlayUrl;

private Integer orderType;

private String videoOrderId;

/\*\*

\* 是否需要中控

\*/

private String centerControl;

/\*\*

\* 中控模式

\*/

private String controlModel;

// @Value(value = "${websocket.realtime.trans.audio.url}")

private String realtime\_trans\_audio\_url;

// @Value(value = "${websocket.realtime.play.audio.url}")

private String realtime\_play\_audio\_url;

public String getVideoOrderId() {

return videoOrderId;

}

public void setVideoOrderId(String videoOrderId) {

this.videoOrderId = videoOrderId;

}

public Integer getOrderType() {

return orderType;

}

public void setOrderType(Integer orderType) {

this.orderType = orderType;

}

public String getWebSocketToTransUrl() {

//获取websocketurl获取实时转写路径

if (StringUtils.isNotBlank(id)) {

return realtime\_trans\_audio\_url + MD5Util.encry(id) + "/" + id;

}

return "";

}

public void setWebSocketToTransUrl(String webSocketToTransUrl) {

this.webSocketToTransUrl = webSocketToTransUrl;

}

public String getCreateTimeStr() {

return createTimeStr;

}

public void setCreateTimeStr(String createTimeStr) {

this.createTimeStr = createTimeStr;

}

public String getWebSocketToPlayUrl() {

//获取websocketurl获取实时转写路径

if (StringUtils.isNotBlank(id)) {

return realtime\_play\_audio\_url + MD5Util.encry(id) + "/" + id;

}

return "";

}

public void setWebSocketToPlayUrl(String webSocketToPlayUrl) {

this.webSocketToPlayUrl = webSocketToPlayUrl;

}

public String getMinWOrderId() {

if (StringUtils.isNotBlank(id)) {

return MD5Util.encry(id);

}

return "";

}

public String getRecordingDeviceId() {

return recordingDeviceId;

}

public void setRecordingDeviceId(String recordingDeviceId) {

this.recordingDeviceId = recordingDeviceId;

}

public void setMinWOrderId(String minWOrderId) {

this.minWOrderId = minWOrderId;

}

public String getCheckStatusName() {

return checkStatusName;

}

public void setCheckStatusName(String checkStatusName) {

this.checkStatusName = checkStatusName;

}

public String getRuleInAdvanceMinutes() {

return ruleInAdvanceMinutes;

}

public void setRuleInAdvanceMinutes(String ruleInAdvanceMinutes) {

this.ruleInAdvanceMinutes = ruleInAdvanceMinutes;

}

public int getNeedSign() {

return needSign;

}

public void setNeedSign(int needSign) {

this.needSign = needSign;

}

public String getRuleNeedSign() {

return ruleNeedSign;

}

public void setRuleNeedSign(String ruleNeedSign) {

this.ruleNeedSign = ruleNeedSign;

}

public String getSignAdvanceMinutes() {

return signAdvanceMinutes;

}

public void setSignAdvanceMinutes(String signAdvanceMinutes) {

this.signAdvanceMinutes = signAdvanceMinutes;

}

public String getTypeAboutMe() {

return typeAboutMe;

}

public void setTypeAboutMe(String typeAboutMe) {

this.typeAboutMe = typeAboutMe;

}

public String getServerPerson() {

return serverPerson;

}

public void setServerPerson(String serverPerson) {

this.serverPerson = serverPerson;

}

public String getServerPersonName() {

return serverPersonName;

}

public void setServerPersonName(String serverPersonName) {

this.serverPersonName = serverPersonName;

}

public String getItServerPerson() {

return itServerPerson;

}

public void setItServerPerson(String itServerPerson) {

this.itServerPerson = itServerPerson;

}

public String getItServerPersonName() {

return itServerPersonName;

}

public void setItServerPersonName(String itServerPersonName) {

this.itServerPersonName = itServerPersonName;

}

public String getNeedCheck() {

return needCheck;

}

public void setNeedCheck(String needCheck) {

this.needCheck = needCheck;

}

public String getServerStatus() {

return serverStatus;

}

public void setServerStatus(String serverStatus) {

this.serverStatus = serverStatus;

}

public String getOrderDateStr() {

if (null != orderDate) {

return new SimpleDateFormat("yyyy-MM-dd").format(orderDate);

}

return orderDateStr;

}

public void setOrderDateStr(String orderDateStr) {

this.orderDateStr = orderDateStr;

}

public Integer getTotalNum() {

return totalNum;

}

public void setTotalNum(Integer totalNum) {

this.totalNum = totalNum;

}

public String getOfficeName() {

return officeName;

}

private Integer msgSendStatus;

public Integer getMsgSendStatus() {

return msgSendStatus;

}

public void setOfficeName(String officeName) {

this.officeName = officeName;

}

public String getTitleHide() {

return titleHide;

}

public void setTitleHide(String titleHide) {

this.titleHide = titleHide;

}

public String getMasterName() {

return masterName;

}

public void setMasterName(String masterName) {

this.masterName = masterName;

}

public void setMsgSendStatus(Integer msgSendStatus) {

this.msgSendStatus = msgSendStatus;

}

public String getEmpNameOrder() {

return empNameOrder;

}

public void setEmpNameOrder(String empNameOrder) {

this.empNameOrder = empNameOrder;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getOfficeId() {

return officeId;

}

public void setOfficeId(String officeId) {

this.officeId = officeId;

}

public String getOfficeTitle() {

return officeTitle;

}

public void setOfficeTitle(String officeTitle) {

this.officeTitle = officeTitle;

}

public String getEmpNoOrder() {

return empNoOrder;

}

public void setEmpNoOrder(String empNoOrder) {

this.empNoOrder = empNoOrder;

}

public Date getOrderDate() {

return orderDate;

}

public void setOrderDate(Date orderDate) {

this.orderDate = orderDate;

}

public String getStartTime() {

return startTime;

}

public void setStartTime(String startTime) {

this.startTime = startTime;

}

public String getEndTime() {

return endTime;

}

public void setEndTime(String endTime) {

this.endTime = endTime;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public String getOfficeNotice() {

return officeNotice;

}

public void setOfficeNotice(String officeNotice) {

this.officeNotice = officeNotice;

}

public String getOfficeServer() {

return officeServer;

}

public void setOfficeServer(String officeServer) {

this.officeServer = officeServer;

}

public Date getCreateTime() {

return createTime;

}

public void setCreateTime(Date createTime) {

this.createTime = createTime;

}

public String getCreateBy() {

return createBy;

}

public void setCreateBy(String createBy) {

this.createBy = createBy;

}

public Date getModifyTime() {

return modifyTime;

}

public void setModifyTime(Date modifyTime) {

this.modifyTime = modifyTime;

}

public String getModifyBy() {

return modifyBy;

}

public void setModifyBy(String modifyBy) {

this.modifyBy = modifyBy;

}

public String getFromType() {

return fromType;

}

public void setFromType(String fromType) {

this.fromType = fromType;

}

public String getImageUrl() {

return imageUrl;

}

public void setImageUrl(String imageUrl) {

this.imageUrl = imageUrl;

}

public String getVideoCode() {

return videoCode;

}

public void setVideoCode(String videoCode) {

this.videoCode = videoCode;

}

public String getCheckStatus() {

return checkStatus;

}

public void setCheckStatus(String checkStatus) {

this.checkStatus = checkStatus;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

public String getAccountNoOrder() {

return accountNoOrder;

}

public void setAccountNoOrder(String accountNoOrder) {

this.accountNoOrder = accountNoOrder;

}

public String getOfficeCode() {

return officeCode;

}

public void setOfficeCode(String officeCode) {

this.officeCode = officeCode;

}

public Integer getInfoStatus() {

return infoStatus;

}

public void setInfoStatus(Integer infoStatus) {

this.infoStatus = infoStatus;

}

public String getBookStartTime() {

return bookStartTime;

}

public void setBookStartTime(String bookStartTime) {

this.bookStartTime = bookStartTime;

}

public String getBookEndTime() {

return bookEndTime;

}

public void setBookEndTime(String bookEndTime) {

this.bookEndTime = bookEndTime;

}

public String getOrderNo() {

return orderNo;

}

public void setOrderNo(String orderNo) {

this.orderNo = orderNo;

}

public String getOfficeManageServer() {

return officeManageServer;

}

public void setOfficeManageServer(String officeManageServer) {

this.officeManageServer = officeManageServer;

}

public String getSerialNumber() {

return serialNumber;

}

public void setSerialNumber(String serialNumber) {

this.serialNumber = serialNumber;

}

public String getMeetingAddress() {

return meetingAddress;

}

public void setMeetingAddress(String meetingAddress) {

this.meetingAddress = meetingAddress;

}

public String getControlModel() {

return controlModel;

}

public void setControlModel(String controlModel) {

this.controlModel = controlModel;

}

public String getCenterControl() {

return centerControl;

}

public void setCenterControl(String centerControl) {

this.centerControl = centerControl;

}

}

package com.xunlu.meeting.controller;

import com.alibaba.fastjson.JSONObject;

import com.xunlu.common.utils.util.StringUtils;

import com.xunlu.digitalpark.commons.vo.BaseResult;

import com.xunlu.digitalpark.commons.vo.MultiResult;

import com.xunlu.meeting.atomService.inter.MtOfficeOrderService;

import com.xunlu.meeting.atomService.inter.MtOfficeScreenService;

import com.xunlu.meeting.dto.\*;

import com.xunlu.meeting.utils.SystemCode;

import com.xunlu.meeting.utils.UserUtil;

import com.xunlu.meeting.utils.fastdfs.FastDFSClient;

import io.swagger.annotations.ApiImplicitParam;

import io.swagger.annotations.ApiImplicitParams;

import io.swagger.annotations.ApiOperation;

import lombok.extern.slf4j.Slf4j;

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

import org.springframework.stereotype.Controller;

import org.springframework.validation.BindingResult;

import org.springframework.validation.FieldError;

import org.springframework.validation.annotation.Validated;

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

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

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

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;

import java.util.HashMap;

import java.util.Map;

/\*\*

\* 描述:pad端预约会议接口类

\*

\*/

@Slf4j

@Controller

@RequestMapping("/officePad/order")

public class PadOfficeOrderController {

@Autowired

private MtOfficeOrderService mtOfficeOrderService;

@Autowired

private MtOfficeScreenService mtOfficeScreenService;

@Autowired

private FastDFSClient fastDFSClient;

/\*\*

\* APP端获取预约会议室列表

\*

\* @param parmDto

\* @param sourceType

\* @return

\* @throws Exception

\*/

@ApiOperation(value = "获取预约会议室列表接口")

@ApiImplicitParams({

@ApiImplicitParam(paramType = "query", name = "sourceType", value = "来源（初始加载：init\_loading，其他时候可不传）", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "date", value = "日期", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "zone", value = "地区", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "building", value = "楼号", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "minFloor", value = "最小楼层", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "maxFloor", value = "最大楼层", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "device", value = "设备", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "minNum", value = "最小人数", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "maxNum", value = "最大人数", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "startOrderDate", value = "预定开始时间", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "endOrderDate", value = "预定结束时间", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "startOrderTime", value = "会议开始时间", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "endOrderTime", value = "会议结束时间", required = false, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "pageSize", value = "分页大小", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "pageNum", value = "当前页", required = true, dataType = "String")})

@PostMapping(value = "/queryOfficeList.do")

@ResponseBody

public BaseResult queryOfficeList(HttpServletRequest request, OfficeSearchParmDto parmDto, String sourceType) throws Exception {

log.info("获取预约会议室列表接口调用开始，接收参数[parmDto={},sourceType={}]",

JSONObject.toJSONString(parmDto), sourceType);

UserInfo info = UserUtil.getUserInfo(request);

parmDto.setEmpAccountNo(info.getUsername());

if(StringUtils.isNotBlank(parmDto.getDevice())){

parmDto.setDeviceTypes(parmDto.getDevice().split(","));

}

BaseResult result = mtOfficeOrderService.queryOfficeListBySourceType(parmDto, sourceType);

log.info("获取预约会议室列表接口调用结束");

return result;

}

/\*\*

\* 获取会议室已定会议信息列表

\*

\* @param paramDto

\* @return

\*/

@RequestMapping(value = "/appointmentMeetingList.do")

@ResponseBody

public BaseResult appointmentMeetingList(QueryOrderParamDto paramDto, HttpServletRequest request) {

log.info("获取会议室已定会议信息列表接口调用开始，参数[paramDto={}]", JSONObject.toJSONString(paramDto));

UserInfo userInfo = UserUtil.getUserInfo(request);

paramDto.setAccountNo(userInfo.getUsername());

BaseResult result = mtOfficeOrderService.queryOfficeOrderByDateAndId(paramDto);

return result;

}

/\*\*

\* 临时会议预定接口

\*

\* @param imageData

\* @param record

\* @return

\*/

@ApiOperation(value = "临时会议预定接口")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "imageData", value = "人脸照片", required = true, dataType = "File")})

@PostMapping(value = "/addTemporaryMeeting.do")

@ResponseBody

public BaseResult addTemporaryMeeting(MultipartFile imageData, @Validated TemporaryMeeting record,

BindingResult bindingResult) throws Exception {

long startTime = System.currentTimeMillis();

log.debug("预定临时会议接口调用开始！");

if (bindingResult.hasErrors()) {

Map<Object, Object> data = new HashMap<Object, Object>();

for (FieldError error : bindingResult.getFieldErrors()) {

data.put(error.getField(), error.getDefaultMessage());

}

MultiResult<String> result = new MultiResult<String>(false, SystemCode.ERROR);

result.setData(data);

log.debug("预定临时会议接收参数验证不成功！接口返回结果为：[" + result.toString() + "],耗时："

+ (System.currentTimeMillis() - startTime) + "ms");

return result;

}

if (null != imageData) {

String imagePath = fastDFSClient.upload(imageData);

record.setImageUrl(imagePath);

}

BaseResult result = mtOfficeScreenService.addTemporaryMeeting(record);

log.debug("预定临时会议接口调用结束，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

}

/\*\*

\* 根据padIp获取对应会议室的规则信息

\*

\* @param padIp

\* @return

\*/

@ApiOperation(value = "根据padIp获取对应会议室的规则信息")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "padIp", value = "padIp", required = true, dataType = "String")})

@PostMapping(value = "/getRuleInfoByPadIp.do")

@ResponseBody

public BaseResult getRuleInfoByPadIp(String padIp) {

long startTime = System.currentTimeMillis();

log.info("根据padIp获取对应会议室的规则信息接口开始，接收参数：[padIp=" + padIp + "]");

BaseResult result = mtOfficeScreenService.getRuleInfoByPadIp(padIp);

log.info("根据padIp获取对应会议室的规则信息接口结束，消耗时间为：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

}

/\*\*

\* 根据padIp查询关联的会议室信息接口

\*

\* @param padIp

\* @return

\*/

@ApiOperation(value = "根据padIp查询关联的会议室信息接口")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "padIp", value = "padIp", required = true, dataType = "String")})

@PostMapping(value = "/getOfficeInfoByPadIp.do")

@ResponseBody

public BaseResult getOfficeInfoByPadIp(String padIp, HttpServletRequest request) {

request.getRemoteHost();

request.getHeader("X-Real-IP");

long startTime = System.currentTimeMillis();

log.info("根据padIp查询关联的会议室信息接口开始，接收参数：[padIp=" + padIp + "]");

BaseResult result = mtOfficeScreenService.getOfficeInfoByPadIp(padIp);

log.info("根据padIp查询关联的会议室信息接口结束，消耗时间为：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

}

/\*\*

\* 根据padIp获取当天会议室预定情况列表

\*

\* @param padIp

\* @param currTime

\* @return

\*/

@ApiOperation(value = "根据padIp获取当天会议室预定情况列表")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "padIp", value = "padIp", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "currTime", value = "当前pad时间", required = false, dataType = "String")})

@PostMapping(value = "/getOfficeOrderLists.do")

@ResponseBody

public BaseResult getOfficeOrderLists(String padIp, String currTime) {

long startTime = System.currentTimeMillis();

log.info("根据padIp获取当天会议室预定情况列表接口开始，接收参数：[padIp=" + padIp + ",currTime=" + currTime + "]");

BaseResult result = mtOfficeScreenService.getOfficeOrderLists(padIp, currTime);

log.info("根据padIp获取当天会议室预定情况列表接口结束，消耗时间：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

}

/\*\*

\* 签到、签退记录保存

\*

\* @param imageData

\* @param record

\* @return

\* @throws Exception

\*/

@ApiOperation(value = "签到、签退记录保存")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "imageData", value = "签到、签退照片", required = false, dataType = "File")})

@PostMapping(value = "/meetingSignInOrOut.action")

@ResponseBody

public BaseResult meetingSignInOrOut(MultipartFile imageData, @Validated MtSignRecord record,

BindingResult bindingResult) throws Exception {

long startTime = System.currentTimeMillis();

log.debug("签到、签退记录保存接口调用开始！接收参数[record=" + record.toString() + "]");

if (bindingResult.hasErrors()) {

Map<Object, Object> data = new HashMap<Object, Object>();

for (FieldError error : bindingResult.getFieldErrors()) {

data.put(error.getField(), error.getDefaultMessage());

}

MultiResult<String> result = new MultiResult<String>(false, SystemCode.ERROR);

result.setData(data);

log.debug("签到数据验证不成功！接口返回结果为：[" + result.toString() + "],耗时："

+ (System.currentTimeMillis() - startTime) + "ms");

return result;

}

if (null != imageData) {

String imagePath = fastDFSClient.upload(imageData);

record.setImageUrl(imagePath);

}

BaseResult result = mtOfficeScreenService.meetingSignInOrOut(record);

log.info("签到、签退记录保存接口调用结束，消耗时间为：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

}

/\*\*

\* 会议时间结束触发会议终止接口

\*

\* @param orderId

\* @return

\*/

@ApiOperation(value = "会议时间结束触发会议终止接口")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "orderId", value = "会议ID", required = true, dataType = "String")})

@PostMapping(value = "/timeOutToStopMeeting.do")

@ResponseBody

public BaseResult timeOutToStopMeeting(String orderId) {

long startTime = System.currentTimeMillis();

log.debug("会议时间结束触发会议终止接口调用开始！");

BaseResult result = mtOfficeScreenService.timeOutToStopMeeting(orderId);

log.debug("会议时间结束触发会议终止接口调用结束，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

}

/\*\*

\* 会议续时接口

\*

\* @param imageData

\* @param record

\* @return

\*/

@ApiOperation(value = "会议续时接口")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "imageData", value = "人脸照片", required = true, dataType = "File")})

@PostMapping(value = "/meetingAddTime.do")

@ResponseBody

public BaseResult meetingAddTime(MultipartFile imageData, @Validated MtOrderAddTime record,

BindingResult bindingResult) throws Exception {

long startTime = System.currentTimeMillis();

log.debug("会议续时接口调用开始！");

if (bindingResult.hasErrors()) {

Map<Object, Object> data = new HashMap<Object, Object>();

for (FieldError error : bindingResult.getFieldErrors()) {

data.put(error.getField(), error.getDefaultMessage());

}

MultiResult<String> result = new MultiResult<String>(false, SystemCode.ERROR);

result.setData(data);

log.debug("参数验证不成功！接口返回结果为：[" + result.toString() + "],耗时："

+ (System.currentTimeMillis() - startTime) + "ms");

return result;

}

if (null != imageData) {

String imagePath = fastDFSClient.upload(imageData);

record.setImageUrl(imagePath);

}

BaseResult result = mtOfficeScreenService.meetingAddTime(record);

log.debug("会议续时接口调用结束，耗时：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

}

@ApiOperation(value = "协同心跳，根据padIp获取当天会议室预定情况列表")

@ApiImplicitParams({@ApiImplicitParam(paramType = "query", name = "padIp", value = "padIp", required = true, dataType = "String"),

@ApiImplicitParam(paramType = "query", name = "officeId", value = "officeId", required = true, dataType = "string"),

@ApiImplicitParam(paramType = "query", name = "currTime", value = "当前pad时间", required = false, dataType = "String")})

@PostMapping(value = "/getOfficeOrderListsWithHeartBeat.do")

@ResponseBody

public BaseResult getOfficeOrderListsWithHeartBeat(String padIp, String officeId, String currTime) {

long startTime = System.currentTimeMillis();

log.info("协同心跳，根据padIp获取当天会议室预定情况列表接口开始，接收参数：[padIp=" + padIp + ",currTime=" + currTime + "]");

BaseResult result = mtOfficeScreenService.getOfficeOrderListsWithHeartBeat(padIp, officeId, currTime);

log.info("协同心跳，根据padIp获取当天会议室预定情况列表接口结束，消耗时间：" + (System.currentTimeMillis() - startTime) + "ms");

return result;

}

}