/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 基于SpringBoot框架的信管专业课程实训系统——前30页源程序代码 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.gdou.teaching.controller.student;

import com.gdou.teaching.Enum.\*;

import com.gdou.teaching.dataobject.\*;

import com.gdou.teaching.dto.CourseDTO;

import com.gdou.teaching.dto.ExperimentDTO;

import com.gdou.teaching.dto.RecordDTO;

import com.gdou.teaching.dto.UserDTO;

import com.gdou.teaching.exception.TeachingException;

import com.gdou.teaching.mbg.model.Achievement;

import com.gdou.teaching.service.\*;

import com.gdou.teaching.util.ResultVOUtil;

import com.gdou.teaching.vo.CourseMainPageVO;

import com.gdou.teaching.vo.CourseVO;

import com.gdou.teaching.vo.ExperimentVO;

import com.gdou.teaching.vo.ResultVO;

import com.gdou.teaching.web.Auth;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.BeanUtils;

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

import org.springframework.util.StringUtils;

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

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/student/course")

@Slf4j

@Auth(user=UserIdentEnum.SUTUDENT)

public class StudentCourseController {

@Autowired

private CourseService courseService;

@Autowired

private ExperimentService experimentService;

@Autowired

private AchievementService achievementService;

@Autowired

private HostHolder hostHolder;

@Autowired

private RecordService recordService;

@Autowired

private FileService fileService;

/\*\*

\* 课程主页

\* @param courseId 课程ID

\* 通过课程ID查询下属的实验列表，然后通过UserId和ExperimentID查询用户的完成情况。返回给前端

\* @return

\*/

@GetMapping(path = "/info/{courseId}")

public ResultVO<CourseVO> info(@PathVariable(value = "courseId") Integer courseId) {

if (courseId == null) {

return ResultVOUtil.fail(ResultEnum.PARAM\_NULL);

}

UserDTO user = hostHolder.getUser();

try {

//查询课程基本信息

CourseDTO courseDTO = courseService.selectOne(courseId);

CourseMainPageVO courseMainPageVO = new CourseMainPageVO();

BeanUtils.copyProperties(courseDTO, courseMainPageVO);

//查询实验列表信息

List<ExperimentDTO> experimentDTOList = experimentService.list(courseId);

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

//结合用户提交记录，将DTO转为VO

List<ExperimentVO> ExperimentVOList = experimentDTOList.stream().map(experimentDTO -> {

ExperimentVO experimentVO = new ExperimentVO();

BeanUtils.copyProperties(experimentDTO, experimentVO);

//设置不需要的字段为空

experimentVO.setCourseId(null);

//设置状态

if(ExperimentStatusEnum.LOCK.getCode().byteValue()==experimentDTO.getExperimentStatus()){

experimentVO.setRecordStatus(RecordStatusEnum.LOCK.getCode());

}else if(ExperimentStatusEnum.END.getCode().byteValue()==experimentDTO.getExperimentStatus()){

experimentVO.setRecordStatus(RecordStatusEnum.FINISH.getCode());

} else{

RecordDTO recordDTO = recordService.selectOne(experimentDTO.getExperimentId(), user.getUserId());

if(recordDTO!=null){

experimentVO.setRecordStatus(recordDTO.getStatus().intValue());

}else{

experimentVO.setRecordStatus(RecordStatusEnum.NOT\_FINISH.getCode());

}

}

return experimentVO;

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

courseMainPageVO.setExperimentDTOList(ExperimentVOList);

}

return ResultVOUtil.success(courseMainPageVO);

} catch (TeachingException e) {

log.info("[StudentCourseController]查询课程主页, 查询异常:{}", e.getMessage());

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR);

}

}

/\*\*

\* 查询课程详情信息

\* @param courseId

\* @return

\*/

@GetMapping("/detail/{courseId}")

public ResultVO detail(@PathVariable(value = "courseId") Integer courseId) {

CourseDTO detail = courseService.detail(courseId);

if(detail==null){

log.info("[StudentCourseController]查询课程详情信息, 查询异常，courseId={}",courseId);

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR);

}

return ResultVOUtil.success(detail);

}

/\*\*

\* 不分页版本-获取课程列表

\* 通过用户ID获取课程列表，并将其分割为结束和未结束两部分

\* @return

\*/

@GetMapping("/listForNoPage")

public ResultVO listForNoPage(@RequestParam(value = "keyword",required = false)String keyword){

UserDTO user = hostHolder.getUser();

//通过ID获取到用户课程数据

List<CourseDTO> list = courseService.listCourseByUserIdAndKeywordForStudent(user.getUserId(),keyword);

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

return ResultVOUtil.success();

}

//拿到了数据，进行分割，将数据分为"未结束"和"已结束"两部分

Map<Boolean, List<CourseVO>> collect = list.stream().map(courseDTO -> {

CourseVO courseVO = new CourseVO();

BeanUtils.copyProperties(courseDTO, courseVO);

return courseVO;

}).collect(Collectors.groupingBy(courseVO -> courseVO.getCourseStatus().intValue() == (CourseStatusEnum.END.getCode())));

//切割正常课程。

List<CourseVO> normal = collect.get(false);

//切割出过期课程

List<CourseVO> end = collect.get(true);

HashMap<String, Object> map = new HashMap<>(2);

map.put("normal", normal);

map.put("end", end);

return ResultVOUtil.success(map);

}

/\*\*

\* 通过课程ID获取该课程的分数

\* @return 该课程下的实验分数。

\*/

@GetMapping("/score/{courseId}")

public ResultVO scoreList(@PathVariable(value = "courseId") Integer courseId){

UserDTO user = hostHolder.getUser();

Achievement achievement = achievementService.getAchievementByUserIdAndCourseId(user.getUserId(), courseId);

Double score = achievement.getCourseAchievement();

//通过课程ID和用户ID获取提交记录列表

List<RecordDTO> recordByUserIdAndCourseId = recordService.getRecordByUserIdAndCourseId(user.getUserId(), courseId);

if(recordByUserIdAndCourseId!=null){

List<HashMap<String, String>> experiments = recordByUserIdAndCourseId.stream().map(r -> {

HashMap<String, String> ex = new HashMap<>(3);

ex.put("experimentName",r.getExperimentName());

ex.put("experimentId", r.getExperimentId().toString());

ex.put("experimentAchievement", r.getExperimentAchievement().toString());

return ex;

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

//提取所需要的字段（实验名-实验ID-实验得分）返回

//获取课程成绩信息

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

map.put("experiments",experiments);

map.put("score",score);

return ResultVOUtil.success(map);

}else {

return ResultVOUtil.success();

}

}

@GetMapping("/resource/{courseId}")

public ResultVO resource(@PathVariable(value = "courseId") Integer courseId, @RequestParam(required = false)String keyword){

if(StringUtils.isEmpty(keyword)){

//通过课程ID获取课程关连的文件

return ResultVOUtil.success(fileService.selectFileByCategoryAndFileCategoryId(FileCategoryEnum.COURSE\_FILE.getCode(), courseId));

}else {

//通过关键字和课程ID获取关联的文件

return ResultVOUtil.success(fileService.selectFileByCategoryAndFileCategoryIdAndKeyword(FileCategoryEnum.COURSE\_FILE.getCode(),courseId,keyword));

}

}

}

package com.gdou.teaching.controller.student;

import com.gdou.teaching.Enum.ResultEnum;

import com.gdou.teaching.Enum.UserIdentEnum;

import com.gdou.teaching.constant.RedisConstant;

import com.gdou.teaching.dataobject.HostHolder;

import com.gdou.teaching.dto.AnswerDTO;

import com.gdou.teaching.dto.ExperimentDTO;

import com.gdou.teaching.dto.RecordDTO;

import com.gdou.teaching.dto.UserDTO;

import com.gdou.teaching.mbg.model.User;

import com.gdou.teaching.service.AnswerService;

import com.gdou.teaching.service.ExperimentService;

import com.gdou.teaching.service.RecordService;

import com.gdou.teaching.util.ResultVOUtil;

import com.gdou.teaching.vo.ResultVO;

import com.gdou.teaching.web.Auth;

import lombok.extern.slf4j.Slf4j;

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

import org.springframework.data.redis.core.StringRedisTemplate;

import org.springframework.stereotype.Controller;

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

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

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

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

import java.util.HashMap;

@Controller

@Slf4j

@RequestMapping("/student/experiment/")

@Auth(user= UserIdentEnum.SUTUDENT)

public class StudentExperimentController {

@Autowired

ExperimentService experimentService;

@Autowired

RecordService recordService;

@Autowired

HostHolder hostHolder;

@Autowired

AnswerService answerService;

@Autowired

StringRedisTemplate redisTemplate;

/\*\*

\* 获取实验信息

\* @param experimentId

\* @return

\*/

@GetMapping("/info/{experimentId}")

@ResponseBody

public ResultVO info(@PathVariable("experimentId") Integer experimentId) {

UserDTO user = hostHolder.getUser();

if (user == null) {

return ResultVOUtil.fail(ResultEnum.USER\_NO\_LOGIN);

}

//实验的具体要求内容和实验文件

ExperimentDTO detail = experimentService.detail(experimentId);

if (detail == null) {

return ResultVOUtil.fail(ResultEnum.EXPERIMENT\_NOT\_EXIST);

}

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

map.put("experiment", detail);

RecordDTO recordDTO = recordService.selectOne(experimentId, user.getUserId());

//若用户为提交过没有记录，则不放提交记录在其中。

if (recordDTO != null) {

map.put("record", recordDTO);

}

return ResultVOUtil.success(map);

}

/\*\*

\* 通过实验ID获取实验的标准答案

\* @param experimentAnswerId

\* 如果查询不为空，在Redis中将对应的查看答案标记将用户设置上去

\* 如果查询结果为空的话返回为空

\* @return

\*/

@GetMapping("/answer/{experimentAnswerId}")

@ResponseBody

public ResultVO answer(@PathVariable("experimentAnswerId")Integer experimentAnswerId){

UserDTO user = hostHolder.getUser();

AnswerDTO detail = answerService.selectOne(experimentAnswerId);

if(detail!=null){

//将用户ID和实验ID在Redis中进行一次标记，之后在Record的时候进行查看答案标记

String key = String.format(RedisConstant.BIZ\_CHECK\_ANSWER, experimentAnswerId);

redisTemplate.opsForSet().add(key,user.getUserId().toString());

return ResultVOUtil.success(detail);

}else {

return ResultVOUtil.success(new AnswerDTO());

}

}

}

package com.gdou.teaching.controller.student;

import com.gdou.teaching.Enum.ResultEnum;

import com.gdou.teaching.Enum.UserIdentEnum;

import com.gdou.teaching.constant.RedisConstant;

import com.gdou.teaching.dataobject.HostHolder;

import com.gdou.teaching.dto.ExperimentDTO;

import com.gdou.teaching.dto.RecordDTO;

import com.gdou.teaching.dto.UserDTO;

import com.gdou.teaching.exception.TeachingException;

import com.gdou.teaching.form.RecordForm;

import com.gdou.teaching.mbg.model.ExperimentMaster;

import com.gdou.teaching.mbg.model.User;

import com.gdou.teaching.service.ExperimentService;

import com.gdou.teaching.service.RecordService;

import com.gdou.teaching.util.ResultVOUtil;

import com.gdou.teaching.vo.ResultVO;

import com.gdou.teaching.web.Auth;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.BeanUtils;

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

import org.springframework.data.redis.core.StringRedisTemplate;

import org.springframework.validation.BindingResult;

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

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

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

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

import javax.validation.Valid;

@RestController

@RequestMapping("/record")

@Slf4j

@Auth(user= UserIdentEnum.SUTUDENT)

public class StudentRecordController {

@Autowired

RecordService recordService;

@Autowired

HostHolder hostHolder;

@Autowired

StringRedisTemplate redisTemplate;

@Autowired

ExperimentService experimentService;

/\*\*

\* 第一次提交完之后需要更新目前已提交的数字

\* @param form

\* @param bindingResult

\* @return

\*/

@PostMapping("/save")

public ResultVO save(@RequestBody @Valid RecordForm form,

BindingResult bindingResult) {

UserDTO user = hostHolder.getUser();

if (bindingResult.hasErrors()) {

log.error("参数不正确：{}" + form);

throw new TeachingException(ResultEnum.PARAM\_ERROR);

}

RecordDTO recordDTO = new RecordDTO();

BeanUtils.copyProperties(form, recordDTO);

recordDTO.setUserId(user.getUserId());

recordDTO.setClassId(user.getClassId());

//需要拿到这个实验的详情数据

ExperimentDTO detail = experimentService.detail(form.getExperimentId());

if(detail.getExperimentAnswerId()!=null){

String key = String.format(RedisConstant.BIZ\_CHECK\_ANSWER, detail.getExperimentAnswerId());

Boolean checkAnswer = redisTemplate.opsForSet().isMember(key, user.getUserId().toString());

recordDTO.setHaveCheckAnswer(checkAnswer);

}

try {

recordService.save(recordDTO);

//UserExperimentId为空,为新增提交记录.

if(form.getUserExperimentId()==null){

//更新提交人数

ExperimentMaster experimentMaster = experimentService.updateCommitNumber(form.getExperimentId());

//判断解锁阈值, 提交人数/总人数>解锁阈值,则自动解锁下一个实验.

if (experimentMaster.getExperimentCommitNum()\*1.0/experimentMaster.getExperimentParticipationNum()>=experimentMaster.getValve()){

experimentService.autoUnLock(form.getExperimentId());

}

}

} catch (TeachingException e) {

log.error("保存记录,发生异常:{}", e);

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR.getCode(),e.getMessage());

}

return ResultVOUtil.success();

}

}

package com.gdou.teaching.controller.common;

import com.gdou.teaching.Enum.ResultEnum;

import com.gdou.teaching.Enum.UserIdentEnum;

import com.gdou.teaching.Enum.UserStatusEnum;

import com.gdou.teaching.constant.CookieConstant;

import com.gdou.teaching.constant.RedisConstant;

import com.gdou.teaching.constant.SecurityConstants;

import com.gdou.teaching.dataobject.HostHolder;

import com.gdou.teaching.dto.CourseDTO;

import com.gdou.teaching.dto.UserDTO;

import com.gdou.teaching.exception.TeachingException;

import com.gdou.teaching.mbg.model.User;

import com.gdou.teaching.service.CourseService;

import com.gdou.teaching.service.UserService;

import com.gdou.teaching.util.CookieUtil;

import com.gdou.teaching.util.JWTUtil;

import com.gdou.teaching.util.ResultVOUtil;

import com.gdou.teaching.vo.ResultVO;

import com.gdou.teaching.vo.UserVO;

import com.gdou.teaching.web.Auth;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.BeanUtils;

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

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

import org.springframework.data.redis.core.StringRedisTemplate;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.util.StringUtils;

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

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import java.util.UUID;

import java.util.concurrent.TimeUnit;

@Controller

@RequestMapping("/user")

@Slf4j

public class UserController {

private final StringRedisTemplate stringRedisTemplate;

private final UserService userService;

private final HostHolder hostHolder;

private final JWTUtil jwtUtil;

@Value("${server.servlet.context-path}")

String urlHead;

@Autowired

public UserController(StringRedisTemplate stringRedisTemplate, UserService userService, HostHolder hostHolder, JWTUtil jwtUtil) {

this.stringRedisTemplate = stringRedisTemplate;

this.userService = userService;

this.hostHolder = hostHolder;

this.jwtUtil = jwtUtil;

}

/\*\*

\* 登陆操作 参数校验在controller完成

\*

\* @param userNumber

\* @param password

\* @param response

\* @return

\*/

@PostMapping("/loginForTest")

@ResponseBody

public ResultVO Login(@RequestParam("userNumber") String userNumber,

@RequestParam("password") String password,

@RequestParam("ident") Integer ident,

HttpServletResponse response

) {

if (StringUtils.isEmpty(userNumber) || StringUtils.isEmpty(password)) {

return ResultVOUtil.fail(ResultEnum.BAD\_REQUEST);

}

try {

User user = userService.login(userNumber, password);

//判断身份

if (user.getUserIdent().intValue() == ident) {

// addToken(user, response);

HashMap map = new HashMap(4);

map.put("userId", user.getUserId());

map.put("nickname", user.getNickname());

map.put("headUrl", user.getHeadUrl());

//TODO 启动jwt模式项目

map.put("token",genToken(user));

return ResultVOUtil.success(map);

} else {

return ResultVOUtil.fail(ResultEnum.USER\_NOT\_EXIST);

}

} catch (TeachingException e) {

return ResultVOUtil.fail(e);

}

}

/\*\*

\* 登出

\*

\* @param request

\* @param response

\* @return

\*/

@RequestMapping(path = {"/logout"}, method = RequestMethod.GET)

@ResponseBody

public ResultVO logout(HttpServletRequest request, HttpServletResponse response) {

UserDTO user = hostHolder.getUser();

cleanToken(user);

log.info("注销成功");

return ResultVOUtil.success();

}

/\*\*

\* 用户个人信息接口

\*

\* @param userId

\* @return

\*/

@GetMapping(path = {"/info/{id}"})

@ResponseBody

public ResultVO<UserDTO> detail(@PathVariable(value = "id") Integer userId) {

try {

UserDTO userByUserId = userService.getUserDetailByUserId(userId);

UserVO userVO = new UserVO();

BeanUtils.copyProperties(userByUserId, userVO);

if (userByUserId.getUserIdent().equals(UserIdentEnum.SUTUDENT.getCode().byteValue())) {

userVO.setUserIdent("学生");

} else if (userByUserId.getUserIdent().equals(UserIdentEnum.TEACHER.getCode().byteValue())) {

userVO.setUserIdent("老师");

} else {

userVO.setUserIdent("管理员");

}

return ResultVOUtil.success(userVO);

} catch (TeachingException e) {

return ResultVOUtil.fail(e);

}

}

@PostMapping("/updatePassWord")

@ResponseBody

public ResultVO updatePassWord(@RequestParam("oldPassword") String oldPassword, @RequestParam("newPassword") String newPassword) {

UserDTO user = hostHolder.getUser();

Boolean updatePassword = userService.updatePassword(user.getUserId(), oldPassword, newPassword);

if (updatePassword) {

return ResultVOUtil.success();

} else {

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR);

}

}

@PostMapping("/login")

@ResponseBody

public ResultVO login(@RequestParam("userNumber") String userNumber,

@RequestParam("password") String password,

@RequestParam("verifyCode") String verifyCode,

@RequestParam("ident") Integer ident,

HttpSession httpSession,

HttpServletResponse response

) {

if (StringUtils.isEmpty(userNumber)) {

return ResultVOUtil.fail(ResultEnum.PARAM\_NULL.getCode(), "学号/工号不能为空");

}

if (StringUtils.isEmpty(password)) {

return ResultVOUtil.fail(ResultEnum.PARAM\_NULL.getCode(), "密码不能为空");

}

if (StringUtils.isEmpty(verifyCode)) {

return ResultVOUtil.fail(ResultEnum.PARAM\_NULL.getCode(), "验证码不能为空");

}

String kaptchaCode = httpSession.getAttribute("verifyCode") + "";

log.info("用户输入的验证码为:{},实际的验证码为:{}", verifyCode, kaptchaCode);

if (StringUtils.isEmpty(kaptchaCode) || !verifyCode.equals(kaptchaCode)) {

return ResultVOUtil.fail(ResultEnum.VERIFYCODE\_ERROR);

}

httpSession.removeAttribute("verifyCode");

try {

User user = userService.login(userNumber, password);

if(!user.getUserStatus().equals(UserStatusEnum.NORMAL.getCode().byteValue())){

return ResultVOUtil.fail(ResultEnum.USER\_STATUS\_ERROR);

}

if (user.getUserIdent().intValue() == ident) {

addToken(user, response);

log.info("登陆的用户为：{}", hostHolder.getUser());

HashMap map = new HashMap(3);

map.put("userId", user.getUserId());

map.put("nickname", user.getNickname());

map.put("headUrl", user.getHeadUrl());

//TODO 启动jwt模式项目

map.put("token",genToken(user));

return ResultVOUtil.success(map);

} else {

return ResultVOUtil.fail(ResultEnum.USER\_NOT\_EXIST);

}

} catch (TeachingException e) {

return ResultVOUtil.fail(e);

}

}

private void addToken(User user, HttpServletResponse response) {

//设置token至redis

String token = UUID.randomUUID().toString();

stringRedisTemplate.opsForValue().set(String.format(RedisConstant.TOKEN\_PREFIX, token), String.valueOf(user.getUserId()), RedisConstant.EXPIRE, TimeUnit.SECONDS);

//设置token到cookie

CookieUtil.set(response, CookieConstant.TOKEN, token, RedisConstant.EXPIRE);

}

private String genToken(User user){

String token = jwtUtil.genToken(user);

stringRedisTemplate.opsForValue().set(String.format(RedisConstant.TOKEN\_PREFIX, user.getUserId()), token, SecurityConstants.EXPIRATION, TimeUnit.SECONDS);

return token;

}

private void cleanToken(UserDTO user){

stringRedisTemplate.opsForValue().getOperations().delete(String.format(RedisConstant.TOKEN\_PREFIX,user.getUserId()));

hostHolder.clear();

}

private void cleanToken(HttpServletRequest request, HttpServletResponse response) {

Cookie cookie = CookieUtil.get(request, CookieConstant.TOKEN);

if (cookie != null) {

//清除Redis

stringRedisTemplate.opsForValue().getOperations().delete(String.format(RedisConstant.TOKEN\_PREFIX, cookie.getValue()));

//清除cookie

CookieUtil.set(response, CookieConstant.TOKEN, null, 0);

hostHolder.clear();

}

hostHolder.clear();

}

@ResponseBody

@PostMapping("/resetPassword")

@Auth(user=UserIdentEnum.ADMIN)

public ResultVO resetPassword(@RequestParam("userId")Integer userId,@RequestParam("password")String newPassword){

userService.resetPassword(userId,newPassword);

return ResultVOUtil.success();

}

@ResponseBody

@Auth(user=UserIdentEnum.ADMIN)

@PostMapping("/invalidUser")

public ResultVO invalidUser(@RequestBody List<Integer> userIds) throws IOException {

List<UserDTO> usersByUserId = userService.getUsersByUserId(userIds);

int findSize = usersByUserId.size();

if(findSize!=userIds.size()){

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR.getCode(),"部分用户ID无效");

}

//检查身份是否符合要求

long adminCount = usersByUserId.stream()

.filter(userDTO -> userDTO.getUserIdent().equals(UserIdentEnum.ADMIN.getCode().byteValue()))

.count();

if(adminCount>0){

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR);

}

//检查状态

long BANCount = usersByUserId.stream()

.filter(userDTO -> userDTO.getUserStatus().equals(UserStatusEnum.BAN.getCode().byteValue()))

.count();

if(BANCount==usersByUserId.size()){

return userService.deleteUserByBatch(userIds)?ResultVOUtil.success():ResultVOUtil.fail(ResultEnum.SERVER\_ERROR);

}else {

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR.getCode(),"用户的状态不符合要求(被停用状态)");

}

}

@PostMapping("/banUsers")

@ResponseBody

@Auth(user=UserIdentEnum.ADMIN)

public ResultVO banUser(@RequestBody List<Integer> userIds) throws IOException {

List<UserDTO> usersByUserId = userService.getUsersByUserId(userIds);

int findSize = usersByUserId.size();

if(findSize!=userIds.size()){

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR.getCode(),"部分用户ID无效");

}

//检查身份是否符合要求

long adminCount = usersByUserId.stream()

.filter(userDTO -> userDTO.getUserIdent().equals(UserIdentEnum.ADMIN.getCode().byteValue()))

.count();

if(adminCount>0){

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR);

}

//检查状态

long NormalCount = usersByUserId.stream()

.filter(userDTO -> userDTO.getUserStatus().equals(UserStatusEnum.NORMAL.getCode().byteValue()))

.count();

if(NormalCount==usersByUserId.size()){

return userService.banUserByBatch(userIds)?ResultVOUtil.success():ResultVOUtil.fail(ResultEnum.SERVER\_ERROR);

}else {

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR.getCode(),"用户的状态不符合要求(正常状态)");

}

}

@PostMapping("/recoverUsers")

@ResponseBody

@Auth(user=UserIdentEnum.ADMIN)

public Object recoverUser(@RequestBody List<Integer> userIds) throws IOException {

List<UserDTO> usersByUserId = userService.getUsersByUserId(userIds);

int findSize = usersByUserId.size();

if(findSize!=userIds.size()){

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR.getCode(),"部分用户ID无效");

}

//检查身份是否符合要求

long adminCount = usersByUserId.stream()

.filter(userDTO -> userDTO.getUserIdent().equals(UserIdentEnum.ADMIN.getCode().byteValue()))

.count();

if(adminCount>0){

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR);

}

long BANCount = usersByUserId.stream()

.filter(userDTO -> userDTO.getUserStatus().equals(UserStatusEnum.BAN.getCode().byteValue()))

.count();

if(BANCount==usersByUserId.size()){

return userService.recoverUserByBatch(userIds)?ResultVOUtil.success():ResultVOUtil.fail(ResultEnum.SERVER\_ERROR);

}else {

return ResultVOUtil.fail(ResultEnum.PARAM\_ERROR.getCode(),"用户的状态不符合要求(被停用状态)");

}

}

}

package com.gdou.teaching.service.impl;

import com.gdou.teaching.Enum.ResultEnum;

import com.gdou.teaching.Enum.UserIdentEnum;

import com.gdou.teaching.Enum.UserStatusEnum;

import com.gdou.teaching.dao.UserDao;

import com.gdou.teaching.dao.UserInfoDao;

import com.gdou.teaching.dto.UserDTO;

import com.gdou.teaching.exception.TeachingException;

import com.gdou.teaching.mbg.mapper.ClassMapper;

import com.gdou.teaching.mbg.mapper.UserInfoMapper;

import com.gdou.teaching.mbg.mapper.UserMapper;

import com.gdou.teaching.mbg.model.Class;

import com.gdou.teaching.mbg.model.User;

import com.gdou.teaching.mbg.model.UserExample;

import com.gdou.teaching.mbg.model.UserInfo;

import com.gdou.teaching.mbg.model.UserInfoExample;

import com.gdou.teaching.service.UserService;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.BeanUtils;

import org.springframework.stereotype.Service;

import org.springframework.util.DigestUtils;

import java.util.List;

import java.util.Random;

import java.util.UUID;

import java.util.stream.Collectors;

import static com.gdou.teaching.Enum.ResultEnum.PARAM\_ERROR;

import static com.gdou.teaching.Enum.ResultEnum.USER\_NOT\_EXIST;

@Service

@Slf4j

public class UserServiceImpl implements UserService {

private final

UserMapper userMapper;

private final

UserDao userDao;

private final

UserInfoDao userInfoDao;

private final

UserInfoMapper userInfoMapper;

private final

ClassMapper classMapper;

public UserServiceImpl(UserMapper userMapper, UserDao userDao, UserInfoDao userInfoDao, UserInfoMapper userInfoMapper, ClassMapper classMapper) {

this.userMapper = userMapper;

this.userDao = userDao;

this.userInfoDao = userInfoDao;

this.userInfoMapper = userInfoMapper;

this.classMapper = classMapper;

}

/\*\*

\* 对user对象进行盐值加密、生成头像、设置用户状态

\* @param user

\*/

private void genUser(User user) {

String salt = UUID.randomUUID().toString().substring(0,5);

user.setSalt(salt);

user.setHeadUrl(String.format("http://images.nowcoder.com/head/%dt.png", new Random().nextInt(1000)));

user.setUserStatus(UserStatusEnum.NORMAL.getCode().byteValue());

user.setPassword(DigestUtils.md5DigestAsHex((user.getPassword()+salt).getBytes()));

}

@Override

public UserDTO selectOne(int id) {

User user = userMapper.selectByPrimaryKey(id);

if (user==null){

log.info("[UserServiceImpl]-,getUserById,学生信息不存在,userId:{}",id);

throw new TeachingException(USER\_NOT\_EXIST);

}

UserDTO userDTO=new UserDTO();

BeanUtils.copyProperties(user,userDTO);

return userDTO;

}

@Override

public Boolean addUser(User user) {

//判断学号信息是否重复

String userNumber = user.getUserNumber();

UserExample userExample = new UserExample();

userExample.createCriteria().andUserNumberEqualTo(userNumber);

List<User> users = userMapper.selectByExample(userExample);

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

return false;

}

genUser(user);

return userMapper.insert(user)>0;

}

@Override

public User login(String username, String password) {

UserExample userExample = new UserExample();

userExample.createCriteria().andUserNumberEqualTo(username);

List<User> users = userMapper.selectByExample(userExample);

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

throw new TeachingException(ResultEnum.USER\_NOT\_EXIST);

}

User user = users.get(0);

if(!DigestUtils.md5DigestAsHex((password+user.getSalt()).getBytes()).equals(user.getPassword())){

throw new TeachingException(ResultEnum.USER\_PASSWORD\_ERROR);

}

return user;

}

@Override

public UserDTO selectOne(String userNumber) {

UserExample userExample = new UserExample();

userExample.createCriteria().andUserNumberEqualTo(userNumber);

List<User> users = userMapper.selectByExample(userExample);

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

return null;

}

UserDTO userDTO=new UserDTO();

BeanUtils.copyProperties(users.get(0),userDTO);

return userDTO;

}

@Override

public Boolean addUserByBatch(List<User> userList) {

List<String> userNumbers = userList.stream().map(User::getUserNumber).collect(Collectors.toList());

UserExample userExample = new UserExample();

userExample.createCriteria().andUserNumberIn(userNumbers);

List<User> users = userMapper.selectByExample(userExample);

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

StringBuilder sb = new StringBuilder();

sb.append("用户");

users.forEach(user -> {

sb.append(" 学号-").append(user.getUserNumber());

sb.append(" 姓名-").append(user.getNickname());

sb.append(",");

});

sb.deleteCharAt(sb.length()-1);

sb.append(" 已存在,请检查数据是否有误");

throw new TeachingException(PARAM\_ERROR.getCode(),sb.toString());

}

//todo 需要在info表插入信息，需要看看怎么调整一下这些信息

userList.forEach(user -> {

genUser(user);

});

return userDao.insertList(userList)==userList.size();

}

@Override

public Boolean addUserInfoByBatch(List<Integer> userIdList, String college, String series, String major) {

List<UserInfo> userInfoList=userIdList.stream().map(userId -> {

UserInfo userInfo = new UserInfo();

userInfo.setUserId(userId);

userInfo.setCollege(college);

userInfo.setSeries(series);

userInfo.setMajor(major);

return userInfo;

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

return userInfoDao.insertList(userInfoList)==userIdList.size();

}

@Override

public UserDTO getUserDetailByUserId(Integer userId) {

UserDTO userDTO = new UserDTO();

User user = userMapper.selectByPrimaryKey(userId);

if(user==null){

throw new TeachingException(USER\_NOT\_EXIST);

}

BeanUtils.copyProperties(user,userDTO);

if(user.getClassId()!=0){

Class aClass = classMapper.selectByPrimaryKey(user.getClassId());

userDTO.setClassName(aClass.getClassName());

}

UserInfoExample userInfoExample = new UserInfoExample();

userInfoExample.createCriteria().andUserIdEqualTo(user.getUserId());

List<UserInfo> userInfos = userInfoMapper.selectByExample(userInfoExample);

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

return userDTO;

}

UserInfo userInfo = userInfos.get(0);

BeanUtils.copyProperties(userInfo,userDTO);

return userDTO;

}

@Override

public List<UserDTO> getStudentListByClassId(Integer classId) {

UserExample userExample = new UserExample();

if(classId!=0){

userExample.createCriteria().andClassIdEqualTo(classId).andUserStatusEqualTo(UserStatusEnum.NORMAL.getCode().byteValue()).andUserIdentEqualTo(UserIdentEnum.SUTUDENT.getCode().byteValue());

}else {

userExample.createCriteria().andUserStatusEqualTo(UserStatusEnum.NORMAL.getCode().byteValue()).andUserIdentEqualTo(UserIdentEnum.SUTUDENT.getCode().byteValue());

}

List<User> users = userMapper.selectByExample(userExample);

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

return null;

}

List<UserDTO> userDTOS = users.stream().map(user -> {

UserDTO userDTO = new UserDTO();

BeanUtils.copyProperties(user, userDTO);

return userDTO;

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

return userDTOS;

}

@Override

public PageInfo getUserListByClassIdAndKeywordAndIdentInPage(Integer classId,Integer page,Integer size,String keyword,Integer ident) {

PageHelper.startPage(page,size);

List<User> users = userDao.selectByClassIdAndKeyword(classId,keyword,ident);

PageInfo pageInfo = new PageInfo(users);

return pageInfo;

}

@Override

public List<UserDTO> getUsersByUserId(List<Integer> userIds) {

UserExample userExample = new UserExample();

userExample.createCriteria().andUserIdIn(userIds);

List<User> users = userMapper.selectByExample(userExample);

List<UserDTO> userDTOS = users.stream().map(user -> {

UserDTO userDTO = new UserDTO();

BeanUtils.copyProperties(user, userDTO);

return userDTO;

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

return userDTOS;

}

@Override

public Boolean updatePassword(Integer userId,String oldPassword ,String newPassword) {

User user = userMapper.selectByPrimaryKey(userId);

if(user==null){

throw new TeachingException(USER\_NOT\_EXIST);

}

if(!DigestUtils.md5DigestAsHex((oldPassword+user.getSalt()).getBytes()).equals(user.getPassword())){

throw new TeachingException(PARAM\_ERROR.getCode(),"原密码错误");

}

if(DigestUtils.md5DigestAsHex((oldPassword+user.getSalt()).getBytes()).equals(DigestUtils.md5DigestAsHex((newPassword+user.getSalt()).getBytes()))){

throw new TeachingException(PARAM\_ERROR.getCode(),"新密码不能与原密码相等");

}

//重新更新盐值

String salt = UUID.randomUUID().toString().substring(0,5);

user.setSalt(salt);

user.setPassword(DigestUtils.md5DigestAsHex((newPassword+salt).getBytes()));

return userMapper.updateByPrimaryKey(user)>0;

}

@Override

public Boolean resetPassword(Integer userId, String newPassword) {

User user = userMapper.selectByPrimaryKey(userId);

if(user==null){

throw new TeachingException(USER\_NOT\_EXIST);

}

//重新更新盐值

String salt = UUID.randomUUID().toString().substring(0,5);

user.setSalt(salt);

user.setPassword(DigestUtils.md5DigestAsHex((newPassword+salt).getBytes()));

return userMapper.updateByPrimaryKey(user)>0;

}

@Override

public Boolean banUserByBatch(List<Integer> userIds){

return updateStatusByUserIds(userIds, UserStatusEnum.BAN);

}

@Override

public Boolean recoverUserByBatch(List<Integer> userIds){

return updateStatusByUserIds(userIds, UserStatusEnum.NORMAL);

}

@Override

public Boolean deleteUserByBatch(List<Integer> userIds) {

return updateStatusByUserIds(userIds, UserStatusEnum.INVALID);

}

private Boolean updateStatusByUserIds(List<Integer> userIds, UserStatusEnum stauts) {

UserExample userExample = new UserExample();

userExample.createCriteria().andUserIdIn(userIds);

User user = new User();

user.setUserStatus(stauts.getCode().byteValue());

int i = userMapper.updateByExampleSelective(user, userExample);

return i==userIds.size();

}

}package com.gdou.teaching.service.impl;

import com.gdou.teaching.Enum.CourseStatusEnum;

import com.gdou.teaching.Enum.FileCategoryEnum;

import com.gdou.teaching.Enum.ResultEnum;

import com.gdou.teaching.dao.CourseMasterDao;

import com.gdou.teaching.dto.CourseDTO;

import com.gdou.teaching.dto.FileDTO;

import com.gdou.teaching.exception.TeachingException;

import com.gdou.teaching.mbg.mapper.\*;

import com.gdou.teaching.mbg.model.\*;

import com.gdou.teaching.service.CourseService;

import com.gdou.teaching.service.FileService;

import com.gdou.teaching.service.UserService;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.BeanUtils;

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

import org.springframework.scheduling.annotation.Async;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.util.StringUtils;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

@Service

@Slf4j

public class CourseServiceImpl implements CourseService {

private final CourseMasterMapper courseMasterMapper;

private final UserMapper userMapper;

private final CourseDetailMapper courseDetailMapper;

private final AchievementMapper achievementMapper;

private final UserService userService;

private final CourseMasterDao courseMasterDao;

private final ExperimentMasterMapper experimentMasterMapper;

private final FileService fileService;

public CourseServiceImpl(CourseMasterMapper courseMasterMapper, UserMapper userMapper, CourseDetailMapper courseDetailMapper, AchievementMapper achievementMapper, UserService userService, CourseMasterDao courseMasterDao, ExperimentMasterMapper experimentMasterMapper, FileService fileService) {

this.courseMasterMapper = courseMasterMapper;

this.userMapper = userMapper;

this.courseDetailMapper = courseDetailMapper;

this.achievementMapper = achievementMapper;

this.userService = userService;

this.courseMasterDao = courseMasterDao;

this.experimentMasterMapper = experimentMasterMapper;

this.fileService = fileService;

}

/\*\*

\* 获取课程的基本信息 --隶属学生

\* @param courseId

\* @return

\*/

@Override

public CourseDTO selectOne(Integer courseId) {

CourseMaster courseMaster = courseMasterMapper.selectByPrimaryKey(courseId);

if(courseMaster==null){

log.info("[CourseServiceImpl]-获取课程基本信息,课程主表不存在,courseId={}",courseId);

throw new TeachingException(ResultEnum.COURSE\_NOT\_EXIST);

}

CourseDTO courseDTO = new CourseDTO();

BeanUtils.copyProperties(courseMaster,courseDTO);

return courseDTO;

}

/\*\*

\* 新增或修改课程 --隶属老师端

\* @param courseDTO

\* @return

\*/

@Override

@Transactional

public CourseDTO save(CourseDTO courseDTO) {

CourseMaster courseMaster = new CourseMaster();

CourseDetail courseDetail = new CourseDetail();

// 先对courseDetail进行新增/更新

BeanUtils.copyProperties(courseDTO,courseDetail);

if (courseDetail.getCourseDetailId()==null){

int insert = courseDetailMapper.insert(courseDetail);

if (insert<=0){

log.error("[CourseServiceImpl-save]新增课程,新增课程详情表详情失败,courseDetail={}",courseDetail);

throw new TeachingException(ResultEnum.COURSE\_SAVE\_ERROR);

}

}else{

int i = courseDetailMapper.updateByPrimaryKeySelective(courseDetail);

if (i<=0){

log.error("[CourseServiceImpl-save]更新课程,更新课程详情表详情失败,courseDetail={}",courseDetail);

throw new TeachingException(ResultEnum.COURSE\_SAVE\_ERROR);

}

}

//回填CourseDetailId

courseDTO.setCourseDetailId(courseDetail.getCourseDetailId());

BeanUtils.copyProperties(courseDTO,courseMaster);

if (courseMaster.getCourseId()==null){

Integer i = courseMasterMapper.insert(courseMaster);

if (i<=0){

log.error("[CourseServiceImpl-save]新增课程,新增课程主表失败,courseMaster={}",courseMaster);

throw new TeachingException(ResultEnum.COURSE\_SAVE\_ERROR);

}

}else{

int i = courseMasterMapper.updateByPrimaryKeySelective(courseMaster);

if (i<=0){

log.error("[CourseServiceImpl-save]更新课程,更新课程主表失败,courseMaster={}",courseMaster);

throw new TeachingException(ResultEnum.COURSE\_SAVE\_ERROR);

}

}

//回填setCourseId

courseDTO.setCourseId(courseMaster.getCourseId());

return courseDTO;

}

/\*\*

\* 获取课程详细信息 通用

\* @param courseId

\* @return

\*/

@Override

public CourseDTO detail(Integer courseId) throws TeachingException{

CourseMaster courseMaster = courseMasterMapper.selectByPrimaryKey(courseId);

if (courseMaster==null){

log.info("[CourseServiceImpl]-查询课程详情,课程主表不存在,courseId={}",courseId);

throw new TeachingException(ResultEnum.COURSE\_NOT\_EXIST);

}

CourseDTO courseDTO = new CourseDTO();

BeanUtils.copyProperties(courseMaster,courseDTO);

User teacher = userMapper.selectByPrimaryKey(courseMaster.getTeacherId());

if (teacher==null){

log.info("[CourseServiceImpl]-查询课程课程详情 用户ID错误,UserId={}",courseMaster.getTeacherId());

}else{

courseDTO.setTeacherNickname(teacher.getNickname());

courseDTO.setTeacherId(teacher.getUserId());

}

//查询课程详情表

CourseDetail courseDetail = courseDetailMapper.selectByPrimaryKey(courseMaster.getCourseDetailId());

if (courseDetail==null){

return courseDTO;

}

BeanUtils.copyProperties(courseDetail,courseDTO);

return courseDTO;

}

/\*\*

\* 注销课程操作 ----隶属老师端

\* @param courseId

\* @return

\*/

@Override

public boolean invalid(Integer courseId) {

CourseMaster courseMaster =

courseMasterMapper.selectByPrimaryKey(courseId);

if (courseMaster == null) {

log.info("注销课程,该课程不存在或已被删除");

throw new TeachingException(ResultEnum.COURSE\_NOT\_EXIST);

}

if (courseMaster.getCourseStatus().intValue()!=(CourseStatusEnum.END.getCode())){

log.info("注销课程,该课程状态异常");

throw new TeachingException(ResultEnum.COURSE\_STATUS\_ERROR);

}

courseMaster.setCourseStatus(CourseStatusEnum.INVALID.getCode().byteValue());

if (courseMasterMapper.updateByPrimaryKeySelective(courseMaster)!=1){

log.info("注销课程,课程注销失败");

throw new TeachingException(ResultEnum.COURSE\_INVALID\_ERROR);

}

return true;

}

/\*\*

\* 解锁课程操作 ----隶属老师端

\* @param courseId

\* @return

\*/

@Override

public boolean unlock(Integer courseId) {

CourseMaster courseMaster =

courseMasterMapper.selectByPrimaryKey(courseId);

if (courseMaster == null) {

log.info("解锁课程,该课程不存在");

throw new TeachingException(ResultEnum.COURSE\_NOT\_EXIST);

}

if (courseMaster.getCourseStatus().intValue()!=(CourseStatusEnum.LOCK.getCode())){

log.info("解锁课程,该课程状态异常");

throw new TeachingException(ResultEnum.COURSE\_STATUS\_ERROR);

}

//更新

courseMaster.setCourseStatus(CourseStatusEnum.NORMAL.getCode().byteValue());

if (courseMasterMapper.updateByPrimaryKeySelective(courseMaster)!=1){

log.info("解锁课程,该课程解锁失败");

throw new TeachingException(ResultEnum.COURSE\_RESTORE\_ERROR);

}

return true;

}

@Override

public boolean lock(Integer courseId) {

CourseMaster courseMaster =

courseMasterMapper.selectByPrimaryKey(courseId);

if (courseMaster == null) {

log.info("锁定课程,该课程不存在");

throw new TeachingException(ResultEnum.COURSE\_NOT\_EXIST);

}

if (courseMaster.getCourseStatus().intValue()!=(CourseStatusEnum.NORMAL.getCode())){

log.info("锁定课程,该课程状态异常");

throw new TeachingException(ResultEnum.COURSE\_STATUS\_ERROR);

}

//更新

courseMaster.setCourseStatus(CourseStatusEnum.LOCK.getCode().byteValue());

if (courseMasterMapper.updateByPrimaryKeySelective(courseMaster)!=1){

log.info("锁定课程,课程锁定失败");

throw new TeachingException(ResultEnum.COURSE\_RESTORE\_ERROR);

}

return true;

}

@Override

public boolean end(Integer courseId) {

CourseMaster courseMaster =

courseMasterMapper.selectByPrimaryKey(courseId);

if (courseMaster == null) {

log.info("完结课程,该课程不存在");

throw new TeachingException(ResultEnum.COURSE\_NOT\_EXIST);

}

if (courseMaster.getCourseStatus().intValue()!=(CourseStatusEnum.NORMAL.getCode())

&&courseMaster.getCourseStatus().intValue()!=(CourseStatusEnum.LOCK.getCode())){

log.info("完结课程,该课程状态异常");

throw new TeachingException(ResultEnum.COURSE\_STATUS\_ERROR);

}

//更新

courseMaster.setCourseStatus(CourseStatusEnum.END.getCode().byteValue());

if ( courseMasterMapper.updateByPrimaryKeySelective(courseMaster)!=1){

log.info("完结课程,课程锁定失败");

throw new TeachingException(ResultEnum.COURSE\_RESTORE\_ERROR);

}

return true;

}

/\*\*

\* 给学生用的课程list

\* @param userId

\* @return

\*/

@Override

public List<CourseDTO> listCourseByUserIdAndKeywordForStudent(Integer userId,String keyword) {

//查询课程主表记录

AchievementExample achievementExample = new AchievementExample();

AchievementExample.Criteria criteria = achievementExample.createCriteria().andUserIdEqualTo(userId);

if(!StringUtils.isEmpty(keyword)){

criteria.andCourseNameLike("%"+keyword+"%");

}

List<Achievement> achievements = achievementMapper.selectByExample(achievementExample);

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

//说明该学生无报任何课程

return null;

}

//获取选修的课程ID

List<Integer> courseIds = achievements.stream().map(achievement -> {

Integer courseId = achievement.getCourseId();

return courseId;

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

//获取选修的课程主表ID

CourseMasterExample courseMasterExample = new CourseMasterExample();

courseMasterExample.createCriteria().andCourseIdIn(courseIds);

List<CourseMaster> courseMasterList = courseMasterMapper.selectByExample(courseMasterExample);

if (courseMasterList==null){

throw new TeachingException(ResultEnum.COURSE\_NOT\_EXIST);

}

List<Integer> ids = courseMasterList.stream().map(courseMaster -> {

Integer teacherId = courseMaster.getTeacherId();

return teacherId;

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

UserExample userExample = new UserExample();

userExample.createCriteria().andUserIdIn(ids);

List<User> teacherList = userMapper.selectByExample(userExample);

//

HashMap<Integer,User>teacherMap=new HashMap<Integer,User>(teacherList.size()){

{

teacherList.forEach(aTeacher->{

put(aTeacher.getUserId(),aTeacher);

});

}

};

//获取到老师的信息

List<CourseDTO> result = courseMasterList.stream().map(courseMaster -> {

CourseDTO courseDTO = new CourseDTO();

courseDTO.setTeacherNickname(teacherMap.get(courseMaster.getTeacherId()).getNickname());

courseDTO.setHeadUrl(teacherMap.get(courseMaster.getTeacherId()).getHeadUrl());

courseDTO.setCourseName(courseMaster.getCourseName());

courseDTO.setCourseId(courseMaster.getCourseId());

courseDTO.setCourseStatus(courseMaster.getCourseStatus());

courseDTO.setCourseCover(courseMaster.getCourseCover());

return courseDTO;

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

return result;

}

@Override

public List<CourseDTO> listCourseByUserIdAndKeywordForTeacher(Integer userId,String keyword) {

User user = userMapper.selectByPrimaryKey(userId);

if(user==null) {

throw new TeachingException(ResultEnum.USER\_NOT\_EXIST);

}

//查询课程主表记录

CourseMasterExample courseMasterExample = new CourseMasterExample();

CourseMasterExample.Criteria criteria = courseMasterExample.createCriteria().andTeacherIdEqualTo(userId).andCourseStatusNotEqualTo(CourseStatusEnum.INVALID.getCode().byteValue());

if(!StringUtils.isEmpty(keyword)){

criteria.andCourseNameLike("%"+keyword+"%");

}

List<CourseMaster> courseMasters = courseMasterMapper.selectByExample(courseMasterExample);

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

//说明该教师暂无授课

return null;

}

//拼装

List<CourseDTO> result = courseMasters.stream().map(courseMaster -> {

CourseDTO courseDTO = new CourseDTO();

courseDTO.setTeacherNickname(user.getNickname());

courseDTO.setHeadUrl(user.getHeadUrl());

courseDTO.setCourseName(courseMaster.getCourseName());

courseDTO.setCourseId(courseMaster.getCourseId());

courseDTO.setCourseStatus(courseMaster.getCourseStatus());

courseDTO.setCourseCover(courseMaster.getCourseCover());

return courseDTO;

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

return result;

}

@Override

@Transactional

@Async

public void updateNumber(Integer courseId) {

//获取人数

AchievementExample example = new AchievementExample();

example.createCriteria().andCourseIdEqualTo(courseId);

int number = achievementMapper.countByExample(example);

//更新人数

CourseMaster courseMaster = new CourseMaster();

courseMaster.setCourseId(courseId);

courseMaster.setCourseNumber(number);

courseMasterMapper.updateByPrimaryKeySelective(courseMaster);

//更新到各个实验

ExperimentMasterExample experimentMasterExample = new ExperimentMasterExample();

experimentMasterExample.createCriteria().andCourseIdEqualTo(courseId);

ExperimentMaster experimentMaster = new ExperimentMaster();

experimentMaster.setExperimentParticipationNum(number);

experimentMasterMapper.updateByExampleSelective(experimentMaster,experimentMasterExample);

}

@Override

public HashMap<String,Object> listCourseForAdminByTeacherIdAndKeywordInPage(Integer userId, Integer page, Integer size, String keyWord, Integer status) {

CourseMasterExample courseMasterExample = new CourseMasterExample();

CourseMasterExample.Criteria criteria = courseMasterExample.createCriteria().andTeacherIdEqualTo(userId).andCourseStatusNotEqualTo(CourseStatusEnum.INVALID.getCode().byteValue());

if(!StringUtils.isEmpty(keyWord)){

keyWord = "%"+keyWord+"%";

criteria.andCourseNameLike(keyWord);

}

if(status!=null){

criteria.andCourseStatusEqualTo(status.byteValue());

}

List<CourseDTO> collect = new ArrayList<>(0);

HashMap<String,Object> result = new HashMap<>(3);

PageHelper.startPage(page,size);

List<CourseMaster> courseMasters = courseMasterMapper.selectByExample(courseMasterExample);

PageInfo<CourseMaster> pageInfo = new PageInfo(courseMasters);

List<CourseMaster> list = pageInfo.getList();

long total = pageInfo.getTotal();

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

//说明该教师暂无授课或分页最末

result.put("list",collect);

result.put("total",total);

return result;

}

collect = list.stream().map(courseMaster -> {

CourseDetail courseDetail = courseDetailMapper.selectByPrimaryKey(courseMaster.getCourseDetailId());

CourseDTO courseDTO = new CourseDTO();

BeanUtils.copyProperties(courseMaster, courseDTO);

BeanUtils.copyProperties(courseDetail, courseDTO);

return courseDTO;

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

result.put("list",collect);

result.put("total",total);

return result;

}

@Override

public HashMap listCourseForAdminByStudentIdAndKeywordAndStatusInPage(Integer page, Integer size,Integer userId,String keyword,Integer status) {

HashMap result = new HashMap(3);

List<CourseDTO> collect = new ArrayList<>(0);

AchievementExample achievementExample = new AchievementExample();

AchievementExample.Criteria criteria = achievementExample.createCriteria().andUserIdEqualTo(userId);

if(!StringUtils.isEmpty(keyword)){

criteria.andCourseNameLike("%"+keyword+"%");

}

List<Achievement> achievements = achievementMapper.selectByExample(achievementExample);

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

//说明该学生无报任何课程

//返回total=0，list=null的map

result.put("total",0);

result.put("list",collect);

return result;

}

Map<Integer, Double> map = achievements.stream().collect(Collectors.toMap(Achievement::getCourseId, Achievement::getCourseAchievement));

List<Integer> courseIds = achievements.stream().map(achievement -> {

Integer courseId = achievement.getCourseId();

return courseId;

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

//拼装数据

//获取选修的课程主表ID

//这里做分页

CourseMasterExample courseMasterExample = new CourseMasterExample();

CourseMasterExample.Criteria criteria1 = courseMasterExample.createCriteria().andCourseIdIn(courseIds);

if(status!=null){

criteria1.andCourseStatusEqualTo(status.byteValue());

}

PageHelper.startPage(page,size);

List<CourseMaster> courseMasterList = courseMasterMapper.selectByExample(courseMasterExample);

PageInfo<CourseMaster> pageInfo = new PageInfo<>(courseMasterList);

long total = pageInfo.getTotal();

courseMasterList = pageInfo.getList();

collect = courseMasterList.stream().map(courseMaster -> {

CourseDetail courseDetail = courseDetailMapper.selectByPrimaryKey(courseMaster.getCourseDetailId());

CourseDTO courseDTO = new CourseDTO();

BeanUtils.copyProperties(courseMaster, courseDTO);

if(courseDetail.getCourseIntroduction().length()<=15){

courseDTO.setCourseIntroduction(courseDetail.getCourseIntroduction());

}else {

courseDTO.setCourseIntroduction(courseDetail.getCourseIntroduction().substring(0, 15));

}

courseDTO.setAchievement(map.get(courseMaster.getCourseId()));

return courseDTO;

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

result.put("total",total);

result.put("list",collect);

return result;

}

}package com.gdou.teaching.server.impl;

import com.gdou.teaching.server.FileServer;

import com.gdou.teaching.util.FileUtil;

import lombok.extern.slf4j.Slf4j;

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

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

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;

import java.io.File;

import java.io.IOException;

import java.io.InputStream;

import java.net.URI;

import java.net.URISyntaxException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import java.util.Random;

@Slf4j

public class LocalFileServer implements FileServer {

@Value("${server.servlet.context-path}")

private String contextPath;

@Autowired

FileUtil fileUtil;

private com.gdou.teaching.mbg.model.File FileName;

@Override

public String uploadFile(HttpServletRequest httpServletRequest,String fileName, MultipartFile file) {

//文件后缀名字

String suffixName = fileName.substring(fileName.lastIndexOf("."));

SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd\_HHmmss");

Random r = new Random();

StringBuilder tempName = new StringBuilder();

tempName.append(sdf.format(new Date())).append(r.nextInt(100)).append(suffixName);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 基于SpringBoot框架的信管专业课程实训系统——后30页源程序代码 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.gdou.teaching.service.impl;

import com.gdou.teaching.Enum.AnswerStatusEnum;

import com.gdou.teaching.Enum.ExperimentStatusEnum;

import com.gdou.teaching.Enum.FileCategoryEnum;

import com.gdou.teaching.Enum.ResultEnum;

import com.gdou.teaching.dao.CourseMasterDao;

import com.gdou.teaching.dto.ExperimentDTO;

import com.gdou.teaching.dto.FileDTO;

import com.gdou.teaching.exception.TeachingException;

import com.gdou.teaching.mbg.mapper.\*;

import com.gdou.teaching.mbg.model.\*;

import com.gdou.teaching.service.ExperimentService;

import com.gdou.teaching.service.FileService;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.BeanUtils;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

@Slf4j

@Service

public class ExperimentServiceImpl implements ExperimentService {

private final ExperimentMasterMapper experimentMasterMapper;

private final ExperimentDetailMapper experimentDetailMapper;

private final FileMapper fileMapper;

private final FileService fileService;

private final CourseMasterMapper courseMasterMapper;

private final ExperimentAnswerMapper answerMapper;

private final UserReExperimentMapper userReExperimentMapper;

private final ExperimentAnswerMapper experimentAnswerMapper;

public ExperimentServiceImpl(ExperimentMasterMapper experimentMasterMapper, ExperimentDetailMapper experimentDetailMapper, FileMapper fileMapper, FileService fileService, CourseMasterMapper courseMasterMapper, ExperimentAnswerMapper answerMapper, UserReExperimentMapper userReExperimentMapper, ExperimentAnswerMapper experimentAnswerMapper) {

this.experimentMasterMapper = experimentMasterMapper;

this.experimentDetailMapper = experimentDetailMapper;

this.fileMapper = fileMapper;

this.fileService = fileService;

this.courseMasterMapper = courseMasterMapper;

this.answerMapper = answerMapper;

this.userReExperimentMapper = userReExperimentMapper;

this.experimentAnswerMapper = experimentAnswerMapper;

}

@Override

public ExperimentDTO detail(Integer experimentId) {

ExperimentDTO experimentDTO = new ExperimentDTO();

//需要查询的数据有：

// 主表数据 副表detail数据 实验文件数据 实验答案数据

ExperimentMaster experimentMaster = experimentMasterMapper.selectByPrimaryKey(experimentId);

if(experimentMaster==null){

log.info("[ExperimentServiceImpl]-datail查询实验详情信息,实验主表不存在,experimentId={}",experimentId);

throw new TeachingException(ResultEnum.EXPERIMENT\_NOT\_EXIST);

}

ExperimentDetail experimentDetail = experimentDetailMapper.selectByPrimaryKey(experimentMaster.getExperimentDetailId());

if(experimentDetail!=null){

BeanUtils.copyProperties(experimentDetail,experimentDTO);

}

List<FileDTO> fileDTOList = fileService.selectFileByCategoryAndFileCategoryId(FileCategoryEnum.EXPERIMENT\_FILE.getCode(), experimentId);

//fileDTOList为null, 则new一个新的List,不为空则设置进去.

experimentDTO.setExperimentDetailFile(fileDTOList==null?new ArrayList<>():fileDTOList);

//属性拷贝

BeanUtils.copyProperties(experimentMaster,experimentDTO);

return experimentDTO;

}

@Override

@Transactional

public ExperimentDTO add(ExperimentDTO experimentDTO) {

ExperimentAnswer experimentAnswer=new ExperimentAnswer();

ExperimentMaster experimentMaster=new ExperimentMaster();

ExperimentDetail experimentDetail=new ExperimentDetail();

//对experimentAnswer进行新增

BeanUtils.copyProperties(experimentDTO,experimentAnswer);

//如果存在答案,则添加答案表信息

if ((experimentAnswer.getExperimentAnswerContent()!=null&&!experimentAnswer.getExperimentAnswerContent().isEmpty())||

experimentDTO.getExperimentAnswerFile()!=null&& !experimentDTO.getExperimentAnswerFile().isEmpty()){

experimentAnswer.setExperimentAnswerStatus(AnswerStatusEnum.NORMAL.getCode().byteValue());

if (experimentAnswerMapper.insert(experimentAnswer)<=0){

log.error("保存实验,保存实验答案失败,ExperimentAnswer={}",experimentAnswer);

throw new TeachingException(ResultEnum.EXPERIMENT\_SAVE\_ERROR);

}

//新增实验答案的文件：

List<FileDTO> experimentAnswerFile = experimentDTO.getExperimentAnswerFile();

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

fileService.saveFile(FileCategoryEnum.EXPERIMENT\_ANSWER\_FILE.getCode(),experimentAnswer.getExperimentAnswerId(),experimentAnswerFile);

}

}

// 先对experimentDetail进行新增

BeanUtils.copyProperties(experimentDTO,experimentDetail);

if( experimentDetailMapper.insert(experimentDetail)<=0){

log.error("[ExperimentServiceImpl]-保存实验,新增实验详情表失败,experimentDetail={}",experimentDetail);

throw new TeachingException(ResultEnum.EXPERIMENT\_SAVE\_ERROR);

}

//回填ExperimentDetailId,ExperimentAnswerId

experimentDTO.setExperimentDetailId(experimentDetail.getExperimentDetailId());

experimentDTO.setExperimentAnswerId(experimentAnswer.getExperimentAnswerId());

//查询应提交人数

Integer courseId = experimentDTO.getCourseId();

CourseMaster courseMaster = courseMasterMapper.selectByPrimaryKey(courseId);

if(courseMaster==null){

log.info("[ExperimentServiceImpl]-save 保存实验信息,课程主表不存在,courseId={}",courseId);

throw new TeachingException(ResultEnum.COURSE\_NOT\_EXIST);

}

experimentDTO.setExperimentParticipationNum(courseMaster.getCourseNumber());

//检查是否需要加上默认值

if(experimentDTO.getValve()==null){

experimentDTO.setValve((float) 0.9);

}

if(experimentDTO.getPunishment()==null){

experimentDTO.setPunishment((float) 0.9);

}

if(experimentDTO.getExperimentStatus()==null){

experimentDTO.setExperimentStatus(ExperimentStatusEnum.NORMAL.getCode().byteValue());

}

if (experimentDTO.getExperimentIntro()==null){

String text=experimentDTO.getExperimentText();

if (text.length()>15){

experimentDTO.setExperimentIntro(text.substring(0,15));

}else{

experimentDTO.setExperimentIntro(text);

}

}

experimentDTO.setExperimentCommitNum(0);

BeanUtils.copyProperties(experimentDTO,experimentMaster);

if (experimentMasterMapper.insert(experimentMaster)<=0){

log.error("[ExperimentServiceImpl]-保存实验,新增实验主表失败,experimentMaster={}",experimentMaster);

throw new TeachingException(ResultEnum.EXPERIMENT\_SAVE\_ERROR);

}

//回填ExperimentId

experimentDTO.setExperimentId(experimentMaster.getExperimentId());

//新增实验的文件：

List<FileDTO> experimentDetailFile = experimentDTO.getExperimentDetailFile();

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

fileService.saveFile(FileCategoryEnum.EXPERIMENT\_FILE.getCode(),experimentMaster.getExperimentId(),experimentDetailFile);

}

return experimentDTO;

}

@Override

public boolean autoUnLock(Integer experimentId) {

ExperimentMaster experimentMaster = experimentMasterMapper.selectByPrimaryKey(experimentId);

ExperimentMasterExample experimentMasterExample = new ExperimentMasterExample();

//取出courseId相同,大于experimentId的实验.

experimentMasterExample.createCriteria().andCourseIdEqualTo(experimentMaster.getCourseId()).andExperimentIdGreaterThan(experimentId);

List<ExperimentMaster> experimentMasterList = experimentMasterMapper.selectByExample(experimentMasterExample);

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

return false;

}

Integer target=Integer.MAX\_VALUE;

for(ExperimentMaster experiment:experimentMasterList){

if (experiment.getExperimentId()<target){

target=experiment.getExperimentId();

}

}

try{

unlock(target);

}catch (TeachingException e){

return false;

}

return true;

}

@Override

public List<ExperimentDTO> list(Integer courseId) {

ExperimentMasterExample experimentMasterExample = new ExperimentMasterExample();

experimentMasterExample.createCriteria().andCourseIdEqualTo(courseId).andExperimentStatusNotEqualTo(ExperimentStatusEnum.INVALID.getCode().byteValue());

List<ExperimentMaster> experimentMasters = experimentMasterMapper.selectByExample(experimentMasterExample);

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

return null;

}

List<ExperimentDTO> experimentDTOList = experimentMasters.stream().map(experimentMaster -> {

ExperimentDTO experimentDTO = new ExperimentDTO();

BeanUtils.copyProperties(experimentMaster,experimentDTO);

return experimentDTO;

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

return experimentDTOList;

}

@Override

public boolean updateExperimentInfo(ExperimentDTO experimentDTO) {

ExperimentMaster experimentMaster=new ExperimentMaster();

BeanUtils.copyProperties(experimentDTO,experimentMaster);

if (experimentMasterMapper.updateByPrimaryKeySelective(experimentMaster)!=1){

log.info("[ExperimentServiceImpl]-更新实验基本信息失败");

throw new TeachingException(ResultEnum.EXPERIMENT\_SAVE\_ERROR);

}

return true;

}

@Override

public boolean updateExperimentDetail(ExperimentDTO experimentDTO) {

ExperimentDetail experimentDetail=new ExperimentDetail();

BeanUtils.copyProperties(experimentDTO,experimentDetail);

if (experimentDetailMapper.updateByPrimaryKeyWithBLOBs(experimentDetail)!=1){

log.info("[ExperimentServiceImpl]-更新实验详情失败");

throw new TeachingException(ResultEnum.EXPERIMENT\_SAVE\_ERROR);

}

return true;

}

@Override

public boolean invalid(Integer experimentId) {

ExperimentMaster experimentMaster =

experimentMasterMapper.selectByPrimaryKey(experimentId);

if (experimentMaster == null) {

log.info("[ExperimentServiceImpl]-删除实验,该实验不存在或已被删除");

throw new TeachingException(ResultEnum.EXPERIMENT\_NOT\_EXIST);

}

//判断当前状态,仅结束状态的时候可被取消

if (experimentMaster.getExperimentStatus().intValue()!=ExperimentStatusEnum.END.getCode()){

log.info("[ExperimentServiceImpl]-完结实验,实验主表状态异常,status={}",experimentMaster.getExperimentStatus());

throw new TeachingException(ResultEnum.EXPERIMENT\_STATUS\_ERROR);

}

experimentMaster.setExperimentStatus(ExperimentStatusEnum.INVALID.getCode().byteValue());

if (experimentMasterMapper.updateByPrimaryKeySelective(experimentMaster)!=1){

log.info("[ExperimentServiceImpl]-删除实验,实验删除失败");

throw new TeachingException(ResultEnum.EXPERIMENT\_INVALID\_ERROR);

}

return true;

}

@Override

public boolean end(Integer experimentId) {

ExperimentMaster experimentMaster =

experimentMasterMapper.selectByPrimaryKey(experimentId);

if (experimentMaster == null) {

log.info("[ExperimentServiceImpl]-完结实验,该实验不存在或已被删除");

throw new TeachingException(ResultEnum.EXPERIMENT\_NOT\_EXIST);

}

//判断当前状态

if (experimentMaster.getExperimentStatus().intValue()!=(ExperimentStatusEnum.NORMAL.getCode())

&&experimentMaster.getExperimentStatus().intValue()!=(ExperimentStatusEnum.LOCK.getCode())){

log.info("[ExperimentServiceImpl]-完结实验,实验主表状态异常,status={}",experimentMaster.getExperimentStatus());

throw new TeachingException(ResultEnum.EXPERIMENT\_STATUS\_ERROR);

}

experimentMaster.setExperimentStatus(ExperimentStatusEnum.END.getCode().byteValue());

if (experimentMasterMapper.updateByPrimaryKeySelective(experimentMaster)!=1){

log.info("[ExperimentServiceImpl]-完结实验,实验完结失败");

throw new TeachingException(ResultEnum.EXPERIMENT\_SAVE\_ERROR);

}

return true;

}

@Override

public boolean lock(Integer experimentId) {

ExperimentMaster experimentMaster =

experimentMasterMapper.selectByPrimaryKey(experimentId);

if (experimentMaster == null) {

log.info("[ExperimentServiceImpl]-锁定实验,该实验不存在或已被删除");

throw new TeachingException(ResultEnum.EXPERIMENT\_NOT\_EXIST);

}

//判断当前状态

if (experimentMaster.getExperimentStatus().intValue()!=ExperimentStatusEnum.NORMAL.getCode()){

log.info("[ExperimentServiceImpl]-锁定实验,实验主表状态异常,status={}",experimentMaster.getExperimentStatus());

throw new TeachingException(ResultEnum.EXPERIMENT\_STATUS\_ERROR);

}

experimentMaster.setExperimentStatus(ExperimentStatusEnum.LOCK.getCode().byteValue());

if (experimentMasterMapper.updateByPrimaryKeySelective(experimentMaster)!=1){

log.info("[ExperimentServiceImpl]-锁定实验,实验锁定失败");

throw new TeachingException(ResultEnum.EXPERIMENT\_SAVE\_ERROR);

}

return true;

}

@Override

public boolean unlock(Integer experimentId) {

ExperimentMaster experimentMaster =

experimentMasterMapper.selectByPrimaryKey(experimentId);

if (experimentMaster == null) {

log.info("[ExperimentServiceImpl]-解锁实验,该实验不存在或已被删除");

throw new TeachingException(ResultEnum.EXPERIMENT\_NOT\_EXIST);

}

//判断当前状态

if (experimentMaster.getExperimentStatus().intValue()!=ExperimentStatusEnum.LOCK.getCode()){

log.info("[ExperimentServiceImpl]-解锁实验主表,实验主表状态异常,status={}",experimentMaster.getExperimentStatus());

throw new TeachingException(ResultEnum.EXPERIMENT\_STATUS\_ERROR);

}

experimentMaster.setExperimentStatus(ExperimentStatusEnum.NORMAL.getCode().byteValue());

if (experimentMasterMapper.updateByPrimaryKeySelective(experimentMaster)!=1){

log.info("[ExperimentServiceImpl]-解锁实验,实验解锁失败");

throw new TeachingException(ResultEnum.EXPERIMENT\_SAVE\_ERROR);

}

return true;

}

@Override

public ExperimentMaster updateCommitNumber(Integer experimentId) {

UserReExperimentExample example = new UserReExperimentExample();

example.createCriteria().andExperimentIdEqualTo(experimentId);

int i = userReExperimentMapper.countByExample(example);

ExperimentMaster experimentMaster = new ExperimentMaster();

experimentMaster.setExperimentId(experimentId);

experimentMaster.setExperimentCommitNum(i);

experimentMasterMapper.updateByPrimaryKeySelective(experimentMaster);

return experimentMasterMapper.selectByPrimaryKey(experimentId);

}

}

package com.gdou.teaching.service.impl;

import com.gdou.teaching.Enum.ExperimentStatusEnum;

import com.gdou.teaching.Enum.FileCategoryEnum;

import com.gdou.teaching.Enum.RecordStatusEnum;

import com.gdou.teaching.Enum.ResultEnum;

import com.gdou.teaching.dao.UserReExperimentDao;

import com.gdou.teaching.dto.FileDTO;

import com.gdou.teaching.dto.RecordDTO;

import com.gdou.teaching.exception.TeachingException;

import com.gdou.teaching.mbg.mapper.ExperimentMasterMapper;

import com.gdou.teaching.mbg.mapper.UserReExperimentMapper;

import com.gdou.teaching.mbg.model.ExperimentMaster;

import com.gdou.teaching.mbg.model.ExperimentMasterExample;

import com.gdou.teaching.mbg.model.UserReExperiment;

import com.gdou.teaching.mbg.model.UserReExperimentExample;

import com.gdou.teaching.service.AchievementService;

import com.gdou.teaching.service.FileService;

import com.gdou.teaching.service.RecordService;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.BeanUtils;

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

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

@Slf4j

@Service

public class RecordServiceImpl implements RecordService {

private final ExperimentMasterMapper experimentMasterMapper;

private final UserReExperimentMapper userReExperimentMapper;

private final UserReExperimentDao userReExperimentDao;

private final FileService fileService;

private final AchievementService achievementService;

public RecordServiceImpl(UserReExperimentMapper userReExperimentMapper, ExperimentMasterMapper experimentMasterMapper, UserReExperimentDao userReExperimentDao, FileService fileService, AchievementService achievementService) {

this.userReExperimentMapper = userReExperimentMapper;

this.experimentMasterMapper = experimentMasterMapper;

this.userReExperimentDao = userReExperimentDao;

this.fileService = fileService;

this.achievementService = achievementService;

}

@Override

public RecordDTO save(RecordDTO recordDTO) {

//查询该实验的状态。

ExperimentMaster experiment = experimentMasterMapper.selectByPrimaryKey(recordDTO.getExperimentId());

if (ExperimentStatusEnum.NORMAL.getCode().byteValue()!=experiment.getExperimentStatus()){

log.error("保存纪录,保存实验记录失败,当前实验不可提交,ExperimentStatus={}",experiment.getExperimentStatus());

throw new TeachingException(ResultEnum.EXPERIMENT\_STATUS\_ERROR);

}

recordDTO.setExperimentName(experiment.getExperimentName());

//准备变量

List<FileDTO> fileDTOList = recordDTO.getUserExperimentFile();

UserReExperiment userReExperiment = new UserReExperiment();

BeanUtils.copyProperties(recordDTO,userReExperiment);

userReExperiment.setExperimentAchievement((double)0);

userReExperiment.setStatus(RecordStatusEnum.REVIEWING.getCode().byteValue());

//查询用户的上一次提交

UserReExperimentExample userReExperimentExample = new UserReExperimentExample();

userReExperimentExample.createCriteria().andUserIdEqualTo(recordDTO.getUserId()).andExperimentIdEqualTo(recordDTO.getExperimentId());

List<UserReExperiment> userReExperiments = userReExperimentMapper.selectByExample(userReExperimentExample);

//上次提交过

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

Byte preStatus = userReExperiments.get(0).getStatus();

//检查状态

if(preStatus.equals(RecordStatusEnum.PASS.getCode().byteValue())||preStatus.equals(RecordStatusEnum.LOCK.getCode().byteValue())){

//如果已经通过审核或者是实验被锁定，无法再次提交

log.error("保存实验记录失败,已有实验提交记录被审核通过或实验已被锁定,preStatus={}",preStatus);

throw new TeachingException(ResultEnum.SUBMIT\_RECORD\_ERROR);

}else{

Integer userExperimentId = userReExperiments.get(0).getUserExperimentId();

//更新，先删除之前所有的文件记录，进行重新插入

fileService.deleteFiles(FileCategoryEnum.RECORD\_FILE.getCode(),userExperimentId);

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

fileService.saveFile(FileCategoryEnum.RECORD\_FILE.getCode(),userExperimentId,recordDTO.getUserExperimentFile());

}

userReExperiment.setUserExperimentId(userReExperiments.get(0).getUserExperimentId());

userReExperimentMapper.updateByPrimaryKeySelective(userReExperiment);

}

}

else{

//没提交过,进行新增存储记录

int insert = userReExperimentMapper.insert(userReExperiment);

if (insert<=0){

log.error("保存记录,保存实验记录失败,record={}",userReExperiment);

throw new TeachingException(ResultEnum.SUBMIT\_RECORD\_ERROR);

}

//存储文件

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

fileService.saveFile(FileCategoryEnum.RECORD\_FILE.getCode(), userReExperiment.getUserExperimentId(), fileDTOList);

}

}

recordDTO.setUserExperimentId(userReExperiment.getUserExperimentId());

return recordDTO;

}

@Override

public RecordDTO selectOne(Integer experimentId, Integer userId) {

UserReExperimentExample userReExperimentExample = new UserReExperimentExample();

userReExperimentExample.setOrderByClause("create\_time desc");

userReExperimentExample.createCriteria().andUserIdEqualTo(userId).andExperimentIdEqualTo(experimentId);

List<UserReExperiment> userReExperiments = userReExperimentMapper.selectByExampleWithBLOBs(userReExperimentExample);

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

return null;

}

UserReExperiment userReExperiment = userReExperiments.get(0);

RecordDTO recordDTO = new RecordDTO();

//查询用户提交时提交的文件记录

List<FileDTO> fileDTOList = fileService.selectFileByCategoryAndFileCategoryId(FileCategoryEnum.RECORD\_FILE.getCode(), userReExperiment.getUserExperimentId());

recordDTO.setUserExperimentFile(fileDTOList);

BeanUtils.copyProperties(userReExperiment,recordDTO);

return recordDTO;

}

@Override

public RecordDTO selectOne(Integer userExperimentId) {

UserReExperiment userReExperiment = userReExperimentMapper.selectByPrimaryKey(userExperimentId);

if(userReExperiment==null){

log.error("[RecordServiceImpl]-selectOne,实验提交表信息不存在,userExperimentId={}",userExperimentId);

throw new TeachingException(ResultEnum.RECORD\_NOT\_EXIST);

}

RecordDTO recordDTO = new RecordDTO();

//查询用户提交时提交的文件记录

List<FileDTO> fileDTOList = fileService.selectFileByCategoryAndFileCategoryId(FileCategoryEnum.RECORD\_FILE.getCode(), userReExperiment.getUserExperimentId());

recordDTO.setUserExperimentFile(fileDTOList);

BeanUtils.copyProperties(userReExperiment,recordDTO);

return recordDTO;

}

@Override

public List<RecordDTO> getRecordByUserIdAndCourseId(Integer userId, Integer courseId) {

//查询课程下属的实验ID

ExperimentMasterExample experimentMasterExample = new ExperimentMasterExample();

experimentMasterExample.createCriteria().andCourseIdEqualTo(courseId);

List<ExperimentMaster> experimentMasters = experimentMasterMapper.selectByExample(experimentMasterExample);

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

//说明这个课程暂时还没有任何实验，直接返回为空即可。

return null;

}

List<Integer> experimentIds = experimentMasters.stream().map(ExperimentMaster::getExperimentId).collect(Collectors.toList());

//查询对应的实验提交记录（不带提交文本）

UserReExperimentExample userReExperimentExample = new UserReExperimentExample();

userReExperimentExample.createCriteria().andExperimentIdIn(experimentIds).andUserIdEqualTo(userId);

List<UserReExperiment> userReExperiments = userReExperimentMapper.selectByExample(userReExperimentExample);

if (userReExperiments==null||experimentMasters.isEmpty()){

//说明用户没有提交过，直接返回空即可

return null;

}

List<RecordDTO> collect = userReExperiments.stream().map(userReExperiment -> {

RecordDTO recordDTO = new RecordDTO();

BeanUtils.copyProperties(userReExperiment, recordDTO);

return recordDTO;

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

return collect;

}

@Override

public List<RecordDTO> getRecordListByExperimentId(Integer experimentId) {

UserReExperimentExample userReExperimentExample = new UserReExperimentExample();

userReExperimentExample.createCriteria().andExperimentIdEqualTo(experimentId);

List<UserReExperiment> userReExperiments = userReExperimentMapper.selectByExample(userReExperimentExample);

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

log.info("[RecordServiceImpl]-getRecordListByExperimentId,实验提交表信息不存在,experimentId={}",experimentId);

return null;

}

List<RecordDTO> recordDTOList = userReExperiments.stream().map(record -> {

RecordDTO recordDTO = new RecordDTO();

BeanUtils.copyProperties(record, recordDTO);

return recordDTO;

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

return recordDTOList;

}

@Override

public void judge(RecordDTO recordDTO) {

//判断是否提前看过答案

ExperimentMaster experiment = experimentMasterMapper.selectByPrimaryKey(recordDTO.getExperimentId());

if (experiment==null){

log.info("[RecordServiceImpl]-judge,实验主表信息不存在,ExperimentId={}",recordDTO.getExperimentId());

throw new TeachingException(ResultEnum.EXPERIMENT\_NOT\_EXIST);

}

if(recordDTO.getHaveCheckAnswer()){

Float punishment = experiment.getPunishment();

recordDTO.setExperimentAchievement(recordDTO.getExperimentAchievement()\*punishment);

}

UserReExperiment userReExperiment = new UserReExperiment();

BeanUtils.copyProperties(recordDTO,userReExperiment);

int i = userReExperimentMapper.updateByPrimaryKeySelective(userReExperiment);

if (i<=0){

log.error("[RecordServiceImpl]-judge,保存实验提交表失败,userReExperiment={}",userReExperiment);

throw new TeachingException(ResultEnum.SUBMIT\_RECORD\_ERROR);

}

//更新课程成绩

achievementService.updateAchievement(experiment.getCourseId(),recordDTO.getUserId());

}

@Override

public void batchJudge(List<RecordDTO> recordDTO) {

Integer experimentId = recordDTO.get(0).getExperimentId();

ExperimentMaster experiment = experimentMasterMapper.selectByPrimaryKey(experimentId);

if(experiment==null){

log.info("[RecordServiceImpl]-batchJudge,实验主表信息不存在,experimentId={}",experimentId);

throw new TeachingException(ResultEnum.EXPERIMENT\_NOT\_EXIST);

}

Float punishment = experiment.getPunishment();

List<UserReExperiment> recordList = recordDTO.stream().map(r -> {

UserReExperiment record = new UserReExperiment();

//判断是否提前查看过答案

if (r.getHaveCheckAnswer()) {

r.setExperimentAchievement(r.getExperimentAchievement() \* punishment);

}

BeanUtils.copyProperties(r, record);

return record;

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

//更新操作

Integer i = userReExperimentDao.updateUserReExperimentByList(recordList);

if (i<=0){

log.error("[RecordServiceImpl]-batchJudge,批量保存实验提交表失败,recordList={}",recordList);

throw new TeachingException(ResultEnum.SUBMIT\_RECORD\_ERROR);

}

//todo 之后需要批量去更新学生的成绩 也许是制作一个异步更新的方法

}

}

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.gdou.teaching.mbg.mapper.CourseMasterMapper">

<resultMap id="BaseResultMap" type="com.gdou.teaching.mbg.model.CourseMaster">

<id column="course\_id" jdbcType="INTEGER" property="courseId" />

<result column="course\_name" jdbcType="VARCHAR" property="courseName" />

<result column="teacher\_id" jdbcType="INTEGER" property="teacherId" />

<result column="course\_detail\_id" jdbcType="INTEGER" property="courseDetailId" />

<result column="course\_status" jdbcType="TINYINT" property="courseStatus" />

<result column="course\_number" jdbcType="INTEGER" property="courseNumber" />

<result column="create\_time" jdbcType="TIMESTAMP" property="createTime" />

<result column="update\_time" jdbcType="TIMESTAMP" property="updateTime" />

<result column="course\_cover" jdbcType="VARCHAR" property="courseCover" />

</resultMap>

<sql id="Example\_Where\_Clause">

<where>

<foreach collection="oredCriteria" item="criteria" separator="or">

<if test="criteria.valid">

<trim prefix="(" prefixOverrides="and" suffix=")">

<foreach collection="criteria.criteria" item="criterion">

<choose>

<when test="criterion.noValue">

and ${criterion.condition}

</when>

<when test="criterion.singleValue">

and ${criterion.condition} #{criterion.value}

</when>

<when test="criterion.betweenValue">

and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}

</when>

<when test="criterion.listValue">

and ${criterion.condition}

<foreach close=")" collection="criterion.value" item="listItem" open="(" separator=",">

#{listItem}

</foreach>

</when>

</choose>

</foreach>

</trim>

</if>

</foreach>

</where>

</sql>

<sql id="Update\_By\_Example\_Where\_Clause">

<where>

<foreach collection="example.oredCriteria" item="criteria" separator="or">

<if test="criteria.valid">

<trim prefix="(" prefixOverrides="and" suffix=")">

<foreach collection="criteria.criteria" item="criterion">

<choose>

<when test="criterion.noValue">

and ${criterion.condition}

</when>

<when test="criterion.singleValue">

and ${criterion.condition} #{criterion.value}

</when>

<when test="criterion.betweenValue">

and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}

</when>

<when test="criterion.listValue">

and ${criterion.condition}

<foreach close=")" collection="criterion.value" item="listItem" open="(" separator=",">

#{listItem}

</foreach>

</when>

</choose>

</foreach>

</trim>

</if>

</foreach>

</where>

</sql>

<sql id="Base\_Column\_List">

course\_id, course\_name, teacher\_id, course\_detail\_id, course\_status, course\_number,

create\_time, update\_time, course\_cover

</sql>

<select id="selectByExample" parameterType="com.gdou.teaching.mbg.model.CourseMasterExample" resultMap="BaseResultMap">

select

<if test="distinct">

distinct

</if>

<include refid="Base\_Column\_List" />

from course\_master

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

<if test="orderByClause != null">

order by ${orderByClause}

</if>

</select>

<select id="selectByPrimaryKey" parameterType="java.lang.Integer" resultMap="BaseResultMap">

select

<include refid="Base\_Column\_List" />

from course\_master

where course\_id = #{courseId,jdbcType=INTEGER}

</select>

<delete id="deleteByPrimaryKey" parameterType="java.lang.Integer">

delete from course\_master

where course\_id = #{courseId,jdbcType=INTEGER}

</delete>

<delete id="deleteByExample" parameterType="com.gdou.teaching.mbg.model.CourseMasterExample">

delete from course\_master

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

</delete>

<insert id="insert" parameterType="com.gdou.teaching.mbg.model.CourseMaster">

<selectKey keyProperty="courseId" order="AFTER" resultType="java.lang.Integer">

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into course\_master (course\_name, teacher\_id, course\_detail\_id,

course\_status, course\_number, create\_time,

update\_time, course\_cover)

values (#{courseName,jdbcType=VARCHAR}, #{teacherId,jdbcType=INTEGER}, #{courseDetailId,jdbcType=INTEGER},

#{courseStatus,jdbcType=TINYINT}, #{courseNumber,jdbcType=INTEGER}, #{createTime,jdbcType=TIMESTAMP},

#{updateTime,jdbcType=TIMESTAMP}, #{courseCover,jdbcType=VARCHAR})

</insert>

<insert id="insertSelective" parameterType="com.gdou.teaching.mbg.model.CourseMaster">

<selectKey keyProperty="courseId" order="AFTER" resultType="java.lang.Integer">

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into course\_master

<trim prefix="(" suffix=")" suffixOverrides=",">

<if test="courseName != null">

course\_name,

</if>

<if test="teacherId != null">

teacher\_id,

</if>

<if test="courseDetailId != null">

course\_detail\_id,

</if>

<if test="courseStatus != null">

course\_status,

</if>

<if test="courseNumber != null">

course\_number,

</if>

<if test="createTime != null">

create\_time,

</if>

<if test="updateTime != null">

update\_time,

</if>

<if test="courseCover != null">

course\_cover,

</if>

</trim>

<trim prefix="values (" suffix=")" suffixOverrides=",">

<if test="courseName != null">

#{courseName,jdbcType=VARCHAR},

</if>

<if test="teacherId != null">

#{teacherId,jdbcType=INTEGER},

</if>

<if test="courseDetailId != null">

#{courseDetailId,jdbcType=INTEGER},

</if>

<if test="courseStatus != null">

#{courseStatus,jdbcType=TINYINT},

</if>

<if test="courseNumber != null">

#{courseNumber,jdbcType=INTEGER},

</if>

<if test="createTime != null">

#{createTime,jdbcType=TIMESTAMP},

</if>

<if test="updateTime != null">

#{updateTime,jdbcType=TIMESTAMP},

</if>

<if test="courseCover != null">

#{courseCover,jdbcType=VARCHAR},

</if>

</trim>

</insert>

<select id="countByExample" parameterType="com.gdou.teaching.mbg.model.CourseMasterExample" resultType="java.lang.Integer">

select count(\*) from course\_master

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

</select>

<update id="updateByExampleSelective" parameterType="map">

update course\_master

<set>

<if test="record.courseId != null">

course\_id = #{record.courseId,jdbcType=INTEGER},

</if>

<if test="record.courseName != null">

course\_name = #{record.courseName,jdbcType=VARCHAR},

</if>

<if test="record.teacherId != null">

teacher\_id = #{record.teacherId,jdbcType=INTEGER},

</if>

<if test="record.courseDetailId != null">

course\_detail\_id = #{record.courseDetailId,jdbcType=INTEGER},

</if>

<if test="record.courseStatus != null">

course\_status = #{record.courseStatus,jdbcType=TINYINT},

</if>

<if test="record.courseNumber != null">

course\_number = #{record.courseNumber,jdbcType=INTEGER},

</if>

<if test="record.createTime != null">

create\_time = #{record.createTime,jdbcType=TIMESTAMP},

</if>

<if test="record.updateTime != null">

update\_time = #{record.updateTime,jdbcType=TIMESTAMP},

</if>

<if test="record.courseCover != null">

course\_cover = #{record.courseCover,jdbcType=VARCHAR},

</if>

</set>

<if test="\_parameter != null">

<include refid="Update\_By\_Example\_Where\_Clause" />

</if>

</update>

<update id="updateByExample" parameterType="map">

update course\_master

set course\_id = #{record.courseId,jdbcType=INTEGER},

course\_name = #{record.courseName,jdbcType=VARCHAR},

teacher\_id = #{record.teacherId,jdbcType=INTEGER},

course\_detail\_id = #{record.courseDetailId,jdbcType=INTEGER},

course\_status = #{record.courseStatus,jdbcType=TINYINT},

course\_number = #{record.courseNumber,jdbcType=INTEGER},

create\_time = #{record.createTime,jdbcType=TIMESTAMP},

update\_time = #{record.updateTime,jdbcType=TIMESTAMP},

course\_cover = #{record.courseCover,jdbcType=VARCHAR}

<if test="\_parameter != null">

<include refid="Update\_By\_Example\_Where\_Clause" />

</if>

</update>

<update id="updateByPrimaryKeySelective" parameterType="com.gdou.teaching.mbg.model.CourseMaster">

update course\_master

<set>

<if test="courseName != null">

course\_name = #{courseName,jdbcType=VARCHAR},

</if>

<if test="teacherId != null">

teacher\_id = #{teacherId,jdbcType=INTEGER},

</if>

<if test="courseDetailId != null">

course\_detail\_id = #{courseDetailId,jdbcType=INTEGER},

</if>

<if test="courseStatus != null">

course\_status = #{courseStatus,jdbcType=TINYINT},

</if>

<if test="courseNumber != null">

course\_number = #{courseNumber,jdbcType=INTEGER},

</if>

<if test="createTime != null">

create\_time = #{createTime,jdbcType=TIMESTAMP},

</if>

<if test="updateTime != null">

update\_time = #{updateTime,jdbcType=TIMESTAMP},

</if>

<if test="courseCover != null">

course\_cover = #{courseCover,jdbcType=VARCHAR},

</if>

</set>

where course\_id = #{courseId,jdbcType=INTEGER}

</update>

<update id="updateByPrimaryKey" parameterType="com.gdou.teaching.mbg.model.CourseMaster">

update course\_master

set course\_name = #{courseName,jdbcType=VARCHAR},

teacher\_id = #{teacherId,jdbcType=INTEGER},

course\_detail\_id = #{courseDetailId,jdbcType=INTEGER},

course\_status = #{courseStatus,jdbcType=TINYINT},

course\_number = #{courseNumber,jdbcType=INTEGER},

create\_time = #{createTime,jdbcType=TIMESTAMP},

update\_time = #{updateTime,jdbcType=TIMESTAMP},

course\_cover = #{courseCover,jdbcType=VARCHAR}

where course\_id = #{courseId,jdbcType=INTEGER}

</update>

</mapper>

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.gdou.teaching.mbg.mapper.ExperimentMasterMapper">

<resultMap id="BaseResultMap" type="com.gdou.teaching.mbg.model.ExperimentMaster">

<id column="experiment\_id" jdbcType="INTEGER" property="experimentId" />

<result column="experiment\_name" jdbcType="VARCHAR" property="experimentName" />

<result column="experiment\_intro" jdbcType="VARCHAR" property="experimentIntro" />

<result column="experiment\_detail\_id" jdbcType="INTEGER" property="experimentDetailId" />

<result column="course\_id" jdbcType="INTEGER" property="courseId" />

<result column="experiment\_answer\_id" jdbcType="INTEGER" property="experimentAnswerId" />

<result column="experiment\_commit\_num" jdbcType="INTEGER" property="experimentCommitNum" />

<result column="experiment\_participation\_num" jdbcType="INTEGER" property="experimentParticipationNum" />

<result column="valve" jdbcType="REAL" property="valve" />

<result column="punishment" jdbcType="REAL" property="punishment" />

<result column="create\_time" jdbcType="TIMESTAMP" property="createTime" />

<result column="update\_time" jdbcType="TIMESTAMP" property="updateTime" />

<result column="experiment\_status" jdbcType="TINYINT" property="experimentStatus" />

</resultMap>

<sql id="Example\_Where\_Clause">

<where>

<foreach collection="oredCriteria" item="criteria" separator="or">

<if test="criteria.valid">

<trim prefix="(" prefixOverrides="and" suffix=")">

<foreach collection="criteria.criteria" item="criterion">

<choose>

<when test="criterion.noValue">

and ${criterion.condition}

</when>

<when test="criterion.singleValue">

and ${criterion.condition} #{criterion.value}

</when>

<when test="criterion.betweenValue">

and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}

</when>

<when test="criterion.listValue">

and ${criterion.condition}

<foreach close=")" collection="criterion.value" item="listItem" open="(" separator=",">

#{listItem}

</foreach>

</when>

</choose>

</foreach>

</trim>

</if>

</foreach>

</where>

</sql>

<sql id="Update\_By\_Example\_Where\_Clause">

<where>

<foreach collection="example.oredCriteria" item="criteria" separator="or">

<if test="criteria.valid">

<trim prefix="(" prefixOverrides="and" suffix=")">

<foreach collection="criteria.criteria" item="criterion">

<choose>

<when test="criterion.noValue">

and ${criterion.condition}

</when>

<when test="criterion.singleValue">

and ${criterion.condition} #{criterion.value}

</when>

<when test="criterion.betweenValue">

and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}

</when>

<when test="criterion.listValue">

and ${criterion.condition}

<foreach close=")" collection="criterion.value" item="listItem" open="(" separator=",">

#{listItem}

</foreach>

</when>

</choose>

</foreach>

</trim>

</if>

</foreach>

</where>

</sql>

<sql id="Base\_Column\_List">

experiment\_id, experiment\_name, experiment\_intro, experiment\_detail\_id, course\_id,

experiment\_answer\_id, experiment\_commit\_num, experiment\_participation\_num, valve,

punishment, create\_time, update\_time, experiment\_status

</sql>

<select id="selectByExample" parameterType="com.gdou.teaching.mbg.model.ExperimentMasterExample" resultMap="BaseResultMap">

select

<if test="distinct">

distinct

</if>

<include refid="Base\_Column\_List" />

from experiment\_master

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

<if test="orderByClause != null">

order by ${orderByClause}

</if>

</select>

<select id="selectByPrimaryKey" parameterType="java.lang.Integer" resultMap="BaseResultMap">

select

<include refid="Base\_Column\_List" />

from experiment\_master

where experiment\_id = #{experimentId,jdbcType=INTEGER}

</select>

<delete id="deleteByPrimaryKey" parameterType="java.lang.Integer">

delete from experiment\_master

where experiment\_id = #{experimentId,jdbcType=INTEGER}

</delete>

<delete id="deleteByExample" parameterType="com.gdou.teaching.mbg.model.ExperimentMasterExample">

delete from experiment\_master

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

</delete>

<insert id="insert" parameterType="com.gdou.teaching.mbg.model.ExperimentMaster">

<selectKey keyProperty="experimentId" order="AFTER" resultType="java.lang.Integer">

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into experiment\_master (experiment\_name, experiment\_intro,

experiment\_detail\_id, course\_id, experiment\_answer\_id,

experiment\_commit\_num, experiment\_participation\_num,

valve, punishment, create\_time,

update\_time, experiment\_status)

values (#{experimentName,jdbcType=VARCHAR}, #{experimentIntro,jdbcType=VARCHAR},

#{experimentDetailId,jdbcType=INTEGER}, #{courseId,jdbcType=INTEGER}, #{experimentAnswerId,jdbcType=INTEGER},

#{experimentCommitNum,jdbcType=INTEGER}, #{experimentParticipationNum,jdbcType=INTEGER},

#{valve,jdbcType=REAL}, #{punishment,jdbcType=REAL}, #{createTime,jdbcType=TIMESTAMP},

#{updateTime,jdbcType=TIMESTAMP}, #{experimentStatus,jdbcType=TINYINT})

</insert>

<insert id="insertSelective" parameterType="com.gdou.teaching.mbg.model.ExperimentMaster">

<selectKey keyProperty="experimentId" order="AFTER" resultType="java.lang.Integer">

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into experiment\_master

<trim prefix="(" suffix=")" suffixOverrides=",">

<if test="experimentName != null">

experiment\_name,

</if>

<if test="experimentIntro != null">

experiment\_intro,

</if>

<if test="experimentDetailId != null">

experiment\_detail\_id,

</if>

<if test="courseId != null">

course\_id,

</if>

<if test="experimentAnswerId != null">

experiment\_answer\_id,

</if>

<if test="experimentCommitNum != null">

experiment\_commit\_num,

</if>

<if test="experimentParticipationNum != null">

experiment\_participation\_num,

</if>

<if test="valve != null">

valve,

</if>

<if test="punishment != null">

punishment,

</if>

<if test="createTime != null">

create\_time,

</if>

<if test="updateTime != null">

update\_time,

</if>

<if test="experimentStatus != null">

experiment\_status,

</if>

</trim>

<trim prefix="values (" suffix=")" suffixOverrides=",">

<if test="experimentName != null">

#{experimentName,jdbcType=VARCHAR},

</if>

<if test="experimentIntro != null">

#{experimentIntro,jdbcType=VARCHAR},

</if>

<if test="experimentDetailId != null">

#{experimentDetailId,jdbcType=INTEGER},

</if>

<if test="courseId != null">

#{courseId,jdbcType=INTEGER},

</if>

<if test="experimentAnswerId != null">

#{experimentAnswerId,jdbcType=INTEGER},

</if>

<if test="experimentCommitNum != null">

#{experimentCommitNum,jdbcType=INTEGER},

</if>

<if test="experimentParticipationNum != null">

#{experimentParticipationNum,jdbcType=INTEGER},

</if>

<if test="valve != null">

#{valve,jdbcType=REAL},

</if>

<if test="punishment != null">

#{punishment,jdbcType=REAL},

</if>

<if test="createTime != null">

#{createTime,jdbcType=TIMESTAMP},

</if>

<if test="updateTime != null">

#{updateTime,jdbcType=TIMESTAMP},

</if>

<if test="experimentStatus != null">

#{experimentStatus,jdbcType=TINYINT},

</if>

</trim>

</insert>

<select id="countByExample" parameterType="com.gdou.teaching.mbg.model.ExperimentMasterExample" resultType="java.lang.Integer">

select count(\*) from experiment\_master

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

</select>

<update id="updateByExampleSelective" parameterType="map">

update experiment\_master

<set>

<if test="record.experimentId != null">

experiment\_id = #{record.experimentId,jdbcType=INTEGER},

</if>

<if test="record.experimentName != null">

experiment\_name = #{record.experimentName,jdbcType=VARCHAR},

</if>

<if test="record.experimentIntro != null">

experiment\_intro = #{record.experimentIntro,jdbcType=VARCHAR},

</if>

<if test="record.experimentDetailId != null">

experiment\_detail\_id = #{record.experimentDetailId,jdbcType=INTEGER},

</if>

<if test="record.courseId != null">

course\_id = #{record.courseId,jdbcType=INTEGER},

</if>

<if test="record.experimentAnswerId != null">

experiment\_answer\_id = #{record.experimentAnswerId,jdbcType=INTEGER},

</if>

<if test="record.experimentCommitNum != null">

experiment\_commit\_num = #{record.experimentCommitNum,jdbcType=INTEGER},

</if>

<if test="record.experimentParticipationNum != null">

experiment\_participation\_num = #{record.experimentParticipationNum,jdbcType=INTEGER},

</if>

<if test="record.valve != null">

valve = #{record.valve,jdbcType=REAL},

</if>

<if test="record.punishment != null">

punishment = #{record.punishment,jdbcType=REAL},

</if>

<if test="record.createTime != null">

create\_time = #{record.createTime,jdbcType=TIMESTAMP},

</if>

<if test="record.updateTime != null">

update\_time = #{record.updateTime,jdbcType=TIMESTAMP},

</if>

<if test="record.experimentStatus != null">

experiment\_status = #{record.experimentStatus,jdbcType=TINYINT},

</if>

</set>

<if test="\_parameter != null">

<include refid="Update\_By\_Example\_Where\_Clause" />

</if>

</update>

<update id="updateByExample" parameterType="map">

update experiment\_master

set experiment\_id = #{record.experimentId,jdbcType=INTEGER},

experiment\_name = #{record.experimentName,jdbcType=VARCHAR},

experiment\_intro = #{record.experimentIntro,jdbcType=VARCHAR},

experiment\_detail\_id = #{record.experimentDetailId,jdbcType=INTEGER},

course\_id = #{record.courseId,jdbcType=INTEGER},

experiment\_answer\_id = #{record.experimentAnswerId,jdbcType=INTEGER},

experiment\_commit\_num = #{record.experimentCommitNum,jdbcType=INTEGER},

experiment\_participation\_num = #{record.experimentParticipationNum,jdbcType=INTEGER},

valve = #{record.valve,jdbcType=REAL},

punishment = #{record.punishment,jdbcType=REAL},

create\_time = #{record.createTime,jdbcType=TIMESTAMP},

update\_time = #{record.updateTime,jdbcType=TIMESTAMP},

experiment\_status = #{record.experimentStatus,jdbcType=TINYINT}

<if test="\_parameter != null">

<include refid="Update\_By\_Example\_Where\_Clause" />

</if>

</update>

<update id="updateByPrimaryKeySelective" parameterType="com.gdou.teaching.mbg.model.ExperimentMaster">

update experiment\_master

<set>

<if test="experimentName != null">

experiment\_name = #{experimentName,jdbcType=VARCHAR},

</if>

<if test="experimentIntro != null">

experiment\_intro = #{experimentIntro,jdbcType=VARCHAR},

</if>

<if test="experimentDetailId != null">

experiment\_detail\_id = #{experimentDetailId,jdbcType=INTEGER},

</if>

<if test="courseId != null">

course\_id = #{courseId,jdbcType=INTEGER},

</if>

<if test="experimentAnswerId != null">

experiment\_answer\_id = #{experimentAnswerId,jdbcType=INTEGER},

</if>

<if test="experimentCommitNum != null">

experiment\_commit\_num = #{experimentCommitNum,jdbcType=INTEGER},

</if>

<if test="experimentParticipationNum != null">

experiment\_participation\_num = #{experimentParticipationNum,jdbcType=INTEGER},

</if>

<if test="valve != null">

valve = #{valve,jdbcType=REAL},

</if>

<if test="punishment != null">

punishment = #{punishment,jdbcType=REAL},

</if>

<if test="createTime != null">

create\_time = #{createTime,jdbcType=TIMESTAMP},

</if>

<if test="updateTime != null">

update\_time = #{updateTime,jdbcType=TIMESTAMP},

</if>

<if test="experimentStatus != null">

experiment\_status = #{experimentStatus,jdbcType=TINYINT},

</if>

</set>

where experiment\_id = #{experimentId,jdbcType=INTEGER}

</update>

<update id="updateByPrimaryKey" parameterType="com.gdou.teaching.mbg.model.ExperimentMaster">

update experiment\_master

set experiment\_name = #{experimentName,jdbcType=VARCHAR},

experiment\_intro = #{experimentIntro,jdbcType=VARCHAR},

experiment\_detail\_id = #{experimentDetailId,jdbcType=INTEGER},

course\_id = #{courseId,jdbcType=INTEGER},

experiment\_answer\_id = #{experimentAnswerId,jdbcType=INTEGER},

experiment\_commit\_num = #{experimentCommitNum,jdbcType=INTEGER},

experiment\_participation\_num = #{experimentParticipationNum,jdbcType=INTEGER},

valve = #{valve,jdbcType=REAL},

punishment = #{punishment,jdbcType=REAL},

create\_time = #{createTime,jdbcType=TIMESTAMP},

update\_time = #{updateTime,jdbcType=TIMESTAMP},

experiment\_status = #{experimentStatus,jdbcType=TINYINT}

where experiment\_id = #{experimentId,jdbcType=INTEGER}

</update>

</mapper>

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.gdou.teaching.mbg.mapper.UserMapper">

<resultMap id="BaseResultMap" type="com.gdou.teaching.mbg.model.User">

<id column="user\_id" jdbcType="INTEGER" property="userId" />

<result column="nickname" jdbcType="VARCHAR" property="nickname" />

<result column="user\_number" jdbcType="VARCHAR" property="userNumber" />

<result column="password" jdbcType="VARCHAR" property="password" />

<result column="head\_url" jdbcType="VARCHAR" property="headUrl" />

<result column="salt" jdbcType="VARCHAR" property="salt" />

<result column="class\_id" jdbcType="INTEGER" property="classId" />

<result column="user\_status" jdbcType="TINYINT" property="userStatus" />

<result column="user\_ident" jdbcType="TINYINT" property="userIdent" />

<result column="create\_time" jdbcType="TIMESTAMP" property="createTime" />

<result column="update\_time" jdbcType="TIMESTAMP" property="updateTime" />

</resultMap>

<sql id="Example\_Where\_Clause">

<where>

<foreach collection="oredCriteria" item="criteria" separator="or">

<if test="criteria.valid">

<trim prefix="(" prefixOverrides="and" suffix=")">

<foreach collection="criteria.criteria" item="criterion">

<choose>

<when test="criterion.noValue">

and ${criterion.condition}

</when>

<when test="criterion.singleValue">

and ${criterion.condition} #{criterion.value}

</when>

<when test="criterion.betweenValue">

and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}

</when>

<when test="criterion.listValue">

and ${criterion.condition}

<foreach close=")" collection="criterion.value" item="listItem" open="(" separator=",">

#{listItem}

</foreach>

</when>

</choose>

</foreach>

</trim>

</if>

</foreach>

</where>

</sql>

<sql id="Update\_By\_Example\_Where\_Clause">

<where>

<foreach collection="example.oredCriteria" item="criteria" separator="or">

<if test="criteria.valid">

<trim prefix="(" prefixOverrides="and" suffix=")">

<foreach collection="criteria.criteria" item="criterion">

<choose>

<when test="criterion.noValue">

and ${criterion.condition}

</when>

<when test="criterion.singleValue">

and ${criterion.condition} #{criterion.value}

</when>

<when test="criterion.betweenValue">

and ${criterion.condition} #{criterion.value} and #{criterion.secondValue}

</when>

<when test="criterion.listValue">

and ${criterion.condition}

<foreach close=")" collection="criterion.value" item="listItem" open="(" separator=",">

#{listItem}

</foreach>

</when>

</choose>

</foreach>

</trim>

</if>

</foreach>

</where>

</sql>

<sql id="Base\_Column\_List">

user\_id, nickname, user\_number, password, head\_url, salt, class\_id, user\_status,

user\_ident, create\_time, update\_time

</sql>

<select id="selectByExample" parameterType="com.gdou.teaching.mbg.model.UserExample" resultMap="BaseResultMap">

select

<if test="distinct">

distinct

</if>

<include refid="Base\_Column\_List" />

from user

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

<if test="orderByClause != null">

order by ${orderByClause}

</if>

</select>

<select id="selectByPrimaryKey" parameterType="java.lang.Integer" resultMap="BaseResultMap">

select

<include refid="Base\_Column\_List" />

from user

where user\_id = #{userId,jdbcType=INTEGER}

</select>

<delete id="deleteByPrimaryKey" parameterType="java.lang.Integer">

delete from user

where user\_id = #{userId,jdbcType=INTEGER}

</delete>

<delete id="deleteByExample" parameterType="com.gdou.teaching.mbg.model.UserExample">

delete from user

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

</delete>

<insert id="insert" parameterType="com.gdou.teaching.mbg.model.User">

<selectKey keyProperty="userId" order="AFTER" resultType="java.lang.Integer">

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into user (nickname, user\_number, password,

head\_url, salt, class\_id,

user\_status, user\_ident, create\_time,

update\_time)

values (#{nickname,jdbcType=VARCHAR}, #{userNumber,jdbcType=VARCHAR}, #{password,jdbcType=VARCHAR},

#{headUrl,jdbcType=VARCHAR}, #{salt,jdbcType=VARCHAR}, #{classId,jdbcType=INTEGER},

#{userStatus,jdbcType=TINYINT}, #{userIdent,jdbcType=TINYINT}, #{createTime,jdbcType=TIMESTAMP},

#{updateTime,jdbcType=TIMESTAMP})

</insert>

<insert id="insertSelective" parameterType="com.gdou.teaching.mbg.model.User">

<selectKey keyProperty="userId" order="AFTER" resultType="java.lang.Integer">

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into user

<trim prefix="(" suffix=")" suffixOverrides=",">

<if test="nickname != null">

nickname,

</if>

<if test="userNumber != null">

user\_number,

</if>

<if test="password != null">

password,

</if>

<if test="headUrl != null">

head\_url,

</if>

<if test="salt != null">

salt,

</if>

<if test="classId != null">

class\_id,

</if>

<if test="userStatus != null">

user\_status,

</if>

<if test="userIdent != null">

user\_ident,

</if>

<if test="createTime != null">

create\_time,

</if>

<if test="updateTime != null">

update\_time,

</if>

</trim>

<trim prefix="values (" suffix=")" suffixOverrides=",">

<if test="nickname != null">

#{nickname,jdbcType=VARCHAR},

</if>

<if test="userNumber != null">

#{userNumber,jdbcType=VARCHAR},

</if>

<if test="password != null">

#{password,jdbcType=VARCHAR},

</if>

<if test="headUrl != null">

#{headUrl,jdbcType=VARCHAR},

</if>

<if test="salt != null">

#{salt,jdbcType=VARCHAR},

</if>

<if test="classId != null">

#{classId,jdbcType=INTEGER},

</if>

<if test="userStatus != null">

#{userStatus,jdbcType=TINYINT},

</if>

<if test="userIdent != null">

#{userIdent,jdbcType=TINYINT},

</if>

<if test="createTime != null">

#{createTime,jdbcType=TIMESTAMP},

</if>

<if test="updateTime != null">

#{updateTime,jdbcType=TIMESTAMP},

</if>

</trim>

</insert>

<select id="countByExample" parameterType="com.gdou.teaching.mbg.model.UserExample" resultType="java.lang.Integer">

select count(\*) from user

<if test="\_parameter != null">

<include refid="Example\_Where\_Clause" />

</if>

</select>

<update id="updateByExampleSelective" parameterType="map">

update user

<set>

<if test="record.userId != null">

user\_id = #{record.userId,jdbcType=INTEGER},

</if>

<if test="record.nickname != null">

nickname = #{record.nickname,jdbcType=VARCHAR},

</if>

<if test="record.userNumber != null">

user\_number = #{record.userNumber,jdbcType=VARCHAR},

</if>

<if test="record.password != null">

password = #{record.password,jdbcType=VARCHAR},

</if>

<if test="record.headUrl != null">

head\_url = #{record.headUrl,jdbcType=VARCHAR},

</if>

<if test="record.salt != null">

salt = #{record.salt,jdbcType=VARCHAR},

</if>

<if test="record.classId != null">

class\_id = #{record.classId,jdbcType=INTEGER},

</if>

<if test="record.userStatus != null">

user\_status = #{record.userStatus,jdbcType=TINYINT},

</if>

<if test="record.userIdent != null">

user\_ident = #{record.userIdent,jdbcType=TINYINT},

</if>

<if test="record.createTime != null">

create\_time = #{record.createTime,jdbcType=TIMESTAMP},

</if>

<if test="record.updateTime != null">

update\_time = #{record.updateTime,jdbcType=TIMESTAMP},

</if>

</set>

<if test="\_parameter != null">

<include refid="Update\_By\_Example\_Where\_Clause" />

</if>

</update>

<update id="updateByExample" parameterType="map">

update user

set user\_id = #{record.userId,jdbcType=INTEGER},

nickname = #{record.nickname,jdbcType=VARCHAR},

user\_number = #{record.userNumber,jdbcType=VARCHAR},

password = #{record.password,jdbcType=VARCHAR},

head\_url = #{record.headUrl,jdbcType=VARCHAR},

salt = #{record.salt,jdbcType=VARCHAR},

class\_id = #{record.classId,jdbcType=INTEGER},

user\_status = #{record.userStatus,jdbcType=TINYINT},

user\_ident = #{record.userIdent,jdbcType=TINYINT},

create\_time = #{record.createTime,jdbcType=TIMESTAMP},

update\_time = #{record.updateTime,jdbcType=TIMESTAMP}

<if test="\_parameter != null">

<include refid="Update\_By\_Example\_Where\_Clause" />

</if>

</update>

<update id="updateByPrimaryKeySelective" parameterType="com.gdou.teaching.mbg.model.User">

update user

<set>

<if test="nickname != null">

nickname = #{nickname,jdbcType=VARCHAR},

</if>

<if test="userNumber != null">

user\_number = #{userNumber,jdbcType=VARCHAR},

</if>

<if test="password != null">

password = #{password,jdbcType=VARCHAR},

</if>

<if test="headUrl != null">

head\_url = #{headUrl,jdbcType=VARCHAR},