# BaseWebController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/10 8:51

\*/

@RestController

public class BaseWebController {

@Autowired

ISysUserService sysUserService;

@Autowired

ISysOrgService sysOrgService;

@Autowired

ISysUserOrgService sysUserOrgService;

@Autowired

ISysFlowRoleService sysFlowRoleService;

public SysUser getUserDetail() {

return TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

}

public String getTenantId() {

return TenantContextHolder.getContext().getTenantContextInfo().getSysUser().getTenantId();

}

/\*\*

\* 获取授权组织

\*

\* @return

\*/

public List<String> getAuthSysOrgIds() {

//查询当前组织以及以下

List<String> authSysOrgIds = sysUserService.findAuthSysOrgIds(getUserDetail().getSysOrgId());

return authSysOrgIds;

}

/\*\*

\* 获取当前用户组织和

\*

\* @return

\*/

public List<String> getOrgIds() {

SysUser sysUser = getUserDetail();

List<String> sysOrgIds = sysUserOrgService.list(new QueryWrapper<SysUserOrg>().lambda().eq(SysUserOrg::getSysUserId, sysUser.getId()).select(SysUserOrg::getSysOrgId)).stream().map(SysUserOrg::getSysOrgId).collect(Collectors.toList());

if (!sysOrgIds.isEmpty()) {

sysOrgIds.add(sysUser.getSysOrgId());

return sysOrgIds;

} else {

return Collections.singletonList(sysUser.getSysOrgId());

}

}

/\*\*

\* 获取主流程角色

\*

\* @return

\*/

public List<SysFlowRole> getMainFlowRoles() {

SysUser sysUser = getUserDetail();

//获取主组织中路程角色编码

return sysUserService.findFlowRoleCodes(sysUser.getId());

}

/\*\*

\* 获取兼职流程角色

\*

\* @return

\*/

public List<SysUserOrg> getSlaveFlowRoles() {

SysUser sysUser = getUserDetail();

//获取兼职

List<SysUserOrg> sysUserOrgs = sysUserOrgService.list(new QueryWrapper<SysUserOrg>().lambda().eq(SysUserOrg::getSysUserId, sysUser.getId()));

if (sysUserOrgs == null) {

sysUserOrgs = new ArrayList<>();

}

sysUserOrgs.forEach(sysUserOrg -> {

if (sysUserOrg.getSysFlowRoleId() != null && !sysUserOrg.getSysFlowRoleId().isEmpty()) {

sysUserOrg.setSysFlowRoles(sysFlowRoleService.list(new QueryWrapper<SysFlowRole>().lambda().in(SysFlowRole::getId, sysUserOrg.getSysFlowRoleId())));

}

});

return sysUserOrgs;

}

public Set<String> getAllFlowRoleCodes() {

Set<String> mainFlowRoleCodes = getMainFlowRoles().stream().map(SysFlowRole::getCode).collect(Collectors.toSet());

Set<String> slaveFlowRoleCodes = new HashSet<>();

getSlaveFlowRoles().forEach(sysUserOrg -> {

if (sysUserOrg.getSysFlowRoles() != null && !sysUserOrg.getSysFlowRoles().isEmpty()) {

slaveFlowRoleCodes.addAll(sysUserOrg.getSysFlowRoles().stream().map(SysFlowRole::getCode).collect(Collectors.toSet()));

}

});

return CollUtil.unionDistinct(mainFlowRoleCodes,slaveFlowRoleCodes);

}

//获取全部的组织包括兼职组织

public Set<String> getAllOrgIds(){

SysUser sysUser = getUserDetail();

Set<String> orgIds= sysUserOrgService.list(new QueryWrapper<SysUserOrg>().lambda()

.select(SysUserOrg::getSysOrgId)

.eq(SysUserOrg::getSysUserId,sysUser.getId())).stream().map(SysUserOrg::getSysOrgId).collect(Collectors.toSet());

orgIds.add(sysUser.getSysOrgId());

return orgIds;

}

public void checkFileSize(MultipartFile file){

if (file == null) return;

// 检测文件大小 1048576 == 1MB

if (file.getSize() > 52428800) {

throw new ServiceException("导入数据不能大于50M");

}

}

}

# LoginController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/22 12:37

\*/

@RestController

@RequestMapping("/user")

@TenantIgnore

@Api(tags = "登录")

public class LoginController {

private static final Logger log = LoggerFactory.getLogger(LoginController.class);

@Autowired

ISysUserService sysUserService;

@Resource

private RedisUtil redisUtil;

@Resource

private WechatUtil wechatUtil;

@PostMapping("login")

@ApiOperation("登录接口")

public SaResult saResult(@Validated @RequestBody LoginDTO loginDTO) {

SysUser sysUser = sysUserService.login(loginDTO.getUsername(), loginDTO.getPassword());

if (sysUser != null) {

StpUtil.login(sysUser.getUsername());

return SaResult.ok().setData(StpUtil.getTokenInfo().tokenValue);

}

return SaResult.error("登录失败,请检查用户密码");

}

@GetMapping("wechat/login")

@ApiOperation(value = "微信登录接口", notes = "<a href=\"https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/getPhoneNumber.html\">手机号获取凭证</a>")

@ApiImplicitParam(name = "code", value = "手机号获取凭证", required = true)

public SaResult wechatLogin(@RequestParam(name = "code") String code) {

String phone;

PhoneInfoResponse info = wechatUtil.getPhoneNumber(code);

log.info("获取手机号码信息：{}", info.toString());

switch (info.getErrcode()) {

case -1:

throw new ServiceException("微信系统繁忙，请稍候再试");

case 40029, 47001:

throw new ServiceException("js\_code【" + code + "】无效,请联系系统管理员");

case 45009:

throw new ServiceException("获取手机号码超过调用额度请联系系统管理员解决。(" + info.getErrmsg() + ")");

case 40001:

// access\_token 失效或者不是最新的，因为我们在系统外面重新获取了 access\_token，导致缓存失效需要手动触发重新获取

String tokenKey = SystemConstant.REDIS\_KEY\_PREFIX + "access\_token";

redisUtil.deleteObject(tokenKey);

info = wechatUtil.getPhoneNumber(code);

break;

case 0:

break;

default:

throw new ServiceException("获取手机号码失败请联系系统管理员解决。(" + info.getErrcode() + "," + info.getErrmsg() + ")");

}

if (info.getErrcode() != 0) {

log.error("获取手机号码失败：{}", info.toString());

throw new ServiceException("获取手机号码失败请联系系统管理员解决(" + info.getErrcode() + "," + info.getErrmsg() + ")");

}

phone = info.getPhoneInfo().getPurePhoneNumber();

if (StringUtils.isBlank(phone)) {

throw new ServiceException("从微信获取手机号码失败！");

}

SysUser sysUser = sysUserService.findSysUserByPhoneNumber(phone);

// 使用手机号码查找对应的用户然后登录

if (sysUser != null) {

StpUtil.login(sysUser.getUsername());

return SaResult.ok().setData(StpUtil.getTokenInfo().tokenValue);

} else {

return SaResult.error("登录失败,请绑定用户数据，该用户数据不唯一");

}

}

}

# device

## DeviceAssetInfoRecordController.java

/\*\*

\* 设备资产信息记录

\*

\* @author wanggang

\* @date 2023/5/6 14:46

\*/

@RestController

@RequestMapping("/DeviceAssetInfoRecord")

@RequiredArgsConstructor

@Api(tags = "设备资产信息记录接口")

public class DeviceAssetInfoRecordController extends BaseWebController {

private final IDeviceAssetInfoRecordService iDeviceAssetInfoRecordService;

@GetMapping("/selectDeviceAssetInfoRecordPage")

@ApiOperation("分页查询设备资产信息记录接口")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceAssetInfoRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceAssetInfoRecordPage(@ApiParam("设备资产信息记录查询条件实体类") DeviceAssetInfoRecordSelectDTO deviceAssetInfoRecordSelectDTO) {

Page<DeviceAssetInfoRecordVO> deviceAssetInfoRecordPage = iDeviceAssetInfoRecordService.selectDeviceAssetInfoRecordPage(deviceAssetInfoRecordSelectDTO);

return SaResult.ok().setData(deviceAssetInfoRecordPage);

}

@GetMapping("/selectDeviceAssetInfoRecordById")

@ApiOperation("根据资产信息ID查询设备资产信息记录数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceAssetInfoRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceAssetInfoRecordById(@ApiParam("资产信息ID") @RequestParam("id") String id) {

DeviceAssetInfoRecordVO deviceAssetInfoRecordVO = iDeviceAssetInfoRecordService.selectDeviceAssetInfoRecordById(id);

return SaResult.ok().setData(deviceAssetInfoRecordVO);

}

@PostMapping("/addDeviceAssetInfoRecord")

@ApiOperation("新增设备资产信息记录")

public SaResult addDeviceAssetInfoRecord(@ApiParam("设备资产信息记录DTO") @RequestBody @Valid DeviceAssetInfoRecordDTO deviceAssetInfoRecordDTO) {

return iDeviceAssetInfoRecordService.addDeviceAssetInfoRecord(deviceAssetInfoRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceAssetInfoRecord")

@ApiOperation("修改设备资产信息记录")

public SaResult updateDeviceAssetInfoRecord(@ApiParam("设备资产信息记录DTO") @RequestBody @Valid DeviceAssetInfoRecordDTO deviceAssetInfoRecordDTO) {

return iDeviceAssetInfoRecordService.updateDeviceAssetInfoRecord(deviceAssetInfoRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceAssetInfoRecord")

@ApiOperation("逻辑删除设备资产信息记录")

public SaResult deleteDeviceAssetInfoRecord(@ApiParam("设备资产信息记录id") @RequestParam("id") String id) {

return iDeviceAssetInfoRecordService.deleteDeviceAssetInfoRecord(id) ? SaResult.ok() : SaResult.error();

}

}

## DeviceAuthorizationController.java

/\*\*

\* 设备授权记录

\*

\* @author wanggang

\* @date 2023/6/9 9:46

\*/

@RestController

@RequestMapping("/deviceAuthorization")

@RequiredArgsConstructor

@Api(tags = "设备授权接口")

public class DeviceAuthorizationController extends BaseWebController {

private final IDeviceAuthorizationService iDeviceAuthorizationService;

@GetMapping("/selectDeviceAuthorizationPage")

@ApiOperation("分页查询设备授权")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceAuthorizationVO.class, message = "查询成功")

})

public SaResult selectDeviceAuthorizationPage(@ApiParam("设备授权查询条件实体类") DeviceAuthorizationSelectDTO deviceAuthorizationSelectDTO) {

Page<DeviceAuthorizationVO> deviceAuthorizationPage = iDeviceAuthorizationService.selectDeviceAuthorizationPage(deviceAuthorizationSelectDTO);

return SaResult.ok().setData(deviceAuthorizationPage);

}

@GetMapping("/selectDeviceAuthorizationById")

@ApiOperation("根据设备授权ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceAuthorizationVO.class, message = "查询成功")

})

public SaResult selectDeviceAuthorizationById(@ApiParam("设备授权ID") @RequestParam("id") String id) {

DeviceAuthorizationVO deviceAuthorizationVO = iDeviceAuthorizationService.selectDeviceAuthorizationById(id);

return SaResult.ok().setData(deviceAuthorizationVO);

}

@PostMapping("/addDeviceAuthorizationBatch")

@ApiOperation("批量新增设备授权")

public SaResult addDeviceAuthorizationBatch(@ApiParam("设备授权DTO") @RequestBody List<DeviceAuthorizationDTO> deviceAuthorizationDTOList) {

if (CollUtil.isEmpty(deviceAuthorizationDTOList)) {

throw new ParameterException("新增参数缺失");

}

return iDeviceAuthorizationService.addDeviceAuthorizationBatch(deviceAuthorizationDTOList) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceAuthorization")

@ApiOperation("修改设备授权")

public SaResult updateDeviceAuthorization(@ApiParam("设备授权DTO") @RequestBody @Valid DeviceAuthorizationDTO deviceAuthorizationDTO) {

return iDeviceAuthorizationService.updateDeviceAuthorization(deviceAuthorizationDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceAuthorization")

@ApiOperation("逻辑删除设备授权")

public SaResult deleteDeviceAuthorization(@ApiParam("设备授权id") @RequestParam("id") String id) {

return iDeviceAuthorizationService.deleteDeviceAuthorization(id) ? SaResult.ok() : SaResult.error();

}

}

## DeviceBorrowRecordController.java

/\*\*

\* 设备借用记录

\*

\* @author wanggang

\* @date 2023/5/22 14:46

\*/

@RestController

@RequestMapping("/deviceBorrowRecord")

@RequiredArgsConstructor

@Api(tags = "设备借用记录接口")

public class DeviceBorrowRecordController extends BaseWebController {

private final IDeviceBorrowRecordService iDeviceBorrowRecordService;

@GetMapping("/selectDeviceBorrowRecordPage")

@ApiOperation("分页查询设备借用记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceBorrowRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceBorrowRecordPage(@ApiParam("设备借用记录查询条件实体类") DeviceBorrowRecordSelectDTO deviceBorrowRecordSelectDTO) {

Page<DeviceBorrowRecordVO> deviceBorrowRecordPage = iDeviceBorrowRecordService.selectDeviceBorrowRecordPage(deviceBorrowRecordSelectDTO);

return SaResult.ok().setData(deviceBorrowRecordPage);

}

@GetMapping("/selectDeviceBorrowRecordById")

@ApiOperation("根据设备借用记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceBorrowRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceBorrowRecordById(@ApiParam("设备借用记录ID") @RequestParam("id") String id) {

DeviceBorrowRecordVO deviceBorrowRecordVO = iDeviceBorrowRecordService.selectDeviceBorrowRecordById(id);

return SaResult.ok().setData(deviceBorrowRecordVO);

}

@PostMapping("/addDeviceBorrowRecord")

@ApiOperation("新增设备借用记录")

public SaResult addDeviceBorrowRecord(@ApiParam("设备借用记录DTO") @RequestBody @Valid DeviceBorrowRecordDTO deviceBorrowRecordDTO) {

return iDeviceBorrowRecordService.addDeviceBorrowRecord(deviceBorrowRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceBorrowRecord")

@ApiOperation("修改设备借用记录")

public SaResult updateDeviceBorrowRecord(@ApiParam("设备借用记录DTO") @RequestBody @Valid DeviceBorrowRecordDTO deviceBorrowRecordDTO) {

return iDeviceBorrowRecordService.updateDeviceBorrowRecord(deviceBorrowRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceBorrowRecord")

@ApiOperation("逻辑删除设备借用记录")

public SaResult deleteDeviceBorrowRecord(@ApiParam("设备借用记录id") @RequestParam("id") String id) {

return iDeviceBorrowRecordService.deleteDeviceBorrowRecord(id) ? SaResult.ok() : SaResult.error();

}

}

## DeviceCirculationRecordController.java

/\*\*

\* 设备流转记录

\*

\* @author wanggang

\* @date 2023/5/10 14:46

\*/

@RestController

@RequestMapping("/deviceCirculationRecord")

@RequiredArgsConstructor

@Api(tags = "设备流转记录接口")

public class DeviceCirculationRecordController extends BaseWebController {

private final IDeviceCirculationRecordService iDeviceCirculationRecordService;

@GetMapping("/selectDeviceCirculationRecordPage")

@ApiOperation("分页查询设备流转记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceCirculationRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceCirculationRecordPage(@ApiParam("设备流转记录查询条件实体类") DeviceCirculationRecordSelectDTO deviceCirculationRecordSelectDTO) {

Page<DeviceCirculationRecordVO> deviceCirculationRecordPage = iDeviceCirculationRecordService.selectDeviceCirculationRecordPage(deviceCirculationRecordSelectDTO);

return SaResult.ok().setData(deviceCirculationRecordPage);

}

@GetMapping("/selectDeviceCirculationRecordById")

@ApiOperation("根据设备设备流转记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceCirculationRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceCirculationRecordById(@ApiParam("设备流转记录ID") @RequestParam("id") String id) {

DeviceCirculationRecordVO deviceCirculationRecordVO = iDeviceCirculationRecordService.selectDeviceCirculationRecordById(id);

return SaResult.ok().setData(deviceCirculationRecordVO);

}

@PostMapping("/addDeviceCirculationRecord")

@ApiOperation("新增设备流转记录")

public SaResult addDeviceCirculationRecord(@ApiParam("设备流转记录DTO") @RequestBody @Valid DeviceCirculationRecordDTO deviceCirculationRecordDTO) {

return iDeviceCirculationRecordService.addDeviceCirculationRecord(deviceCirculationRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceCirculationRecord")

@ApiOperation("修改设备流转记录")

public SaResult updateDeviceCirculationRecord(@ApiParam("设备流转记录DTO") @RequestBody @Valid DeviceCirculationRecordDTO deviceCirculationRecordDTO) {

return iDeviceCirculationRecordService.updateDeviceCirculationRecord(deviceCirculationRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceCirculationRecord")

@ApiOperation("逻辑删除设备流转记录")

public SaResult deleteDeviceCirculationRecord(@ApiParam("设备流转记录id") @RequestParam("id") String id) {

return iDeviceCirculationRecordService.deleteDeviceCirculationRecord(id) ? SaResult.ok() : SaResult.error();

}

}

## DeviceClassifyInfoController.java

@RestController

@RequestMapping("/device/classify")

@Api(tags = "设备分类管理接口")

@RequiredArgsConstructor

@Validated

public class DeviceClassifyInfoController extends BaseWebController {

private final DeviceClassifyInfoService deviceClassifyInfoService;

@GetMapping("/select")

@ApiOperation("查询设备分类信息")

public SaResult selectDeviceClassifyInfo(DeviceClassifyInfoSelectDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(deviceClassifyInfoService.findByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据主键查询设备分类信息")

@ApiImplicitParam(name = "id", value = "主键", required = true, example = "1658755598444441601")

public SaResult selectDeviceClassifyInfo(String id) {

return SaResult.ok().setData(deviceClassifyInfoService.selectById(id));

}

@GetMapping("/selectClassifyNames")

@ApiOperation("查询设备分类名称数据")

public SaResult selectClassifyNames() {

return SaResult.ok().setData(deviceClassifyInfoService.selectDeviceClassifyNames(this.getTenantId()));

}

// TODO 设备编号接口需要提供

@PostMapping("/add")

@ApiOperation("添加设备分类信息、复制功能")

public SaResult addDeviceClassifyInfo(@RequestBody DeviceClassifyInfoDTO add) {

return deviceClassifyInfoService.add(add) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/update")

@ApiOperation("修改设备分类信息")

public SaResult updateDeviceClassifyInfo(@RequestBody DeviceClassifyInfoDTO update) {

return deviceClassifyInfoService.update(update) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteByIds")

@ApiOperation("删除设备分类信息")

public SaResult deleteDeviceClassifyInfo(String[] ids) {

deviceClassifyInfoService.logicDelete(ids);

return SaResult.ok().setData("删除成功！");

}

@PostMapping("/import")

@ApiOperation("导入设备分类信息")

@ApiImplicitParams({

@ApiImplicitParam(name = "file", value = "文件", required = true),

@ApiImplicitParam(name = "isOverride", value = "是否覆盖已经存在的数据"),

@ApiImplicitParam(name = "isFirstDelete", value = "导入之前是否删除数据")

})

public SaResult upload(@RequestParam(value = "file") MultipartFile file, Boolean isOverride, Boolean isFirstDelete) {

// 检测文件大小 1048576 == 1MB

if (file.getSize() > 52428800) {

return SaResult.error("导入数据不能大于50M");

}

if ((Boolean.TRUE.equals(isOverride) && Boolean.TRUE.equals(isFirstDelete)) ||

Boolean.FALSE.equals(isOverride) && Boolean.FALSE.equals(isFirstDelete)) {

return SaResult.error("是否覆盖和是否删除参数不能同时选中或不选");

}

try {

List<DeviceClassifyInfoVO> data = EasyExcelUtil.readExcel(file.getInputStream(), DeviceClassifyInfoVO.class, new EasyExcelUtil.ExcelListener<>());

deviceClassifyInfoService.importData(data, isOverride, isFirstDelete);

} catch (IOException e) {

throw new ServiceException(e);

}

return SaResult.ok();

}

@GetMapping("/downloadTemplate")

@ApiOperation("下载模板设备分类信息")

public void downloadDeviceClassifyInfo(HttpServletResponse response) {

try {

EasyExcelUtil.writeExcel(response, DeviceClassifyInfoVO.class, Collections.emptyList(), "设备分类信息模板" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

@GetMapping("/download")

@ApiOperation("导出设备分类信息")

public void downloadDeviceClassifyInfo(DeviceClassifyInfoSelectDTO dto, HttpServletResponse response) {

deviceClassifyInfoService.exportData(dto, response);

}

}

## DeviceInfoController.java

/\*\*

\* 设备信息

\*

\* @author wanggang

\* @date 2023/5/6 14:46

\*/

@RestController

@RequestMapping("/deviceInfo")

@RequiredArgsConstructor

@Api(tags = "设备信息接口")

public class DeviceInfoController extends BaseWebController {

private final IDeviceInfoService iDeviceInfoService;

@GetMapping("/selectDevicePage")

@ApiOperation("分页查询设备信息接口")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceInfoVO.class, message = "查询成功")

})

public SaResult selectDevicePage(@ApiParam("设备信息查询条件实体类") DeviceInfoSelectDTO deviceInfoSelectDTO) {

Page<DeviceInfoVO> deviceInfoPage = iDeviceInfoService.selectDevicePage(deviceInfoSelectDTO);

return SaResult.ok().setData(deviceInfoPage);

}

@GetMapping("/selectDeviceInfoById")

@ApiOperation("根据设备信息ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceInfoVO.class, message = "查询成功")

})

public SaResult selectDeviceInfoById(@ApiParam("设备信息ID") @RequestParam("id") String id) {

DeviceInfoVO deviceInfoVO = iDeviceInfoService.selectDeviceInfoById(id);

return SaResult.ok().setData(deviceInfoVO);

}

@PostMapping("/addDeviceInfo")

@ApiOperation("新增设备信息")

public SaResult addDeviceInfo(@ApiParam("设备信息DTO") @RequestBody @Valid DeviceInfoDTO deviceInfoDTO) {

return iDeviceInfoService.addDeviceInfo(deviceInfoDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceInfo")

@ApiOperation("修改设备信息")

public SaResult updateDeviceInfo(@ApiParam("设备信息DTO") @RequestBody @Valid DeviceInfoDTO deviceInfoDTO) {

return iDeviceInfoService.updateDeviceInfo(deviceInfoDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceInfo")

@ApiOperation("逻辑删除设备信息")

public SaResult deleteDeviceInfo(@ApiParam("设备信息id集") @RequestParam("id") String id) {

return iDeviceInfoService.deleteDeviceInfo(id) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/importDeviceInfo")

@ApiOperation("导入设备信息")

public SaResult importDeviceInfo(@ApiParam("导入的文件") @RequestParam(value = "file") MultipartFile file,

@ApiParam("是否覆盖更新字段(true是覆盖更新false先删除后插入)") @RequestParam(value = "isOverride") Boolean isOverride) {

// 检测文件大小 1048576 == 1MB

if (file.getSize() > 52428800) {

return SaResult.error("导入数据不能大于50M");

}

iDeviceInfoService.importDeviceInfo(file, isOverride);

return SaResult.ok();

}

@GetMapping("/exportDeviceInfo")

@ApiOperation("导出设备信息")

public void exportDeviceInfo(@ApiParam("设备信息查询条件实体类") DeviceInfoSelectDTO deviceInfoSelectDTO, HttpServletResponse response) {

iDeviceInfoService.exportDeviceInfo(deviceInfoSelectDTO, response);

}

}

## DevicePlaceController.java

@RestController

@RequestMapping("/device/place")

@Api(tags = "场所管理接口")

@RequiredArgsConstructor

@Validated

public class DevicePlaceController extends BaseWebController {

private final DevicePlaceService devicePlaceService;

@GetMapping("/select")

@ApiOperation("分页查询场所信息")

@ApiResponse(code = 200, response = DevicePlaceVO.class, message = "查询成功")

public SaResult select(CommonSelectEntity dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(devicePlaceService.findByPage(dto, true));

}

@GetMapping("/selectAllInfo")

@ApiOperation("查询当前租户下场所信息")

@ApiResponse(code = 200, response = DevicePlaceVO.class, message = "查询成功")

public SaResult select() {

CommonSelectEntity dto = new CommonSelectEntity();

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(devicePlaceService.findByPage(dto, false).getRecords());

}

@GetMapping("/treePlaceAndRoom")

@ApiOperation("查询所有的场所和下面的存放室，以树状结构返回")

public SaResult treePlaceAndRoom() {

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

List<DevicePlaceTreeVO> devicePlaceTreeVOS = devicePlaceService.treePlaceAndRoom();

for (DevicePlaceTreeVO first : devicePlaceTreeVOS) {

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

result.add(firstMap);

firstMap.put(SystemConstant.TREE\_TITLE, first.getPlaceName());

firstMap.put("placeCode", first.getPlaceCode());

firstMap.put(SystemConstant.TREE\_VALUE, first.getId());

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

firstMap.put(SystemConstant.TREE\_CHILDREN, subList);

for (DeviceRoomVO deviceRoomVO : first.getDeviceRoomVOS()) {

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

subList.add(secondMap);

secondMap.put(SystemConstant.TREE\_TITLE, deviceRoomVO.getRoomName());

secondMap.put("roomCode", deviceRoomVO.getRoomCode());

secondMap.put(SystemConstant.TREE\_VALUE, deviceRoomVO.getId());

secondMap.put(SystemConstant.TREE\_CHILDREN, Collections.emptyList());

}

}

return SaResult.ok().setData(result);

}

@GetMapping("/treePlaceAndRoomAndShelves")

@ApiOperation("查询所有的场所和下面的存放室以及存放架，以树状结构返回")

public SaResult treePlaceAndRoomAndShelves() {

return SaResult.ok().setData(devicePlaceService.treePlaceAndRoomAndShelves());

}

@GetMapping("/selectById")

@ApiOperation("根据主键查询场所信息")

@ApiImplicitParam(name = "id", value = "主键", required = true)

public SaResult selectById(String id) {

return SaResult.ok().setData(devicePlaceService.selectById(id));

}

@GetMapping("/init")

@ApiOperation("初始化场所信息")

public SaResult addDeviceClassifyInfo() {

devicePlaceService.add();

return SaResult.ok();

}

@PostMapping("/update")

@ApiOperation("修改场所信息")

public SaResult updateDeviceClassifyInfo(@RequestBody DevicePlaceDTO update) {

return devicePlaceService.update(update) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/batchUpdate")

@ApiOperation("批量修改场所信息")

public SaResult updateDeviceClassifyInfo(@RequestBody List<DevicePlaceDTO> update) {

update.forEach(devicePlaceService::update);

return SaResult.ok();

}

@DeleteMapping("/deleteByIds")

@ApiOperation("删除场所信息(没有使用)")

public SaResult deleteDeviceClassifyInfo(String[] ids) {

devicePlaceService.logicDeleteByIds(ids, getTenantId());

return SaResult.ok().setData("删除成功！");

}

@GetMapping("/surplusCode")

@ApiOperation("查询场所剩余编码")

public SaResult selectSurplusPlaceCode() {

List<Character> characters = devicePlaceService.selectSurplusPlaceCode();

List<Map<String, Character>> result = new ArrayList<>(characters.size());

for (Character v : characters) {

Map<String, Character> temp = new HashMap<>(1);

temp.put("label", v);

temp.put("value", v);

result.add(temp);

}

return SaResult.data(result);

}

}

## DeviceRepairMaintenanceRecordController.java

/\*\*

\* 设备保养维修记录

\*

\* @author wanggang

\* @date 2023/5/10 14:46

\*/

@RestController

@RequestMapping("/deviceRepairMaintenanceRecord")

@RequiredArgsConstructor

@Api(tags = "设备保养维修记录接口")

public class DeviceRepairMaintenanceRecordController extends BaseWebController {

private final IDeviceRepairMaintenanceRecordService iDeviceRepairMaintenanceRecordService;

@GetMapping("/selectDeviceRepairMaintenanceRecordPage")

@ApiOperation("分页查询设备保养维修记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceRepairMaintenanceRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceRepairMaintenanceRecordPage(@ApiParam("设备保养维修记录查询条件实体类") DeviceRepairMaintenanceRecordSelectDTO deviceRepairMaintenanceRecordSelectDTO) {

Page<DeviceRepairMaintenanceRecordVO> deviceRepairMaintenanceRecordPage = iDeviceRepairMaintenanceRecordService.selectDeviceRepairMaintenanceRecordPage(deviceRepairMaintenanceRecordSelectDTO);

return SaResult.ok().setData(deviceRepairMaintenanceRecordPage);

}

@GetMapping("/selectDeviceRepairMaintenanceRecordById")

@ApiOperation("根据设备设备保养维修记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceRepairMaintenanceRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceRepairMaintenanceRecordById(@ApiParam("设备保养维修记录ID") @RequestParam("id") String id) {

DeviceRepairMaintenanceRecordVO deviceRepairMaintenanceRecordVO = iDeviceRepairMaintenanceRecordService.selectDeviceRepairMaintenanceRecordById(id);

return SaResult.ok().setData(deviceRepairMaintenanceRecordVO);

}

@PostMapping("/addDeviceRepairMaintenanceRecord")

@ApiOperation("新增设备保养维修记录")

public SaResult addDeviceRepairMaintenanceRecord(@ApiParam("设备保养维修记录DTO") @RequestBody @Valid DeviceRepairMaintenanceRecordDTO deviceRepairMaintenanceRecordDTO) {

return iDeviceRepairMaintenanceRecordService.addDeviceRepairMaintenanceRecord(deviceRepairMaintenanceRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceRepairMaintenanceRecord")

@ApiOperation("修改设备保养维修记录")

public SaResult updateDeviceRepairMaintenanceRecord(@ApiParam("设备保养维修记录DTO") @RequestBody @Valid DeviceRepairMaintenanceRecordDTO deviceRepairMaintenanceRecordDTO) {

return iDeviceRepairMaintenanceRecordService.updateDeviceRepairMaintenanceRecord(deviceRepairMaintenanceRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceRepairMaintenanceRecord")

@ApiOperation("逻辑删除设备保养维修记录")

public SaResult deleteDeviceRepairMaintenanceRecord(@ApiParam("设备保养维修记录id") @RequestParam("id") String id) {

return iDeviceRepairMaintenanceRecordService.deleteDeviceRepairMaintenanceRecord(id) ? SaResult.ok() : SaResult.error();

}

}

## DeviceRoomController.java

@RestController

@RequestMapping("/device/room")

@Api(tags = "存放室管理接口")

@RequiredArgsConstructor

@Validated

public class DeviceRoomController extends BaseWebController {

private final DeviceRoomService deviceRoomService;

@GetMapping("/select")

@ApiOperation("分页查询存放室信息")

@ApiResponse(code = 200, response = DeviceRoomVO.class, message = "查询成功")

public SaResult select(CommonSelectEntity dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(deviceRoomService.findByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据主键查询存放室信息")

@ApiImplicitParam(name = "id", value = "主键", required = true)

public SaResult selectById(String id) {

return SaResult.ok().setData(deviceRoomService.selectById(id));

}

@PostMapping("/add")

@ApiOperation("添加存放室信息")

public SaResult addDeviceClassifyInfo(@RequestBody DeviceRoomDTO add) {

deviceRoomService.add(add);

return SaResult.ok();

}

@PostMapping("/update")

@ApiOperation("修改存放室信息")

public SaResult updateDeviceClassifyInfo(@RequestBody DeviceRoomDTO update) {

return deviceRoomService.update(update) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteByIds")

@ApiOperation("删除存放室信息")

public SaResult deleteDeviceClassifyInfo(String[] ids) {

deviceRoomService.logicDeleteByIds(ids, getTenantId());

return SaResult.ok().setData("删除成功！");

}

}

## DeviceShelvesInfoController.java

@RestController

@RequestMapping("/device/shelves")

@Api(tags = "货架管理")

@RequiredArgsConstructor

@Validated

public class DeviceShelvesInfoController extends BaseWebController {

private final DeviceShelvesInfoService deviceShelvesInfoService;

@GetMapping("/select")

@ApiOperation("分页查询货架信息")

@ApiResponse(code = 200, response = DeviceShelvesInfoVO.class, message = "查询成功")

public SaResult select(DeviceShelvesInfoDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(deviceShelvesInfoService.findByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据id查询货架信息")

@ApiResponse(code = 200, response = SupplierContactsInfoVO.class, message = "查询成功")

public SaResult selectById(String id) {

return SaResult.ok().setData(deviceShelvesInfoService.selectById(id));

}

@PostMapping("/add")

@ApiOperation(value = "添加货架信息")

public SaResult add(@RequestBody DeviceShelvesAddInfoDTO add) {

return deviceShelvesInfoService.add(add) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/update")

@ApiOperation("修改货架信息")

public SaResult update(@RequestBody DeviceShelvesInfoUpdateDTO update) {

return deviceShelvesInfoService.update(update) ? SaResult.ok() : SaResult.error();

}

@GetMapping("/deleteByIds")

@ApiOperation("删除货架信息")

public SaResult deleteByIds(@RequestParam("ids") String[] ids) {

deviceShelvesInfoService.logicDeleteByIds(ids, null);

return SaResult.ok().setData("删除成功！");

}

@GetMapping("/download")

@ApiOperation("导出货架信息")

public void downloadDeviceClassifyInfo(DeviceShelvesInfoDTO dto, HttpServletResponse response) {

Page<DeviceShelvesInfoVO> data = deviceShelvesInfoService.findByPage(dto, false);

dto.setTenantId(this.getTenantId());

try {

EasyExcelUtil.writeExcel(response, DeviceShelvesInfoVO.class, data.getRecords(), "货架信息" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

}

## DeviceVerificationRecordController.java

/\*\*

\* 设备检定记录

\*

\* @author wanggang

\* @date 2023/5/10 15:46

\*/

@RestController

@RequestMapping("/deviceVerificationRecord")

@RequiredArgsConstructor

@Api(tags = "设备检定记录接口")

public class DeviceVerificationRecordController extends BaseWebController {

private final IDeviceVerificationRecordService iDeviceVerificationRecordService;

@GetMapping("/selectDeviceVerificationRecordPage")

@ApiOperation("分页查询设备检定记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceVerificationRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceVerificationRecordPage(@ApiParam("设备检定记录查询条件实体类") DeviceVerificationRecordSelectDTO deviceVerificationRecordSelectDTO) {

Page<DeviceVerificationRecordVO> deviceVerificationRecordPage = iDeviceVerificationRecordService.selectDeviceVerificationRecordPage(deviceVerificationRecordSelectDTO);

return SaResult.ok().setData(deviceVerificationRecordPage);

}

@GetMapping("/selectDeviceVerificationRecordById")

@ApiOperation("根据设备检定记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceVerificationRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceVerificationRecordById(@ApiParam("设备检定记录ID") @RequestParam("id") String id) {

DeviceVerificationRecordVO deviceVerificationRecordVO = iDeviceVerificationRecordService.selectDeviceVerificationRecordById(id);

return SaResult.ok().setData(deviceVerificationRecordVO);

}

@PostMapping("/addDeviceVerificationRecord")

@ApiOperation("新增设备检定记录")

public SaResult addDeviceVerificationRecord(@ApiParam("设备检定记录DTO") @RequestBody @Valid DeviceVerificationRecordDTO deviceVerificationRecordDTO) {

return iDeviceVerificationRecordService.addDeviceVerificationRecord(deviceVerificationRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceVerificationRecord")

@ApiOperation("修改设备检定记录")

public SaResult updateDeviceVerificationRecord(@ApiParam("设备检定记录DTO") @RequestBody @Valid DeviceVerificationRecordDTO deviceVerificationRecordDTO) {

return iDeviceVerificationRecordService.updateDeviceVerificationRecord(deviceVerificationRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceVerificationRecord")

@ApiOperation("逻辑删除设备检定记录")

public SaResult deleteDeviceVerificationRecord(@ApiParam("设备检定记录id") @RequestParam("id") String id) {

return iDeviceVerificationRecordService.deleteDeviceVerificationRecord(id) ? SaResult.ok() : SaResult.error();

}

}

## DeviceWorkingRecordController.java

/\*\*

\* 设备运行记录

\*

\* @author wanggang

\* @date 2023/5/9 14:46

\*/

@RestController

@RequestMapping("/deviceWorkingRecord")

@RequiredArgsConstructor

@Api(tags = "设备运行记录接口")

public class DeviceWorkingRecordController extends BaseWebController {

private final IDeviceWorkingRecordService iDeviceWorkingRecordService;

@GetMapping("/selectDeviceWorkingRecordPage")

@ApiOperation("分页查询设备运行记录")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceWorkingRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceWorkingRecordPage(@ApiParam("设备运行记录查询条件实体类") DeviceWorkingRecordSelectDTO deviceWorkingRecordSelectDTO) {

Page<DeviceWorkingRecordVO> deviceWorkingRecordPage = iDeviceWorkingRecordService.selectDeviceWorkingRecordPage(deviceWorkingRecordSelectDTO);

return SaResult.ok().setData(deviceWorkingRecordPage);

}

@GetMapping("/selectDeviceWorkingRecordById")

@ApiOperation("根据设备运行记录ID查询修改回显数据")

@ApiResponses({

@ApiResponse(code = 200, response = DeviceWorkingRecordVO.class, message = "查询成功")

})

public SaResult selectDeviceWorkingRecordById(@ApiParam("设备运行记录ID") @RequestParam("id") String id) {

DeviceWorkingRecordVO deviceWorkingRecordVO = iDeviceWorkingRecordService.selectDeviceWorkingRecordById(id);

return SaResult.ok().setData(deviceWorkingRecordVO);

}

@PostMapping("/addDeviceWorkingRecord")

@ApiOperation("新增设备运行记录")

public SaResult addDeviceWorkingRecord(@ApiParam("设备运行记录DTO") @RequestBody @Valid DeviceWorkingRecordDTO deviceWorkingRecordDTO) {

return iDeviceWorkingRecordService.addDeviceWorkingRecord(deviceWorkingRecordDTO) ? SaResult.ok() : SaResult.error();

}

@PutMapping("/updateDeviceWorkingRecord")

@ApiOperation("修改设备运行记录")

public SaResult updateDeviceWorkingRecord(@ApiParam("设备运行记录DTO") @RequestBody @Valid DeviceWorkingRecordDTO deviceWorkingRecordDTO) {

return iDeviceWorkingRecordService.updateDeviceWorkingRecord(deviceWorkingRecordDTO) ? SaResult.ok() : SaResult.error();

}

@DeleteMapping("/deleteDeviceWorkingRecord")

@ApiOperation("逻辑删除设备运行记录")

public SaResult deleteDeviceWorkingRecord(@ApiParam("设备运行记录id") @RequestParam("id") String id) {

return iDeviceWorkingRecordService.deleteDeviceWorkingRecord(id) ? SaResult.ok() : SaResult.error();

}

}

## SupplierContactsInfoController.java

@RestController

@RequestMapping("/supplier/contacts")

@Api(tags = "厂商联系人管理")

@RequiredArgsConstructor

@Validated

public class SupplierContactsInfoController extends BaseWebController {

private final SupplierContactsInfoService supplierContactsInfoService;

private final ISysDictCategoryService sysDictCategoryService;

@GetMapping("/select")

@ApiOperation("分页查询厂商联系人信息")

@ApiResponse(code = 200, response = SupplierContactsInfoVO.class, message = "查询成功")

public SaResult select(SupplierContactsInfoSelectDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(supplierContactsInfoService.findByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据id查询 联系人信息")

@ApiResponse(code = 200, response = SupplierContactsInfoVO.class, message = "查询成功")

public SaResult selectById(@RequestParam("id") String id) {

return SaResult.ok().setData(supplierContactsInfoService.selectById(id));

}

@PostMapping("/add")

@ApiOperation("添加厂商联系人信息")

public SaResult add(@RequestBody SupplierContactsInfoDTO add) {

return supplierContactsInfoService.addSupplierContactsInfo(add) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/update")

@ApiOperation("修改厂商联系人信息")

public SaResult update(@RequestBody SupplierContactsInfoDTO update) {

return supplierContactsInfoService.updateSupplierContactsInfo(update) ? SaResult.ok() : SaResult.error();

}

@GetMapping("/deleteByIds")

@ApiOperation("删除厂商联系人信息")

public SaResult deleteByIds(@RequestParam("ids") String[] ids) {

supplierContactsInfoService.logicDeleteByIds(ids, null, null);

return SaResult.ok().setData("删除成功！");

}

// // 提示：此接口没有使用、测试。没有使用场景。暂时保留

@PostMapping("/import")

@ApiOperation(value = "导入厂商联系人信息" , hidden = true)

public SaResult upload(@RequestParam(value = "file") MultipartFile file) {

this.checkFileSize(file);

try {

List<SupplierContactsInfoDTO> excelData = EasyExcelUtil.readExcel(file.getInputStream(), SupplierContactsInfoDTO.class, new EasyExcelUtil.ExcelListener<>());

if (CollUtil.isEmpty(excelData)) {

return SaResult.data("导入成功！");

}

// 批量导入数据

List<SupplierContactsInfo> collect = excelData.stream().map(SupplierContactsInfoDTO::transferEntity).toList();

supplierContactsInfoService.saveBatch(collect);

} catch (IOException e) {

throw new ServiceException(e);

}

return SaResult.ok();

}

@GetMapping("/downloadTemplate")

@ApiOperation("下载模板厂商联系人信息")

public void downloadTemplate(HttpServletResponse response) {

try {

EasyExcelUtil.writeExcel(response, SupplierContactsInfoDTO.class, Collections.emptyList(), "厂商联系人信息模板" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

@GetMapping("/download")

@ApiOperation("导出厂商联系人信息")

public void downloadDeviceClassifyInfo(SupplierContactsInfoSelectDTO dto, HttpServletResponse response) {

Page<SupplierContactsInfoVO> deviceClassifyByPage = supplierContactsInfoService.findByPage(dto, false);

dto.setTenantId(this.getTenantId());

// 获取性别字典

Map<String, String> gender = sysDictCategoryService.selectDictBycategoryKey(null, "gender" , true);

if (CollectionUtils.isEmpty(gender)) {

throw new ServiceException("性别字典没有数据，请联系管理员添加");

}

List<SupplierContactsInfoDTO> collect = deviceClassifyByPage.getRecords().stream().map(c -> {

SupplierContactsInfoDTO data = c.transferDto();

data.setGender(gender.get(String.valueOf(data.getGender())));

return data;

}).toList();

try {

EasyExcelUtil.writeExcel(response, SupplierContactsInfoDTO.class, collect, "厂商联系人信息" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

}

## SupplierInfoController.java

@RestController

@RequestMapping("/supplier")

@Api(tags = "厂商管理")

@RequiredArgsConstructor

@Validated

public class SupplierInfoController extends BaseWebController {

private final SupplierInfoService supplierInfoService;

private final SupplierContactsInfoService contactInfoService;

@GetMapping("/select")

@ApiOperation(value = "分页查询厂商和联系人信息", notes = "分页查询，响应参数省略了分页数据")

@ApiResponse(code = 200, response = SupplierInfoVO.class, message = "查询成功")

public SaResult select(SupplierInfoSelectDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(supplierInfoService.findByPage(dto, true));

}

@GetMapping("/selectSupplier")

@ApiOperation(value = "分页查询厂商信息", notes = "分页查询")

@ApiResponse(code = 200, response = SupplierInfoVO.class, message = "查询成功")

public SaResult selectSupplier(SupplierInfoSelectDTO dto) {

dto.setTenantId(this.getTenantId());

return SaResult.ok().setData(supplierInfoService.findSupplierByPage(dto, true));

}

@GetMapping("/selectById")

@ApiOperation("根据id查询 厂商和联系人信息")

@ApiResponse(code = 200, response = SupplierInfoDTO.class, message = "查询成功")

@ApiImplicitParam(value = "id", example = "1656847984220512257", required = true)

public SaResult selectById(@NotBlank(message = "主键不能为空") String id) {

SupplierInfo byId = supplierInfoService.getById(id);

SupplierInfoDTO dto = BeanUtil.copyProperties(byId, SupplierInfoDTO.class);

dto.setContactList(contactInfoService.selectListBySupplierId(id));

return SaResult.ok().setData(dto);

}

@PostMapping("/add")

@ApiOperation("添加厂商和联系人信息")

@ApiResponse(code = 200, response = SaResult.class, message = "查询成功")

public SaResult add(@RequestBody SupplierInfoDTO add) {

return supplierInfoService.addSupplierInfo(add) ? SaResult.ok() : SaResult.error();

}

@PostMapping("/update")

@ApiOperation(value = "修改厂商信息", notes = "注意这个接口只能修改厂商信息")

@ApiResponse(code = 200, response = SaResult.class, message = "查询成功")

public SaResult update(@RequestBody SupplierInfoUpdateDTO update) {

return supplierInfoService.updateSupplierInfo(update) ? SaResult.ok() : SaResult.error();

}

@GetMapping("/deleteByIds")

@ApiOperation("删除厂商和联系人信息")

@ApiResponse(code = 200, response = SaResult.class, message = "查询成功")

@ApiImplicitParam(name = "ids", example = "1656847984220512257,1656847984220512290")

public SaResult deleteByIds(@RequestParam(name = "ids") String[] ids) {

if (ids == null || ids.length == 0) {

return SaResult.ok().setData("删除成功！");

}

supplierInfoService.logicDelete(ids);

return SaResult.ok().setData("删除成功！");

}

@PostMapping("/import")

@ApiOperation(value = "导入厂商和联系人分类信息", notes = "isOverride 和 isFirstDelete 参数只能有一个为true")

@ApiImplicitParams({

@ApiImplicitParam(name = "file", value = "文件", required = true),

@ApiImplicitParam(name = "isOverride", value = "是否覆盖已经存在的数据"),

@ApiImplicitParam(name = "isFirstDelete", value = "导入之前是否删除数据")

})

public SaResult upload(@RequestParam(value = "file") MultipartFile file, Boolean isOverride, Boolean isFirstDelete) {

this.checkFileSize(file);

if ((Boolean.TRUE.equals(isOverride) && Boolean.TRUE.equals(isFirstDelete)) ||

Boolean.FALSE.equals(isOverride) && Boolean.FALSE.equals(isFirstDelete)) {

return SaResult.error("是否覆盖和是否删除参数不能同时选中或不选");

}

try {

List<SupplierInfoVO> excelData = EasyExcelUtil.readExcel(file.getInputStream(), SupplierInfoVO.class, new EasyExcelUtil.ExcelListener<>());

supplierInfoService.importData(excelData, isOverride, isFirstDelete);

} catch (IOException e) {

throw new ServiceException(e);

}

return SaResult.ok();

}

@GetMapping("/downloadTemplate")

@ApiOperation("下载模板厂商和联系人")

public void downloadTemplate(HttpServletResponse response) {

try {

EasyExcelUtil.writeExcel(response, SupplierInfoVO.class, Collections.emptyList(), "厂商和联系人信息模板" + System.currentTimeMillis() + ".xlsx");

} catch (IOException | IllegalAccessException | NoSuchFieldException e) {

throw new ServiceException(e);

}

}

@GetMapping("/download")

@ApiOperation("导出厂商和联系人数据")

public void download(SupplierInfoSelectDTO dto, HttpServletResponse response) {

supplierInfoService.exportData(dto, response);

}

}

# flow

## FlowableHistoryController.java

/\*\*

\* 历史信息服务

\* @author wanggang

\* @date 2023/4/15 10:30

\*

\*/

@Api(tags = "历史信息服务")

@RestController

@RequestMapping(value = "/flowableHistory")

public class FlowableHistoryController {

@Autowired

IFlowableHistoryService iFlowableHistoryService;

@ApiOperation("查询所有历史任务信息")

@GetMapping(value = "/selectHistoryTask")

public SaResult selectHistoryTask() {

List<HistoricTaskInstanceDTO> historicTaskInstanceDTOList = iFlowableHistoryService.selectHistoryTask();

return SaResult.ok().setData(historicTaskInstanceDTOList);

}

@ApiOperation("查询已完成历史任务信息")

@GetMapping(value = "/selectHistoryTaskFinish")

public SaResult selectHistoryTaskFinish() {

List<HistoricTaskInstanceDTO> historicTaskInstanceDTOList = iFlowableHistoryService.selectHistoryTaskFinish();

return SaResult.ok().setData(historicTaskInstanceDTOList);

}

}

## FlowableIdentityController.java

/\*\*

\* 初始化用户服务

\*

\* @author wanggang

\* @date 2023/4/15 10:30

\*/

@Api(tags = "初始化用户服务")

@RestController

@RequestMapping(value = "/flowableIdentity")

public class FlowableIdentityController {

@Autowired

IFlowableIdentityService iFlowableIdentityService;

/\*\*

\* 新增组

\*

\* @param groupDTO

\* @return void

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

@ApiOperation("新增用户组")

@PostMapping("/addGroup")

public SaResult addGroup(@ApiParam("组DTO") @Valid @RequestBody GroupDTO groupDTO) {

iFlowableIdentityService.addGroup(groupDTO);

return SaResult.ok();

}

/\*\*

\* 查询组

\*

\* @param param

\* @return void

\* @author wanggang

\* @date 2023/4/14 17:10

\*/

@ApiOperation("查询组")

@GetMapping("/selectGroup")

public SaResult selectGroup(@ApiParam("查询参数(可以传组ID、组Name、用户ID、不传为全查询)")

@RequestParam(value = "param", required = false) String param) {

List<GroupDTO> groups = iFlowableIdentityService.selectGroup(param);

return SaResult.ok().setData(groups);

}

/\*\*

\* 添加用户和组的关联

\*

\* @param groupAndUserDTOList

\* @return void

\* @author wanggang

\* @date 2023/4/17 10:32

\*/

@ApiOperation("添加用户和组的关联")

@PostMapping("/addUserAndGroup")

public SaResult addUserAndGroupRelationBatch(@ApiParam("用户和组") @RequestBody List<GroupAndUserDTO> groupAndUserDTOList) {

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

throw new ParameterException("参数缺失");

}

iFlowableIdentityService.addUserAndGroupRelationBatch(groupAndUserDTOList);

return SaResult.ok();

}

/\*\*

\* 修改组名信息

\*

\* @param groupDTO

\* @return void

\* @author wanggang

\* @date 2023/4/17 17:10

\*/

@ApiOperation("修改组信息")

@PutMapping("/updateGroup")

public SaResult updateGroup(@ApiParam("组DTO") @Valid @RequestBody GroupDTO groupDTO) {

iFlowableIdentityService.updateGroup(groupDTO);

return SaResult.ok();

}

/\*\*

\* 删除组

\*

\* @param groupId

\* @return void

\* @author wanggang

\* @date 2023/4/17 17:10

\*/

@ApiOperation("删除组")

@DeleteMapping("/deleteGroup")

public SaResult deleteGroup(@ApiParam("组Id") @RequestParam(value = "groupId") String groupId) {

iFlowableIdentityService.deleteGroup(groupId);

return SaResult.ok();

}

/\*\*

\* 新增用户

\*

\* @param userDTO

\* @return void

\* @author wanggang

\* @date 2023/4/17 17:10

\*/

@ApiOperation("新增用户")

@PostMapping("/addUser")

public SaResult addUser(@ApiParam("用户DTO") @Valid @RequestBody UserDTO userDTO) {

iFlowableIdentityService.addUser(userDTO);

return SaResult.ok();

}

/\*\*

\* 修改除密码外的其他信息

\*

\* @param userDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:41

\*/

@ApiOperation("修改用户")

@PutMapping("/updateUser")

public SaResult updateUser(@ApiParam("用户DTO") @Valid @RequestBody UserDTO userDTO) {

iFlowableIdentityService.updateUser(userDTO);

return SaResult.ok();

}

/\*\*

\* 删除用户

\*

\* @param userId

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:46

\*/

@ApiOperation("删除用户")

@DeleteMapping("/deleteUser")

public SaResult deleteUser(@ApiParam("用户Id") @RequestParam(value = "userId") String userId) {

iFlowableIdentityService.deleteUser(userId);

return SaResult.ok();

}

/\*\*

\* 查询用户

\*

\* @param param

\* @return void

\* @author wanggang

\* @date 2023/4/18 9:10

\*/

@ApiOperation("查询用户")

@GetMapping("/selectUser")

public SaResult selectUser(@ApiParam("查询参数(可以传用户ID、用户显示名、不传为全查询)")

@RequestParam(value = "param", required = false) String param) {

List<UserDTO> users = iFlowableIdentityService.selectUser(param);

return SaResult.ok().setData(users);

}

/\*\*

\* 删除用户和组的关联关系

\*

\* @param groupAndUserDTO

\* @return void

\* @author wanggang

\* @date 2023/4/19 16:46

\*/

@ApiOperation("删除用户和组的关联关系")

@DeleteMapping("/deleteUserAndGroupRelation")

public SaResult deleteUserAndGroupRelation(@ApiParam("用户和组DTO") @Valid @RequestBody GroupAndUserDTO groupAndUserDTO) {

iFlowableIdentityService.deleteUserAndGroupRelation(groupAndUserDTO);

return SaResult.ok();

}

}

## FlowableRepositoryController.java

/\*\*

\* 流程定义和部署服务

\*

\* @author wanggang

\* @date 2023/4/15 10:33

\*/

@Api(tags = "流程定义和部署服务")

@RestController

@RequestMapping(value = "/flowableRepository")

public class FlowableRepositoryController {

@Autowired

IFlowableRepositoryService iFlowableRepositoryService;

@ApiOperation("根据flowId和resourceName查询流程图或xml")

@GetMapping(value = "/loadFlowXml")

public void loadFlowXml(HttpServletResponse httpServletResponse,

@ApiParam("流程Id") @RequestParam(value = "flowId") String flowId,

@ApiParam("流程图名") @RequestParam(value = "resourceName") String resourceName) {

iFlowableRepositoryService.loadFlowXml(httpServletResponse, flowId, resourceName);

}

// @ApiOperation("下载资源文件")

// @GetMapping(value = "/downloadFlowXml")

// public void downloadFlowXml(@ApiParam("流程Id") @RequestParam(value = "flowId") String flowId,

// @ApiParam("流程图名") @RequestParam(value = "resourceName") String resourceName,

// HttpServletRequest request, HttpServletResponse response) {

// iFlowableRepositoryService.downloadFlowXml(flowId, resourceName, request, response);

// }

@ApiOperation("查询流程定义信息(不传参数为全查询)")

@GetMapping(value = "/selectProcessDefinition")

public SaResult selectProcessDefinition(@ApiParam("流程名称") @RequestParam(value = "name", required = false) String name,

@ApiParam("流程key") @RequestParam(value = "key", required = false) String key) {

List<ProcessDefinitionDTO> processDefinitions = iFlowableRepositoryService.selectProcessDefinition(name, key);

return SaResult.ok().setData(processDefinitions);

}

}

## FlowableTaskController.java

@Api(tags = "流程相关操作接口")

@RestController

@RequestMapping(value = "/flowableTask")

public class FlowableTaskController extends BaseWebController {

@Autowired

IFlowableTaskService flowableTaskService;

@Autowired

IFlowFactory flowFactory;

@Autowired

ISysOrgService sysOrgService;

@Autowired

IFlowBaseInfoService flowBaseInfoService;

@Autowired

ISysUserService sysUserService;

@ApiOperation("查询待办任务列表")

@GetMapping(value = "/findMyTask")

@ApiResponses({

@ApiResponse(code = 200,message = "返回成功",response = FlowBaseInfo.class)

})

public SaResult getFlowableMyTaskByUserId(@RequestParam Integer current,

@RequestParam Integer pageSize,

String searchKey,

String processKey,

String processState,

String startTime,

String endTime

) {

SysUser sysUser = getUserDetail();

IPage<FlowBaseInfo> page = flowableTaskService.findPageFlowBaseInfo(current, pageSize, sysUser, getMainFlowRoles(), getSlaveFlowRoles(),searchKey,processKey,processState,startTime,endTime);

return SaResult.ok().setData(page);

}

@ApiOperation("审核通过任务")

@PostMapping(value = "/completeTask")

public SaResult completeTask(@Validated @RequestBody CompleteTaskDTO completeTaskDTO) {

//

SysUser sysUser=getUserDetail();

if (Strings.isNullOrEmpty(sysUser.getSignatureBase64())){

return SaResult.error("您还未设置签名，无法审核，请到右上角个人中心设置签名");

}

flowableTaskService.completeTask(completeTaskDTO, getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes());

return SaResult.ok().setMsg("审核成功");

}

@ApiOperation("拒绝任务，驳回到任务发起人")

@PostMapping(value = "/rejectTask")

public SaResult myCompletedProcessPage(@Validated @RequestBody RejectTaskDTO rejectTaskDTO) {

//

//判断当前流程的审核人和发起人都是登登录人才能修改

if (!flowableTaskService.canApproval(rejectTaskDTO.getProcessId(), getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes())) {

return SaResult.error("任务审核人不是您,无法拒绝");

}

flowableTaskService.rejectTask(rejectTaskDTO);

return SaResult.data(true).setMsg("拒绝成功,已回到申请人节点");

}

@ApiOperation("查询已经办理的流程")

@GetMapping("myCompletedProcessPage")

@ApiResponses({

@ApiResponse(code = 200,message = "返回成功",response = FlowBaseInfo.class)

})

public SaResult myCompletedProcessPage(@RequestParam Integer current,

@RequestParam Integer pageSize,

String searchKey,

@ApiParam("流程key") String processKey,

@ApiParam("流程状态") String processState,

String startTime, String endTime) {

//

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

IPage<FlowBaseInfo> flowBaseInfoIPage = flowBaseInfoService.findCompletedProcessPage(new Page<>(current, pageSize), sysUser.getId()

,processState,processKey,searchKey,startTime,endTime);

return SaResult.data(flowBaseInfoIPage).setMsg("查询成功");

}

@ApiOperation("查询我发起的流程")

@GetMapping("myStartProcessPage")

@ApiResponses({

@ApiResponse(code = 200,message = "返回成功",response = FlowBaseInfo.class)

})

public SaResult myStartProcessPage(@RequestParam Integer current,

@RequestParam Integer pageSize,

String searchKey,

@ApiParam("流程key") String processKey,

@ApiParam("流程状态") String processState,String startTime, String endTime) {

SysUser sysUser = TenantContextHolder.getContext().getTenantContextInfo().getSysUser();

IPage<FlowBaseInfo> flowBaseInfoIPage = flowBaseInfoService.page(new Page<>(current, pageSize), new QueryWrapper<FlowBaseInfo>() .lambda()

.eq(!Strings.isNullOrEmpty(processState)&&ProcessState.match(processState)!=null,FlowBaseInfo::getState,ProcessState.match(processState))

.eq(!Strings.isNullOrEmpty(processKey),FlowBaseInfo::getProcessKey,processKey)

.eq(BaseEntity::getCreateBy, sysUser.getId()).orderByDesc(BaseEntity::getCreateTime)

.and(!Strings.isNullOrEmpty(searchKey), flowBaseInfoLambdaQueryWrapper -> flowBaseInfoLambdaQueryWrapper.like(FlowBaseInfo::getProcessId, searchKey)

.or().like(FlowBaseInfo::getProcessName, searchKey)

.or().like(FlowBaseInfo::getCreateName, searchKey)

.or().like(FlowBaseInfo::getSysOrgFullName, searchKey))

.gt(!Strings.isNullOrEmpty(startTime),BaseEntity::getCreateTime,startTime)

.lt(!Strings.isNullOrEmpty(endTime),BaseEntity::getCreateTime,endTime)

);

return SaResult.data(flowBaseInfoIPage).setMsg("查询成功");

}

@ApiOperation("查询流程的审核记录")

@GetMapping("processHis/{processId}")

@ApiResponses({

@ApiResponse(code = 200,message = "返回成功",response = FlowHistVO.class)

})

public SaResult processHis(@PathVariable String processId) {

List<FlowHistVO> flowHistVOS =flowableTaskService.findFlowHistVOs(processId);

return SaResult.data(flowHistVOS).setMsg("查询成功");

}

/\*\*

\* 获取流程图并显示当前流程

\*

\* @param httpServletResponse, processId

\* @return void

\* @author wanggang

\* @date 2023/4/18 11:13

\*/

@ApiOperation("获取流程图并显示当前流程")

@GetMapping(value = "/processDiagram")

public void genProcessDiagram(HttpServletResponse httpServletResponse,

@ApiParam("流程Id") @RequestParam(value = "processId") String processId) throws Exception {

flowableTaskService.genProcessDiagram(httpServletResponse, processId);

}

// /\*\*

// \* 根据任务Id获取流程图并显示当前流程

// \*

// \* @param httpServletResponse, taskId

// \* @return void

// \* @author wanggang

// \* @date 2023/4/18 11:13

// \*/

// @ApiOperation("根据任务Id获取流程图并显示当前流程")

// @GetMapping(value = "/processDiagramByTaskId")

// public void genProcessDiagramByTaskId(HttpServletResponse httpServletResponse,

// @ApiParam("任务Id") @RequestParam(value = "taskId") String taskId) throws Exception {

// flowableTaskService.genProcessDiagramByTaskId(httpServletResponse, taskId);

// }

}

## FlowDeviceVerificationRecordController.java

/\*\*

\* 检定校验申请记录

\*

\* @author wanggang

\* @date 2023/6/2 14:46

\*/

@RestController

@RequestMapping("/flowDeviceVerificationRecord")

@Api(tags = "检定校验申请记录接口")

public class FlowDeviceVerificationRecordController extends BaseWebController {

@Resource

IFlowDeviceVerificationRecordService iFlowDeviceVerificationRecordService;

@Resource

IFlowDeviceVerificationService iFlowDeviceVerificationService;

@GetMapping("/selectFlowDeviceVerificationRecordPage")

@ApiOperation("分页查询检定校准申请流水记录")

@ApiResponses({

@ApiResponse(code = 200, response = FlowDeviceVerificationVO.class, message = "查询成功")

})

public SaResult selectFlowDeviceVerificationRecordPage(@ApiParam("检定校准申请流水记录查询条件实体类") FlowDeviceVerificationRecordSelectDTO flowDeviceVerificationRecordSelectDTO) {

Page<FlowDeviceVerificationVO> flowDeviceVerificationPage = iFlowDeviceVerificationService.selectFlowDeviceVerificationRecordPage(flowDeviceVerificationRecordSelectDTO);

return SaResult.ok().setData(flowDeviceVerificationPage);

}

@PostMapping("/storageFlowDeviceVerificationRecord")

@ApiOperation("暂存检定校验申请记录")

public SaResult storageFlowDeviceVerificationRecord(@ApiParam("检定校验申请记录DTO") @RequestBody List<FlowDeviceVerificationDTO> flowDeviceVerificationDTOList) {

if (CollUtil.isEmpty(flowDeviceVerificationDTOList)) {

throw new ParameterException("修改参数缺失!");

}

return iFlowDeviceVerificationService.updateFlowDeviceVerificationBatch(flowDeviceVerificationDTOList) ? SaResult.ok() : SaResult.error();

}

@GetMapping("/selectByProcessRecordId")

@ApiOperation("根据流程记录ID查询明细")

@ApiResponses({

@ApiResponse(code = 200, response = FlowDeviceVerificationVO.class, message = "查询成功")

})

public SaResult selectByProcessRecordId(@ApiParam("流程记录ID") @RequestParam("processRecordId") String processRecordId) {

List<FlowDeviceVerificationVO> flowDeviceVerificationVO = iFlowDeviceVerificationService.selectByProcessRecordId(processRecordId);

return SaResult.ok().setData(flowDeviceVerificationVO);

}

}

## FlowProcessBizController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/6/2 15:13

\*/

@RestController

@RequestMapping("flowProcessBiz")

@Api(tags = "流程相关操作接口")

public class FlowProcessBizController extends BaseWebController {

@Autowired

FlowBaseHandlerFactory flowBaseHandlerFactory;

@Autowired

IFlowFactory flowFactory;

@Autowired

IFlowableTaskService flowableTaskService;

@Autowired

IFlowBaseInfoService flowBaseInfoService;

/\*\*

\* @return

\*/

@ApiOperation(value = "创建流程传入参数格式{processKey:'',processName:'',......}")

@PostMapping("createProcess")

public SaResult createProcess(@RequestBody @Validated JSONObject jsonObject) {

if (Strings.isNullOrEmpty(jsonObject.getString("processKey"))) {

return SaResult.error("请选择需要发起的流程key");

}

FlowBaseFactory flowBaseFactory = flowBaseHandlerFactory.createHandler(jsonObject.getString("processKey"));

flowBaseFactory.createProcess(jsonObject, null);

return SaResult.data(true).setMsg("流程创建成功");

}

@ApiOperation("获取流程业务数据")

@GetMapping("getProcessBiz/{processId}")

@ApiResponses({

@ApiResponse(code = 200, response = FlowBizVO.class, message = "流程业务数据")

})

public SaResult getProcessBiz(@PathVariable String processId) {

String processKey = "";

boolean isEnd=false;

ProcessInstance processInstance = flowFactory.getRuntimeService().createProcessInstanceQuery().processInstanceId(processId).singleResult();

if (processInstance == null) {

HistoricProcessInstance historicProcessInstance = flowFactory.getHistoryService().createHistoricProcessInstanceQuery().processInstanceId(processId).singleResult();

if (historicProcessInstance == null) {

return SaResult.error("不存在的流程");

} else {

isEnd=true;

processKey = historicProcessInstance.getProcessDefinitionKey();

}

} else {

processKey = processInstance.getProcessDefinitionKey();

}

FlowBaseFactory flowBaseFactory = flowBaseHandlerFactory.createHandler(processKey);

FlowBizVO flowBizVO = new FlowBizVO();

flowBizVO.setCanUpdate(flowableTaskService.canUpdate(processId, getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes()));

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, processId));

flowBizVO.setCanDelete(getUserDetail().getId().equals(flowBaseInfo.getCreateBy())&& !isEnd);

flowBizVO.setCanApproval(flowableTaskService.canApproval(processId, getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes()));

flowBizVO.setCanReject(flowBizVO.getCanApproval() && !flowBizVO.getCanDelete());

flowBizVO.setBizObj(flowBaseFactory.getProcessBiz(processId));

flowBizVO.setProcessKey(ProcessKey.match(processKey));

return SaResult.data(flowBizVO).setMsg("查询成功");

}

@ApiOperation(value = "修改业务数据,传入格式为{processId:'xxxx',.....}")

@PutMapping("updateProcessBiz")

public SaResult updateProcessBiz(@RequestBody @Validated JSONObject flowBaseInfoUpdateDTO) {

SysUser sysUser=getUserDetail();

if (Strings.isNullOrEmpty(sysUser.getSignatureBase64())){

return SaResult.error("您还未设置签名，请到右上角个人中心设置签名");

}

ProcessInstance processInstance = flowFactory.getRuntimeService().createProcessInstanceQuery().processInstanceId(flowBaseInfoUpdateDTO.getString("processId")).singleResult();

if (processInstance == null) {

return SaResult.error("不存在的流程");

}

//判断当前流程的审核人和发起人都是登登录人才能修改

if (!flowableTaskService.canUpdate(processInstance.getProcessInstanceId(), getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes())) {

return SaResult.error("您不是当前流程的发起人且任务审核人不是您,无法修改");

}

FlowBaseFactory flowBaseFactory = flowBaseHandlerFactory.createHandler(processInstance.getProcessDefinitionKey());

flowBaseFactory.updateProcessBiz(flowBaseInfoUpdateDTO, processInstance.getProcessInstanceId());

CompleteTaskDTO completeTaskDTO=new CompleteTaskDTO();

completeTaskDTO.setProcessId(processInstance.getProcessInstanceId());

completeTaskDTO.setComment("再次发起申请");

flowableTaskService.completeTask(completeTaskDTO, getMainFlowRoles(), getSlaveFlowRoles(), getAllFlowRoleCodes());

return SaResult.data(null).setMsg("修改成功");

}

@ApiOperation("废弃流程")

@PostMapping("deleteProcess")

@Transactional

public SaResult deleteProcess(@RequestBody @Validated DeleteProcessDTO deleteProcessDTO) {

ProcessInstance processInstance = flowFactory.getRuntimeService().createProcessInstanceQuery().processInstanceId(deleteProcessDTO.getProcessId()).singleResult();

if (processInstance==null){

return SaResult.error("流程已经结束或不存在的流程！");

}

FlowBaseInfo flowBaseInfo = flowBaseInfoService.getOne(new QueryWrapper<FlowBaseInfo>().lambda().eq(FlowBaseInfo::getProcessId, deleteProcessDTO.getProcessId()));

if (!getUserDetail().getId().equals(flowBaseInfo.getCreateBy())) {

return SaResult.error("您不是当前流程发起人，无法废弃流程！");

}

//

flowFactory.getRuntimeService().deleteProcessInstance(deleteProcessDTO.getProcessId(), deleteProcessDTO.getDeleteReason());

flowBaseInfo.setState(ProcessState.REVOKE);

flowBaseInfoService.updateById(flowBaseInfo);

return SaResult.data(true).setMsg("废弃成功");

}

@ApiOperation("查询全部流程定义")

@GetMapping("flowProcessKeys")

public SaResult flowProcessKeys() {

return SaResult.data(ProcessKey.values());

}

@ApiOperation("查询全部流程状态定义")

@GetMapping("flowProcessStates")

public SaResult flowProcessStates() {

return SaResult.data(ProcessState.values());

}

}

# h5

## H5Controller.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/27 17:10

\*/

@RestController

public class H5Controller {

@Autowired

ISysUserService sysUserService;

// 当前是否登录

@RequestMapping("/sso/isLogin")

@TenantIgnore

public Object isLogin() {

return SaResult.data(StpUtil.isLogin());

}

// 返回SSO认证中心登录地址

@RequestMapping("/sso/getSsoAuthUrl")

@TenantIgnore

public SaResult getSsoAuthUrl(String clientLoginUrl, String back) {

String serverAuthUrl = SaSsoUtil.buildServerAuthUrl(clientLoginUrl, back);

return SaResult.data(serverAuthUrl);

}

// 根据ticket进行登录

@RequestMapping("/sso/doLoginByTicket")

@TenantIgnore

public SaResult doLoginByTicket(String ticket) {

String username = (String) SaSsoProcessor.instance.checkTicket(ticket, "/sso/doLoginByTicket");

if (username != null) {

//查询在当前系统是否存在相同账号

SysUser sysUser = sysUserService.findSysUserByUsername(username);

if (sysUser == null) {

return SaResult.error("当前系统无认证中心统一账户");

}

StpUtil.login(username);

return SaResult.data(StpUtil.getTokenValue());

}

return SaResult.error("无效ticket：" + ticket);

}

}

# sso

## SsoClientController.java

/\*\*

\* Sa-Token-SSO Client端 Controller

\*

\* @author kong

\*/

@RestController

@Slf4j

public class SsoClientController {

/\*

\* SSO-Client端：处理所有SSO相关请求

\* http://{host}:{port}/sso/login -- Client端登录地址，接受参数：back=登录后的跳转地址

\* http://{host}:{port}/sso/logout -- Client端单点注销地址（isSlo=true时打开），接受参数：back=注销后的跳转地址

\* http://{host}:{port}/sso/logoutCall -- Client端单点注销回调地址（isSlo=true时打开），此接口为框架回调，开发者无需关心

\*/

@RequestMapping("/sso/\*")

@TenantIgnore

public Object ssoRequest() {

return SaSsoProcessor.instance.clientDister();

}

// 配置SSO相关参数

@Autowired

@TenantIgnore

private void configSso(SaSsoConfig sso) {

// 配置Http请求处理器

sso.setSendHttp(url -> {

log.debug("------ 发起请求：" + url);

return Forest.get(url).executeAsString();

});

}

// 查询我的账号信息

@RequestMapping("/sso/myinfo")

@TenantIgnore

public Object myinfo() {

Object userinfo = SaSsoUtil.getUserinfo(StpUtil.getLoginId());

log.debug("--------info：" + userinfo);

return userinfo;

}

}

# sys

## FileUploadController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/3/17 11:21

\*/

@RestController

@RequestMapping("file")

public class FileUploadController {

@Autowired

MinioUtil minioUtil;

@PostMapping("upload")

public Object upload(@RequestParam(value = "file") MultipartFile file) throws IOException {

String url = minioUtil.upload(file);

return SaResult.data(url);

}

@PostMapping("/uploadFile")

public SaResult uploadFile(@RequestParam(value = "file") MultipartFile file) {

return SaResult.ok().setData(minioUtil.uploadFile(file));

}

@GetMapping("down")

public void down(String fileName, HttpServletResponse response) {

minioUtil.download(fileName, response);

}

}

## SysDictCategoryController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/15 15:57

\*/

@RestController

@RequestMapping("sysDictCategory")

@Api(tags = "字典分类")

@RequiredArgsConstructor

public class SysDictCategoryController extends BaseWebController {

private final ISysDictCategoryService sysDictCategoryService;

private final ISysDictService sysDictService;

@GetMapping("listPage")

public Object listPage(@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer pageSize,

String categoryName,

String categoryKey) {

// String tenantId = this.getTenantId();

LambdaQueryWrapper<SysDictCategory> queryWrapper = new QueryWrapper<SysDictCategory>().lambda()

.like(!Strings.isNullOrEmpty(categoryName), SysDictCategory::getCategoryName, categoryName)

.like(!Strings.isNullOrEmpty(categoryKey), SysDictCategory::getCategoryKey, categoryKey)

.orderByDesc(SysDictCategory::getCreateTime);

// if (!SystemConstant.SYSTEM\_TENANT\_ID.equals(tenantId)) {

// queryWrapper.eq(true, SysDictCategory::getTenantId, tenantId);

// }

Page<SysDictCategory> page = sysDictCategoryService.page(new Page<>(current, pageSize), queryWrapper);

return SaResult.data(page);

}

/\*\*

\* 添加数据字典

\*

\* @return

\*/

@PostMapping("add")

public SaResult add(@RequestBody SysDictCategory sysDictCategory) {

long count=sysDictCategoryService.count(new QueryWrapper<SysDictCategory>().lambda()

.eq(SysDictCategory::getCategoryKey,sysDictCategory.getCategoryKey())

);

if (count>0){

throw new JcglException("当前字典类别key不能重复");

}

sysDictCategory.setTenantId(this.getTenantId());

return SaResult.ok("添加成功").setData(sysDictCategoryService.save(sysDictCategory));

}

@ApiOperation("修改数据字典分类")

@PostMapping("update")

@Transactional(rollbackFor = JcglException.class)

public SaResult update(@RequestBody SysDictCategory sysDictCategory) {

sysDictCategoryService.updateById(sysDictCategory);

long count=sysDictCategoryService.count(new QueryWrapper<SysDictCategory>().lambda()

.eq(SysDictCategory::getCategoryKey,sysDictCategory.getCategoryKey())

);

if (count>1){

throw new JcglException("当前字典类别key不能重复");

}

List<SysDict> sysDictList = sysDictService.list(new QueryWrapper<SysDict>().lambda().eq(SysDict::getSysDictCategoryId, sysDictCategory.getId()));

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

sysDictList.forEach(sysDict -> {

sysDict.setType(sysDictCategory.getCategoryKey());

sysDictService.updateById(sysDict);

});

}

return SaResult.ok("修改成功");

}

@DeleteMapping("delete/{id}")

public SaResult update(@PathVariable String id) {

if (sysDictService.count(new QueryWrapper<SysDict>().lambda().eq(SysDict::getSysDictCategoryId, id)) > 0) {

throw new JcglException("当前数据字典存在不能删除!");

}

return SaResult.ok("删除成功").setData(sysDictCategoryService.removeById(id));

}

@GetMapping("/initDictData")

@ApiOperation("初始化当前租户字典数据或者系统升级")

public SaResult initdefault() {

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

sysDictCategoryService.initDictData(tenantId);

return SaResult.ok();

}

@GetMapping("/dictDataReset")

@ApiOperation("字典数据初始化为系统数据")

public SaResult dictDataReset() {

String tenantId = TenantContextHolder.getContext().getTenantContextInfo().getTenantId();

sysDictCategoryService.dictDataReset(tenantId);

return SaResult.ok();

}

}

## SysDictController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/15 9:17

\*/

@RestController

@RequestMapping("sysDict")

@RequiredArgsConstructor

@Api(tags = "数据字典")

public class SysDictController extends BaseWebController {

private final ISysDictService sysDictService;

private final ISysDictCategoryService sysDictCategoryService;

/\*\*

\* 查询全部数据字典

\*

\* @return

\*/

@ApiOperation("查询全部")

@GetMapping("listAll")

public Object listAll() {

List<SysDict> sysDictList=sysDictService.findListAll(getTenantId());

return SaResult.data(sysDictList);

}

@ApiOperation("分页查询")

@GetMapping("listPage")

public Object listPage(@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer pageSize,

String sysDictCategoryId) {

Page<SysDict> page = sysDictService.page(new Page<>(current, pageSize), new QueryWrapper<SysDict>().lambda()

.eq(SysDict::getSysDictCategoryId,sysDictCategoryId)

.orderByAsc(SysDict::getSort)

);

return SaResult.data(page);

}

@ApiOperation("添加数据字典")

@PostMapping("add")

public SaResult add(@RequestBody SysDict sysDict) {

long count=sysDictService.count(new QueryWrapper<SysDict>().lambda()

.eq(SysDict::getVal,sysDict.getVal())

.eq(SysDict::getSysDictCategoryId,sysDict.getSysDictCategoryId()));

if (count>0){

throw new JcglException("当前字典类目下值不能重复");

}

sysDict.setType(sysDictCategoryService.getById(sysDict.getSysDictCategoryId()).getCategoryKey());

return SaResult.ok("添加成功").setData(sysDictService.save(sysDict));

}

@ApiOperation("修改数据字典")

@PostMapping("update")

@Transactional(rollbackFor = JcglException.class)

public SaResult update(@RequestBody SysDict sysDict) {

sysDict.setType(sysDictCategoryService.getById(sysDict.getSysDictCategoryId()).getCategoryKey());

sysDictService.updateById(sysDict);

long count=sysDictService.count(new QueryWrapper<SysDict>().lambda()

.eq(SysDict::getVal,sysDict.getVal())

.eq(SysDict::getSysDictCategoryId,sysDict.getSysDictCategoryId()));

if (count>1){

throw new JcglException("当前字典类目下值不能重复");

}

return SaResult.ok("修改成功");

}

@DeleteMapping("delete/{id}")

public SaResult update(@PathVariable String id) {

return SaResult.ok("删除成功").setData(sysDictService.removeById(id));

}

}

## SysFlowRoleController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/30 13:50

\*/

@Api(tags = "用户流程角色")

@RestController

@RequestMapping("/sysFlowRole")

public class SysFlowRoleController extends BaseWebController {

@Autowired

ISysFlowRoleService sysFlowRoleService;

@ApiOperation("分页查询流程角色")

@GetMapping("listPage")

public SaResult listPage(@RequestParam Integer current,

@RequestParam Integer pageSize,

String name,

String code) {

IPage<SysFlowRole> sysFlowRoleIPage = sysFlowRoleService.page(

new Page<>(current, pageSize),

new QueryWrapper<SysFlowRole>().lambda()

.like(!Strings.isNullOrEmpty(code), SysFlowRole::getCode, code)

.like(!Strings.isNullOrEmpty(name), SysFlowRole::getName, name));

return SaResult.data(sysFlowRoleIPage);

}

@ApiOperation("查询全部流程角色")

@GetMapping("listAll")

public SaResult listAll() {

return SaResult.data(sysFlowRoleService.list());

}

@ApiOperation("新增流程角色")

@PostMapping("add")

public SaResult add(@RequestBody SysFlowRole sysFlowRole) {

if (sysFlowRoleService.getOne(new QueryWrapper<SysFlowRole>().lambda().eq(SysFlowRole::getCode, sysFlowRole.getCode())) != null) {

throw new JcglException("角色编码不能重复");

}

return SaResult.ok("添加成功").setData(sysFlowRoleService.save(sysFlowRole));

}

@ApiOperation("修改流程角色")

@PostMapping("update")

public SaResult update(@RequestBody SysFlowRole sysFlowRole) {

return SaResult.ok("修改成功").setData(sysFlowRoleService.updateById(sysFlowRole));

}

@ApiOperation("删除流程角色")

@DeleteMapping("delete/{id}")

public SaResult delete(@PathVariable String id) {

return SaResult.ok("删除成功").setData(sysFlowRoleService.removeById(id));

}

}

## SysMenuController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/14 8:56

\*/

@RestController

@RequestMapping("sysMenu")

public class SysMenuController extends BaseWebController {

@Autowired

ISysMenuService sysMenuService;

@GetMapping("menuListTree")

public Object menuListTree(String title,String menuSys) {

return SaResult.data(sysMenuService.findMenuListTree(title,menuSys));

}

/\*\*

\* 新增

\*

\* @param sysMenu

\* @return

\*/

@PostMapping("add")

public Object add(@RequestBody @Validated SysMenu sysMenu) {

if (Strings.isNullOrEmpty(sysMenu.getParentId())) {

sysMenu.setParentId("0");

}

return SaResult.ok("保存成功").setData(sysMenuService.save(sysMenu));

}

/\*\*

\* 删除菜单

\*

\* @param id

\* @return

\*/

@DeleteMapping("delete/{id}")

public Object delete(@PathVariable String id) {

return SaResult.ok("删除成功").setData(sysMenuService.removeById(id));

}

/\*\*

\* 删除菜单

\*

\* @param sysMenu

\* @return

\*/

@PostMapping("update")

public Object update(@RequestBody SysMenu sysMenu) {

if (Strings.isNullOrEmpty(sysMenu.getParentId())) {

sysMenu.setParentId("0");

}

return SaResult.ok("修改成功").setData(sysMenuService.updateById(sysMenu));

}

}

## SysOrgController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/22 16:48

\*/

@RestController

@RequestMapping("sysOrg")

public class SysOrgController extends BaseWebController {

@Autowired

ISysOrgService sysOrgService;

/\*\*

\* 查询组织树

\*

\* @return

\*/

@GetMapping("myTree")

public SaResult myTree() {

SysUser sysUser = getUserDetail();

SysOrg sysOrg = sysOrgService.getById(sysUser.getSysOrgId());

if (sysOrg == null) {

return SaResult.ok().setData(new ArrayList<>());

}

List<SysOrg> sysOrgs = sysOrgService.findTreeByOrg(sysOrg);

return SaResult.ok().setData(sysOrgs);

}

@GetMapping("subOrg")

public SaResult subOrg(String sysOrgId,

@RequestParam(defaultValue = "1") Integer current,

@RequestParam(defaultValue = "10") Integer pageSize) {

SysUser sysUser = getUserDetail();

SysOrg psysOrg = sysOrgService.getById(sysUser.getSysOrgId());

if (psysOrg == null) {

return SaResult.error("当前登录用户组织信息不对");

}

if (Strings.isNullOrEmpty(sysOrgId)) {

sysOrgId = psysOrg.getId();

}

//判断当前新增的父级ID是否在登录人权限内

Set<String> authOrgIds = sysOrgService.findAuthIdsByOrgId(psysOrg.getParentId());

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

return SaResult.error("当前所选父级组织不在权限范围内");

} else {

if (!authOrgIds.contains(sysOrgId)) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

}

return SaResult.data(sysOrgService.page(new Page<>(current, pageSize),

new QueryWrapper<SysOrg>().lambda().eq(SysOrg::getParentId, sysOrgId).orderByAsc(SysOrg::getSort)));

}

@PostMapping("add")

public SaResult add(@RequestBody SysOrg sysOrg) {

SysUser sysUser = getUserDetail();

SysOrg psysOrg = sysOrgService.getById(sysUser.getSysOrgId());

if (psysOrg == null) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

//判断当前新增的父级ID是否在登录人权限内

Set<String> authOrgIds = sysOrgService.findAuthIdsByOrgId(psysOrg.getParentId());

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

return SaResult.error("当前所选父级组织不在权限范围内");

} else {

if (!authOrgIds.contains(sysOrg.getParentId())) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

}

//顶级节点第一级才是公司租户级别

if ("0".equals(sysUser.getSysOrgId())) {

sysOrg.setOrgType(1);

} else {

sysOrg.setOrgType(2);

}

sysOrg.setCreateBy(sysUser.getId());

sysOrg.setCreateTime(new Date());

sysOrgService.save(sysOrg);

//设置分级列表ID和完整部门路径

SupOrgListVO supOrgListVO = sysOrgService.findSupOrgListVOByOrgId(sysOrg.getId());

sysOrg.setAncestors(supOrgListVO.getAncestors() != null ? Joiner.on(",").join(supOrgListVO.getAncestors()) : null);

sysOrg.setFullOrgName(supOrgListVO.getPOrgNames() != null ? Joiner.on("-").join(supOrgListVO.getPOrgNames()) : null);

sysOrgService.updateById(sysOrg);

return SaResult.ok("保存成功");

}

@PostMapping("update")

public SaResult update(@RequestBody SysOrg sysOrg) {

SysUser sysUser = getUserDetail();

SysOrg psysOrg = sysOrgService.getById(sysUser.getSysOrgId());

if (psysOrg == null) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

//判断当前新增的父级ID是否在登录人权限内

Set<String> authOrgIds = sysOrgService.findAuthIdsByOrgId(psysOrg.getParentId());

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

return SaResult.error("当前所选父级组织不在权限范围内");

} else {

if (!authOrgIds.contains(sysOrg.getParentId())) {

return SaResult.error("当前所选父级组织不在权限范围内");

}

}

//顶级节点第一级才是公司租户级别

if ("0".equals(sysUser.getSysOrgId())) {

sysOrg.setOrgType(1);

} else {

sysOrg.setOrgType(2);

}

sysOrg.setUpdateBy(sysUser.getId());

sysOrg.setUpdateTime(new Date());

sysOrgService.updateById(sysOrg);

SupOrgListVO supOrgListVO = sysOrgService.findSupOrgListVOByOrgId(sysOrg.getId());

sysOrg.setAncestors(supOrgListVO.getAncestors() != null ? Joiner.on(",").join(supOrgListVO.getAncestors()) : null);

sysOrg.setFullOrgName(supOrgListVO.getPOrgNames() != null ? Joiner.on("-").join(supOrgListVO.getPOrgNames()) : null);

sysOrgService.updateById(sysOrg);

return SaResult.ok("保存成功");

}

@DeleteMapping("delete/{sysOrgId}")

public SaResult delete(@PathVariable String sysOrgId) {

SysUser sysUser = getUserDetail();

SysOrg psysOrg = sysOrgService.getById(sysUser.getSysOrgId());

//判断当前新增的父级ID是否在登录人权限内

Set<String> authOrgIds = sysOrgService.findAuthIdsByOrgId(psysOrg.getParentId());

if (authOrgIds == null || authOrgIds.isEmpty() || !authOrgIds.contains(sysOrgId)) {

return SaResult.error("没有删除该组织的权限");

}

sysOrgService.removeById(sysOrgId);

return SaResult.ok("删除成功");

}

}

## SysPostController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/23 17:07

\*/

@RestController

@RequestMapping("sysPost")

public class SysPostController {

@Autowired

ISysPostService sysPostService;

@GetMapping("listPage")

public SaResult listPage(@RequestParam Integer current,

@RequestParam Integer pageSize, String postName) {

return SaResult.data(sysPostService.page(new Page<>(current, pageSize), new QueryWrapper<SysPost>().lambda()

.like(!Strings.isNullOrEmpty(postName), SysPost::getPostName, postName)));

}

@GetMapping("listAll")

public SaResult listAll() {

return SaResult.data(sysPostService.list());

}

@PostMapping("add")

public SaResult add(@RequestBody SysPost sysPost) {

return SaResult.ok("添加成功").setData(sysPostService.save(sysPost));

}

@PostMapping("update")

public SaResult update(@RequestBody SysPost sysPost) {

return SaResult.ok("添加成功").setData(sysPostService.updateById(sysPost));

}

@DeleteMapping("delete/{postId}")

public SaResult delete(@PathVariable String postId) {

return SaResult.ok("删除成功").setData(sysPostService.removeById(postId));

}

}

## SysRoleController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/2/23 13:34

\*/

@RestController

@RequestMapping("sysRole")

public class SysRoleController extends BaseWebController {

@Autowired

ISysRoleService sysRoleService;

@Autowired

ISysRoleMenuService sysRoleMenuService;

@GetMapping("listPage")

public SaResult listPage(@RequestParam Integer current, @RequestParam Integer pageSize, String name) {

return SaResult.data(sysRoleService.page(new Page<>(current, pageSize), new QueryWrapper<SysRole>().lambda()

.like(!Strings.isNullOrEmpty(name), SysRole::getRoleName, name)));

}

@GetMapping("listAll")

public SaResult listAll() {

return SaResult.data(sysRoleService.list());

}

@PostMapping("add")

public SaResult add(@RequestBody SysRole sysRole) {

if (sysRoleService.getOne(new QueryWrapper<SysRole>().lambda().eq(SysRole::getCode,sysRole.getCode()))!=null){

throw new JcglException("角色编码不能重复");

}

return SaResult.ok("添加成功").setData(sysRoleService.save(sysRole));

}

@PostMapping("update")

public SaResult update(@RequestBody SysRole sysRole) {

return SaResult.ok("修改成功").setData(sysRoleService.updateById(sysRole));

}

@DeleteMapping("delete/{roleId}")

public SaResult delete(@PathVariable String roleId) {

return SaResult.ok("删除成功").setData(sysRoleService.removeById(roleId));

}

@PostMapping("roleAuthMenusSave")

public SaResult roleAuthMenusSave(@RequestBody RoleAuthMenusDTO roleAuthMenusDTO) {

SysUser sysUser = getUserDetail();

sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().lambda().eq(SysRoleMenu::getSysRoleId, roleAuthMenusDTO.getRoleId()));

List<SysRoleMenu> sysRoleMenuList = new ArrayList<>();

roleAuthMenusDTO.getMenuIds().forEach(menuId -> {

SysRoleMenu sysRoleMenu = new SysRoleMenu();

sysRoleMenu.setSysRoleId(roleAuthMenusDTO.getRoleId());

sysRoleMenu.setSysMenuId(menuId);

sysRoleMenu.setCreateTime(new Date());

sysRoleMenu.setCreateBy(sysUser.getId());

sysRoleMenuList.add(sysRoleMenu);

});

sysRoleMenuService.saveBatch(sysRoleMenuList);

//当拥有当前角色的用户踢下线

List<String> usernames = sysRoleService.findUsernameByRoleId(roleAuthMenusDTO.getRoleId());

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

usernames.forEach(StpUtil::logout);

}

return SaResult.ok("保存成功");

}

@GetMapping("menuIdsByRoleId/{roleId}")

public SaResult roleAuthMenus(@PathVariable String roleId) {

List<SysRoleMenu> sysRoleMenuList = sysRoleMenuService.list(new QueryWrapper<SysRoleMenu>().lambda().eq(SysRoleMenu::getSysRoleId, roleId));

List<String> menuIds = sysRoleMenuList.stream().map(SysRoleMenu::getSysMenuId).collect(Collectors.toList());

return SaResult.ok("保存成功").setData(menuIds);

}

}

## SysTenantController.java

/\*\*

\* @Description:

\* @Author: yb

\* @Date: 2023/5/9 17:21

\*/

@Api(tags = "租户列表")

@RestController

@RequestMapping("sysTenant")

@RequiredArgsConstructor

public class SysTenantController {

private final ISysTenantService sysTenantService;

@GetMapping("listPage")

@ApiOperation("分页查询")

@ApiResponses({

@ApiResponse(code = 200, response = SysTenant.class, message = "查询成功")

})

public SaResult listPage(@Validated SysTenantPageDTO sysTenantPageDTO) {

Page<SysTenantVO> sysTenantVOPage = sysTenantService.findPage(new Page<>(sysTenantPageDTO.getCurrent(), sysTenantPageDTO.getPageSize()), sysTenantPageDTO.getTenantName());

return SaResult.data(sysTenantVOPage);

}

@PostMapping("add")

@ApiOperation("新增租户")

@ApiResponses({

@ApiResponse(code = 200, response = SaResult.class, message = "返回成功")

})

public SaResult add(@Validated @RequestBody SysTenantAddDTO sysTenantAddDTO) {

sysTenantService.addSysTenant(sysTenantAddDTO);

return SaResult.ok("新增成功");

}

@PutMapping("update")

@ApiOperation("修改租户")

@ApiResponses({

@ApiResponse(code = 200, response = SaResult.class, message = "返回成功")

})

public SaResult update(@Validated @RequestBody SysTenant sysTenant) {

sysTenantService.updateTenant(sysTenant);

return SaResult.ok("修改成功");

}

@DeleteMapping("delete/{id}")

@ApiOperation("删除租户")

@ApiResponses({

@ApiResponse(code = 200, response = SaResult.class, message = "返回成功")

})

public SaResult delete(@PathVariable String id) {

sysTenantService.removeTenant(id);

return SaResult.ok("删除成功");

}

}

## SysTenantPkgController.java

/\*\*

\* @Description: 租户套餐管理

\* @Author: yb

\* @Date: 2023/5/10 9:58

\*/

@RestController

@RequestMapping("sysTenantPkg")

@Api(tags = "租户套餐")

@RequiredArgsConstructor

public class SysTenantPkgController {

private final ISysTenantPkgService sysTenantPkgService;

@ApiOperation("分页查询")

@GetMapping("listPage")

@ApiResponses({

@ApiResponse(code = 200,response = SysTenantPkgVO.class,message = "查询成功")

})

public SaResult listPage(@Validated SysTenantPkgPageDTO sysTenantPkgPageDTO){

Page<SysTenantPkgVO> sysTenantPkgVOPage=sysTenantPkgService.findPage(new Page<>(sysTenantPkgPageDTO.getCurrent(), sysTenantPkgPageDTO.getPageSize()),sysTenantPkgPageDTO.getPkgName());

return SaResult.data(sysTenantPkgVOPage);

}

@ApiOperation("查询全部套餐")

@GetMapping("listAll")

@ApiResponses({

@ApiResponse(code = 200,response = SysTenantPkg.class,message = "查询成功")

})

public SaResult listAll(){

return SaResult.data(sysTenantPkgService.list());

}

@ApiOperation("新增套餐")

@PostMapping("add")

@ApiResponses({

@ApiResponse(code = 200,response = SaResult.class,message = "操作成功")

})

public SaResult add(@Validated @RequestBody SysTenantPkgDTO sysTenantPkgDTO){

sysTenantPkgDTO.setId(null);

sysTenantPkgService.savePkg(sysTenantPkgDTO);

return SaResult.ok("新增成功");

}

@ApiOperation("修改套餐")

@PutMapping("update")

@ApiResponses({

@ApiResponse(code = 200,response = SaResult.class,message = "操作成功")

})

public SaResult update(@Validated @RequestBody SysTenantPkgDTO sysTenantPkgDTO){

//修改套餐

sysTenantPkgService.updatePkg(sysTenantPkgDTO);

return SaResult.ok("修改成功");

}

@ApiOperation("删除套餐")

@DeleteMapping("delete/{id}")

@ApiResponses({

@ApiResponse(code = 200,response = SaResult.class,message = "操作成功")

})

public SaResult add(@PathVariable String id){

sysTenantPkgService.removePkg(id);

return SaResult.ok("删除成功");

}

}

## SysUserController.java

@RestController

@RequestMapping("sysUser")

@Api(tags = "用户管理")

public class SysUserController extends BaseWebController {

@Autowired

ISysUserService sysUserService;

@Autowired

ISysMenuService sysMenuService;

@Autowired

ISysUserRoleService sysUserRoleService;

@Autowired

ISysUserFlowRoleService sysUserFlowRoleService;

@Autowired

ISysUserOrgService sysUserOrgService;

@GetMapping("listPage")

@ApiResponses({@ApiResponse(code = 200,message = "返回成功",response =SysUserVO.class )})

public SaResult listPage(@RequestParam Integer current,

@RequestParam Integer pageSize,

String sysOrgId,

String empName,

String mobile) {

if (Strings.isNullOrEmpty(sysOrgId)){

sysOrgId=getUserDetail().getSysOrgId();

}

Page<SysUserVO> page = sysUserService.findUserPage(new Page<>(current, pageSize), sysOrgId, empName, mobile);

page.getRecords().forEach(sysUserVO -> {

List<SysRole> sysRoles=sysUserService.findRolesByUserId(sysUserVO.getId());

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

sysUserVO.setRoleNames(Joiner.on(",").join(sysRoles.stream().map(SysRole::getRoleName).collect(Collectors.toList())));

sysUserVO.setRoleIds(sysRoles.stream().map(SysRole::getId).collect(Collectors.toList()));

}

sysUserVO.setFlowRoleIds(sysUserFlowRoleService.list(new QueryWrapper<SysUserFlowRole>().lambda()

.eq(SysUserFlowRole::getSysUserId,sysUserVO.getId())

.select(SysUserFlowRole::getSysFlowRoleId)).stream().map(SysUserFlowRole::getSysFlowRoleId).collect(Collectors.toList()));

//查询兼职组织

sysUserVO.setUserOrgs(sysUserOrgService.list(new QueryWrapper<SysUserOrg>().lambda().eq(SysUserOrg::getSysUserId,sysUserVO.getId())));

});

return SaResult.data(page);

}

/\*\*

\* 查询我的信息

\*

\* @return

\*/

@GetMapping("myUserInfo")

public Object myUserInfo() {

SysUserVO sysUserVO = sysUserService.findUserVOByUserId(getUserDetail().getId());

return SaResult.data(sysUserVO);

}

@GetMapping("getMyMenusPc")

public Object getMyMenusPc( ) {

SysUser sysUser = getUserDetail();

String userId = sysUser.getId();

List<SysMenu> sysMenuList = sysMenuService.findMyMenuListTree(userId,"pc");

return SaResult.data(sysMenuList);

}

@GetMapping("getMyMenusApp")

public Object getMyMenusApp( ) {

SysUser sysUser = getUserDetail();

String userId = sysUser.getId();

List<SysMenu> sysMenuList = sysMenuService.findMyMenuListTree(userId,"app");

return SaResult.data(sysMenuList);

}

@ApiOperation("新增用户")

@PostMapping("add")

@Transactional

public SaResult add(@Validated @RequestBody SysUserAddDTO sysUserAddDTO) {

SysUser sysUser = new SysUser();

BeanUtils.copyProperties(sysUserAddDTO, sysUser);

String password = RandomUtil.randomString(8);

sysUser.setPassword(BCrypt.hashpw(password));

sysUserService.save(sysUser);

//功能角色新增

List<SysUserRole> sysUserRoles=new ArrayList<>();

sysUserAddDTO.getRoleIds().forEach(roleId->{

SysUserRole sysUserRole=new SysUserRole();

sysUserRole.setSysRoleId(roleId);

sysUserRole.setSysUserId(sysUser.getId());

sysUserRoles.add(sysUserRole);

});

if(!sysUserRoles.isEmpty()){

sysUserRoleService.saveBatch(sysUserRoles);

}

//流程角色新增

List<SysUserFlowRole> sysUserFlowRoles=new ArrayList<>();

sysUserAddDTO.getFlowRoleIds().forEach(flowRoleId->{

SysUserFlowRole sysUserFlowRole=new SysUserFlowRole();

sysUserFlowRole.setSysFlowRoleId(flowRoleId);

sysUserFlowRole.setSysUserId(sysUser.getId());

sysUserFlowRoles.add(sysUserFlowRole);

});

if (!sysUserFlowRoles.isEmpty()){

sysUserFlowRoleService.saveBatch(sysUserFlowRoles);

}

//兼职组织添加

if (sysUserAddDTO.getUserOrgs()!=null&&!sysUserAddDTO.getUserOrgs().isEmpty()){

List<SysUserOrg> sysUserOrgs=new ArrayList<>();

sysUserAddDTO.getUserOrgs().forEach(sysUserOrg -> {

sysUserOrg.setSysUserId(sysUser.getId());

sysUserOrgs.add(sysUserOrg);

});

if (!sysUserOrgs.isEmpty()){

sysUserOrgService.saveBatch(sysUserOrgs);

}

}

return SaResult.ok("新增成功:初始密码为:" + password);

}

@ApiOperation("修改用户信息")

@PostMapping("update")

@Transactional

public SaResult update(@Validated @RequestBody SysUserUpdateDTO sysUserUpdateDTO) {

SysUser sysUser = sysUserService.getById(sysUserUpdateDTO.getId());

BeanUtils.copyProperties(sysUserUpdateDTO, sysUser);

sysUserService.updateById(sysUser);

sysUserRoleService.remove(new QueryWrapper<SysUserRole>().lambda().eq(SysUserRole::getSysUserId,sysUser.getId()));

sysUserFlowRoleService.remove(new QueryWrapper<SysUserFlowRole>().lambda().eq(SysUserFlowRole::getSysUserId,sysUser.getId()));

//功能角色新增

List<SysUserRole> sysUserRoles=new ArrayList<>();

sysUserUpdateDTO.getRoleIds().forEach(roleId->{

SysUserRole sysUserRole=new SysUserRole();

sysUserRole.setSysRoleId(roleId);

sysUserRole.setSysUserId(sysUser.getId());

sysUserRoles.add(sysUserRole);

});

if(!sysUserRoles.isEmpty()){

sysUserRoleService.saveBatch(sysUserRoles);

}

//流程角色新增

List<SysUserFlowRole> sysUserFlowRoles=new ArrayList<>();

sysUserUpdateDTO.getFlowRoleIds().forEach(flowRoleId->{

SysUserFlowRole sysUserFlowRole=new SysUserFlowRole();

sysUserFlowRole.setSysFlowRoleId(flowRoleId);

sysUserFlowRole.setSysUserId(sysUser.getId());

sysUserFlowRoles.add(sysUserFlowRole);

});

if (!sysUserFlowRoles.isEmpty()){

sysUserFlowRoleService.saveBatch(sysUserFlowRoles);

}

//兼职组织删除

sysUserOrgService.remove(new QueryWrapper<SysUserOrg>().lambda().eq(SysUserOrg::getSysUserId,sysUser.getId()));

//兼职组织添加

if (sysUserUpdateDTO.getUserOrgs()!=null&&!sysUserUpdateDTO.getUserOrgs().isEmpty()){

List<SysUserOrg> sysUserOrgs=new ArrayList<>();

sysUserUpdateDTO.getUserOrgs().forEach(sysUserOrg -> {

sysUserOrg.setSysUserId(sysUser.getId());

sysUserOrgs.add(sysUserOrg);

});

if (!sysUserOrgs.isEmpty()){

sysUserOrgService.saveBatch(sysUserOrgs);

}

}

StpUtil.logout(sysUser.getUsername());

return SaResult.ok("修改成功,当前用户已经强制下线");

}

//删除用户

@DeleteMapping("delete/{sysUserId}")

@Transactional

public SaResult delete(@PathVariable String sysUserId) {

SysUser sysUser = sysUserService.getById(sysUserId);

if (sysUser==null){

throw new JcglException("不存在的用户");

}

sysUserService.removeById(sysUserId);

sysUserRoleService.remove(new QueryWrapper<SysUserRole>().lambda().eq(SysUserRole::getSysUserId,sysUserId));

sysUserFlowRoleService.remove(new QueryWrapper<SysUserFlowRole>().lambda().eq(SysUserFlowRole::getSysUserId,sysUserId));

StpUtil.logout(sysUser.getUsername());

return SaResult.ok("删除成功,当前用户已经强制下线");

}

//停用用户

@PostMapping("stop/{sysUserId}")

public SaResult stop(@PathVariable String sysUserId) {

sysUserService.updateEnabled(sysUserId, 0);

//员工状态1 在职2 离职

sysUserService.updateEmpStatus(sysUserId, 2);

SysUser sysUser = sysUserService.getById(sysUserId);

StpUtil.logout(sysUser.getUsername());

return SaResult.ok("停用成功,当前用户已经强制下线");

}

//启用用户

@PostMapping("start/{sysUserId}")

public SaResult start(@PathVariable String sysUserId) {

sysUserService.updateEnabled(sysUserId, 1);

sysUserService.updateEmpStatus(sysUserId, 1);

return SaResult.ok("启用成功");

}

/\*\*

\* 重置密码

\* @param resetPasswordDTO

\* @return

\*/

@PostMapping("resetPassword")

public SaResult resetPassword(@RequestBody @Validated ResetPasswordDTO resetPasswordDTO) {

SysUser sysUser = sysUserService.getById(resetPasswordDTO.getSysUserId());

if (sysUser == null) {

return SaResult.error("用户不存在");

} else {

sysUser.setPassword(BCrypt.hashpw(resetPasswordDTO.getPassword()));

sysUserService.updateById(sysUser);

}

//重置成功后把该用户踢下线

StpUtil.logout(sysUser.getUsername());

return SaResult.ok("重置成功,当前用户已经强制下线");

}

/\*\*

\* 查询用户拥有的角色

\*

\* @param sysUserId

\* @return

\*/

@GetMapping("rolesByUerId/{sysUserId}")

public SaResult getRolesByUserId(@PathVariable String sysUserId) {

return SaResult.ok("查询成功").setData(sysUserRoleService.list(new QueryWrapper<SysUserRole>()

.lambda().eq(SysUserRole::getSysUserId, sysUserId)));

}

//重新授权角色

@PostMapping("saveUserRoles")

public SaResult saveUserRoles(@RequestBody @Validated SaveUserRolesDTO saveUserRolesDTO) {

SysUser sysUser = getUserDetail();

//获取授权用户

SysUser sysUserAuth = sysUserService.getById(saveUserRolesDTO.getSysUserId());

StpUtil.logout(sysUserAuth.getUsername());

sysUserRoleService.remove(new QueryWrapper<SysUserRole>().lambda().eq(SysUserRole::getSysUserId, saveUserRolesDTO.getSysUserId()));

List<SysUserRole> sysUserRoles = new ArrayList<>();

saveUserRolesDTO.getRoleIds().forEach(roleId -> {

SysUserRole sysUserRole = new SysUserRole();

sysUserRole.setSysUserId(saveUserRolesDTO.getSysUserId());

sysUserRole.setSysRoleId(roleId);

sysUserRole.setCreateTime(new Date());

sysUserRole.setCreateBy(sysUser.getId());

sysUserRoles.add(sysUserRole);

});

sysUserRoleService.saveBatch(sysUserRoles);

return SaResult.ok("保存成功,关联角色的用户已经强制下线").setData(true);

}

@PostMapping("updateMyInfo")

public SaResult updateMyInfo(@RequestBody UpdateMyInfoDTO updateMyInfoDTO) {

SysUser sysUser = getUserDetail();

BeanUtils.copyProperties(updateMyInfoDTO, sysUser);

return SaResult.ok("保存成功").setData(sysUserService.updateById(sysUser));

}

@PostMapping("exitLogin")

public SaResult exitLogin() {

StpUtil.logout();

return SaResult.ok("退出成功");

}

}