package com.dazuizui.online\_judge.service.impl;  
  
import cn.hutool.core.util.ObjectUtil;  
import cn.hutool.json.JSONArray;  
import cn.hutool.json.JSONObject;  
import cn.hutool.json.JSONUtil;  
import com.dazuizui.basicapi.entry.bo.ProgramBo;  
import com.dazuizui.online\_judge.service.OnlineJudgeService;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.http.HttpEntity;  
import org.springframework.http.HttpHeaders;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.stereotype.Service;  
import org.springframework.web.client.RestTemplate;  
  
import java.util.Arrays;  
import java.util.List;  
  
  
*/\*\*  
 \* 在线判断业务接口实现类  
 \*/*@Service  
public class OnlineJudgeServiceImpl implements OnlineJudgeService {  
 @Value("${OJServer}")  
 private String url;  
 */\*\*  
 \* 判决代码业务实现类  
 \** ***@param*** *programBo  
 \** ***@return*** *\*/* @Override  
 public JSONObject judgeTheProgram(ProgramBo programBo) {  
 JSONObject cmd = new JSONObject();  
 //设置args参数  
 cmd.set("args",programBo.getParseCodeArgs());  
 //设置env  
 cmd.set("env",Arrays.*asList*("PATH=/usr/bin:/bin"));  
  
 //设置cmd的file参数  
 JSONArray files = new JSONArray();  
 files.put(new JSONObject().set("name","stdout").set("max",programBo.getProblemLimit().getStdoutMax()));  
 files.put(new JSONObject().set("name","stderr").set("max",programBo.getProblemLimit().getStderrMax()));  
 cmd.set("files", files);  
 // 设置cpu时间限制  
 cmd.set("cpuLimit",programBo.getProblemLimit().getCpuLimit()\*1000\*1000\*1000L);  
 // 设置内存限制  
 cmd.set("memoryLimit",programBo.getProblemLimit().getMemoryLimit()\*1024\*1024L);  
 // 设置线程限制  
 cmd.set("procLimit",programBo.getProblemLimit().getProcLimit());  
 //设置copyin以及代码内容  
 JSONObject copyIn = new JSONObject();  
 copyIn.set(programBo.getCopyOutCached().get(0),new JSONObject().set("content",programBo.getCode()));  
  
 cmd.set("copyIn", copyIn);  
 // 设置copyOut  
 cmd.set("copyOut", Arrays.*asList*("stdout","stderr"));  
 // 设置文件缓存  
 cmd.set("copyOutCached", programBo.getCopyOutCached());  
  
 JSONArray request = request(cmd);  
 JSONObject jsonObject = new JSONObject(request.get(0));  
 //获取返回状态  
 String status = jsonObject.get("status").toString();  
 */\*\*  
 \* 如果通过就去查看执行结果，如果不通过就返回状态  
 \*/* if (status.equals("Accepted")){  
 JSONObject jsonObject1 = new JSONObject(jsonObject.get("fileIds"));  
 String fileUUID = jsonObject1.get(programBo.getCopyOutCached().get(1)).toString();  
 JSONArray jsonArray = executeCode(fileUUID, programBo.getInput(), programBo);  
 JSONObject jsonObject2 = new JSONObject(jsonArray.get(0));  
 return jsonObject2;  
 }  
  
 return jsonObject;  
 }  
  
 */\*\*  
  
 \* 10/23/2022 5:21 Am  
 \* 代码执行  
 \** ***@param*** *fileId  
 \** ***@param*** *input  
 \*/* public JSONArray executeCode(String fileId,String input,ProgramBo programBo){  
 JSONObject cmd = new JSONObject();  
 System.*out*.println(programBo.getRunCommandArgs());  
 //args配置  
 cmd.set("args",programBo.getRunCommandArgs());  
 //设置env  
 cmd.set("env",Arrays.*asList*("PATH=/usr/bin:/bin"));  
 // 设置cmd的files参数  
 JSONArray files = new JSONArray();  
 //设置输入样例  
 if (ObjectUtil.*isNotNull*(input)){  
 files.put(new JSONObject().set("content",input));  
 }else{  
 files.put(new JSONObject().set("content",""));  
 }  
 files.put(new JSONObject().set("name","stdout").set("max",programBo.getProblemLimit().getStdoutMax()));  
 files.put(new JSONObject().set("name","stderr").set("max",programBo.getProblemLimit().getStderrMax()));  
 cmd.set("files", files);  
 // 设置cpu时间限制  
 cmd.set("cpuLimit",programBo.getProblemLimit().getCpuLimit()\*1000\*1000\*1000L);  
 // 设置内存限制  
 cmd.set("memoryLimit",programBo.getProblemLimit().getMemoryLimit()\*1024\*1024L);  
 // 设置线程限制  
 cmd.set("procLimit",programBo.getProblemLimit().getProcLimit());  
 JSONObject copyIn = new JSONObject();  
 copyIn.set(programBo.getCopyOutCached().get(1),new JSONObject().set("fileId",fileId));  
 cmd.set("copyIn", copyIn);  
  
 JSONArray request = request(cmd);  
 return request;  
 }  
  
 public JSONArray request(JSONObject cmd){  
 RestTemplate restTemplate = new RestTemplate();  
 JSONObject cmd1 = new JSONObject().set("cmd", new JSONArray().put(cmd));  
 System.*err*.println(cmd1);  
 //System.err.println();  
 HttpHeaders headers = new HttpHeaders();  
 headers.setContentType(MediaType.*APPLICATION\_JSON*);  
 HttpEntity<String> request = new HttpEntity<>(JSONUtil.*toJsonStr*(cmd1), headers);  
 System.*out*.println(request);  
 ResponseEntity<String> responseEntity = restTemplate.postForEntity(url,request,String.class);  
 System.*out*.println(responseEntity);  
 return JSONUtil.*parseArray*(responseEntity.getBody());  
 }  
}

核心判决业务代码

package com.dazuizui.business.service.onlineJudge.impl;  
  
import cn.hutool.json.JSONObject;  
import com.alibaba.fastjson2.JSONArray;  
import com.dazuizui.basicapi.entry.\*;  
import com.dazuizui.basicapi.entry.bo.ProgramBo;  
import com.dazuizui.business.domain.CodeInContest;  
import com.dazuizui.business.domain.RedisKey;  
import com.dazuizui.business.mapper.ContestMapper;  
import com.dazuizui.business.mapper.LanguageCommandMapper;  
import com.dazuizui.business.mapper.ProblemLimitMapper;  
import com.dazuizui.business.mapper.QuestionCaseMapper;  
import com.dazuizui.business.messageQueue.cofnig.MessageSource;  
import com.dazuizui.business.service.onlineJudge.AcContestQuestionSerivce;  
import com.dazuizui.business.service.onlineJudge.OnlineJudgeService;  
import com.dazuizui.business.service.onlineJudge.ProblemLimitService;  
import com.dazuizui.business.util.HttpUtil;  
import com.dazuizui.business.util.RedisUtil;  
import com.dazuizui.business.util.ThreadLocalUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.integration.support.MessageBuilder;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.Arrays;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.List;  
  
@Service  
public class OnlineJudgeServiceImpl implements OnlineJudgeService {  
 @Autowired  
 private LanguageCommandMapper languageCommandMapper;  
 @Autowired  
 private ProblemLimitMapper problemLimitMapper;  
 @Autowired  
 private QuestionCaseMapper questionCaseMapper;  
 @Autowired  
 private RedisUtil redisUtil;  
 @Autowired  
 private AcContestQuestionSerivce acContestQuestionSerivce;  
 @Autowired  
 private ProblemLimitService problemLimitService;  
 //消息队列  
 @Autowired  
 private MessageSource source;  
 */\*\*  
 \* 判决代码  
 \* 不要动 危险  
 \* -- 杨易达 Bryan yang  
 \** ***@param*** *programBo  
 \** ***@return*** *\*/* @Override  
 @Transactional  
 public String judgeTheProgram(ProgramBo programBo){  
 //封装代码记录  
 CodeInContest codeInContest = new CodeInContest();  
 //获取操作人id  
 String JWTStringID = (String) ThreadLocalUtil.*mapThreadLocalOfJWT*.get().get("userinfo").get("id");  
 Long id = Long.*valueOf*(JWTStringID);  
  
 */\*\*  
 \* 初始化代码运行  
 \*/* HashMap<Integer, List<String>> map = InitializerData.*langSystem*.get(programBo.getLanguageId());  
 programBo.setEnv(Arrays.*asList*("PATH=/usr/bin:/bin"));  
 programBo.setParseCodeArgs(map.get(new Integer(0)));  
 programBo.setCopyOutCached(map.get(new Integer(2)));  
 programBo.setRunCommandArgs(map.get(new Integer(1)));  
  
 */\*\*  
 \* 初始化题目限制  
 \*/* ProblemLimit problemLimit = problemLimitService.getProblemLimitById(programBo.getTopicId());  
 programBo.setProblemLimit(problemLimit);  
  
 */\*\*  
 \* 获取案例he todo 改成mongodb  
 \*/* List<QuestionCase> questionCases = questionCaseMapper.queryTheQuestionCasesByQuestionId(programBo.getTopicId());  
 /\*  
 淘汰方案  
 List<QuestionCase> questionCases = redisUtil.getListInRedis(RedisKey.ZuiOJQuestionCase +programBo.getTopicId());  
 System.out.println(questionCases.size());  
 if (questionCases.size() == 0){  
 questionCases = questionCaseMapper.queryTheQuestionCasesByQuestionId(programBo.getTopicId());  
  
 redisUtil.putListInRedis(RedisKey.ZuiOJQuestionCase +programBo.getTopicId(),60\*60\*24\*15,questionCases);  
 //System.out.println("in db");  
 }  
 \*/  
  
 //封装请求  
 JSONObject request = new JSONObject();  
  
 for (QuestionCase questionCase : questionCases) {  
 programBo.setInput(questionCase.getInputs());  
 //发起请求  
 request = HttpUtil.*request*(programBo);  
  
 if (!request.get("status").equals("Accepted")) {  
 break;  
 }  
  
 //判断答案是否正确  
 JSONObject jsonObject1 = new JSONObject(request.get("files"));  
 String stdout = jsonObject1.get("stdout").toString() ;  
 stdout = stdout.replace("\n","\\n").trim();  
 questionCase.setAnswer( questionCase.getAnswer().replace("\n","\\n").trim());  
  
 */\*\*  
 \* 查看stdout最后两位是否为\n 如果是\n则忽略  
 \* 此处我想不到更好的业务解决方案了，如果后续有人有更好的解决方案请联系我通过email  
 \* y51288033@outlook.com  
 \* y51288033@gmail.com  
 \* 出现的问题是 如果我们正确答案是Hello World  
 \* 但是我们某些语言比如使用println， 打印出来的结果是Hello World\n 我们也想让他通过  
 \* 所以采用此处优化  
 \*/* //System.out.println(stdout.substring(stdout.length() - 2).equals("\\n"));  
 if (stdout.length() >= 2 && stdout.substring(stdout.length()-2).equals("\\n")){  
 stdout = stdout.substring(0,stdout.length()-2);  
 }  
  
 System.*out*.println(questionCase.getAnswer()+" and "+stdout);  
 if (!stdout.trim().equals(questionCase.getAnswer())) {  
 request.set("status","Answer error");  
 break;  
 }  
 }  
  
  
 //如果通过并且是比赛类型的题目  
 if (programBo.getQuestionType() != 1 ) {  
 //查看是否为比赛题目  
 AcContestQuestion acContestQuestion = new AcContestQuestion();  
 acContestQuestion.setContestId(programBo.getContestId());  
 acContestQuestion.setUserId(id);  
 acContestQuestion.setQuestionId(programBo.getTopicId());  
  
 //查看当前啊比赛是否结束  
 if (checkIfTheCurrentGameIsOver(acContestQuestion.getContestId())){  
 //在比赛中标记记  
 Long acContestQuestionId = acContestQuestionSerivce.submitAnswer(acContestQuestion, (String) request.get("status"));  
  
 //封装竞赛id  
 codeInContest.setContestId(programBo.getContestId());  
  
 //获取状态  
 String status = (String) request.get("status");  
 //封装判决状态  
 if (status.equals("File Error")){  
 codeInContest.setStatus(1);  
 }else if (status.equals("Nonzero Exit Status")){  
 codeInContest.setStatus(2);  
 }else if (status.equals("Answer error")){  
 codeInContest.setStatus(3);  
 }else if (status.equals("Accepted")){  
 codeInContest.setStatus(0);  
 }else if (status.equals("Time Limit Exceeded")){  
 codeInContest.setStatus(5);  
 }else{  
 codeInContest.setStatus(6);  
 }  
 //绑定删除状态  
 codeInContest.setDelFlag(0);  
 //绑定通过几率id  
 codeInContest.setAcContestQuestionId(acContestQuestionId);  
 //封装判决代码  
 codeInContest.setCode(programBo.getCode());  
 //封装题目id  
 codeInContest.setQuestionId(programBo.getTopicId());  
 //封装通过者和创建人和创建时间  
 codeInContest.setCreateBy(id);  
 codeInContest.setUserId(id);  
 codeInContest.setCreateTime(new Date());  
  
  
  
 //放入消息队列  
 source.addContestSubmittionCodeOutput().send(MessageBuilder.*withPayload*(codeInContest).build());  
 }  
 }  
  
  
 return JSONArray.*toJSONString*(request);  
 }  
  
 @Autowired  
 private ContestMapper contestMapper;  
 */\*\*  
 \* 检查比赛是否结束  
 \*  
 \* 没结束返回true，结束返回false  
 \** ***@return*** *\*/* public boolean checkIfTheCurrentGameIsOver(Long contestId){  
 //获取比赛数据  
// 因redis Key混乱 后续优化  
// Contest contest = (Contest) redisUtil.getStringInRedis(RedisKey.ZuiOJContestInfo + contestId);  
// if (contest == null){  
// contest = contestMapper.getEventById(contestId);  
// if (contest == null){  
// return false;  
// }else{  
// redisUtil.setStringInRedis(RedisKey.ZuiOJContestInfo + contestId,RedisKey.OutTime,contest);  
// }  
// }  
 Contest contest = contestMapper.getEventById(contestId);;  
  
 //查看是否结束  
 Date endTime = contest.getEndTime();  
 Date cur = new Date();  
 // System.out.println(endTime);  
 //System.out.println(cur);  
 //System.out.println(endTime.after(cur));  
 //int b = endTime.compareTo(cur);  
 return endTime.after(cur);  
 }  
  
}

核心竞赛代码

package com.dazuizui.business.service.onlineJudge.impl;  
  
import com.alibaba.fastjson2.JSONArray;  
import com.dazuizui.basicapi.entry.\*;  
import com.dazuizui.basicapi.entry.vo.ContestInfoVo;  
import com.dazuizui.basicapi.entry.vo.ResponseVo;  
import com.dazuizui.business.domain.RedisKey;  
import com.dazuizui.business.domain.vo.AdminGetArticleByPaginVo;  
import com.dazuizui.business.domain.vo.GetContestDateVo;  
import com.dazuizui.business.mapper.CompetitionInfoMapper;  
import com.dazuizui.business.mapper.ContestMapper;  
import com.dazuizui.business.mapper.ProctorAttributeMapper;  
import com.dazuizui.business.service.onlineJudge.AcContestQuestionSerivce;  
import com.dazuizui.business.service.onlineJudge.ContestSerivce;  
import com.dazuizui.business.util.RedisUtil;  
import com.dazuizui.business.util.ThreadLocalUtil;  
import com.dazuizui.business.util.TransactionUtils;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.TransactionDefinition;  
import org.springframework.transaction.TransactionStatus;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.\*;  
  
*/\*\*  
 \** ***@author*** *yida yang  
 \* 竞赛模块接口实现类  
 \*/*@Service  
public class ContestSerivceImpl implements ContestSerivce {  
  
 @Autowired  
 private ContestMapper conTestMapper;  
 @Autowired  
 private CompetitionInfoMapper competitionInfoMapper;  
 @Autowired  
 private RedisUtil redisUtil;  
 @Autowired  
 private TransactionUtils transactionUtils;  
 @Autowired  
 private ProctorAttributeMapper proctorAttributeMapper;  
 @Autowired  
 private AcContestQuestionSerivce acContestQuestionSerivce;  
  
 @Override  
 public GetContestDateVo getContestData(Long contestId) {  
 GetContestDateVo getContestDateVo = new GetContestDateVo();  
 //获取日志  
 ResponseVo responseVo = acContestQuestionSerivce.querySubmitLogByContestIdOrderUpdateTimeDesc(contestId);  
 getContestDateVo.setAcContestQuestions(getContestDateVo.getAcContestQuestions());  
 return getContestDateVo;  
 }  
  
 */\*\*  
 \* 移除比赛信息  
 \* remove the competition  
 \** ***@param*** *id 比赛id competition ID  
 \*  
 \* 在进行删除了比赛的简介信息和比赛的详细信息和该考场的监考人员和redis中缓存的比赛内容信息数据和所有参赛选手的记录。  
 \*  
 \* on the business layer,the profile info and detailed info of the competition,the invigilators of the  
 \* examination room ,the contest info date of competition cached in redis and records of contestands are deleted  
 \*  
 \** ***@Param*** *id 比赛id Competition ID  
 \*/* @Override  
 public String removeTheContestById(Long id){  
 TransactionStatus transactionStatus = transactionUtils.begin(TransactionDefinition.*ISOLATION\_READ\_COMMITTED*);  
 try {  
 //删除比赛简介信息  
 Long aLong = conTestMapper.removeTheContestById(id);  
 if (aLong == 0){  
 transactionUtils.rollback(transactionStatus);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 //删除比赛详细信息  
 aLong = conTestMapper.removeTheContestDetailedById(id);  
 if (aLong == 0){  
 transactionUtils.rollback(transactionStatus);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 //*todo 删除监考数据* //删除参赛选手信息  
 List<String> competitionInfos = competitionInfoMapper.selectAllUserInTheContestByContestId(id);  
  
 //删除redis中的缓存数据  
  
 redisUtil.batchDeletion(competitionInfos);  
 //删除数据库  
 competitionInfoMapper.deleteCompetitionInfoByContestId(id);  
  
 //删除redis 比赛详细页面  
 redisUtil.deleteKey(RedisKey.*ZuiOJContestInfo*+id);  
 } catch (Exception e) {  
 e.printStackTrace();  
 transactionUtils.rollback(transactionStatus);  
 }  
  
 //提交事物  
 transactionUtils.commit(transactionStatus);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,null, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 修改比赛信息  
 \* 修改了比赛的简介信息和详细信息和更新redis事务  
 \*  
 \* Modify contest info  
 \* modifyed the profile info and details of the contet and updated the redis data  
 \*  
 \** ***@param*** *contest 比赛数据实体  
 \** ***@return*** *\*/* @Override  
 public String updateContest(Contest contest){  
 TransactionStatus transactionStatus = transactionUtils.begin(TransactionDefinition.*ISOLATION\_READ\_COMMITTED*);  
  
 //操作熟练  
 Long numbersOfOpention = 0l;  
  
 try {  
 //修改竞赛简介信息  
 numbersOfOpention = conTestMapper.updateContest(contest);  
 //添加失败  
 if (numbersOfOpention.longValue() == 0){  
 transactionUtils.rollback(transactionStatus);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 //修改竞赛详细Md文档信息  
 numbersOfOpention = conTestMapper.updateContestDetailed(contest);  
 //添加失败  
 if (numbersOfOpention.longValue() == 0){  
 transactionUtils.rollback(transactionStatus);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 //RedisKey.ZuiOJContestId+id  
 System.*err*.println(RedisKey.*ZuiOJContestId*+contest.getId());  
 //修改redis  
  
 redisUtil.setStringInRedis(RedisKey.*ZuiOJContestId*+contest.getId(),RedisKey.*OutTime*,contest);  
// System.err.println(redisUtil.getStringInRedis(RedisKey.ZuiOJContestId+contest.getId()));  
  
 transactionUtils.commit(transactionStatus);  
 } catch (Exception e) {  
 transactionUtils.rollback(transactionStatus);  
 e.printStackTrace();  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
  
  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,null, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \*  
 \* 管理员分页查询比赛数据  
 \* Admin query game data by page  
 \*  
 \* 根据AdminQueryGameInformationByPageVo查询条件进行分页查询  
 \*  
 \* Perform pagination query according to AdminQueryGameInformationByPageVo query conditions  
 \*  
 \** ***@param*** *adminQueryGameInformationByPageBo 分页  
 \** ***@return*** *String  
 \*/* @Override  
 public String adminQueryGameInformationByPage( AdminGetArticleByPaginVo.AdminQueryGameInformationByPageVo adminQueryGameInformationByPageBo){  
 //获取比赛数据  
 List<Contest> contests = conTestMapper.adminQueryGameInformationByPage(adminQueryGameInformationByPageBo);  
 //查询多少个比赛  
 Long count = conTestMapper.queryCoubtOfContest();  
 com.dazuizui.business.domain.vo.AdminQueryGameInformationByPageVo adminQueryGameInformationByPageVo = new com.dazuizui.business.domain.vo.AdminQueryGameInformationByPageVo();  
 adminQueryGameInformationByPageVo.setList(contests);  
 adminQueryGameInformationByPageVo.setCount(count);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,adminQueryGameInformationByPageVo, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 创建比赛  
 \* Creat Contest  
 \* 该接口实现做了将比赛信息添加到数据库和初始化当前赛事的监考信息和设置对应的比赛信息缓存。  
 \* this interface impl adds competition info to the DB,initialize the invigilation info for the current competition  
 \* ,and sets the corresponding competition info Redis cache.  
 \*  
 \** ***@param*** *conTest  
 \** ***@return*** *\*/* @Override  
 @Transactional  
 public String postContest(Contest conTest) {  
 TransactionStatus begin = transactionUtils.begin(TransactionDefinition.*ISOLATION\_READ\_COMMITTED*);  
  
 //设置创建人  
 String strId = (String) ThreadLocalUtil.*mapThreadLocalOfJWT*.get().get("userinfo").get("id");  
 Long id = Long.*valueOf*(strId);  
 conTest.setCreateById(id);  
 conTest.setCreateTime(new Date());  
  
 //添加比赛简略信息  
 long numbersOfOpetion = conTestMapper.insertConTest(conTest);  
 if (numbersOfOpetion == 0){  
 transactionUtils.rollback(begin);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
  
 //添加contest详细页面  
 numbersOfOpetion = conTestMapper.insertConTestDetailed(conTest);  
 if (numbersOfOpetion == 0){  
 transactionUtils.rollback(begin);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
  
 //添加监考信息  
 numbersOfOpetion = proctorAttributeMapper.createAttributeOfProctors(conTest.getId());  
 if (numbersOfOpetion == 0){  
 transactionUtils.rollback(begin);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
  
 transactionUtils.commit(begin);  
  
 System.*err*.println(RedisKey.*ZuiOJContestId*+conTest.getId());  
 redisUtil.setStringInRedis(RedisKey.*ZuiOJContestId*+conTest.getId(),RedisKey.*OutTime*,conTest);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*SuccessfullyCreatedTheCompetition*,null,StatusCode.*SuccessfullyCreatedTheCompetition*));  
 }  
  
 */\*\*  
 \** ***@authro*** *Bryan Yang(大嘴) 16/010/2022  
 \*  
 \* 获取未来赛制  
 \* get future events  
 \*  
 \** ***@return*** *String  
 \*/* @Override  
 public String getFutureEvents() {  
 //获取未来时间比赛  
 List<Contest> futureEvents = conTestMapper.getFutureEvents();  
 return JSONArray.*toJSONString*(new ResponseVo<>("获取未来赛制",futureEvents,"0x1003"));  
 }  
  
  
 */\*\*  
 \** ***@author*** *Bryan Yang(Dazui)  
 \* 获取全部赛制  
 \*  
 \* 业务层  
 \* 获取未来进行时赛事和以前进行时赛事。  
 \*  
 \* Business Layer  
 \* Get future on-going events and previous on-going events.  
 \*  
 \** ***@return*** *String  
 \*/* @Override  
 public String getAllEvents() {  
 //获取未来赛制  
 List<Contest> futureEvents = conTestMapper.getFutureEvents();  
 //获取往期赛制  
 List<Contest> pastevents = conTestMapper.getPastevents();  
 Map<String,List<Contest>> map = new HashMap<>();  
 map.put("future",futureEvents);  
 map.put("past",pastevents);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>("获取全部赛制",map,"200"));  
 }  
  
 */\*\*  
 \* 比赛选手举报  
 \* todo  
 \** ***@param*** *reportMessageText  
 \** ***@return*** *\*/* @Override  
 public String competitorReport(String reportMessageText) {  
 return null;  
 }  
  
 */\*\*  
 \** ***@author*** *03/10/2022 Bryan  
 \*  
 \* 通过id获取赛制  
 \* get event by id  
 \** ***@param*** *id 赛制id  
 \** ***@return*** *\*/* @Override  
 public String getEventById(Long id) {  
 //信息初始化  
 ContestInfoVo contestInfoVo = new ContestInfoVo(); //返回数据  
 Map<String, Object> map = ThreadLocalUtil.*mapThreadLocalOfJWT*.get().get("userinfo");  
 String idInJWTString = (String) map.get("id");  
 Long idInJWt = Long.*valueOf*(idInJWTString);  
 CompetitionInfo competitionInfo = new CompetitionInfo();  
 competitionInfo.setUserId(idInJWt);  
 competitionInfo.setContestId(id);  
  
 //获取比赛  
 Contest contest = (Contest) redisUtil.getStringInRedis(RedisKey.*ZuiOJContestId*+id);  
 if (contest == null) {  
 contest = conTestMapper.getEventById(id);  
 contestInfoVo.setContest(contest);  
 redisUtil.setStringInRedis(RedisKey.*ZuiOJContestId*+id,RedisKey.*OutTime*,contest);  
 }  
 contestInfoVo.setContest(contest);  
  
 //查看是否已经报名  
 CompetitionInfo competitionInfoInDB  
 = (CompetitionInfo) redisUtil.getStringInRedis(RedisKey.*ZuiOJConetstCompetitionInfo* + competitionInfo.getContestId() + ":" + competitionInfo.getUserId());  
 if (competitionInfoInDB == null){  
 competitionInfoInDB = competitionInfoMapper.checkForEntry(competitionInfo);  
 if (competitionInfoInDB != null){  
 redisUtil.setStringInRedis(RedisKey.*ZuiOJConetstCompetitionInfo* + competitionInfo.getContestId() + ":" + competitionInfo.getUserId(),RedisKey.*OutTime*,competitionInfoInDB);  
 }  
 }  
  
 //如果已经报名了则检查检测是否被封禁和是否满足获取题库  
 if (competitionInfoInDB != null){  
 contestInfoVo.setCheckForEntry(true);  
 //*todo 检测是否被封禁* //*todo 是否满足获取题库* return JSONArray.*toJSONString*(new ResponseVo<>("获取赛制通过id",contestInfoVo,"666"));  
 }  
 //*todo 是否满足获取题库* //未报名的结果  
 return JSONArray.*toJSONString*(new ResponseVo<>("获取赛制通过id",contestInfoVo,"200"));  
 }  
  
 */\*\*  
 \*  
 \* 报名比赛  
 \* sign up for the competition  
 \*  
 \* 业务层:  
 \* 首先检测参赛人的id是否已经报名，如果没有报名则写入数据库。  
 \* Business Layer  
 \* First check whether the id of the contestant has registered, and if not, write it into the database.  
 \** ***@param*** *contestId  
 \** ***@return*** *\*/* @Override  
 public String signUpForTheCompetition(Long contestId) {  
  
 Map<String, Object> map = ThreadLocalUtil.*mapThreadLocalOfJWT*.get().get("userinfo");  
 String idInJWTString = (String) map.get("id");  
 Long idInJWt = Long.*valueOf*(idInJWTString);  
 CompetitionInfo competitionInfo = new CompetitionInfo();  
 competitionInfo.setContestId(contestId);  
 competitionInfo.setUserId(idInJWt);  
 //*todo 查看此人是否为封禁状态* //查看此人是否有报名此比赛  
 CompetitionInfo competitionInfoInDB = competitionInfoMapper.checkForEntry(competitionInfo);  
 if (competitionInfoInDB != null){  
 return JSONArray.*toJSONString*(new ResponseVo<>("报名失败，您已经参加此比赛",null,"666"));  
 }  
  
 //报名比赛  
 String username = (String) map.get("username");  
 competitionInfo.setCreateById(idInJWt);  
 competitionInfo.setCreateByName(username);  
 competitionInfo.setCreateTime(new Date());  
 competitionInfoMapper.addCompetitionInfo(competitionInfo);  
  
  
 return JSONArray.*toJSONString*(new ResponseVo<>("报名成功，请注意您的邮箱。",null,"200"));  
 }  
}

比赛业务判决code

package com.dazuizui.business.service.onlineJudge.impl;  
  
import com.alibaba.fastjson2.JSONArray;  
import com.dazuizui.basicapi.entry.AcContestQuestion;  
import com.dazuizui.basicapi.entry.StatusCode;  
import com.dazuizui.basicapi.entry.StatusCodeMessage;  
import com.dazuizui.basicapi.entry.vo.ContestQuestionVo;  
import com.dazuizui.basicapi.entry.vo.ResponseVo;  
import com.dazuizui.business.domain.CodeInContest;  
import com.dazuizui.business.domain.bo.CheckTheSubmitQuesitonDetailInfoBo;  
import com.dazuizui.business.domain.bo.ElementOfQueryLogBo;  
import com.dazuizui.business.domain.bo.QueryContestSubmissionLogBo;  
import com.dazuizui.business.domain.vo.\*;  
import com.dazuizui.business.mapper.AcContestQuestionMapper;  
import com.dazuizui.business.mapper.CompetitionQuestionBankMapper;  
import com.dazuizui.business.mongodao.SubmmitionCodeInContestRepository;  
import com.dazuizui.business.service.onlineJudge.AcContestQuestionSerivce;  
import com.dazuizui.business.service.onlineJudge.SubmmitionCodeInContestSerivce;  
import com.dazuizui.business.util.ThreadLocalUtil;  
import org.apache.ibatis.annotations.Param;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.stereotype.Service;  
  
import java.util.Date;  
import java.util.HashMap;  
import java.util.List;  
  
*/\*\*  
 \* 比赛题目提交业务层操作  
 \*/*@Service  
public class AcContestQuestionSerivceImpl implements AcContestQuestionSerivce {  
 @Autowired  
 private AcContestQuestionMapper acContestQuestionMapper;  
 @Autowired  
 private SubmmitionCodeInContestSerivce submmitionCodeInContestSerivce;  
 @Autowired  
 private SubmmitionCodeInContestRepository submmitionCodeInContestRepository;  
 @Autowired  
 private CompetitionQuestionBankMapper competitionQuestionBankMapper;  
  
 */\*\*  
 \* 提供给oj大屏使用，查看每个题的通过人数量  
 \** ***@param*** *contestId  
 \** ***@return*** *\*/* @Override  
 public ResponseVo findAcCountEveryQuestionByContestIdAndStatus(@Param("contestId")Long contestId){  
 List<FindAcCountEveryQuestionByContestIdAndStatusVo> acCountEveryQuestionByContestIdAndStatus = acContestQuestionMapper.findAcCountEveryQuestionByContestIdAndStatus(contestId);  
  
 return new ResponseVo<>(StatusCodeMessage.*OK*,acCountEveryQuestionByContestIdAndStatus,StatusCode.*OK*);  
 }  
  
 */\*\*  
 \** ***@author*** *Bryan yang 2023 11 11  
 \* 此业务用于websokcet OJ数据大屏的提交日志的数据交互使用  
 \* 在一次获取数据中只会获取20条  
 \** ***@param*** *contestId 比赛id  
 \** ***@return*** *\*/* public ResponseVo querySubmitLogByContestIdOrderUpdateTimeDesc(Long contestId){  
// List<AcContestQuestion> acContestQuestions = acContestQuestionMapper.querySubmitLogByContestIdOrderUpdateTimeDesc(contestId);  
 //获取提交日志  
 Page<CodeInContest> byContestIdOrderByCreateTimeDesc = submmitionCodeInContestRepository.findByContestIdOrderByCreateTimeDesc(contestId, PageRequest.*of*(0, 100));  
 byContestIdOrderByCreateTimeDesc.getTotalElements();  
 List<CodeInContest> content = byContestIdOrderByCreateTimeDesc.getContent();  
 //获取比赛题目  
 List<ContestQuestionVo> questionListByContestId = competitionQuestionBankMapper.getQuestionListByContestId(contestId);  
  
 HashMap<String,Object> map = new HashMap<>();  
 map.put("questionList",questionListByContestId);  
 map.put("codingLog",content);  
  
 return new ResponseVo<>(StatusCodeMessage.*OK*,map,StatusCode.*OK*);  
 }  
  
 */\*\*  
 \** ***@author*** *Bryan Yang(Dazui) 06/08/2022 22:00:00 PM  
 \*  
 \* 通过比赛id还有问题id查询提交日志  
 \* query log by contestId and questionId  
 \*  
 \** ***@param*** *elementOfQueryLogBo 查询元素实体  
 \** ***@return*** *\*/* @Override  
 public String queryLogByContestIdAndQuestionId(ElementOfQueryLogBo elementOfQueryLogBo){  
 List<AcContestQuestion> acContestQuestions = acContestQuestionMapper.queryLogByContestIdAndQuestionId(elementOfQueryLogBo);  
 //Long count = acContestQuestionMapper.queryCountByContestIdAndQuestionId(elementOfQueryLogBo);  
 QueryLogByContestIdAndQuestionIdVo queryLogByContestIdAndQuestionIdVo = new QueryLogByContestIdAndQuestionIdVo();  
 //queryLogByContestIdAndQuestionIdVo.setCount(count);  
 queryLogByContestIdAndQuestionIdVo.setAcContestQuestions(acContestQuestions);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,queryLogByContestIdAndQuestionIdVo, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 通过QuestionId和ContestId查询提交数据  
 \** ***@param*** *elementOfQueryLogBo  
 \** ***@return*** *\*/* @Override  
 public String queryCountByContestIdAndQuestionId(ElementOfQueryLogBo elementOfQueryLogBo){  
 Long count = acContestQuestionMapper.queryCountByContestIdAndQuestionId(elementOfQueryLogBo.getContestId(),elementOfQueryLogBo.getQuestionId());  
 List<AcContestQuestion> acContestQuestions = acContestQuestionMapper.queryLogByElement(elementOfQueryLogBo);  
 QueryLogByElementVo queryLogByElementVo = new QueryLogByElementVo();  
 queryLogByElementVo.setAcContestQuestions(acContestQuestions);  
 queryLogByElementVo.setCount(count);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,queryLogByElementVo, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \** ***@author*** *Bryan Yang y51288033@outlook.com  
 \* 查看详细信息通过user\_id contest\_id  
 \** ***@param*** *checkTheDetailInfoBo  
 \** ***@return*** *\*/* @Override  
 public String checkTheDetailInfoByUserIdAndContestId(CheckTheSubmitQuesitonDetailInfoBo checkTheDetailInfoBo) {  
 List<CheckTheSubmitQuesitonDetailInfoVo> checkTheSubmitQuesitonDetailInfoBos = acContestQuestionMapper.checkTheDetailInfoByUserIdAndContestId(checkTheDetailInfoBo);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,checkTheSubmitQuesitonDetailInfoBos, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \** ***@author*** *Bryan Yang(Dazui) 07/08/2022 22:00:00 PM  
 \*  
 \* 查看提交日志  
 \* query commition log  
 \*  
 \* 业务层  
 \* 通过比赛id和用户名  
 \*  
 \* Business  
 \* query log by contestId and username  
 \*  
 \** ***@param*** *elementOfQueryLogBo 查询元素实体  
 \** ***@return*** *String  
 \*/* @Override  
 public String queryLogByElement(ElementOfQueryLogBo elementOfQueryLogBo) {  
 List<AcContestQuestion> acContestQuestions = acContestQuestionMapper.queryLogByElement(elementOfQueryLogBo);  
 Long count = acContestQuestionMapper.queryCountByElement(elementOfQueryLogBo);  
 QueryLogByElementVo queryLogByElementVo = new QueryLogByElementVo();  
 queryLogByElementVo.setAcContestQuestions(acContestQuestions);  
 queryLogByElementVo.setCount(count);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,queryLogByElementVo, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 删除此用户的提交记录通过Id  
 \** ***@param*** *id  
 \** ***@return*** *\*/* @Override  
 public String deleteLogById(@Param("id")Long id){  
 //删除数据  
 Long aLong = acContestQuestionMapper.deleteLogById(id);  
 //删除现有提交数量  
 if (aLong == 0){  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,null, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \** ***@author*** *Bryan Yang(Dazui) 07/08/2022 22:00:00 PM  
 \*  
 \* 查看本体提交日志  
 \* query commition log  
 \*  
 \* 业务层  
 \* 通过比赛id查询提交日志  
 \*  
 \* Business  
 \* Query submission logs through competition ID  
 \*  
 \** ***@param*** *queryContestSubmissionLogBo 查询元素实体  
 \** ***@return*** *String  
 \*/* public String queryContestSubmissionLog(QueryContestSubmissionLogBo queryContestSubmissionLogBo){  
 //查询分页数据  
 List<AcContestQuestion> acContestQuestions = acContestQuestionMapper.queryContestSubmissionLog(queryContestSubmissionLogBo);  
 //查询个数  
 //*todo 赶工期，暂不优化，等主要功能开发完毕回来优化* Long count = acContestQuestionMapper.queryCount(queryContestSubmissionLogBo.getContestId());  
 //封装  
 QueryContestSubmissionLogVo queryContestSubmissionLogVo = new QueryContestSubmissionLogVo();  
 queryContestSubmissionLogVo.setCount(count);  
 queryContestSubmissionLogVo.setAcContestQuestions(acContestQuestions);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,queryContestSubmissionLogVo, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \*  
 \* 移除当前比赛提交数据  
 \** ***@param*** *id  
 \** ***@return*** *\*/* @Override  
 public String removeAllContestSubmissionLogbyContestId(Long id) {  
 //删除提交日志  
 Long aLong = acContestQuestionMapper.removeAllContestSubmissionLogbyContestId(id);  
 //删除代码存储  
 submmitionCodeInContestSerivce.deleteByContestId(id);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,null, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 提交答案  
 \** ***@param*** *acContestQuestion  
 \** ***@param*** *status 代码判决状态  
 \*/* @Override  
 public Long submitAnswer(AcContestQuestion acContestQuestion,String status) {  
 //在比赛信息标记通过  
 AcContestQuestion acContestQuestionInDB = acContestQuestionMapper.checkSubmissions(acContestQuestion);  
 acContestQuestion.setCreateByName((String) ThreadLocalUtil.*mapThreadLocalOfJWT*.get().get("userinfo").get("name"));  
 String idstring = (String) ThreadLocalUtil.*mapThreadLocalOfJWT*.get().get("userinfo").get("id");  
 acContestQuestion.setCreateById(Long.*valueOf*(idstring));  
 acContestQuestion.setCreateTime(new Date());  
 acContestQuestion.setUpdateTime(new Date());  
 */\*\*  
 \* 如果没有提交记录就创建提交记录  
 \* 如果有提交记录，就当前比赛的题目的提交次数++  
 \* 第一次提交就ac的情况  
 \*/* if (acContestQuestionInDB == null){  
  
 if (status.equals("Accepted")){  
 acContestQuestion.setFirstAc(new Date());  
 acContestQuestion.setNumberOfAttempts(0);  
 acContestQuestionMapper.acTheQuestionInDB(acContestQuestion,1,acContestQuestion.getUpdateTime());  
 }else{  
 acContestQuestion.setNumberOfAttempts(1);  
 acContestQuestionMapper.acTheQuestionInDB(acContestQuestion,0,acContestQuestion.getUpdateTime());  
 }  
 return acContestQuestion.getId();  
 }else{  
 */\*\*  
 \* acContestQuestionInDB.getStatus() == 1 代表已经AC  
 \* acContestQuestionInDB.getStatus() == 0 还未AC  
 \* 此if条件是罚时  
 \* 如果已经AC但是没有通过，或者没有ac也没有通过就增加错误次数  
 \*/* if (acContestQuestionInDB.getStatus() == 1 && !status.equals("Accepted")){  
 acContestQuestion.setFirstAc(acContestQuestionInDB.getFirstAc() == null ? null : acContestQuestionInDB.getFirstAc());  
 acContestQuestion.setNumberOfAttempts(acContestQuestionInDB.getNumberOfAttempts()+1);  
 acContestQuestionMapper.recordSubmissions(acContestQuestion,1,acContestQuestion.getUpdateTime());  
 }  
 //  
 else if (acContestQuestionInDB.getStatus() == 0 && !status.equals("Accepted")) {  
 acContestQuestion.setFirstAc(acContestQuestionInDB.getFirstAc() == null ? null : acContestQuestionInDB.getFirstAc());  
 acContestQuestion.setNumberOfAttempts(acContestQuestionInDB.getNumberOfAttempts()+1);  
 acContestQuestionMapper.recordSubmissions(acContestQuestion,0,acContestQuestion.getUpdateTime());  
 }  
 */\*\*  
 \* 如果是首次通过  
 \*/* else if (status.equals("Accepted") && acContestQuestionInDB.getStatus() == 0){  
 acContestQuestion.setFirstAc(new Date());  
 acContestQuestion.setNumberOfAttempts(acContestQuestionInDB.getNumberOfAttempts());  
 acContestQuestionMapper.recordSubmissions(acContestQuestion,1,acContestQuestion.getUpdateTime());  
 }  
  
  
  
 return acContestQuestionInDB.getId();  
 }  
 }  
}

代码持久化业务

package com.dazuizui.business.service.onlineJudge.impl;  
  
import cn.hutool.core.lang.hash.Hash;  
import com.alibaba.fastjson2.JSONArray;  
import com.dazuizui.basicapi.entry.AcContestQuestion;  
import com.dazuizui.basicapi.entry.StatusCode;  
import com.dazuizui.basicapi.entry.StatusCodeMessage;  
import com.dazuizui.basicapi.entry.vo.ResponseVo;  
import com.dazuizui.business.domain.CodeInContest;  
import com.dazuizui.business.domain.StatusCountMapEntry;  
import com.dazuizui.business.domain.bo.DuplicateCodeBo;  
import com.dazuizui.business.domain.bo.FilterQueryMatchSaveCodeBo;  
import com.dazuizui.business.domain.vo.DuplicateCodeVo;  
import com.dazuizui.business.domain.vo.FilterQueryMatchSaveCodeVo;  
import com.dazuizui.business.mapper.AcContestQuestionMapper;  
import com.dazuizui.business.mapper.CodeDetailedInContestMapper;  
import com.dazuizui.business.mapper.CodeInContestMapper;  
import com.dazuizui.business.mongodao.SubmmitionCodeInContestRepository;  
import com.dazuizui.business.service.onlineJudge.SubmmitionCodeInContestSerivce;  
import com.dazuizui.business.util.ThreadLocalUtil;  
import com.dazuizui.business.util.TransactionUtils;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.data.mongodb.core.MongoTemplate;  
import org.springframework.data.mongodb.core.aggregation.Aggregation;  
import org.springframework.data.mongodb.core.aggregation.AggregationResults;  
import org.springframework.data.mongodb.core.aggregation.TypedAggregation;  
import org.springframework.data.mongodb.core.query.Criteria;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.TransactionDefinition;  
import org.springframework.transaction.TransactionStatus;  
  
import java.util.\*;  
  
*/\*\*  
 \* 存储在比赛时候提交的代码的表业务接口  
 \*/*@Service  
public class SubmmitionCodeInContestSerivceImpl implements SubmmitionCodeInContestSerivce {  
 @Autowired  
 private SubmmitionCodeInContestRepository submmitionCodeInContestRepository;  
 @Autowired  
 private CodeInContestMapper codeInContestMapper;  
 @Autowired  
 private CodeDetailedInContestMapper codeDetailedInContestMapper;  
 @Autowired  
 private AcContestQuestionMapper acContestQuestionMapper;  
 @Autowired  
 private TransactionUtils transactionUtils;  
 @Autowired  
 private MongoTemplate mongoTemplate;  
  
 */\*\*  
 \* 该业务接口主要给websocket OJ数据大屏使用.  
 \* 查询当前比赛status所有状态的数量，  
 \** ***@param*** *contestId  
 \** ***@return*** *\*/* @Override  
 public ResponseVo findStatusCountsByContestId(Long contestId) {  
 TypedAggregation<CodeInContest> typedAggregation = TypedAggregation.*newAggregation*(CodeInContest.class,  
 Aggregation.*match*(Criteria.*where*("contestId").is(contestId)),Aggregation.*group*("status").count().as("count"));  
  
 AggregationResults<Map> aggregate = mongoTemplate.aggregate(typedAggregation, Map.class);  
 List<Map> mappedResults = aggregate.getMappedResults();  
 System.*err*.println(mappedResults);  
  
 List<StatusCountMapEntry> statusCountLists = new ArrayList<>();  
  
 for (Map map : mappedResults) {  
 Object status = map.get("\_id"); // Assuming "\_id" is the key for the grouped field  
 Integer count = (Integer) map.get("count");  
 String key = status.toString();  
 StatusCountMapEntry entry = new StatusCountMapEntry();  
 entry.setKey(key);  
 entry.setCount(count);  
 statusCountLists.add(entry);  
  
 }  
  
  
 HashMap res = new HashMap();  
 res.put("statusCountMap",statusCountLists);  
  
 return new ResponseVo(StatusCodeMessage.*OK*,res,StatusCode.*OK*);  
 }  
  
 */\*\*  
 \* 获取涉嫌重复的代码  
 \** ***@param*** *duplicateCode  
 \** ***@return*** *\*/* @Override  
 public String getDuplicateCode(DuplicateCodeBo duplicateCode){  
  
 CodeInContest guestCode = codeDetailedInContestMapper.findOneById(duplicateCode.getGuestCodeId());  
 CodeInContest masterCOde = codeDetailedInContestMapper.findOneById(duplicateCode.getMasterCodeId());  
  
 DuplicateCodeVo duplicateCodeVo = new DuplicateCodeVo();  
 duplicateCodeVo.setGuestCode(guestCode.getCode());  
 duplicateCodeVo.setMasterCode(masterCOde.getCode());  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,duplicateCodeVo, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 通过比赛id还有题目id还有用户id指定提交代码数据业务接口  
 \** ***@param*** *findByContestIdAndQuestionIdAndUserIdBo  
 \** ***@return*** *\*/* @Override  
 public String filterQueryMatchSaveCode(FilterQueryMatchSaveCodeBo findByContestIdAndQuestionIdAndUserIdBo) {  
 //查询mongodb  
 Page<CodeInContest> byContestIdAndQuestionIdAndUserId = submmitionCodeInContestRepository  
 .findByContestIdAndQuestionIdAndUserIdAndDelFlag(  
 findByContestIdAndQuestionIdAndUserIdBo.getContestId(),  
 findByContestIdAndQuestionIdAndUserIdBo.getQuestionId(),  
 findByContestIdAndQuestionIdAndUserIdBo.getUserId(),  
 0,  
 PageRequest.*of*(findByContestIdAndQuestionIdAndUserIdBo.getPage(), findByContestIdAndQuestionIdAndUserIdBo.getSize()));  
  
  
 FilterQueryMatchSaveCodeVo filterQueryMatchSaveCodeVo = new FilterQueryMatchSaveCodeVo();  
 filterQueryMatchSaveCodeVo.setCodeInContestList(byContestIdAndQuestionIdAndUserId.getContent());  
 filterQueryMatchSaveCodeVo.setCount(byContestIdAndQuestionIdAndUserId.getTotalElements());  
  
 //System.out.println(filterQueryMatchSaveCodeVo);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,filterQueryMatchSaveCodeVo, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 通过id查询详细代码  
 \* todo redis优化  
 \** ***@param*** *id  
 \** ***@return*** *\*/* @Override  
 public String findOneById(Long id) {  
 CodeInContest codeInContest = codeDetailedInContestMapper.findOneById(id);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,codeInContest, StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 通过比赛id删除代码信息  
 \** ***@param*** *contestId  
 \** ***@return*** *\*/* @Override  
 public String deleteByContestId(Long contestId) {  
 submmitionCodeInContestRepository.deleteByContestId(contestId);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,"", StatusCode.*OK*));  
 }  
  
 */\*\*  
 \* 通过提交记录id删除记录  
 \** ***@param*** *id mongoDb主键id  
 \** ***@return*** *\*/* @Override  
 public String deleteById(String id) {  
 //解析操作人id  
 //通过id查询到指定数据  
 Optional<CodeInContest> byId = submmitionCodeInContestRepository.findById(id);  
 CodeInContest codeInContest = byId.get();  
 codeInContest.setCreateBy((Long.*valueOf*( ThreadLocalUtil.*mapThreadLocalOfJWT*.get().get("userinfo").get("id")+"")));  
 //获取当前提交信息  
 AcContestQuestion acContestQuestion = acContestQuestionMapper.findOneById(codeInContest.getAcContestQuestionId());  
 acContestQuestion.setCreateById((Long.*valueOf*( ThreadLocalUtil.*mapThreadLocalOfJWT*.get().get("userinfo").get("id")+"")));  
 //设置修改人和修改时间  
 acContestQuestion.setUpdateTime(new Date());  
 //减少通关次数  
 acContestQuestion.setNumberOfAttempts(acContestQuestion.getNumberOfAttempts()-1);  
 //ac次数  
 long acCount = 0;  
 //查看是否为通过状态  
 if (codeInContest.getStatus() == 0){  
 //查看是否拥有ac记录  
 Page<CodeInContest> byContestIdAndQuestionIdAndUserIdAndDelFlagAndStatus = submmitionCodeInContestRepository.findByContestIdAndQuestionIdAndUserIdAndDelFlagAndStatus(  
 codeInContest.getContestId(),  
 codeInContest.getQuestionId(),  
 codeInContest.getUserId(),  
 codeInContest.getDelFlag(),  
 codeInContest.getStatus(),  
 PageRequest.*of*(0, 2)  
 );  
 //如果只存在一个ac记录，那么就清除该ac记录  
 acCount = byContestIdAndQuestionIdAndUserIdAndDelFlagAndStatus.getTotalElements();  
 if (acCount == 1){  
 //清除首次ac时间  
 acContestQuestion.setFirstAc(null);  
 acContestQuestion.setStatus(0);  
 }  
 }  
  
 TransactionStatus begin = transactionUtils.begin(TransactionDefinition.*ISOLATION\_READ\_COMMITTED*);  
  
 try {  
 //删除代码数据 todo改成逻辑删除  
 Long aLong = codeDetailedInContestMapper.deleteCodeById(codeInContest.getCodeId());  
 if (aLong == 0){  
 transactionUtils.rollback(begin);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
  
 //如果尝试次数大于0并且尝试的数据不是AC数据那么就减少一次数据，否则就清除数据  
 if (codeInContest.getStatus() != 0 && acContestQuestion.getNumberOfAttempts() >= 0){  
 //开始修改mysql的数据  
 aLong = acContestQuestionMapper.updateAcContestQuestionById(acContestQuestion);  
 if (aLong == 0){  
 transactionUtils.rollback(begin);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 }else{  
  
 // 如果状态为AC，并且只通关一次 并且尝试一次为 0次（ < 0）则清除数据  
 // 如果没有ac并且尝试一次为 0次（ < 0）则清除数据  
 if (  
 codeInContest.getStatus() ==0 && acCount ==1 && acContestQuestion.getNumberOfAttempts() < 0||  
 codeInContest.getStatus() !=0 && acContestQuestion.getNumberOfAttempts() < 0  
 ){  
 //通过id删除数据  
 aLong = acContestQuestionMapper.deleteLogById(acContestQuestion.getId());  
 if (aLong == 0){  
 transactionUtils.rollback(begin);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 }  
 //如果状态为ac 并且尝试次数大于0（ >= 0），那么就更新通过状态  
 else if (codeInContest.getStatus() == 0 && acContestQuestion.getNumberOfAttempts() >= 0){  
 acContestQuestion.setNumberOfAttempts(acContestQuestion.getNumberOfAttempts()+1);  
 aLong = acContestQuestionMapper.updateAcContestQuestionById(acContestQuestion);  
 if (aLong == 0){  
 transactionUtils.rollback(begin);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 }  
 }  
  
 //删除mongoDB提交数据  
 submmitionCodeInContestRepository.deleteById(id);  
 } catch (Exception e) {  
 e.printStackTrace();  
 transactionUtils.rollback(begin);  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*Error*,null, StatusCode.*Error*));  
 }  
 transactionUtils.commit(begin);  
  
 return JSONArray.*toJSONString*(new ResponseVo<>(StatusCodeMessage.*OK*,null, StatusCode.*OK*));  
 }  
}

查重代码核心业务

package com.dazuizui.business.util;  
  
import java.math.BigInteger;  
import java.util.\*;  
  
public class CodeDuplicationChecker {  
 private static Set<String> *whitelist*; // 白名单集合，存放不需要查重的代码片段  
  
 static {  
 *whitelist* = new HashSet<>();  
 // 在白名单中添加不需要查重的代码片段  
 *whitelist*.add("import");  
 *whitelist*.add("package");  
 *whitelist*.add("public");  
 *whitelist*.add("private");  
 *whitelist*.add("protected");  
 *whitelist*.add("class");  
 *whitelist*.add("interface");  
 *whitelist*.add("extends");  
 *whitelist*.add("implements");  
 *whitelist*.add("void");  
 *whitelist*.add("static");  
 *whitelist*.add("new");  
 *whitelist*.add("this");  
 *whitelist*.add("super");  
 *whitelist*.add("if");  
 *whitelist*.add("else");  
 *whitelist*.add("for");  
 *whitelist*.add("while");  
 *whitelist*.add("do");  
 *whitelist*.add("switch");  
 *whitelist*.add("case");  
 *whitelist*.add("break");  
 *whitelist*.add("continue");  
 *whitelist*.add("return");  
 *whitelist*.add("try");  
 *whitelist*.add("catch");  
 *whitelist*.add("finally");  
 *whitelist*.add("throw");  
 *whitelist*.add("throws");  
 *whitelist*.add("assert");  
 *whitelist*.add("true");  
 *whitelist*.add("false");  
 *whitelist*.add("null");  
 // 添加更多白名单条目...  
 }  
  
  
 public static void main(String[] args) {  
 // 假设您有一批学生提交的代码，存放在一个列表中  
 List<String> studentCodes = new ArrayList<>();  
 studentCodes.add("你好");  
 studentCodes.add("你好");  
  
  
 // 检查每对学生代码的相似性，并输出结果  
 int totalComparisons = 0;  
 int similarPairs = 0;  
  
 for (int i = 0; i < studentCodes.size(); i++) {  
 for (int j = i + 1; j < studentCodes.size(); j++) {  
 totalComparisons++;  
  
 String code1 = studentCodes.get(i);  
 String code2 = studentCodes.get(j);  
  
 // 检查代码片段是否在白名单中  
 if (*isWhitelisted*(code1) || *isWhitelisted*(code2)) {  
 continue; // 在白名单中的代码片段不需要查重，跳过  
 }  
  
 // 计算SimHash相似性  
 double simHashSimilarity = *calculateSimHashSimilarity*(code1, code2);  
  
 // 计算Jaccard相似性  
 double jaccardSimilarity = *calculateJaccardSimilarity*(code1, code2);  
  
 // 计算编辑距离  
 int editDistance = *calculateEditDistance*(code1, code2);  
  
 // 综合考虑相似性度量，并根据需要进行权衡  
 double combinedSimilarity = *combineSimilarities*(simHashSimilarity, jaccardSimilarity, editDistance);  
  
 // 如果相似性超过阈值，认为存在查重  
 double threshold = 0.5; // 设置相似性阈值  
 if (combinedSimilarity >= threshold) {  
 similarPairs++;  
 System.*out*.println("存在代码查重：");  
 System.*out*.println("代码片段1：");  
 System.*out*.println(code1);  
 System.*out*.println("代码片段2：");  
 System.*out*.println(code2);  
  
 }  
 System.*out*.println("综合相似性：" + combinedSimilarity);  
 System.*out*.println("------------------------");  
 }  
 }  
  
 // 计算最终查重率  
 double duplicationRate = (double) similarPairs / totalComparisons;  
 System.*out*.println("最终查重率：" + duplicationRate);  
 }  
  
 // 计算SimHash相似性的方法  
 private static double calculateSimHashSimilarity(String code1, String code2) {  
 BigInteger simHash1 = *calculateSimHash*(code1);  
 BigInteger simHash2 = *calculateSimHash*(code2);  
  
 int hammingDistance = *calculateHammingDistance*(simHash1, simHash2);  
  
 return *calculateSimilarityScore*(hammingDistance);  
 }  
  
 // 计算SimHash值  
 private static BigInteger calculateSimHash(String code) {  
 // 设置SimHash的位数（根据需要调整）  
 int bitCount = 64;  
 int[] simHashArray = new int[bitCount];  
  
 // 分词并计算每个词的哈希值  
 String[] tokens = code.split("\\s+");  
 Map<Integer, Integer> wordFrequencyMap = new HashMap<>();  
  
 for (String token : tokens) {  
 int hash = token.hashCode();  
 wordFrequencyMap.put(hash, wordFrequencyMap.getOrDefault(hash, 0) + 1);  
 }  
  
 // 计算SimHash值  
 for (int hash : wordFrequencyMap.keySet()) {  
 int hashValue = hash;  
 for (int i = 0; i < bitCount; i++) {  
 int bit = (hashValue & 1) == 1 ? 1 : -1;  
 simHashArray[i] += bit;  
 hashValue >>= 1;  
 }  
 }  
  
 BigInteger simHash = BigInteger.*ZERO*;  
 for (int i = 0; i < bitCount; i++) {  
 if (simHashArray[i] > 0) {  
 simHash = simHash.setBit(i);  
 }  
 }  
  
 return simHash;  
 }  
  
 // 计算汉明距离  
 private static int calculateHammingDistance(BigInteger simHash1, BigInteger simHash2) {  
 BigInteger xor = simHash1.xor(simHash2);  
 int hammingDistance = xor.bitCount();  
 return hammingDistance;  
 }  
  
 // 计算相似性分数  
 private static double calculateSimilarityScore(int hammingDistance) {  
 return 1.0 / (hammingDistance + 1);  
 }  
  
 // 计算Jaccard相似性的方法  
 private static double calculateJaccardSimilarity(String code1, String code2) {  
 Set<String> words1 = new HashSet<>(Arrays.*asList*(code1.split("\\s+")));  
 Set<String> words2 = new HashSet<>(Arrays.*asList*(code2.split("\\s+")));  
  
 // 计算交集大小  
 int intersectionSize = 0;  
 for (String word : words1) {  
 if (*isWhitelisted*(word)) {  
 continue; // 如果单词在白名单中，跳过  
 }  
 if (words2.contains(word)) {  
 intersectionSize++;  
 }  
 }  
  
 // 计算并集大小  
 int unionSize = words1.size() + words2.size() - intersectionSize;  
  
 // 计算Jaccard相似性  
 if (unionSize == 0) {  
 return 0.0; // 避免除以零错误  
 } else {  
 return (double) intersectionSize / unionSize;  
 }  
 }  
  
 // 检查单词是否在白名单中  
 private static boolean isWhitelisted(String word) {  
 return *whitelist*.contains(word);  
 }  
  
  
 // 计算编辑距离的方法  
 private static int calculateEditDistance(String code1, String code2) {  
 // 使用动态规划计算编辑距离  
 int m = code1.length();  
 int n = code2.length();  
 int[][] dp = new int[m + 1][n + 1];  
  
 for (int i = 0; i <= m; i++) {  
 for (int j = 0; j <= n; j++) {  
 if (i == 0) {  
 dp[i][j] = j;  
 } else if (j == 0) {  
 dp[i][j] = i;  
 } else if (code1.charAt(i - 1) == code2.charAt(j - 1)) {  
 dp[i][j] = dp[i - 1][j - 1];  
 } else {  
 dp[i][j] = 1 + Math.*min*(Math.*min*(dp[i - 1][j], dp[i][j - 1]), dp[i - 1][j - 1]);  
 }  
 }  
 }  
  
 return dp[m][n];  
 }  
  
 // 综合考虑相似性度量并根据需要进行权衡  
 private static double combineSimilarities(double simHashSimilarity, double jaccardSimilarity, int editDistance) {  
 // 这里可以根据需要进行权衡不同相似性度量的贡献  
 // 这个示例中，平均了三个相似性分数，您可以根据实际情况进行调整  
 return (simHashSimilarity + jaccardSimilarity + (1.0 / (editDistance + 1))) / 3.0;  
 }  
  
  
}