```
package com.cy.iamlbmp.mapper;
import java.util.List;
import com.cy.iamlbmp.domain.Report;
import com.baomidou.mybatisplus.core.mapper.BaseMapper;
/**
 * 报告 Dao 接口
 */
public interface ReportMapper extends BaseMapper<Report>{
    /**
     * 新增报告
     *
     * @param report 报告
     * @return
     */
    int insertReport(Report report);

    /**
     * 修改报告
     *
     * @param report 报告
     * @return
     */
    int updateReport(Report report);

    /**
     * 查询报告列表
     *
     * @param reportId 报告主键
     * @return
     */
    Report selectReportByReportId(Integer reportId);

    /**
     * 根据条件查询报告
     */
    List<ReportVo> selectByCondition(ReportQueryDto queryReportDto);

    /**
     * 删除报告
     * @param reportId 报告主键
     * @return
     */
    int deleteReportByReportId(Integer reportId);

    /**
     * 批量删除报告
     * @param reportIds  ID 集合
     * @return
     */
    int deleteReportByReportIds(Integer[] reportIds);
}

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/myba
tis-3-mapper.dtd">
<mapper namespace="com.cy.iamlbmp.mapper.BlacklistMapper">
    <resultMap type="Blacklist" id="BlacklistResult">
        <result property="blacklistId" column="blacklist_ID"    />
        <result property="name" column="name"    />
        <result property="customerId" column="customer_ID"    />
        <result property="expirationDate" column="expiration_date"    />
        <result property="createdAt" column="created_at"    />
        <result property="reason" column="reason"    />
        <result property="updatedAt" column="updated_at"    />
        <result property="source" column="source"    />
        <result property="nationality" column="nationality"    />
        <result property="listedDate" column="listed_date"    />
    </resultMap>
    <sql id="selectBlacklistVo">
        select blacklist_ID, name, customer_ID, expiration_date, created_at, reason, upda
ted_at, source, nationality, listed_date
        from iamlbmp_blacklist
    </sql>
    <select id="selectBlacklistList" parameterType="Blacklist" resultMap="BlacklistResult
">
        <include refid="selectBlacklistVo"/>
        <where>
            <if test="blacklistId != null ">
            and blacklist_ID = #{blacklistId}
            </if>
            <if test="name != null and name != ''">
            and name = #{name}
            </if>
            <if test="customerId != null ">
            and customer_ID = #{customerId}
            </if>
            <if test="expirationDate != null and expirationDate != ''">
            and expiration_date = #{expirationDate}
            </if>
            <if test="createdAt != null ">
            and created_at = #{createdAt}
            </if>
            <if test="reason != null and reason != ''">
            and reason = #{reason}
            </if>
            <if test="updatedAt != null ">
            and updated_at = #{updatedAt}
            </if>
            <if test="source != null and source != ''">
            and source = #{source}
            </if>
            <if test="nationality != null and nationality != ''">
            and nationality = #{nationality}
            </if>
            <if test="listedDate != null and listedDate != ''">
            and listed_date = #{listedDate}
            </if>
        </where>
    </select>
    <delete id="deleteBlacklistByBlacklistId" parameterType="Integer">
        delete from iamlbmp_blacklist where blacklist_ID = #{blacklistId}
    </delete>

    <insert id="insertBlacklist" parameterType="Blacklist" useGeneratedKeys="true" keyPro
perty="blacklistId">
        insert into iamlbmp_blacklist
        <trim prefix="(" suffix=")" suffixOverrides=",">
            <if test="name != null and name != ''">name,</if>
            <if test="customerId != null">customer_ID,</if>
            <if test="expirationDate != null and expirationDate != ''">expiration_date,</
if>
            <if test="createdAt != null">created_at,</if>
            <if test="reason != null and reason != ''">reason,</if>
            <if test="updatedAt != null">updated_at,</if>
            <if test="source != null and source != ''">source,</if>
            <if test="nationality != null and nationality != ''">nationality,</if>
            <if test="listedDate != null and listedDate != ''">listed_date,</if>
         </trim>
        <trim prefix="values (" suffix=")" suffixOverrides=",">
            <if test="name != null and name != ''">#{name},</if>
            <if test="customerId != null">#{customerId},</if>
            <if test="expirationDate != null and expirationDate != ''">#{expirationDate},
</if>
            <if test="createdAt != null">#{createdAt},</if>
            <if test="reason != null and reason != ''">#{reason},</if>
            <if test="updatedAt != null">#{updatedAt},</if>
            <if test="source != null and source != ''">#{source},</if>
            <if test="nationality != null and nationality != ''">#{nationality},</if>
            <if test="listedDate != null and listedDate != ''">#{listedDate},</if>
         </trim>
    </insert>
    <delete id="deleteBlacklistByBlacklistIds" parameterType="String">
        delete from iamlbmp_blacklist where blacklist_ID in
        <foreach item="blacklistId" collection="array" open="(" separator="," close=")">
            #{blacklistId}
        </foreach>
    </delete>
</mapper>
package com.cy.iamlbmp.io.file;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufAllocator;
import io.netty.buffer.ByteBufUtil;
import io.netty.buffer.Unpooled;
import io.scalecube.services.annotations.ServiceMethod;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.codec.digest.DigestUtils;
import org.hswebframework.ezorm.rdb.mapping.ReactiveRepository;
import org.hswebframework.web.crud.events.EntityDeletedEvent;
import org.hswebframework.web.exception.BusinessException;
import org.hswebframework.web.exception.NotFoundException;
import org.hswebframework.web.id.IDGenerator;
import com.cy.iamlbmp.config.ConfigManager;
import org.jetlinks.core.rpc.RpcManager;
import org.springframework.context.event.EventListener;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.buffer.*;
import org.springframework.http.codec.multipart.FilePart;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;
import java.io.File;
import java.nio.file.NoSuchFileException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.security.MessageDigest;
import java.time.Duration;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Objects;
import java.util.function.Function;
@Slf4j
public class ClusterFileManager implements FileManager {
    public static final String API_PATH_CONFIG_NAME = "paths";
    public static final String API_PATH_CONFIG_KEY = "base-path";
    private final FileProperties properties;
    private final NettyDataBufferFactory bufferFactory = new NettyDataBufferFactory(ByteB
ufAllocator.DEFAULT);
    private final ReactiveRepository<FileEntity, String> repository;
    private final RpcManager rpcManager;
    private final ConfigManager configManager;
    public ClusterFileManager(RpcManager rpcManager,
                              FileProperties properties,
                              ReactiveRepository<FileEntity, String> repository,
                              ConfigManager configManager) {
        new File(properties.getStorageBasePath()).mkdirs();
        this.properties = properties;
        this.rpcManager = rpcManager;
        this.repository = repository;
        this.configManager = configManager;
        rpcManager.registerService(new ServiceImpl());
        if (!properties.getTempFilePeriod().isZero()) {
            Duration duration = Duration.ofHours(1);
            if (duration.toMillis() > properties.getTempFilePeriod().toMillis()) {
                duration = properties.getTempFilePeriod();
            }
            Flux.interval(duration)
                .onBackpressureDrop()
                .concatMap(ignore -> repository
                    .createDelete()
                    .where(FileEntity::getServerNodeId, rpcManager.currentServerId())
                    .lte(FileEntity::getCreateTime,
                        System.currentTimeMillis() - properties.getTempFilePeriod().toMil
lis())
                    .and(FileEntity::getOptions, "in$any", FileOption.tempFile)
                    .execute()
                    .onErrorResume(err -> {
                        log.warn("delete temp file error", err);
                        return Mono.empty();
                    }))
                .subscribe();
        }

    }

    private Mono<String> getApiBasePath() {
        return configManager
            .getProperties(API_PATH_CONFIG_NAME)
            .mapNotNull(val -> val.getString(API_PATH_CONFIG_KEY, null));
    }

    @Override
    public Mono<FileInfo> saveFile(FilePart filePart, FileOption... options) {
        return saveFile(filePart.filename(), filePart.content(), options);
    }

    private DataBuffer updateDigest(MessageDigest digest, DataBuffer dataBuffer) {
        dataBuffer = DataBufferUtils.retain(dataBuffer);
        digest.update(dataBuffer.asByteBuffer());
        DataBufferUtils.release(dataBuffer);
        return dataBuffer;
    }

    public Mono<FileInfo> doSaveFile(String name, Flux<DataBuffer> stream, FileOption...
options) {
        LocalDate now = LocalDate.now();
        FileInfo fileInfo = new FileInfo();
        fileInfo.setId(IDGenerator.MD5.generate());
        fileInfo.withFileName(name);
        String storagePath = now.format(DateTimeFormatter.BASIC_ISO_DATE)
            + "/" + fileInfo.getId() + "." + fileInfo.getExtension();
        MessageDigest md5 = DigestUtils.getMd5Digest();
        MessageDigest sha256 = DigestUtils.getSha256Digest();
        String storageBasePath = properties.getStorageBasePath();
        String serverNodeId = rpcManager.currentServerId();
        Path path = Paths.get(storageBasePath, storagePath);
        path.toFile().getParentFile().mkdirs();
        return stream
            .map(buffer -> updateDigest(md5, updateDigest(sha256, buffer)))
            .as(buf -> DataBufferUtils
                .write(buf, path,
                    StandardOpenOption.WRITE,
                    StandardOpenOption.CREATE_NEW,
                    StandardOpenOption.TRUNCATE_EXISTING))
            .then(Mono.defer(() -> {
                File savedFile = Paths.get(storageBasePath, storagePath).toFile();
                if (!savedFile.exists()) {
                    return Mono.error(new BusinessException("error.file_storage_failed"));
                }
                fileInfo.withAccessKey(IDGenerator.MD5.generate());
                fileInfo.setMd5(ByteBufUtil.hexDump(md5.digest()));
                fileInfo.setSha256(ByteBufUtil.hexDump(sha256.digest()));
                fileInfo.setLength(savedFile.length());
                fileInfo.setCreateTime(System.currentTimeMillis());
                fileInfo.setOptions(options);
                FileEntity entity = FileEntity.of(fileInfo, storagePath, serverNodeId);
                return repository
                    .insert(entity)
                    .then(Mono.defer(() -> {
                        FileInfo response = entity.toInfo();
                        return this
                            .getApiBasePath().doOnNext(response::withBasePath)
                            .thenReturn(response);
                    }));
            }));
    }

    @Override
    public Mono<FileInfo> saveFile(String name, Flux<DataBuffer> stream, FileOption... op
tions) {
        return doSaveFile(name, stream, options);
    }

    @Override
    public Mono<FileInfo> getFileByMd5(String md5) {
        return repository
            .createQuery()
            .where(FileEntity::getMd5, md5)
            .fetchOne()
            .map(FileEntity::toInfo);
    }

    @Override
    public Mono<FileInfo> getFileBySha256(String sha256) {
        return repository
            .createQuery()
            .where(FileEntity::getSha256, sha256)
            .fetchOne()
            .map(FileEntity::toInfo);
    }

    @Override
    public Mono<FileInfo> getFile(String id) {
        return repository
            .findById(id)
            .map(FileEntity::toInfo);
    }

    private Flux<DataBuffer> readFile(String filePath, long position) {
        return DataBufferUtils
            .read(new FileSystemResource(Paths.get(properties.getStorageBasePath(), fileP
ath)),
                position,
                bufferFactory,
                (int) properties.getReadBufferSize().toBytes())
            .onErrorMap(NoSuchFileException.class, e -> new NotFoundException());
    }

    private Flux<DataBuffer> readFile(FileEntity file, long position) {
        if (Objects.equals(file.getServerNodeId(), rpcManager.currentServerId())) {
            return readFile(file.getStoragePath(), position);
        }
        return readFromAnotherServer(file, position);
    }

    protected Flux<DataBuffer> readFromAnotherServer(FileEntity file, long position) {

        return rpcManager
            .getService(file.getServerNodeId(), Service.class)
            .switchIfEmpty(Mono.error(NotFoundException::new))
            .flatMapMany(service -> service.read(new ReadRequest(file.getId(), position)))
            .<DataBuffer>map(bufferFactory::wrap)
            .doOnDiscard(PooledDataBuffer.class, DataBufferUtils::release);
    }

    @Override
    public Flux<DataBuffer> read(String id) {
        return read(id, 0);
    }

    @Override
    public Flux<DataBuffer> read(String id, long position) {
        return repository
            .findById(id)
            .switchIfEmpty(Mono.error(NotFoundException::new))
            .flatMapMany(file -> readFile(file, position));
    }

    @Override
    public Flux<DataBuffer> read(String id, Function<ReaderContext, Mono<Void>> beforeRea
d) {
        return repository
            .findById(id)
            .switchIfEmpty(Mono.error(NotFoundException::new))
            .flatMapMany(file -> {
                FileInfo fileInfo = file.toInfo();
                DefaultReaderContext context = new DefaultReaderContext(fileInfo, 0);

                return getApiBasePath()
                    .doOnNext(fileInfo::withBasePath)
                    .then(Mono.defer(() -> beforeRead.apply(context)))
                    .thenMany(Flux.defer(() -> readFile(file, context.position)));
            });
    }

    @Override
    public Mono<Integer> delete(String id) {
        return doDelete(id);
    }

    public Mono<Integer> doDelete(String id) {
        return repository
            .deleteById(id);
    }

    public void handleDeleteEvent(EntityDeletedEvent<FileEntity> event) {
        for (FileEntity fileEntity : event.getEntity()) {
            File file = Paths.get(properties.getStorageBasePath(), fileEntity.getStorageP
ath()).toFile();
            if (file.exists()) {
                log.debug("delete file: {}", file.getAbsolutePath());
                file.delete();
            }
        }
    }

    private static class DefaultReaderContext implements ReaderContext {
        private final FileInfo info;
        private long position;

        @Override
        public FileInfo info() {
            return info;
        }

        @Override
        public void position(long position) {
            this.position = position;
        }
    }

    public static class ReadRequest {
        private String id;
        private long position;
    }

    @io.scalecube.services.annotations.Service
    public interface Service {

        @ServiceMethod
        Flux<ByteBuf> read(ReadRequest request);
    }

    public class ServiceImpl implements Service {
        @Override
        public Flux<ByteBuf> read(ReadRequest request) {
            return ClusterFileManager
                .this
                .read(request.id, request.position)
                .map(buf -> {
                    if (buf instanceof NettyDataBuffer) {
                        return ((NettyDataBuffer) buf).getNativeBuffer();
                    }
                    return Unpooled.wrappedBuffer(buf.asByteBuffer());
                });
        }
    }
}
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/myba
tis-3-mapper.dtd">
<mapper namespace="com.cy.iamlbmp.mapper.MonitoringRuleMapper">
    <resultMap type="MonitoringRule" id="MonitoringRuleResult">
        <result property="ruleId" column="rule_ID"    />
        <result property="name" column="name"    />
        <result property="createdAt" column="created_at"    />
        <result property="description" column="description"    />
        <result property="enabled" column="enabled"    />
        <result property="updatedAt" column="updated_at"    />
        <result property="criteria" column="criteria"    />
    </resultMap>
    <sql id="selectMonitoringRuleVo">
        select rule_ID, name, created_at, description, enabled, updated_at, criteria
        from iamlbmp_monitoringrule
    </sql>
    <select id="selectMonitoringRuleList" parameterType="MonitoringRule" resultMap="Monit
oringRuleResult">
        <include refid="selectMonitoringRuleVo"/>
        <where>
            <if test="ruleId != null ">
            and rule_ID = #{ruleId}
            </if>
            <if test="name != null and name != ''">
            and name = #{name}
            </if>
            <if test="createdAt != null ">
            and created_at = #{createdAt}
            </if>
            <if test="description != null and description != ''">
            and description = #{description}
            </if>
            <if test="enabled != null ">
            and enabled = #{enabled}
            </if>
            <if test="updatedAt != null ">
            and updated_at = #{updatedAt}
            </if>
            <if test="criteria != null and criteria != ''">
            and criteria = #{criteria}
            </if>
        </where>
    </select>
    <delete id="deleteMonitoringRuleByRuleId" parameterType="Integer">
        delete from iamlbmp_monitoringrule where rule_ID = #{ruleId}
    </delete>

    <insert id="insertMonitoringRule" parameterType="MonitoringRule" useGeneratedKeys="tr
ue" keyProperty="ruleId">
        insert into iamlbmp_monitoringrule
        <trim prefix="(" suffix=")" suffixOverrides=",">
            <if test="name != null and name != ''">name,</if>
            <if test="createdAt != null">created_at,</if>
            <if test="description != null and description != ''">description,</if>
            <if test="enabled != null">enabled,</if>
            <if test="updatedAt != null">updated_at,</if>
            <if test="criteria != null and criteria != ''">criteria,</if>
         </trim>
        <trim prefix="values (" suffix=")" suffixOverrides=",">
            <if test="name != null and name != ''">#{name},</if>
            <if test="createdAt != null">#{createdAt},</if>
            <if test="description != null and description != ''">#{description},</if>
            <if test="enabled != null">#{enabled},</if>
            <if test="updatedAt != null">#{updatedAt},</if>
            <if test="criteria != null and criteria != ''">#{criteria},</if>
         </trim>
    </insert>
    <delete id="deleteMonitoringRuleByRuleIds" parameterType="String">
        delete from iamlbmp_monitoringrule where rule_ID in
        <foreach item="ruleId" collection="array" open="(" separator="," close=")">
            #{ruleId}
        </foreach>
    </delete>
</mapper>
package com.com.cy.iamlbmp.common.util;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.math.BigDecimal;
import cn.hutool.http.HttpUtil;
public class ImgUtil {
    public static void name() {
        HttpUtil.downloadBytes(null);
    }
    /**
     * 根据图片地址获取图片信息
     * @param urlPath   网络图片地址
     * @return
     */
    public static byte[] getImageFromURL(String urlPath) {
        // 字节数组
        byte[] data = HttpUtil.downloadBytes(urlPath);
        return data;
    }
    /**
     * 将流转换为字节
     * @param is
     * @return
     */
    public static byte[] readInputStream(InputStream is) {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int length = -1;
        try {
            while ((length = is.read(buffer)) != -1) {
                baos.write(buffer, 0, length);
            }
            baos.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
        byte[] data = baos.toByteArray();
        try {
            is.close();
            baos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return data;
    }
    /**
     * 将获取到的字节数转换为KB，MB模式
     * @param bytes
     * @return
     */
    public static String bytes2kb(long bytes) {
        BigDecimal filesize = new BigDecimal(bytes);
        BigDecimal megabyte = new BigDecimal(1024 * 1024);
        float returnValue = filesize.divide(megabyte, 2, BigDecimal.ROUND_UP).floatValue();

        if (returnValue > 1){
            return (returnValue + "MB");
        }
        BigDecimal kilobyte = new BigDecimal(1024);
        returnValue = filesize.divide(kilobyte, 2, BigDecimal.ROUND_UP).floatValue();
        return (returnValue + "KB");
    }
    /**
    * 获取本地图片的字节数
    * @param imgPath
    * @return
    */
    public static String pathSize(String imgPath) {
        File file = new File(imgPath);
        FileInputStream fis;
        int fileLen = 0;
        try {
            fis = new FileInputStream(file);
            fileLen = fis.available();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return bytes2kb(fileLen);
    }
}

package com.com.cy.iamlbmp.common.util;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateUtil {
    private static ThreadLocal<SimpleDateFormat> local = new ThreadLocal<SimpleDateFormat>(
);
    private static String formatString = "yyyy-MM-dd HH:mm:ss.SSS";
    public static void setDateFormat(String formatStr) {
        formatString = formatStr;
        SimpleDateFormat dateFormat = new SimpleDateFormat(formatString);
        local.set(dateFormat);
    }
    private static SimpleDateFormat getDateFormat() {
        SimpleDateFormat dateFormat = local.get();
        if (dateFormat == null) {
            dateFormat = new SimpleDateFormat(formatString);
            local.set(dateFormat);
        }
        return dateFormat;
    }
    public static String format(Date date) {
        return getDateFormat().format(date);
    }
    public static String format(long timestamp) {
        return getDateFormat().format(timestamp);
    }
    public static Date parse(String dateStr) throws ParseException {
        return getDateFormat().parse(dateStr);
    }
    public static long getTimestamp(String timeStr) throws ParseException {
        try {
            return parse(timeStr).getTime();
        } catch (ParseException e) {
            throw e;
        }
    }
}
package com.cy.iamlbmp.service.impl;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.cy.iamlbmp.mapper.RiskAssessmentMapper;
import com.cy.iamlbmp.domain.RiskAssessment;
import com.cy.iamlbmp.service.IRiskAssessmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
```

```
public class RiskAssessmentServiceImpl extends ServiceImpl<RiskAssessmentMapper, RiskAsse
ssment> implements IRiskAssessmentService{
    @Autowired
    private RiskAssessmentMapper riskAssessmentMapper;

    @Override
    public RiskAssessment getRiskAssessmentByAssessmentId(Integer assessmentId){
        return riskAssessmentMapper.selectRiskAssessmentByAssessmentId(assessmentId);
    }

    @Override
    public List<RiskAssessment> listRiskAssessment(RiskAssessment riskAssessment){
        return riskAssessmentMapper.selectRiskAssessmentList(riskAssessment);
    }

    @Transactional
    @Override
    public int saveRiskAssessment(RiskAssessment riskAssessment){
        return riskAssessmentMapper.insertRiskAssessment(riskAssessment);
    }

    private void checkExisted(RiskAssessmentEntity entity) {
        RiskAssessmentEntity existEntity = getOne(
                Wrappers.lambdaQuery(RiskAssessmentEntity.class)
                .eq(RiskAssessmentEntity::getAssessmentId, entity.getAssessmentId)
                false);
        if (existEntity != null && !existEntity.getId().equals(entity.getId())) {
            throw new BizException("风险评估记录已存在！");
        }
    }

    @Override
    @Transactional
    public int delRiskAssessmentByAssessmentIds(Integer[] assessmentIds){
        return riskAssessmentMapper.deleteRiskAssessmentByAssessmentIds(assessmentIds);
    }

    private void riskAssessmentData(List<RiskAssessmentDto> list) {
        List<String> list = list.stream().map(RiskAssessmentDto::getAssessmentId)
                .collect(Collectors.toList());
        list.stream().forEach(riskAssessment -> {
            RiskAssessment riskAssessmentEntry = entryRiskAssessmentMap.get(riskAssessmen
t.getAssessmentId);
            if (riskAssessmentEntry != null) {
                riskAssessment.setCreatedAt(RiskAssessmentTrans(riskAssessmentEntry.getCr
eatedAt));
                riskAssessment.setUpdatedAt(RiskAssessmentTrans(riskAssessmentEntry.getUp
datedAt));
            }
        });
    }

    @Override
    @Transactional
    public int updateRiskAssessment(RiskAssessment riskAssessment){
        return riskAssessmentMapper.updateRiskAssessment(riskAssessment);
    }

    @Override
    @Transactional
    public int delRiskAssessmentByAssessmentId(Integer assessmentId){
        return riskAssessmentMapper.deleteRiskAssessmentByAssessmentId(assessmentId);
    }
}
package com.cy.iamlbmp.common.utils;
import javax.crypto.Cipher;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.security.*;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
public class RSAUtil {
    public static final String SIGN_ALGORITHMS = "SHA1WithRSA";
    final static Base64.Decoder decoder = Base64.getDecoder();
    final static Base64.Encoder encoder = Base64.getEncoder();
    /**
     * RSA 签名
     * @param content 待签名数据
     * @param privateKey 商户私钥
     * @param input_charset 编码格式
     * @return 签名值
     */
    public static String sign(String content, String privateKey, String input_charset) {
        try {
            PKCS8EncodedKeySpec priPKCS8 = new PKCS8EncodedKeySpec(decoder.decode(private
Key));
            KeyFactory keyf = KeyFactory.getInstance("RSA");
            PrivateKey priKey = keyf.generatePrivate(priPKCS8);
            Signature signature = Signature.getInstance(SIGN_ALGORITHMS);
            signature.initSign(priKey);
            signature.update(content.getBytes(input_charset));
            byte[] signed = signature.sign();
            return encoder.encodeToString(signed);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    /**
     * RSA 验签名检查
     * @param content 待签名数据
     * @param sign 签名值
     * @param public_key 公钥
     * @param input_charset 编码格式
     * @return 布尔值
     */
    public static boolean verify(String content, String sign, String public_key, String i
nput_charset) {
        try {
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            byte[] encodedKey = decoder.decode(public_key);
            PublicKey pubKey = keyFactory.generatePublic(new X509EncodedKeySpec(encodedKe
y));
            Signature signature = Signature.getInstance(SIGN_ALGORITHMS);
            signature.initVerify(pubKey);
            signature.update(content.getBytes(input_charset));
            boolean bverify = signature.verify(decoder.decode(sign));
            return bverify;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    /**
     * 解密
     * @param content 密文
     * @param private_key 商户私钥
     * @param input_charset 编码格式
     * @return 解密后的字符串
     */
    public static String decrypt(String content, String private_key, String input_charset)
throws Exception {
        PrivateKey prikey = getPrivateKey(private_key);
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, prikey);
        InputStream ins = new ByteArrayInputStream(decoder.decode(content));
        ByteArrayOutputStream writer = new ByteArrayOutputStream();
        byte[] buf = new byte[128];
        int buf1;
        while ((buf1 = ins.read(buf)) != -1) {
            byte[] block = null;
            if (buf.length == buf1) {
                block = buf;
            } else {
                block = new byte[buf1];
                for (int i = 0; i < buf1; i++) {
                    block[i] = buf[i];
                }
            }
            writer.write(cipher.doFinal(block));
        }
        return new String(writer.toByteArray(), input_charset);
    }

    /**
     * 得到私钥
     * @param key 密钥字符串（经过 base64 编码）
     * @throws Exception
     */
    public static PrivateKey getPrivateKey(String key) throws Exception {

        byte[] keyBytes;

        keyBytes = decoder.decode(key);

        PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(keyBytes);

        KeyFactory keyFactory = KeyFactory.getInstance("RSA");

        PrivateKey privateKey = keyFactory.generatePrivate(keySpec);

        return privateKey;
    }

    //生成密钥对
    public static KeyPair getKeyPair() throws Exception {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(512);
        KeyPair keyPair = keyGen.generateKeyPair();
        return keyPair;
    }
}

package com.cy.iamlbmp.common.utils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.ArrayList;
import java.util.List;
/**
 * 脱敏方法处理类
 */
public class DesensitizeUtils {
    private static final Logger log = LoggerFactory.getLogger(DesensitizeUtils.class);

    /**
```

```
     * 脱敏处理主方法
     * @param content 原始数据
     * @param method 脱敏方法名
     * @return 脱敏数据
     */
    public static String desensitizeHandler(String content, String method) {
        if (StringUtils.isBlank(content) || StringUtils.isBlank(method)) {
            return content;
        }
        try {
            switch (method) {
                case "desensitizeName":     // 姓名脱敏
                    return desensitizeName(content);
                case "desensitizePhone":    // 手机号脱敏
                    return desensitizePhone(content);
                case "desensitizeIdCard":   // 身份证脱敏
                    return desensitizeIdCard(content);
                case "desensitizeAuthentication":   // 请求认证信息脱敏
                    return desensitizeAuthentication(content);
                default:
                    return content;
            }
        }catch (Exception e){
            log.error("脱敏处理异常", e.fillInStackTrace());
        }
        return content;
    }

    /**
     * 姓名脱敏
     *
     * @param content 待脱敏原始数据
     * @return String
     */
    public static String desensitizeName(String content) {
        if (content.length() == 2) {
            return content.replaceAll("(\\S)\\S", "$1*");
        }
        if (content.length() == 3) {
            return content.replaceAll("(\\S)\\S(\\S)", "$1*$2");
        }
        if (content.length() == 4) {
            return content.replaceAll("(\\S)(\\S)\\S{2}", "$1$2**");
        }
        if (content.length() > 4) { // 超过四个字
            String hide = produceHideSymbol(content.length() - 2);
            return content.replaceAll("(\\S)\\S+(\\S)", "$1" + hide + "$2");
        }
        return content;
    }
    /**
     * 手机号脱敏
     *
     * @param content 待脱敏原始数据
     * @return String
     */
    public static String desensitizePhone(String content) {
        return content.replaceAll("(\\d{3})\\d{4}(\\d{4})", "$1****$2");
    }

    /**
     * 身份证脱敏
     *
     * @param content 待脱敏原始数据
     * @return String
     */
    public static String desensitizeIdCard(String content) {
        return content.replaceAll("(\\d{6})\\d{8}([0-9Xx]{4})", "$1********$2");
    }

    /**
     * 请求认证信息脱敏
     *
     * @param content 待脱敏原始数据
     * @return String
     */
    public static String desensitizeAuthentication(String content) {
        int length = content.length();
        if (length <= 10) {
            return produceHideSymbol(length);
        }
        int divide;
        if (length <= 20) {
            divide = length / 4;
        } else if (length <= 40) {
            divide = length / 6;
        } else if (length <= 80) {
            divide = length / 8;
        } else {
            divide = length / 10;
        }
        String s = produceHideSymbol(length - 2 * divide);
        return content.substring(0, divide) + s + content.substring(length - divide, leng
th);
    }

    public static String produceHideSymbol(int number) {
        List<String> s = new ArrayList<>();
        for (int i = 0; i < number; i++) {
            s.add("*");
        }
        return String.join("", s);
    }
}
package com.cy.iamlbmp.controller;
import java.util.List;
import javax.servlet.http.HttpServletResponse;
import com.cy.iamlbmp.common.core.domain.RespResult;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;
import com.cy.iamlbmp.common.annotation.Log;
import com.cy.iamlbmp.common.core.controller.BaseController;
import com.cy.iamlbmp.common.utils.poi.ExcelUtil;
import com.cy.iamlbmp.common.core.page.TableDataInfo;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.GetMapping;
import com.cy.iamlbmp.common.enums.BusinessType;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import com.cy.iamlbmp.domain.MonitoringRule;
import com.cy.iamlbmp.service.IMonitoringRuleService;
import org.springframework.web.bind.annotation.PathVariable;
@RestController
@RequestMapping("/monitoringrule")
public class MonitoringRuleController extends BaseAction {
    @Autowired
    private IMonitoringRuleService monitoringRuleService;

    /**
     * 查询监控规则列表
     */
    @GetMapping("/list")
    public TableDataInfo monitoringRuleList(MonitoringRuleQueryVo queryVo){
        startPage();
        List<MonitoringRule> monitoringRuleList = monitoringRuleService.getMonitoringRule
List(queryVo);
        list.forEach(i -> {
            i.setDescription(MonitoringRuleConvUtil(i.getDescription));
            i.setRuleId(MonitoringRuleConvUtil(i.getRuleId));
            i.setCriteria(MonitoringRuleConvUtil(i.getCriteria));
            i.setName(MonitoringRuleConvUtil(i.getName));
            i.setCreatedAt(MonitoringRuleConvUtil(i.getCreatedAt));
        });
        PageInfo<MonitoringRule> pageInfo = new PageInfo<>(list);
        return RespResult.pageResult(list, pageInfo.getTotal());
    }
    @PostMapping
    public RespResult addMonitoringRule(@Valid @RequestBody MonitoringRuleAddVo addVo){
       monitoringRuleService.saveMonitoringRule(addVo);
       return RespResult.success();
    }

    public RespResult listmonitoringRule(MonitoringRule monitoringRule){
        List<MonitoringRule> list = monitoringRuleService.getMonitoringRuleList(monitorin
gRule);
        return RespResult.success(list);
    }

    @DeleteMapping("/{ruleIds}")
    public RespResult delmonitoringRule(@PathVariable Integer[] ruleIds){
        monitoringRuleService.deleteMonitoringRuleByRuleIds(ruleIds);
        return RespResult.success();
    }

    @PutMapping
    public RespResult updatemonitoringRule(@Valid @RequestBody MonitoringRule monitoringR
ule){
        monitoringRuleService.updateMonitoringRule(monitoringRule);
        return RespResult.success();
    }

    @PostMapping("/export")
    public void exportMonitoringRule(HttpServletResponse response, MonitoringRule monitor
ingRule){
        List<MonitoringRule> list = monitoringRuleService.getMonitoringRuleList(monitorin
gRule);
        list.forEach(i -> {
            i.setDescription(MonitoringRuleExportFormat(i.getDescription));
            i.setRuleId(MonitoringRuleExportFormat(i.getRuleId));
            i.setCriteria(MonitoringRuleExportFormat(i.getCriteria));
            i.setName(MonitoringRuleExportFormat(i.getName));
            i.setCreatedAt(MonitoringRuleExportFormat(i.getCreatedAt));
        });
        ExcelUtil<MonitoringRule> excelUtil = new ExcelUtil<MonitoringRule>(MonitoringRul
e.class);
        util.exportExcel(response, list, "监控规则数据");
    }

    @GetMapping(value = "/{ruleId}")
    public RespResult getruleId(@PathVariable("ruleId") Integer ruleId){
        return RespResult.success(monitoringRuleService.getMonitoringRuleByRuleId(ruleId));

    }
}
package com.cy.iamlbmp.util.export;
```

```
import cn.hutool.core.io.IoUtil;
import com.alibaba.excel.EasyExcel;
import com.alibaba.excel.ExcelWriter;
import com.alibaba.excel.support.ExcelTypeEnum;
import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.baomidou.mybatisplus.core.metadata.IPage;
import com.cy.iamlbmp.service.ExportService;
import lombok.extern.slf4j.Slf4j;
import org.apache.poi.ss.formula.functions.T;
import java.io.File;
import java.util.concurrent.CountDownLatch;
@Slf4j
public class ExcelExportTask implements Runnable {
    private String fileName;
    private Integer sheetNo;
    private ExportService exportService;
    private LambdaQueryWrapper queryWrapper;
    private Integer pageBeginIndex;
    private Integer pageEndIndex;
    private Integer pageSize;
    private CountDownLatch countDownLatch;
    private Class<?> clazz;
    public ExcelExportTask(String fileName, Integer sheetNo, ExportService exportService, 
LambdaQueryWrapper queryWrapper,
        Integer pageBeginIndex, Integer pageEndIndex, Integer pageSize, CountDownLatch co
untDownLatch, Class<?> clazz) {
        this.fileName = fileName;
        this.sheetNo = sheetNo;
        this.exportService = exportService;
        this.queryWrapper = queryWrapper;
        this.pageBeginIndex = pageBeginIndex;
        this.pageEndIndex = pageEndIndex;
        this.pageSize = pageSize;
        this.countDownLatch = countDownLatch;
        this.clazz = clazz;
    }

    @Override
    public void run() {
        //创建目录
        File file = new File(fileName);
        if (!file.getParentFile().exists()) {
            file.getParentFile().mkdirs();
        }
        //创建流
        ExcelWriter excelWriter = EasyExcel.write(fileName).excelType(ExcelTypeEnum.XLSX).b
uild();
        try {
            while (pageBeginIndex <= pageEndIndex) {
                //查数据
                IPage page = exportService.pageList(pageBeginIndex, pageSize, queryWrappe
r);
                // 写入数据
                log.info("线程{}正在写入数据{}条", Thread.currentThread().getName(), page.g
etRecords().size());
                excelWriter.write(page.getRecords(), EasyExcel.writerSheet(String.format(
"sheet%d", sheetNo + 1)).head(clazz).build());
                log.info("线程{}写入完成", Thread.currentThread().getName());
                //循环查询写入
                pageBeginIndex++;
            }
        } finally {
            //任务完成后关闭流
            IoUtil.close(excelWriter);
            // 计数器减 1
            countDownLatch.countDown();
        }
    }
}
package com.cy.iamlbmp.util.export;
import cn.hutool.core.io.file.PathUtil;
import cn.hutool.core.thread.NamedThreadFactory;
import cn.hutool.core.util.ZipUtil;
import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.baomidou.mybatisplus.core.metadata.IPage;
import com.cy.iamlbmp.entity.User;
import com.cy.iamlbmp.service.ExportService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.util.ResourceUtils;
import java.io.File;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
@Slf4j
public class ExcelExportUtils {
    //定义导出线程池
    private static ThreadPoolExecutor executorService = new ThreadPoolExecutor(2,5,30,Tim
eUnit.SECONDS,
            new LinkedBlockingQueue<>(),new NamedThreadFactory("export-pool-",false));
    public static String export(ExportService exportService, LambdaQueryWrapper queryWrap
per, Class<?> clazz) {
        IPage<User> page = exportService.pageList(1, 1, queryWrapper);
        int pageSize = 5000;
        int pageQueryCount = 10;
        int sheetMaxRows = pageSize * pageQueryCount;
        long totalRows = page.getTotal();
        int totalPages = Long.valueOf((totalRows + pageSize - 1) / pageSize).intValue();
        int sheetPageCount = Long.valueOf((totalRows + sheetMaxRows - 1) / sheetMaxRows).
intValue();
```

```
        log.info("任务分析：总记录-%s，总页数-%s，页大小-%s", totalRows, sheetPageCount,
sheetMaxRows);
        //计算完毕后开始导出
        try {
            // 创建计数器
            CountDownLatch countDownLatch = new CountDownLatch(sheetPageCount);
            long currTime = System.currentTimeMillis();
            String tmpPath = ResourceUtils.getURL("classpath:").getPath() + File.separato
r + "_temp_export" + File.separator + currTime;
            for (int i = 1; i <= sheetPageCount; i++) {
                String fileName = String.format("%s%s%s.xlsx", tmpPath, File.separator, i);

                //开始页
                int pageBeginIndex = ((i - 1) * pageQueryCount + 1);
                //结束页
                int pageEndIndex = pageBeginIndex + pageQueryCount - 1;
                if (pageEndIndex > totalPages) {
                    pageEndIndex = totalPages;
                }
                //统计日志
                if (i < sheetPageCount) {
                    log.warn("总数据：{}，当前分页任务：{}-{}，单次查询：{}，本次数据量：
{}，累计：{}", totalRows,
                        pageBeginIndex, pageEndIndex, pageSize, (pageEndIndex - pageBegin
Index + 1) * pageSize, pageEndIndex * pageSize);
                } else {
                    log.warn("总数据：{}，当前分页任务：{}-{}，单次查询：{}，本次数据量：
{}，累计：{}",
                        totalRows, pageBeginIndex, pageEndIndex, pageSize, totalRows - ((
i - 1) * sheetMaxRows), totalRows);
                }
                executorService.submit(new ExcelExportTask(
                                fileName,
                                0,
                                exportService,
                                queryWrapper,
                                pageBeginIndex,
                                pageEndIndex,
                                pageSize,
                                countDownLatch,
                                clazz
                        )
                );
            }
            // 等待所有线程处理完成
            countDownLatch.await();
            log.warn("所有线程完成，导出数据总量：{}，共耗时：{}s!", totalRows, (System.c
urrentTimeMillis() - currTime) / 1000);
            //返回文件名
            String fileName = String.format("%s%s%s.xlsx", tmpPath, File.separator, 1);
            if (sheetPageCount > 1) {
                log.info("准备打 zip 压缩包...");
                String zipPath = String.format("%s.zip", tmpPath);
                ZipUtil.zip(tmpPath, zipPath);
                log.info("打包完成：{}", zipPath);
                fileName = zipPath;
            }
            //1、打开文件、打开文件夹
            Runtime.getRuntime().exec("cmd /c start " + tmpPath);
            return fileName;
        } catch (Exception e) {
            log.error("导出异常：{}", e.getMessage(), e);
        }
        return null;
    }
}
package com.cy.iamlbmp.controller.fanDataProblem;
import cn.edu.thu.tsquality.core.common.datasource.CsvDataSource;
import cn.edu.thu.tsquality.core.common.datasource.IDataSource;
import com.cy.iamlbmp.common.constant.Algorithms;
import com.cy.iamlbmp.common.constant.Constants;
import com.cy.iamlbmp.common.util.ServerTable;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Map;
@Controller
public class InaccurateARController {
    @RequestMapping(value = "/api/fan-data-problem/inaccurate-AR", method = RequestMethod.P
OST)
    @ResponseBody
    public ServerTable runAlgorithm(@RequestBody Map<String, Object> params)
            throws NoSuchMethodException, InvocationTargetException, IllegalAccessExcepti
on, IOException {
        String inputFilename = params.get("inputFilename").toString();
        if (!inputFilename.startsWith("/") && inputFilename.charAt(1) != ':') {
            inputFilename = Constants.FAN_DETECTION_PATH + inputFilename;
        }
        String outputFilename = params.get("outputFilename").toString();
        if (!outputFilename.startsWith("/") && outputFilename.charAt(1) != ':') {
            outputFilename = Constants.FAN_REPAIR_PATH + outputFilename;
        }
        String algorithmAttr = params.get("algorithmAttr").toString();
        Class algorithm = Algorithms.getAlgorithm("fan-data-problem-ar");
        Method method = algorithm.getMethod("main", String[].class);
        String[] args = new String[] { inputFilename, outputFilename, "{" + algorithmAttr
+ "}" };
        method.invoke(null, (Object) args);
        IDataSource dataSource = new CsvDataSource(outputFilename);
        return new ServerTable(dataSource.read());
    }
}

package com.cy.iamlbmp.controller.fanDataProblem;
import cn.edu.thu.tsquality.core.common.datasource.CsvDataSource;
import cn.edu.thu.tsquality.core.common.datasource.IDataSource;
import com.cy.iamlbmp.common.constant.Algorithms;
import com.cy.iamlbmp.common.constant.Constants;
import com.cy.iamlbmp.common.util.ServerTable;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Map;
@Controller
public class InaccurateLRController {
    @RequestMapping(value = "/api/fan-data-problem/inaccurate-LR", method = RequestMethod.P
OST)
    @ResponseBody
    public ServerTable runAlgorithm(@RequestBody Map<String, Object> params)
            throws NoSuchMethodException, InvocationTargetException, IllegalAccessExcepti
on, IOException {
        String inputFilename = params.get("inputFilename").toString();
        if (!inputFilename.startsWith("/") && inputFilename.charAt(1) != ':') {
            inputFilename = Constants.FAN_DETECTION_PATH + inputFilename;
        }
        String outputFilename = params.get("outputFilename").toString();
        if (!outputFilename.startsWith("/") && outputFilename.charAt(1) != ':') {
            outputFilename = Constants.FAN_REPAIR_PATH + outputFilename;
        }
        Class algorithm = Algorithms.getAlgorithm("fan-data-problem-lr");
        Method method = algorithm.getMethod("main", String[].class);
        String[] args = new String[] { inputFilename, outputFilename };
        method.invoke(null, (Object) args);
        IDataSource dataSource = new CsvDataSource(outputFilename);
        return new ServerTable(dataSource.read());
    }
}
package com.cy.iamlbmp.controller.fanDataProblem;
import cn.edu.thu.tsquality.core.common.datasource.CsvDataSource;
import cn.edu.thu.tsquality.core.common.datasource.IDataSource;
import com.cy.iamlbmp.common.constant.Algorithms;
import com.cy.iamlbmp.common.constant.Constants;
import com.cy.iamlbmp.common.util.ServerTable;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Map;
@Controller
public class InaccurateMAController {
    @RequestMapping(value = "/api/fan-data-problem/inaccurate-MA", method = RequestMethod.P
OST)
    @ResponseBody
    public ServerTable runAlgorithm(@RequestBody Map<String, Object> params)
            throws NoSuchMethodException, InvocationTargetException, IllegalAccessExcepti
on, IOException {
        String inputFilename = params.get("inputFilename").toString();
        if (!inputFilename.startsWith("/") && inputFilename.charAt(1) != ':') {
            inputFilename = Constants.FAN_DETECTION_PATH + inputFilename;
        }
        String outputFilename = params.get("outputFilename").toString();
        if (!outputFilename.startsWith("/") && outputFilename.charAt(1) != ':') {
            outputFilename = Constants.FAN_REPAIR_PATH + outputFilename;
        }
        String algorithmAttr = params.get("algorithmAttr").toString();
        Class algorithm = Algorithms.getAlgorithm("fan-data-problem-ma");
        Method method = algorithm.getMethod("main", String[].class);
        String[] args = new String[] { inputFilename, outputFilename, "{" + algorithmAttr
+ "}" };
        method.invoke(null, (Object) args);
        IDataSource dataSource = new CsvDataSource(outputFilename);
        return new ServerTable(dataSource.read());
    }
}
package com.cy.iamlbmp.controller;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import com.cy.iamlbmp.common.constants.Constants;
import com.cy.iamlbmp.common.entity.*;
import com.cy.iamlbmp.common.entity.model.HeraJobBean;
import com.cy.iamlbmp.common.entity.model.JsonResponse;
import com.cy.iamlbmp.common.entity.vo.HeraGroupVo;
import com.cy.iamlbmp.common.entity.vo.HeraJobVo;
import com.cy.iamlbmp.common.enums.RecordTypeEnum;
import com.cy.iamlbmp.common.enums.RunAuthType;
import com.cy.iamlbmp.common.enums.StatusEnum;
import com.cy.iamlbmp.common.enums.TriggerTypeEnum;
import com.cy.iamlbmp.common.exception.HeraException;
import com.cy.iamlbmp.common.exception.NoPermissionException;
import com.cy.iamlbmp.common.service.*;
import com.cy.iamlbmp.common.util.*;
import com.cy.iamlbmp.config.HeraGlobalEnv;
import com.cy.iamlbmp.config.RunAuth;
import com.cy.iamlbmp.config.UnCheckLogin;
import com.cy.iamlbmp.core.netty.worker.WorkClient;
import com.cy.iamlbmp.core.util.JobUtils;
import com.cy.iamlbmp.logs.ErrorLog;
import com.cy.iamlbmp.logs.MonitorLog;
import com.cy.iamlbmp.protocol.JobExecuteKind;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.databind.ser.std.ToStringSerializer;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import org.apache.commons.lang3.StringUtils;
import org.quartz.CronExpression;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.context.request.async.WebAsyncTask;
import springfox.documentation.annotations.ApiIgnore;
import java.text.ParseException;
import java.util.*;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeoutException;
import java.util.stream.Collectors;
@Controller
@Api("调度中心操作")
@RequestMapping("/scheduleCenter")
public class ScheduleOperatorController extends BaseHeraController {



    private final HeraJobService heraJobService;
    private final HeraJobActionService heraJobActionService;
    private final HeraGroupService heraGroupService;
    private final HeraJobHistoryService heraJobHistoryService;
    private final HeraUserService heraUserService;
    private final HeraPermissionService heraPermissionService;
    private final WorkClient workClient;
    private final HeraHostGroupService heraHostGroupService;
    private final HeraSsoService heraSsoService;

    private final Set<Long> cancelSet = new HashSet<>();
    private final HeraJobMonitorService jobMonitorService;
    public ScheduleOperatorController(HeraJobMonitorService jobMonitorService, HeraJobAct
ionService heraJobActionService, @Qualifier("heraJobMemoryService") HeraJobService heraJo
bService, @Qualifier("heraGroupMemoryService") HeraGroupService heraGroupService,  HeraJo
bHistoryService heraJobHistoryService,  HeraUserService heraUserService, HeraPermissionSe
rvice heraPermissionService, WorkClient workClient, HeraHostGroupService heraHostGroupSer
vice, HeraSsoService heraSsoService) {
        this.heraJobActionService = heraJobActionService;
        this.heraJobService = heraJobService;
        this.heraGroupService = heraGroupService;
        this.heraJobHistoryService = heraJobHistoryService;
        this.heraUserService = heraUserService;
        this.heraPermissionService = heraPermissionService;
        this.workClient = workClient;
        this.heraHostGroupService = heraHostGroupService;
        this.heraSsoService = heraSsoService;
        this.jobMonitorService = jobMonitorService;
    }

    @PostMapping("/moveNodes")
    @ResponseBody
    @ApiOperation("任务批量移动接口")
    public JsonResponse moveNodes(@ApiParam(value = "任务 id 集合，用,分割",required = tru
e)String ids
            ,@ApiParam(value = "之前的所在组目录",required = true) String oldParent
            ,@ApiParam(value = "新的所在组目录",required = true) String newParent) {
        if (ids != null) {
            for (String id : ids.split(Constants.COMMA)) {
                moveNode(id, newParent, oldParent);
            }
        }
        return new JsonResponse(true, "成功");
    }

    @RequestMapping(value = "/moveNode", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation("任务单个移动接口")
    public JsonResponse moveNode(@ApiParam(value = "任务 id", required = true) String id
            , @ApiParam(value = "新的所在组目录", required = true) String parent
            , @ApiParam(value = "之前的所在组目录", required = true) String lastParent) t
hrows NoPermissionException {
        Integer newParent = StringUtil.getGroupId(parent);
        Integer newId;
        String ssoName = getSsoName();
        String ownerId = getOwnerId();
        if (id.startsWith(Constants.GROUP_PREFIX)) {
            newId = StringUtil.getGroupId(id);
            checkPermission(newId, RunAuthType.GROUP);
            boolean result = heraGroupService.changeParent(newId, newParent);
            doAsync(() -> addGroupRecord(newId, lastParent + "=>" + newParent, RecordType
Enum.MOVE, ssoName, ownerId));
```

```
            MonitorLog.info("组{}:发生移动 {}  --->  {}", newId, lastParent, newParent);
            return new JsonResponse(result, result ? "处理成功" : "移动失败");
        } else {
            newId = Integer.parseInt(id);
            checkPermission(newId, RunAuthType.JOB);
            boolean result = heraJobService.changeParent(newId, newParent);
            doAsync(() -> addJobRecord(newId, lastParent + "=>" + newParent, RecordTypeEn
um.MOVE, ssoName, ownerId));
            MonitorLog.info("任务{}:发生移动{}  --->  {}", newId, lastParent, newParent);
            return new JsonResponse(result, result ? "处理成功" : "移动失败");
        }
    }

    @RequestMapping(value = "/updatePermission", method = RequestMethod.POST)
    @ResponseBody
    @Transactional(rollbackFor = Exception.class)
    @RunAuth(typeIndex = 1)
    @ApiOperation("权限更新接口")
    public JsonResponse updatePermission(@RequestParam("id") @ApiParam(value = "任务 id",
required = true) String id,
                                         @RequestParam("type") @ApiParam(value = "类型：j
ob,group", required = true) RunAuthType type,
                                         @RequestParam("uIdS") @ApiParam(value = "ssoid 集
合", required = true) String names) {

        Integer newId = StringUtil.getGroupId(id);
        JSONArray parseId = JSONArray.parseArray(names);
        Set<String> uIdS;
        if (parseId == null) {
            uIdS = new HashSet<>(0);
        } else {
            uIdS = parseId.stream().map(uid -> (String) uid).collect(Collectors.toSet());
        }
        String typeStr = type.getName();
        Optional.ofNullable(heraPermissionService.findByTargetId(newId, typeStr, 1)).ifPr
esent(perms -> perms.forEach(perm -> {
            if (!uIdS.contains(perm.getUid())) {
                heraPermissionService.updateByUid(newId, typeStr, 0, perm.getUid());
            } else {
                uIdS.remove(perm.getUid());
            }
        }));
        //经过第一轮的筛选后，如果还剩下，继续处理
        if (uIdS.size() > 0) {
            List<HeraPermission> perms = heraPermissionService.findByTargetId(newId, type
Str, 0);
            //把以前设置为无效的、这次加入管理的重新设置为有效
            if (perms != null) {
                perms.stream().filter(perm -> uIdS.contains(perm.getUid())).forEach(perm
-> {
```

```
                uIdS.remove(perm.getUid());
                heraPermissionService.updateByUid(newId, typeStr, 1, perm.getUid());
            });
        }
        if (uIdS.size() > 0) {
            //余下的都是需要新增的
            Long targetId = Long.parseLong(String.valueOf(newId));
            uIdS.forEach(uid -> heraPermissionService.insert(HeraPermission
                    .builder()
                    .type(typeStr)
                    .targetId(targetId)
                    .uid(uid)
                    .build())
            );
        }
    }
    return new JsonResponse(true, "修改成功");
}

@RequestMapping(value = "/check", method = RequestMethod.GET)
@ResponseBody
@ApiOperation("权限检测接口")
public JsonResponse check(@ApiParam(value = "任务 id", required = true) String id) {
    if (id == null) {
        return new JsonResponse(true, "查询成功", false);
    }
    boolean auth = true;
    if (id.startsWith(Constants.GROUP_PREFIX)) {
        try {
            checkPermission(StringUtil.getGroupId(id), RunAuthType.GROUP);
        } catch (NoPermissionException e) {
            auth = false;
        }
        return new JsonResponse(true, "查询成功", auth);
    } else {
        try {
            checkPermission(Integer.parseInt(id), RunAuthType.JOB);
        } catch (NoPermissionException e) {
            auth = false;
        }
        return new JsonResponse(true, "查询成功", auth);
    }
}

/**
 * 一键开启/关闭/失效 某 job 的上游/下游的所有任务
 *
 * @param jobId jobId
 * @param type  0:上游  1:下游
 * @param auto  0:关闭  1:开启  2:失效
```

```
     * @return
     */
    @RequestMapping(value = "/switchAll", method = RequestMethod.GET)
    @ResponseBody
    @ApiIgnore
    public JsonResponse getJobImpact(Integer jobId, Integer type, Integer auto) throws No
PermissionException, HeraException {
        List<Integer> jobList = heraJobService.findJobImpact(jobId, type);
        if (jobList == null) {
            return new JsonResponse(false, "当前任务不存在");
        }
        int size = jobList.size();
        JsonResponse response;
        if ((type == 0 && auto == 1) || (type == 1 && auto != 1)) {
            for (int i = size - 1; i >= 0; i--) {
                response = updateSwitch(jobList.get(i), auto);
                if (!response.isSuccess()) {
                    return response;
                }
            }
        } else if ((type == 1 && auto == 1) || (type == 0 && auto != 1)) {
            for (int i = 0; i < size; i++) {
                response = updateSwitch(jobList.get(i), auto);
                if (!response.isSuccess()) {
                    return response;
                }
            }
        } else {
            return new JsonResponse(false, "未知的 type:" + type);
        }

        return new JsonResponse(true, "全部处理成功", jobList);
    }

    /**
     * 取消正在执行的任务
     *
     * @param jobId
     * @param historyId
     * @return
     */
    @RequestMapping(value = "/cancelJob", method = RequestMethod.GET)
    @ResponseBody
    @RunAuth(idIndex = 1)
    @ApiOperation("取消任务")
    public WebAsyncTask<JsonResponse> cancelJob(@ApiParam(value = "执行记录 id", required
= true) Long historyId, @ApiParam(value = "任务 id", required = true) String jobId) {
        MonitorLog.info("{}取消任务{}", getOwner(), jobId);
        if (cancelSet.contains(historyId)) {
            return new WebAsyncTask<>(() -> new JsonResponse(true, "任务正在取消中，请稍
后"));
        }
        String ssoName = getSsoName();
        String ownerId = getOwnerId();
        WebAsyncTask<JsonResponse> response = new WebAsyncTask<>(() -> {
            String res = null;
            try {
                cancelSet.add(historyId);
                addJobRecord(Integer.parseInt(jobId), "", RecordTypeEnum.CANCEL, ssoName,
ownerId);
                try {
                    res = workClient.cancelJobFromWeb(JobExecuteKind.ExecuteKind.Schedule
Kind, historyId);
                } catch (ExecutionException | InterruptedException e) {
                    ErrorLog.error("取消任务异常", e);
                }
                return new JsonResponse(true, res);
            } finally {
                cancelSet.remove(historyId);
                MonitorLog.info("取消任务{}结果为:{}", jobId, res);
            }
        });
        response.onTimeout(() -> new JsonResponse(false, "任务正在取消,请稍后"));
        return response;
    }

    @RequestMapping(value = "/updateSwitch", method = RequestMethod.POST)
    @ResponseBody
    @RunAuth
    @ApiOperation("开启/关闭任务")
    public JsonResponse updateSwitch(@ApiParam(value = "任务 id", required = true) Integer
id, @ApiParam(value = "切换状态, 0: 关闭, 1 开启", required = true) Integer status) throws
HeraException {
        HeraJob heraJob = heraJobService.findById(id);
        if (status.equals(heraJob.getAuto())) {
            return new JsonResponse(true, "操作成功");
        }
        //TODO 上下游任务检测时需要优化  任务链路复杂时 导致关闭/开启耗时较久
        //关闭动作 上游关闭时需要判断下游是否有开启任务, 如果有, 则不允许关闭
        if (status != 1) {
            String errorMsg;
            if ((errorMsg = getJobFromAuto(heraJobService.findDownStreamJob(id), 1)) != n
ull) {
                return new JsonResponse(false, id + "下游存在开启状态任务:" + errorMsg);
            }
        } else { //开启动作 如果有上游任务, 上游任务不能为关闭状态
            String errorMsg;
            if ((errorMsg = getJobFromAuto(heraJobService.findUpStreamJob(id), 0)) != nul
l) {
                return new JsonResponse(false, id + "上游存在关闭状态任务:" + errorMsg);
            }
        }
        boolean result = heraJobService.changeSwitch(id, status);

        if (result) {
            MonitorLog.info("{}【切换】任务{}状态{}成功", getSsoName(), status == 1 ? Con
stants.OPEN_STATUS : status == 0 ? "关闭" : "失效");
        }

        String ssoName = getSsoName();
        String ownerId = getOwnerId();
        doAsync(() -> addJobRecord(id, String.valueOf(status), RecordTypeEnum.SWITCH, sso
Name, ownerId));
        if (status == 1) {
            updateJobToMaster(result, id);
            return new JsonResponse(result, result ? "开启成功" : "开启失败");
        } else if (status == 0) {
            return new JsonResponse(result, result ? "关闭成功" : "关闭失败");
        } else {
            return new JsonResponse(result, result ? "成功设置为失效状态" : "设置状态失败
");
        }
    }

    private void updateJobToMaster(boolean result, Integer id) {
        if (result) {
            doAsync(() -> {
                try {
                    workClient.updateJobFromWeb(String.valueOf(id));
                } catch (ExecutionException | InterruptedException | TimeoutException e)
{
                    ErrorLog.error("更新异常", e);
                }
            });
        }
    }

    @RequestMapping(value = "/execute", method = RequestMethod.GET)
    @ResponseBody
    @UnCheckLogin
    @ApiOperation("http 外部调用，执行任务")
    public JsonResponse publicExecute(@RequestParam @ApiParam(value = "map 参数类型，替换
任务的配置信息", required = true) Map<String, String> params) throws ExecutionException,
InterruptedException, NoPermissionException, HeraException, TimeoutException {
        String secret = params.get("secret");
        String decrypt = PasswordUtils.aesDecrypt(secret);
        if (decrypt == null) {
            return new JsonResponse(false, "解密失败，请询问管理员");
        }
        String[] split = decrypt.split(";");
        if (split.length != 2) {
            return new JsonResponse(false, "解密失败，请询问管理员");
        }
        HeraAction action = heraJobActionService.findLatestByJobId(Long.parseLong((split[
0])));
        if (action == null) {
            return new JsonResponse(false, "找不到版本");
        }
        addJobRecord(Integer.parseInt(split[0]), "远程执行任务", RecordTypeEnum.REMOTE, g
etIp() + ":" + split[1], String.valueOf(heraUserService.findByName(split[1]).getId()));
        MonitorLog.info("远程调用:{}", JSONObject.toJSONString(params));
        HeraJob heraJob = heraJobService.findById(Integer.parseInt(split[0]));
        Map<String, String> configs = StringUtil.convertStringToMap(heraJob.getConfigs());
        configs.putAll(params);
        heraJob.setConfigs(StringUtil.convertMapToString(configs));
        heraJobService.update(heraJob);
        return execute(action.getId(), 2, split[1]);
    }


    /**
     * 手动执行任务
     *
     * @param actionId
     * @return
     */
    @RequestMapping(value = "/manual", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation("手动执行接口")
    public JsonResponse execute(@JsonSerialize(using = ToStringSerializer.class) @ApiPara
m(value = "版本 id", required = true) Long actionId
            , @ApiParam(value = "触发类型，2 手动执行，3 手动恢复，6 超级恢复", required =
true) Integer triggerType,
                                @RequestParam(required = false) @ApiParam(value = "任务执
行组", required = false) String execUser) throws InterruptedException, ExecutionException,
HeraException, TimeoutException {
        if (actionId == null) {
            return new JsonResponse(false, "请先生成版本再执行");
        }
        if (execUser == null) {
            checkPermission(ActionUtil.getJobId(actionId), RunAuthType.JOB);
        }
        TriggerTypeEnum triggerTypeEnum = TriggerTypeEnum.parser(triggerType);
        if (triggerTypeEnum == null) {
            return new JsonResponse(false, " 无法识别的触发类型，请联系管理员");
        }

        HeraAction heraAction = heraJobActionService.findById(actionId);
        HeraJob heraJob = heraJobService.findById(heraAction.getJobId());
        if (execUser == null) {
            execUser = super.getSsoName();
        }
        if (execUser == null) {
            return new JsonResponse(false, "任务执行人为空");
        }

        String configs = heraJob.getConfigs();
        HeraJobHistory actionHistory = HeraJobHistory.builder().build();
        actionHistory.setJobId(heraAction.getJobId());
        actionHistory.setActionId(heraAction.getId());
        actionHistory.setTriggerType(triggerTypeEnum.getId());
        actionHistory.setOperator(heraJob.getOwner());
        actionHistory.setIllustrate(execUser);
        actionHistory.setStatus(StatusEnum.RUNNING.toString());
        actionHistory.setStatisticEndTime(heraAction.getStatisticEndTime());
        actionHistory.setHostGroupId(heraAction.getHostGroupId());
        heraJobHistoryService.insert(actionHistory);
        heraAction.setScript(heraJob.getScript());
        heraAction.setHistoryId(actionHistory.getId());
        heraAction.setConfigs(configs);
        heraAction.setAuto(heraJob.getAuto());
        heraAction.setHostGroupId(heraJob.getHostGroupId());
        heraJobActionService.update(heraAction);
        workClient.executeJobFromWeb(JobExecuteKind.ExecuteKind.ManualKind, actionHistory.g
etId());

        String ownerId = getOwnerId();
        if (ownerId == null) {
            ownerId = "0";
        }
        addJobRecord(heraJob.getId(), String.valueOf(actionId), RecordTypeEnum.Execute, e
xecUser, ownerId);
        return new JsonResponse(true, String.valueOf(actionId));
    }

    @RequestMapping(value = "/updateJobMessage", method = RequestMethod.POST)
    @ResponseBody
    @RunAuth(idIndex = -1)
    @ApiOperation("更新任务信息")
    public JsonResponse updateJobMessage(@ApiParam(value = "任务 vo 对象", required = true)
HeraJobVo heraJobVo) {
        if (StringUtils.isBlank(heraJobVo.getDescription())) {
            return new JsonResponse(false, "描述不能为空");
        }
        try {
            new CronExpression(heraJobVo.getCronExpression());
        } catch (ParseException e) {
            return new JsonResponse(false, "定时表达式不准确，请核实后再保存");
        }
        HeraHostGroup hostGroup = heraHostGroupService.findById(heraJobVo.getHostGroupId())
;
        if (hostGroup == null) {
            return new JsonResponse(false, "机器组不存在，请选择一个机器组");
        }

        if (StringUtils.isBlank(heraJobVo.getAreaId())) {
            return new JsonResponse(false, "至少选择一个任务所在区域");
        }

        //如果是依赖任务
        if (heraJobVo.getScheduleType() == 1) {
            String dependencies = heraJobVo.getDependencies();
            if (StringUtils.isNotBlank(dependencies)) {
                String[] jobs = dependencies.split(Constants.COMMA);
                HeraJob heraJob;
                boolean jobAuto = true;
                StringBuilder sb = null;
                for (String job : jobs) {
                    heraJob = heraJobService.findById(Integer.parseInt(job));
                    if (heraJob == null) {
                        return new JsonResponse(false, "任务:" + job + "为空");
                    }
                    if (heraJob.getAuto() != 1) {
                        if (jobAuto) {
                            jobAuto = false;
                            sb = new StringBuilder();
                            sb.append(job);
                        } else {
                            sb.append(",").append(job);
                        }
                    }
                }
                if (!jobAuto) {
                    return new JsonResponse(false, "不允许依赖关闭状态的任务:" + sb.toStr
ing());
                }
            } else {
                return new JsonResponse(false, "请勾选你要依赖的任务");
            }
        } else if (heraJobVo.getScheduleType() == 0) {
            heraJobVo.setDependencies("");
        } else {
            return new JsonResponse(false, "无法识别的调度类型");
        }
        HeraJob memJob = heraJobService.findById(heraJobVo.getId());
        Map<String, String> configMap = StringUtil.configsToMap(heraJobVo.getSelfConfigs())
;
        configEncry(configMap);
        heraJobVo.setSelfConfigs(StringUtil.mapToConfigs(configMap));
        HeraJob newJob = BeanConvertUtils.convertToHeraJob(heraJobVo);
        int maxTimeOut = HeraGlobalEnv.getTaskTimeout() * 60;
        if (newJob.getMustEndMinute() > maxTimeOut) {
            return new JsonResponse(false, "超出最大超时限制,最大为:" + maxTimeOut);
        } else if (newJob.getMustEndMinute() == 0) {
            newJob.setMustEndMinute(60);
        }
        if (StringUtils.isNotBlank(newJob.getDependencies())) {
            if (!newJob.getDependencies().equals(memJob.getDependencies())) {
                List<HeraJob> relation = heraJobService.getAllJobDependencies();

                DagLoopUtil dagLoopUtil = new DagLoopUtil(heraJobService.selectMaxId());

                for (HeraJob job : relation) {
                    String dependencies;
                    if (job.getId() == newJob.getId()) {
                        dependencies = newJob.getDependencies();
                    } else {
                        dependencies = job.getDependencies();
                    }
                    if (StringUtils.isNotBlank(dependencies)) {
                        String[] split = dependencies.split(",");
                        for (String s : split) {
                            HeraJob memById = heraJobService.findMemById(Integer.parseInt
(s));
                            if (memById == null) {
                                return new JsonResponse(false, "依赖任务:" + s + "不存在");

                            }
                            dagLoopUtil.addEdge(job.getId(), Integer.parseInt(s));
                        }
                    }
                }

                if (dagLoopUtil.isLoop()) {
                    return new JsonResponse(false, "出现环形依赖，请检测依赖关系:" + dagL
oopUtil.getLoop());
                }
            }
        }


        newJob.setAuto(memJob.getAuto());

        String ssoName = getSsoName();
        String ownerId = getOwnerId();
        Integer update = heraJobService.update(newJob);
        if (update == null || update == 0) {
            return new JsonResponse(false, "更新失败");
        }
        doAsync(() -> {
            //脚本更新
            if (!newJob.getScript().equals(memJob.getScript())) {
                addJobRecord(newJob.getId(), memJob.getScript(), RecordTypeEnum.SCRIPT, s
soName, ownerId);
            }
            //依赖任务更新
            if (newJob.getDependencies() != null && !newJob.getDependencies().equals(memJ
ob.getDependencies())) {
                addJobRecord(newJob.getId(), memJob.getDependencies(), RecordTypeEnum.DEP
END, ssoName, ownerId);
            }
            //定时表达式更新
            if (newJob.getCronExpression() != null && !newJob.getCronExpression().equals(
memJob.getCronExpression())) {
                addJobRecord(newJob.getId(), memJob.getCronExpression(), RecordTypeEnum.C
RON, ssoName, ownerId);
            }
            //执行区域更新
            if (newJob.getAreaId() != null && !newJob.getAreaId().equals(memJob.getAreaId
())) {
                addJobRecord(newJob.getId(), memJob.getAreaId(), RecordTypeEnum.AREA, sso
Name, ownerId);
            }
            //脚本配置项变化
            if (newJob.getConfigs() != null && !newJob.getConfigs().equals(memJob.getConf
igs())) {
                addJobRecord(newJob.getId(), memJob.getConfigs(), RecordTypeEnum.CONFIG,
ssoName, ownerId);
            }
            if (newJob.getRunType() != null && !newJob.getRunType().equals(memJob.getRunT
ype())) {
                addJobRecord(newJob.getId(), memJob.getRunType(), RecordTypeEnum.RUN_TYPE,
ssoName, ownerId);
            }
        });

        return new JsonResponse(true, "更新成功");
    }




    @GetMapping(value = "previewJob")
    @ResponseBody
    @ApiOperation("预览任务接口")
    public JsonResponse previewJobScript(@ApiParam(value = "任务版本 id", required = true)
Long actionId) throws HeraException {
        return new JsonResponse("", true, getRenderScript(actionId));
    }
    private String getRenderScript(Long actionId) throws HeraException {
        return this.getRenderScript(actionId, null);
    }

    private String getRenderScript(Long actionId, String script) throws HeraException {
        Integer jobId = ActionUtil.getJobId(String.valueOf(actionId));
        HeraJobBean jobBean = heraGroupService.getUpstreamJobBean(jobId);
        if (script == null) {
            script = jobBean.getHeraJob().getScript();
        }

        RenderHierarchyProperties renderHierarchyProperties = new RenderHierarchyProperti
es(jobBean.getHierarchyProperties());
        script = JobUtils.previewScript(renderHierarchyProperties.getAllProperties(), scr
ipt);
        script = RenderHierarchyProperties.render(script, String.valueOf(actionId).substr
ing(0, 12));
        return script;
    }

    @RequestMapping(value = "/updateGroupMessage", method = RequestMethod.POST)
    @ResponseBody
    @ApiOperation("更新组信息")
    @RunAuth(authType = RunAuthType.GROUP, idIndex = 1)
    public JsonResponse updateGroupMessage(@ApiParam(value = "组信息对象", required = tru
e) HeraGroupVo groupVo, @ApiParam(value = "组 id", required = true) String groupId) {
        groupVo.setId(StringUtil.getGroupId(groupId));

        Map<String, String> configMap = StringUtil.configsToMap(groupVo.getSelfConfigs());
        configEncry(configMap);

        groupVo.setSelfConfigs(StringUtil.mapToConfigs(configMap));
        HeraGroup heraGroup = BeanConvertUtils.convert(groupVo);

        String ownerId = getOwnerId();
        String ssoName = getSsoName();
        doAsync(() -> {
            HeraGroup lastGroup = heraGroupService.findById(heraGroup.getId());
            if (lastGroup.getConfigs() != null && !lastGroup.getConfigs().equals(heraGrou
p.getConfigs())) {
                addGroupRecord(heraGroup.getId(), lastGroup.getConfigs(), RecordTypeEnum.
CONFIG, ssoName, ownerId);
            }
        });

        boolean res = heraGroupService.update(heraGroup) > 0;
        return new JsonResponse(res, res ? "更新成功" : "系统异常,请联系管理员");
    }
    private void configEncry(Map<String, String> config) {
        Optional.ofNullable(config)
                .ifPresent(cxf -> cxf.entrySet()
                        .stream()
                        .filter(pair -> pair.getKey().toLowerCase().contains(Constants.SE
CRET_PREFIX))
                        .forEach(entry -> entry.setValue(PasswordUtils.aesEncryption(entr
y.getValue()))));
    }

    private String getJobFromAuto(List<HeraJob> streamJob, Integer auto) {
        boolean has = false;
        StringBuilder filterJob = null;
        for (HeraJob job : streamJob) {
            if (job.getAuto().equals(auto)) {
                if (!has) {
                    has = true;
                    filterJob = new StringBuilder();
                    filterJob.append(job.getId());
                } else {
                    filterJob.append(",").append(job.getId());
                }
            }
        }
        if (has) {
            return filterJob.toString();
        }
        return null;
    }

    @RequestMapping(value = "/deleteJob", method = RequestMethod.POST)
    @ResponseBody
    @RunAuth(typeIndex = 1)
    @ApiOperation("删除任务接口")
    public JsonResponse deleteJob(@ApiParam(value = "任务 id", required = true) String id,
                              @ApiParam(value = "类型：job、group", required = true)
RunAuthType type) throws NoPermissionException {
        Integer xId = StringUtil.getGroupId(id);
        boolean res;
        String check = heraJobService.checkDependencies(xId, type);
        if (StringUtils.isNotBlank(check)) {
            return new JsonResponse(false, check);
        }
        String ssoName = getSsoName();
        String ownerId = getOwnerId();
        if (type == RunAuthType.GROUP) {
            res = heraGroupService.delete(xId) > 0;
            MonitorLog.info("{}【删除】组{}成功", ssoName, xId);
            doAsync(() -> addGroupRecord(xId, null, RecordTypeEnum.DELETE, ssoName, owner
Id));
```

```
            return new JsonResponse(res, res ? "删除成功" : "系统异常,请联系管理员");
        }
        res = heraJobService.delete(xId) > 0;
        MonitorLog.info("{}【删除】任务{}成功", getOwner(), xId);
        updateJobToMaster(res, xId);
        doAsync(() -> addJobRecord(xId, null, RecordTypeEnum.DELETE, ssoName, ownerId));
        return new JsonResponse(res, res ? "删除成功" : "系统异常,请联系管理员");
    }
}
package com.cy.iamlbmp.service.impl;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.cy.iamlbmp.mapper.BlacklistMapper;
import com.cy.iamlbmp.domain.Blacklist;
import com.cy.iamlbmp.service.IBlacklistService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class BlacklistServiceImpl extends ServiceImpl<BlacklistMapper, Blacklist> impleme
nts IBlacklistService{
    @Autowired
    private BlacklistMapper blacklistMapper;

    @Override
    public Blacklist getBlacklistByBlacklistId(Integer blacklistId){
        return blacklistMapper.selectBlacklistByBlacklistId(blacklistId);
    }

    @Override
    public List<Blacklist> listBlacklist(Blacklist blacklist){
        return blacklistMapper.selectBlacklistList(blacklist);
    }

    @Transactional
    @Override
    public int saveBlacklist(Blacklist blacklist){
        return blacklistMapper.insertBlacklist(blacklist);
    }

    private void checkExisted(BlacklistEntity entity) {
        BlacklistEntity existEntity = getOne(
                Wrappers.lambdaQuery(BlacklistEntity.class)
                .eq(BlacklistEntity::getBlacklistId, entity.getBlacklistId)
                false);
        if (existEntity != null && !existEntity.getId().equals(entity.getId())) {
            throw new BizException("黑名单记录已存在！");
        }
    }

    @Override
    @Transactional
    public int delBlacklistByBlacklistIds(Integer[] blacklistIds){
        return blacklistMapper.deleteBlacklistByBlacklistIds(blacklistIds);
    }

    private void blacklistData(List<BlacklistDto> list) {
        List<String> list = list.stream().map(BlacklistDto::getBlacklistId)
                .collect(Collectors.toList());
        list.stream().forEach(blacklist -> {
            Blacklist blacklistEntry = entryBlacklistMap.get(blacklist.getBlacklistId);
            if (blacklistEntry != null) {
                blacklist.setExpirationDate(BlacklistTrans(blacklistEntry.getExpirationDa
te));
                blacklist.setName(BlacklistTrans(blacklistEntry.getName));
            }
        });
    }

    @Override
    @Transactional
    public int updateBlacklist(Blacklist blacklist){
        return blacklistMapper.updateBlacklist(blacklist);
    }

    @Override
    @Transactional
    public int delBlacklistByBlacklistId(Integer blacklistId){
        return blacklistMapper.deleteBlacklistByBlacklistId(blacklistId);
    }
}
package com.cy.iamlbmp.domain;
import java.util.Date;
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Data;
import com.cy.iamlbmp.common.annotation.Excel;
import com.baomidou.mybatisplus.annotation.TableName;
import com.cy.iamlbmp.common.core.domain.BaseEntity;
/**
 * 报告实体类
 */
@TableName("iamlbmp_report")
@Data
public class Report extends BaseEntity {
    private static final long serialVersionUID = --1813483107844663615LL;
    // 报告 ID
    private Integer reportId;
    // 报告日期
    private String reportDate;
    // 创建时间
    private Date createdAt;
    // 报告类型
    private String reportType;
    // 报告接收方
    private String recipient;
    // 更新时间
    private Date updatedAt;
    // 关联的预警 ID
    private Integer alertId;
    // 报告状态
    private String status;
    // 报告描述
    private String description;
}
package com.cy.iamlbmp.service.impl;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.cy.iamlbmp.mapper.ReportMapper;
import com.cy.iamlbmp.domain.Report;
import com.cy.iamlbmp.service.IReportService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class ReportServiceImpl extends ServiceImpl<ReportMapper, Report> implements IRepo
rtService{
    @Autowired
    private ReportMapper reportMapper;

    @Override
    public Report getReportByReportId(Integer reportId){
        return reportMapper.selectReportByReportId(reportId);
    }

    @Override
    public List<Report> listReport(Report report){
        return reportMapper.selectReportList(report);
    }

    @Transactional
    @Override
    public int saveReport(Report report){
        return reportMapper.insertReport(report);
    }

    private void checkExisted(ReportEntity entity) {
        ReportEntity existEntity = getOne(
                Wrappers.lambdaQuery(ReportEntity.class)
                .eq(ReportEntity::getReportId, entity.getReportId)
                false);
        if (existEntity != null && !existEntity.getId().equals(entity.getId())) {
            throw new BizException("报告记录已存在！");
        }
    }

    @Override
    @Transactional
    public int delReportByReportIds(Integer[] reportIds){
        return reportMapper.deleteReportByReportIds(reportIds);
    }

    private void reportData(List<ReportDto> list) {
        List<String> list = list.stream().map(ReportDto::getReportId)
                .collect(Collectors.toList());
        list.stream().forEach(report -> {
            Report reportEntry = entryReportMap.get(report.getReportId);
            if (reportEntry != null) {
                report.setStatus(ReportTrans(reportEntry.getStatus));
                report.setReportId(ReportTrans(reportEntry.getReportId));
                report.setRecipient(ReportTrans(reportEntry.getRecipient));
            }
        });
    }

    @Override
    @Transactional
    public int updateReport(Report report){
        return reportMapper.updateReport(report);
    }

    @Override
    @Transactional
    public int delReportByReportId(Integer reportId){
        return reportMapper.deleteReportByReportId(reportId);
    }
}
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/myba
tis-3-mapper.dtd">
<mapper namespace="com.cy.iamlbmp.mapper.ReviewRecordMapper">
    <resultMap type="ReviewRecord" id="ReviewRecordResult">
        <result property="reviewId" column="review_ID"    />
        <result property="findings" column="findings"    />
        <result property="updatedAt" column="updated_at"    />
        <result property="status" column="status"    />
        <result property="reviewDate" column="review_date"    />
        <result property="recommendations" column="recommendations"    />
        <result property="customerId" column="customer_ID"    />
        <result property="createdAt" column="created_at"    />
        <result property="reviewer" column="reviewer"    />
        <result property="conclusion" column="conclusion"    />
    </resultMap>
    <sql id="selectReviewRecordVo">
        select review_ID, findings, updated_at, status, review_date, recommendations, cus
tomer_ID, created_at, reviewer, conclusion
        from iam1bmp_reviewrecord
    </sql>
    <select id="selectReviewRecordList" parameterType="ReviewRecord" resultMap="ReviewRec
ordResult">
        <include refid="selectReviewRecordVo"/>
        <where>
            <if test="reviewId != null ">
            and review_ID = #{reviewId}
            </if>
            <if test="findings != null and findings != ''">
            and findings = #{findings}
            </if>
            <if test="updatedAt != null ">
            and updated_at = #{updatedAt}
            </if>
            <if test="status != null and status != ''">
            and status = #{status}
            </if>
            <if test="reviewDate != null and reviewDate != ''">
            and review_date = #{reviewDate}
            </if>
            <if test="recommendations != null and recommendations != ''">
            and recommendations = #{recommendations}
            </if>
            <if test="customerId != null ">
            and customer_ID = #{customerId}
            </if>
            <if test="createdAt != null ">
            and created_at = #{createdAt}
            </if>
            <if test="reviewer != null and reviewer != ''">
            and reviewer = #{reviewer}
            </if>
            <if test="conclusion != null and conclusion != ''">
            and conclusion = #{conclusion}
            </if>
        </where>
    </select>
    <delete id="deleteReviewRecordByReviewId" parameterType="Integer">
        delete from iam1bmp_reviewrecord where review_ID = #{reviewId}
    </delete>

    <insert id="insertReviewRecord" parameterType="ReviewRecord" useGeneratedKeys="true"
keyProperty="reviewId">
        insert into iam1bmp_reviewrecord
        <trim prefix="(" suffix=")" suffixOverrides=",">
            <if test="findings != null and findings != ''">findings,</if>
            <if test="updatedAt != null">updated_at,</if>
            <if test="status != null and status != ''">status,</if>
            <if test="reviewDate != null and reviewDate != ''">review_date,</if>
            <if test="recommendations != null and recommendations != ''">recommendations,
</if>
            <if test="customerId != null">customer_ID,</if>
            <if test="createdAt != null">created_at,</if>
            <if test="reviewer != null and reviewer != ''">reviewer,</if>
            <if test="conclusion != null and conclusion != ''">conclusion,</if>
         </trim>
        <trim prefix="values (" suffix=")" suffixOverrides=",">
            <if test="findings != null and findings != ''">#{findings},</if>
            <if test="updatedAt != null">#{updatedAt},</if>
            <if test="status != null and status != ''">#{status},</if>
            <if test="reviewDate != null and reviewDate != ''">#{reviewDate},</if>
            <if test="recommendations != null and recommendations != ''">#{recommendation
s},</if>
            <if test="customerId != null">#{customerId},</if>
            <if test="createdAt != null">#{createdAt},</if>
            <if test="reviewer != null and reviewer != ''">#{reviewer},</if>
            <if test="conclusion != null and conclusion != ''">#{conclusion},</if>
         </trim>
    </insert>
    <delete id="deleteReviewRecordByReviewIds" parameterType="String">
        delete from iamlbmp_reviewrecord where review_ID in
        <foreach item="reviewId" collection="array" open="(" separator="," close=")">
            #{reviewId}
        </foreach>
    </delete>
</mapper>
package com.cy.iamlbmp.controller;
import java.util.List;
import javax.servlet.http.HttpServletResponse;
import com.cy.iamlbmp.common.core.domain.RespResult;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;
import com.cy.iamlbmp.common.annotation.Log;
import com.cy.iamlbmp.common.core.controller.BaseController;
import com.cy.iamlbmp.common.utils.poi.ExcelUtil;
import com.cy.iamlbmp.common.core.page.TableDataInfo;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.GetMapping;
import com.cy.iamlbmp.common.enums.BusinessType;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import com.cy.iamlbmp.domain.Blacklist;
import com.cy.iamlbmp.service.IBlacklistService;
import org.springframework.web.bind.annotation.PathVariable;
@RestController
@RequestMapping("/blacklist")
public class BlacklistController extends BaseAction {
    @Autowired
    private IBlacklistService blacklistService;

    /**
     * 查询黑名单列表
     */
    @GetMapping("/list")
    public TableDataInfo blacklistList(BlacklistQueryVo queryVo){
        startPage();
        List<Blacklist> blacklistList = blacklistService.getBlacklistList(queryVo);
        list.forEach(i -> {
            i.setExpirationDate(BlacklistConvUtil(i.getExpirationDate));
            i.setName(BlacklistConvUtil(i.getName));
        });
        PageInfo<Blacklist> pageInfo = new PageInfo<>(list);
        return RespResult.pageResult(list, pageInfo.getTotal());
    }

    @PostMapping
    public RespResult addBlacklist(@Valid @RequestBody BlacklistAddVo addVo){
       blacklistService.saveBlacklist(addVo);
       return RespResult.success();
    }

    public RespResult listblacklist(Blacklist blacklist){
        List<Blacklist> list = blacklistService.getBlacklistList(blacklist);
        return RespResult.success(list);
    }

    @DeleteMapping("/{blacklistIds}")
    public RespResult delblacklist(@PathVariable Integer[] blacklistIds){
        blacklistService.deleteBlacklistByBlacklistIds(blacklistIds);
        return RespResult.success();
    }

    @PutMapping
    public RespResult updateblacklist(@Valid @RequestBody Blacklist blacklist){
        blacklistService.updateBlacklist(blacklist);
        return RespResult.success();
    }

    @PostMapping("/export")
    public void exportBlacklist(HttpServletResponse response, Blacklist blacklist){
        List<Blacklist> list = blacklistService.getBlacklistList(blacklist);
        list.forEach(i -> {
            i.setExpirationDate(BlacklistExportFormat(i.getExpirationDate));
            i.setName(BlacklistExportFormat(i.getName));
        });
        ExcelUtil<Blacklist> excelUtil = new ExcelUtil<Blacklist>(Blacklist.class);
        util.exportExcel(response, list, "黑名单数据");
    }

    @GetMapping(value = "/{blacklistId}")
    public RespResult getblacklistId(@PathVariable("blacklistId") Integer blacklistId){
        return RespResult.success(blacklistService.getBlacklistByBlacklistId(blacklistId));

    }
}
package com.cy.iamlbmp.controller;
import java.util.List;
import javax.servlet.http.HttpServletResponse;
import com.cy.iamlbmp.common.core.domain.RespResult;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;
import com.cy.iamlbmp.common.annotation.Log;
import com.cy.iamlbmp.common.core.controller.BaseController;
import com.cy.iamlbmp.common.utils.poi.ExcelUtil;
import com.cy.iamlbmp.common.core.page.TableDataInfo;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.GetMapping;
import com.cy.iamlbmp.common.enums.BusinessType;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import com.cy.iamlbmp.domain.Report;
import com.cy.iamlbmp.service.IReportService;
import org.springframework.web.bind.annotation.PathVariable;
@RestController
@RequestMapping("/report")
public class ReportController extends BaseAction {
    @Autowired
    private IReportService reportService;

    /**
     * 查询报告列表
     */
    @GetMapping("/list")
    public TableDataInfo reportList(ReportQueryVo queryVo){
        startPage();
        List<Report> reportList = reportService.getReportList(queryVo);
        list.forEach(i -> {
            i.setStatus(ReportConvUtil(i.getStatus));
            i.setReportId(ReportConvUtil(i.getReportId));
            i.setRecipient(ReportConvUtil(i.getRecipient));
        });
        PageInfo<Report> pageInfo = new PageInfo<>(list);
        return RespResult.pageResult(list, pageInfo.getTotal());
    }

    @PostMapping
    public RespResult addReport(@Valid @RequestBody ReportAddVo addVo){
       reportService.saveReport(addVo);
       return RespResult.success();
    }

    public RespResult listreport(Report report){
        List<Report> list = reportService.getReportList(report);
        return RespResult.success(list);
    }

    @DeleteMapping("/{reportIds}")
    public RespResult delreport(@PathVariable Integer[] reportIds){
        reportService.deleteReportByReportIds(reportIds);
        return RespResult.success();
    }

    @PutMapping
    public RespResult updatereport(@Valid @RequestBody Report report){
        reportService.updateReport(report);
        return RespResult.success();
    }

    @PostMapping("/export")
    public void exportReport(HttpServletResponse response, Report report){
        List<Report> list = reportService.getReportList(report);
        list.forEach(i -> {
            i.setStatus(ReportExportFormat(i.getStatus));
            i.setReportId(ReportExportFormat(i.getReportId));
            i.setRecipient(ReportExportFormat(i.getRecipient));
        });
        ExcelUtil<Report> excelUtil = new ExcelUtil<Report>(Report.class);
        util.exportExcel(response, list, "报告数据");
    }

    @GetMapping(value = "/{reportId}")
    public RespResult getreportId(@PathVariable("reportId") Integer reportId){
        return RespResult.success(reportService.getReportByReportId(reportId));
    }
}
package com.cy.iamlbmp.domain;
import java.util.Date;
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Data;
import com.cy.iamlbmp.common.annotation.Excel;
import com.baomidou.mybatisplus.annotation.TableName;
import com.cy.iamlbmp.common.core.domain.BaseEntity;
/**
 * 黑名单实体类
 */
@TableName("iamlbmp_blacklist")
@Data
public class Blacklist extends BaseEntity {
    private static final long serialVersionUID = --1009693654514049023LL;
    // 黑名单 ID
    private Integer blacklistId;
    // 名称
    private String name;
    // 客户 ID
    private Integer customerId;
    // 过期日期
    private String expirationDate;
    // 创建时间
    private Date createdAt;
    // 列入黑名单原因
    private String reason;
    // 更新时间
    private Date updatedAt;
    // 信息来源
    private String source;
    // 国籍
    private String nationality;
    // 列入日期
    private String listedDate;
}
package com.cy.iamlbmp.controller.fanDataProblem;
import javafx.util.Pair;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.Map;
@Controller
public class StatisticsController{
    static ArrayList<Pair<Integer, Integer>> ret = new ArrayList<Pair<Integer, Integer>>(){

        {add(new Pair<>(0,  0)); add(new Pair<>(0,  0)); add(new Pair<>(0,  0));}};
    @RequestMapping(value = "/api/fan-data-problem/statistics", method = RequestMethod.PO
ST)
    @ResponseBody
    public ArrayList<Pair<Integer, Integer>> getStatistics(@RequestBody Map<String, Objec
t> params)
        throws NoSuchMethodException, InvocationTargetException, IllegalAccessException,
IOException{
        String operation = params.get("operation").toString();
        if(operation.equals("query")){
            return ret;
        } else if(operation.equals("clear")){
            for(int i = 0; i < ret.size(); i++)
                ret.set(i, new Pair<Integer, Integer>(0, 0));
            return ret;
        }else if(operation.equals("update")){
            Integer id = Integer.parseInt(params.get("id").toString());
            Integer key = Integer.parseInt(params.get("key").toString());
            Integer value = Integer.parseInt(params.get("value").toString());
            ret.set(id, new Pair<>(ret.get(id).getKey() + key, ret.get(id).getValue() + v
alue));
            return ret;
        }else{
            return ret;
        }
    }
}
package com.cy.iamlbmp.controller.util;
import cn.edu.thu.tsquality.core.common.datasource.CsvDataSource;
import cn.edu.thu.tsquality.core.common.datasource.IDataSource;
import com.cy.iamlbmp.common.constant.Constants;
import com.cy.iamlbmp.common.util.DataUtil;
import com.cy.iamlbmp.common.util.ServerTable;
import com.cy.iamlbmp.record.TabularData;
import org.apache.tomcat.util.http.fileupload.IOUtils;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
@Controller
public class FileController {
  @RequestMapping(value = "/api/chooseFile", method = RequestMethod.POST)
  @ResponseBody
  @CrossOrigin
  public Map<String, String> chooseFile(@RequestParam("file") MultipartFile file) {
    Map<String, String> fileMap = new HashMap<String, String>();
    String fileName = file.getOriginalFilename();
    fileMap.put("fileName", fileName);
    if (file.isEmpty()) {
      return fileMap;
    }
    String path = Constants.DATA_PATH;
    File dest = new File(path + "/" + fileName);
    if (!dest.getParentFile().exists()) { //判断文件父目录是否存在
      dest.getParentFile().mkdirs();
    }
    try {
      file.transferTo(dest); //保存文件
    } catch (IllegalStateException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
    return fileMap;
  }
  @RequestMapping(value = "/api/downloadFile", method = RequestMethod.GET)
  @CrossOrigin
  public void downloadFile(
      @RequestParam String type,
      @RequestParam String fileName,
      HttpServletRequest request, HttpServletResponse response) {

    String folder;
    switch (type.toUpperCase()) {
      case "DATA":
        folder = Constants.DATA_PATH;
        break;
      case "RULE":
        folder = Constants.RULE_PATH;
        break;
      case "ERROR":
        folder = Constants.ERROR_PATH;
        break;
      case "REPAIR":
        folder = Constants.REPAIR_PATH;
        break;
      default:
        folder = Constants.DATA_PATH;
        break;
    }
    String csvPath = folder + fileName;

    response.setCharacterEncoding(request.getCharacterEncoding());
    response.setContentType("application/octet-stream");
    FileInputStream fis = null;
    try {
      File file = new File(csvPath);
      fis = new FileInputStream(file);
      response.setHeader("Content-Disposition", "attachment; filename=" + file.getName());
      IOUtils.copy(fis, response.getOutputStream());
      response.flushBuffer();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      if (fis != null) {
        try {
          fis.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }
  }
  @RequestMapping(value = "/api/viewTable", method = RequestMethod.GET)
  @ResponseBody
  @CrossOrigin
  public TabularData showData(
      @RequestParam String type,
      @RequestParam String csvName) {
    String folder;
    switch (type.toUpperCase()) {
      case "DATA":
        folder = Constants.DATA_PATH;
        break;
      case "RULE":
        folder = Constants.RULE_PATH;
        break;
      case "ERROR":
        folder = Constants.ERROR_PATH;
        break;
      case "REPAIR":
        folder = Constants.REPAIR_PATH;
        break;
      default:
        folder = Constants.DATA_PATH;
        break;
    }
    String csvPath = folder + csvName;
    TabularData csvData = DataUtil.readCSV(csvPath);
    csvData.setFileName(csvName);
    return csvData;
  }
  /**
   * @param filename
   */
  @RequestMapping(value = "/api/data/csv", method = RequestMethod.GET)
  @ResponseBody
  public ServerTable viewCsvData(@RequestParam String filename) {
    try {
      if (!filename.startsWith("/")) {
        filename = Constants.DATA_PATH + filename;
      }
      IDataSource csvSource = new CsvDataSource(filename);
      return new ServerTable(csvSource.read());
    } catch (IOException | IllegalStateException e) {
      return null;
    }
  }
}
package com.cy.iamlbmp.service.impl;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.cy.iamlbmp.mapper.ReviewRecordMapper;
import com.cy.iamlbmp.domain.ReviewRecord;
import com.cy.iamlbmp.service.IReviewRecordService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class ReviewRecordServiceImpl extends ServiceImpl<ReviewRecordMapper, ReviewRecord>
implements IReviewRecordService{
    @Autowired
    private ReviewRecordMapper reviewRecordMapper;

    @Override
    public ReviewRecord getReviewRecordByReviewId(Integer reviewId){
        return reviewRecordMapper.selectReviewRecordByReviewId(reviewId);
    }

    @Override
    public List<ReviewRecord> listReviewRecord(ReviewRecord reviewRecord){
        return reviewRecordMapper.selectReviewRecordList(reviewRecord);
    }

    @Transactional
    @Override
    public int saveReviewRecord(ReviewRecord reviewRecord){
        return reviewRecordMapper.insertReviewRecord(reviewRecord);
    }

    private void checkExisted(ReviewRecordEntity entity) {
        ReviewRecordEntity existEntity = getOne(
                Wrappers.lambdaQuery(ReviewRecordEntity.class)
                .eq(ReviewRecordEntity::getReviewId, entity.getReviewId)
                false);
        if (existEntity != null && !existEntity.getId().equals(entity.getId())) {
            throw new BizException("审核记录记录已存在！");
        }
    }

    @Override
    @Transactional
    public int delReviewRecordByReviewIds(Integer[] reviewIds){
        return reviewRecordMapper.deleteReviewRecordByReviewIds(reviewIds);
    }

    private void reviewRecordData(List<ReviewRecordDto> list) {
        List<String> list = list.stream().map(ReviewRecordDto::getReviewId)
                .collect(Collectors.toList());
        list.stream().forEach(reviewRecord -> {
            ReviewRecord reviewRecordEntry = entryReviewRecordMap.get(reviewRecord.getRev
iewId);
            if (reviewRecordEntry != null) {
                reviewRecord.setStatus(ReviewRecordTrans(reviewRecordEntry.getStatus));
                reviewRecord.setReviewDate(ReviewRecordTrans(reviewRecordEntry.getReviewD
ate));
                reviewRecord.setUpdatedAt(ReviewRecordTrans(reviewRecordEntry.getUpdatedA
t));
            }
        });
    }

    @Override
    @Transactional
    public int updateReviewRecord(ReviewRecord reviewRecord){
        return reviewRecordMapper.updateReviewRecord(reviewRecord);
    }

    @Override
    @Transactional
    public int delReviewRecordByReviewId(Integer reviewId){
        return reviewRecordMapper.deleteReviewRecordByReviewId(reviewId);
    }
}
package com.cy.iamlbmp.dao;
import java.util.List;
import java.util.Map;
import com.cy.iamlbmp.model.Pager;
import com.cy.iamlbmp.model.User;
import com.cy.iamlbmp.util.StringUtils;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.springframework.stereotype.Repository;
@Repository
public class UsersDao extends BaseDao {
    public List<User> getAllUser() throws Exception {
        String hsql = "from t_tmp_users";
        Session session = sessionFactory.getCurrentSession();
        Query query = session.createQuery(hsql);
        return query.list();
    }
    /**
     * 查询用户
     *
     * @param map
     * @return
     * @throws Exception
     */
    public Map<String, Object> findUser(Map<String, Object> map) throws Exception {
        StringBuffer hsql = new StringBuffer("from t_tmp_users where 1=1");
        String id = (String) map.get("id");
        String searchText = (String) map.get("searchText");
        Pager page = (Pager) map.get("page");
        if (id != null) {
            hsql.append(" and id=");
            hsql.append(id);
        }
        if (searchText != null) {
            searchText = StringUtils.concat("'%", searchText, "%'");
            hsql.append(" and (user_name like " + searchText + " mobile like " + searchTe
xt);
        }
        Session session = sessionFactory.getCurrentSession();
        Query query = session.createQuery(hsql.toString());
        page.setTotalPage(query.list().size());
        query.setMaxResults(page.getPageSize());
        query.setFirstResult(page.getFirstIndex());
        List<User> list = query.list();
        if (list != null) {
            for (User u : list) {
                u.setPassword("");
            }
        }
        return packData(list, page);
    }

    /**
     * 修改密码
     *
     * @param id
     * @param md5DigestAsHex
     * @throws Exception
     */
    public int updatePassword(Integer id, String newPwd) throws Exception {
        Session session = sessionFactory.openSession();
        Transaction trans = session.beginTransaction();
        Query query = session.createQuery("update t_tmp_users t set t.password = ? where
id = ?");
        query.setParameter(0, newPwd);
        query.setParameter(1, id);
        int result = query.executeUpdate();
        trans.commit();
        session.close();
        return result;
    }
}
package com.cy.iamlbmp.common.util;
import java.io.IOException;
import java.io.OutputStream;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.math.BigDecimal;
import java.net.URL;
import java.util.function.Function;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;
public class ZipFileUtil {
    /**
     * zip 文件压缩
     *
     * @param inputFile    待压缩文件夹/文件名
     * @param outputFile   生成的压缩包
     * @param containsRoot 是否包含根目录
     * @throws IOException 压缩异常
     */
    public static void zip(File inputFile, File outputFile, boolean containsRoot) throws
IOException {
        if (outputFile.getParentFile() != null && !outputFile.getParentFile().exists()) {
            outputFile.getParentFile().mkdirs();
        }
        try (ZipOutputStream out = new ZipOutputStream(Files.newOutputStream(outputFile.t
oPath()))) {
            zip(inputFile, containsRoot ? inputFile.getName() : "", out);
        }
    }

    /**
     * 将 fileToZip 文件夹及其子目录文件递归压缩到 zip 文件中
     *
     * @param sourceFile 递归当前处理对象，可能是文件夹，也可能是文件
     * @param fileName   fileToZip 文件或文件夹名称
     * @param zipOut     压缩文件输出流
     * @throws IOException 压缩异常
     */
    private static void zip(File sourceFile, String fileName, ZipOutputStream zipOut) thr
ows IOException {
        //不压缩隐藏文件夹
        if (sourceFile.isHidden()) {
            return;
        }
        if (sourceFile.isDirectory()) {
            if (fileName.endsWith("/")) {
                zipOut.putNextEntry(new ZipEntry(fileName));
            } else if (!fileName.isEmpty()) {
                zipOut.putNextEntry(new ZipEntry(fileName + "/"));
            }
            zipOut.closeEntry();
            //遍历文件夹子目录，进行递归的 zipFile
            File[] children = sourceFile.listFiles();
            for (File childFile : children) {
                zip(childFile, fileName.isEmpty() ? childFile.getName() : (fileName + "/"
+ childFile.getName()),
                        zipOut);
            }
            return;
        }
        FileInputStream fis = new FileInputStream(sourceFile);
        ZipEntry zipEntry = new ZipEntry(fileName);
        zipOut.putNextEntry(zipEntry);
        byte[] bytes = new byte[1024];
        int length;
        while ((length = fis.read(bytes)) >= 0) {
            zipOut.write(bytes, 0, length);
        }
        fis.close();
    }

    /**
     * 解压 jar 文件到指定目录
     *
     * @param file      jar 文件
     * @param outputDir 输出目录
     * @throws IOException IO error
     */
    public static void unJar(File file, File outputDir) throws IOException {
        if (outputDir.exists() && outputDir.listFiles() != null) {
            delFile(outputDir);
        }
        if (outputDir.isFile()) {
            throw new IOException("Can not write to file " + outputDir + " (must be direc
tory)");
        }
        outputDir.mkdirs();
        try (JarFile jarFile = new JarFile(file)) {
            Enumeration<JarEntry> entries = jarFile.entries();
            while (entries.hasMoreElements()) {
                JarEntry entry = entries.nextElement();
                if (!entry.isDirectory()) {
                    File f = new File(outputDir, entry.getName());
                    if (f.getParentFile() != null) {
                        f.getParentFile().mkdirs();
                    }
                    try (InputStream is = jarFile.getInputStream(entry);
                            FileOutputStream fos = new FileOutputStream(f)) {
                        byte[] bytes = new byte[SAFE_BYTE_LENGTH];
                        int read = 0;
                        while ((read = is.read(bytes, 0, bytes.length)) > 0) {
                            fos.write(bytes, 0, read);
                        }
                    }
                }
            }
        }
    }

    /**
     * 删除文件，如果 file 是目录，递归删除目录下所有文件
     *
     * @param file 被删除的文件或文件目录
     * @return 是否删除成功
     */
    public static boolean delFile(File file) {
        if (!file.exists()) {
            return true;
        }
        if (file.isDirectory()) {
            File[] files = file.listFiles();
            for (File f : files) {
                if (!delFile(f)) {
                    return false;
                }
            }
        }
        return file.delete();
    }

    /**
     * 读取文件成为字符串
     *
     * @param f 源文件
     * @return hex string
     * @throws IOException when file is not exists
     */
    public static String readToHexStr(File f) throws IOException {
        if (!f.exists()) {
            throw new IOException("File not found: " + f);
        }
        byte[] bytes = Files.readAllBytes(f.toPath());
        return CodingUtil.bytes2HexString(bytes);
    }

    /**
     * 将 hex 字符串转为 bytes 并写成文件
     *
     * @param to     target file
     * @param hexStr hex string sources
     * @throws IOException when write file failed
     */
    public static void saveFromHexStr(File to, String hexStr) throws IOException {
        if (!to.getParentFile().exists()) {
            to.getParentFile().mkdirs();
        }
        if (!to.exists()) {
            to.createNewFile();
        }
        Files.write(to.toPath(), CodingUtil.hexStr2Bytes(hexStr));
    }

    /**
     * 从 Jar 包中删除文件
     *
     * @param jarPath Jar
     * @param regex   正则匹配
     */
    public static void delFromJar(String jarPath, String regex) throws IOException {
        File tmpFile = new File(getTempPath(), UUID.randomUUID().toString());
        try (JarFile jarFile = new JarFile(jarPath)) {
            boolean found = jarFile.stream().anyMatch(o -> !o.isDirectory() && o.getName().
matches(regex));
            if (found) {
                Enumeration<JarEntry> entries = jarFile.entries();
                while (entries.hasMoreElements()) {
                    JarEntry entry = entries.nextElement();
                    if (!entry.isDirectory() && !entry.getName().matches(regex)) {
                        File f = new File(tmpFile, entry.getName());
                        if (f.getParentFile() != null && !f.getParentFile().exists()) {
                            f.getParentFile().mkdirs();
                        }
                        if (f.exists()) {
                            f.delete();
                        }
                        try (InputStream is = jarFile.getInputStream(entry);
                                FileOutputStream fos = new FileOutputStream(f)) {
                            byte[] bytes = new byte[SAFE_BYTE_LENGTH];
                            int read = 0;
                            while ((read = is.read(bytes, 0, bytes.length)) > 0) {
                                fos.write(bytes, 0, read);
                            }
                        }
                    }
                }
            }
        }
        if (tmpFile.exists()) {
            File file = new File(jarPath);
            file.delete();
            zip(tmpFile, file, false);
            delFile(tmpFile);
        }
    }
}
package com.cy.iamlbmp.mapper;
import java.util.List;
import com.cy.iamlbmp.domain.RiskAssessment;
import com.baomidou.mybatisplus.core.mapper.BaseMapper;
/**
 * 风险评估 Dao 接口
 */
public interface RiskAssessmentMapper extends BaseMapper<RiskAssessment>{
    /**
     * 新增风险评估
     *
     * @param riskAssessment 风险评估
     * @return
     */
    int insertRiskAssessment(RiskAssessment riskAssessment);

    /**
     * 修改风险评估
     *
     * @param riskAssessment 风险评估
     * @return
     */
    int updateRiskAssessment(RiskAssessment riskAssessment);

    /**
     * 查询风险评估列表
     *
     * @param assessmentId 风险评估主键
     * @return
     */
    RiskAssessment selectRiskAssessmentByAssessmentId(Integer assessmentId);

    /**
     * 根据条件查询风险评估
     */
    List<RiskAssessmentVo> selectByCondition(RiskAssessmentQueryDto queryRiskAssessmentDt
o);

    /**
     * 删除风险评估
     * @param assessmentId 风险评估主键
     * @return
     */
    int deleteRiskAssessmentByAssessmentId(Integer assessmentId);

    /**
     * 批量删除风险评估
     * @param assessmentIds  ID 集合
     * @return
     */
    int deleteRiskAssessmentByAssessmentIds(Integer[] assessmentIds);
}

package com.cy.iamlbmp.learn.component;
import com.cy.iamlbmp.learn.Tensor;
import com.cy.iamlbmp.learn.layer.Layer;
import java.lang.reflect.Field;
import java.util.*;
/**
 * 高层神经网络组件
 */
public abstract class Component {
    private Map<String, Tensor> parameters = new HashMap<>();
    private String name;

    public Component(){
        this("layer");
    }

    public Component(String name){
        this.name = String.valueOf(name);
    }

    /**
     * 前向计算
     * @param inputs 输入
     * @return 输出
     */
    public Tensor forward(Tensor... inputs){
        //为参数创建对应的结点
        for (String parameterName : parameters.keySet()){
            Tensor tensor = parameters.get(parameterName);
            Layer.Node n = new Layer.Node();
            n.layer = null;
            n.output = tensor;
            tensor.setAttribute(Layer.NODE_ADDR_NAME,n );
        }
        return inputs[0];
    }

    /**
     * 创建参数
     * @param shape 参数的形状
     * @return 参数
     */
    protected Tensor createParameter(int... shape){
        //创建一个参数
        Tensor tensor = new Tensor(true, shape);  //使用 BP 算法来优化
        String name = UUID.randomUUID().toString();
        parameters.put(name, tensor);
        return tensor;
    }

    /**
     * 获取所有子计算结点
     * @return 计算结点
     */
    public List<Component> getChildren(){
        Class<?> cls = this.getClass();
        Field[] declaredFields = cls.getDeclaredFields();
        List<Component> components = new LinkedList<>();
        for (Field field : declaredFields){
            try {
                field.setAccessible(true);
                Object object = field.get(this);
                if(object instanceof Component){
                    Component component = (Component)object ;
                    components.add(component);
                }
            } catch (IllegalAccessException ignored) {}
        }
        return components;
    }
}
package com.cy.iamlbmp.learn.component;
import com.cy.iamlbmp.learn.Tensor;
import com.cy.iamlbmp.learn.component.Component;
import com.cy.iamlbmp.learn.layer.ElementAdd;
import com.cy.iamlbmp.learn.layer.MatDot;
/**
 * 线性变换层，默认没有激活函数
 */
public class Linear extends Component {
    private final Tensor W;
    private final Tensor b;
    private final MatDot matDot;
    private final ElementAdd elementAdd;

    public Linear(int inputSize, int outputSize){
        this.W = createParameter(outputSize, inputSize);
        this.b = createParameter(outputSize);
        this.matDot = new MatDot();
        this.elementAdd = new ElementAdd();
    }

    @Override
    public Tensor forward(Tensor... input) {
        super.forward(input);
        Tensor x = input[0];
        x = matDot.call(W, x);
        x = elementAdd.call(x, b);
        return x;
    }
}
package com.cy.iamlbmp.learn.metric;
import com.cy.iamlbmp.learn.Tensor;
import com.cy.iamlbmp.learn.layer.ElementAvg;
import com.cy.iamlbmp.learn.layer.ElementPow2;
import com.cy.iamlbmp.learn.layer.ElementSub;
import java.util.List;
/**
 * 均方误差
 * loss = (1 / n) \sum_i (y_i - y_hat_i)^2
 * @author yx.sdfbilblm
 * @date 2021/12/29
 */
public class MeanSquareError implements Metric {
    @Override
    public Tensor measure(List<Tensor> groundTruth, List<Tensor> output) {
        Tensor[] distances = new Tensor[output.size()];
        for (int i = 0; i < groundTruth.size(); i++){
            Tensor truth = groundTruth.get(i);
            Tensor pred = output.get(i);
            Tensor distance = new ElementSub().call(truth, pred);
            Tensor distance2 = new ElementPow2().call(distance);
            distances[i] = distance2;
        }
        return new ElementAvg().call(distances);
    }
}

package com.cy.iamlbmp.learn;
import com.cy.iamlbmp.learn.component.Component;
import com.cy.iamlbmp.learn.layer.Layer;
import com.cy.iamlbmp.learn.metric.Metric;
import com.cy.iamlbmp.learn.optimizer.Optimizer;
import com.cy.iamlbmp.learn.optimizer.OptimizerBuilder;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.*;
import java.util.concurrent.*;
public class Model {
    private final Component component;
    private final Random random;
    private final Map<Tensor, Optimizer> optimizers;

    public Model(Component component){
        this.component = component;
        this.random = new Random(new Date().getTime());
        this.optimizers = new HashMap<>();
    }

    /**
     * 训练模型
     * @param X  一批 x 数据
     * @param Y  一批 y 数据
     * @param lossFunction 损失函数
     * @param optimizerBuilder     优化器
     */
    public void train(List<Tensor[]> X, List<Tensor> Y, Metric lossFunction, OptimizerBui
lder optimizerBuilder){

        //创建与核心数相同的线程个数
        final ExecutorService executorService = Executors.newFixedThreadPool(Runtime.getR
untime().availableProcessors());
        //前向传播
        List<Tensor> outputs = new LinkedList<>();
        for (Tensor[] inputs : X){
            outputs.add(component.forward(inputs));
        }

        //executorService.shutdown();
        //计算损失函数
        Tensor loss = lossFunction.measure(Y, outputs);
        System.out.println(String.format("loss: %s", loss.get(0)));
        //反向传播算法(图的编历)
        Layer.Node root = (Layer.Node) loss.getAttribute("node");
```

```
Tensor rootGrid = new Tensor(loss.getShape());
rootGrid.fill(1);
root.grids.add(rootGrid);
Queue<Layer.Node> queue = new PriorityQueue<>();
queue.add(root);
while (!queue.isEmpty()){
    Layer.Node node = queue.poll();
    if(node.next.size() != 0 || node.grids.size() == 0){
        throw new RuntimeException("计算图出错");
    }
    //把所有梯度加起来
    Tensor gridAll = new Tensor(node.grids.get(0).getShape());
    for (int i = 0; i < gridAll.getSize(); i++){
        double sum = 0;
        for (Tensor grid : node.grids){
            sum += grid.getByLocation(i);
        }
        gridAll.setByLocation(i, sum);
    }
    //如果是参数，则梯度下降
    if (node.layer == null){
        Tensor parameter = node.output;
        Optimizer optimizer = optimizers.get(parameter);
        if(optimizer == null){
            optimizer = optimizerBuilder.build();
            optimizers.put(parameter, optimizer);
        }
        optimizer.optimize(parameter, gridAll);
    }
    // 计算得到目标函数对算子输入变量 X 的梯度
    else{
        Tensor[] gridX = node.layer.backward(node.context, gridAll, node.inputs);
        for (int i = 0; i < node.prev.size(); i++){
            Layer.Node prev = node.prev.get(i);
            if(prev != null && prev.layer == null){
                var j = 1 + 1;
            }
            if (prev != null && prev.next.size() > 0){
                //把梯度传播到上一个结点
                prev.grids.add(gridX[i]);
                //完成传播并把引用减 1
                prev.next.remove(0);
                if(!queue.contains(prev)){
                    queue.add(prev);
                }
            }
        }
    }

}
```

```
    }

    /**
     * 测试性能
     * @param X   一批 x 数据
     * @param Y   一批 y 数据
     * @param metrics 一批度量指标
     * @return 指标的值
     */
    public List<Double> Test(List<Tensor[]> X, List<Tensor> Y, List<Metric> metrics){
        //TODO: 实现测试代码
        return null;
    }

    /**
     * 做预测
     * @param inputs 输入
     * @return 输出
     */
    public Tensor predict(Tensor[] inputs){
        return component.forward(inputs);
    }

    /**
     * 保存模拟
     * @param outputStream 输出流
     */
    public void save(OutputStream outputStream){

    }

    /**
     * 加载模型
     * @param inputStream 输入流
     */
    public void load(InputStream inputStream){

    }
}
package com.cy.iamlbmp.bert;
import org.tensorflow.Graph;
import org.tensorflow.Session;
import org.tensorflow.Tensor;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.*;
public class BertNerPredict {
    private static String vocabPath = "src/main/resources/vocab.txt";
    private static Map<String, Integer> word2id = new HashMap<String, Integer>();
    static {
        try {
            BufferedReader buffer = null;
            buffer = new BufferedReader(new InputStreamReader(new FileInputStream(vocabPa
th)));
            int i = 0;
            String line = buffer.readLine().trim();
            while (line!=null){
                word2id.put(line, i++);
                line = buffer.readLine().trim();
            }
            buffer.close();
        }catch (Exception e){
        }

    }

    private static byte[] readAllByteOrExit(Path path){
        try{
            return Files.readAllBytes(path);
        }catch (IOException e){
            System.out.println("Failed to read[" + path + "]:" + e.getMessage());
            System.exit(1);
        }
        return null;
    }

    public static void getTextToId(int[][] inputIds, int[][] inputMask, String[] text){
        for(int i=0; i<text.length; i++){
            char[] chs = text[i].trim().toLowerCase().toCharArray();

            List<Integer> list = new ArrayList<>();
            List<Integer> mask = new ArrayList<>();
            list.add(word2id.get("[CLS]"));
            mask.add(1);
            for(int j=0; j<chs.length; j++){
                String element = Character.toString(chs[j]);
                if(word2id.containsKey(element)){
                    list.add(word2id.get(element));
                    mask.add(1);
                }
            }
            list.add(word2id.get("[SEP]"));
            mask.add(1);
            int size = list.size();
            Integer[] targetInter = list.toArray(new Integer[size]);
            int[] idResult = Arrays.stream(targetInter).mapToInt(Integer::valueOf).toArra
y();
            Integer[] maskInter = mask.toArray(new Integer[size]);
            int[] maskResult = Arrays.stream(maskInter).mapToInt(Integer::valueOf).toArra
y();
            System.arraycopy(idResult,0,inputIds[i], 0, idResult.length);
            System.arraycopy(maskResult,0,inputMask[i], 0, maskResult.length);
        }
    }
}
package com.cy.iamlbmp.controller;
import com.cy.iamlbmp.common.constants.Constants;
import com.cy.iamlbmp.common.entity.*;
import com.cy.iamlbmp.common.entity.model.JsonResponse;
import com.cy.iamlbmp.common.entity.model.TablePageForm;
import com.cy.iamlbmp.common.entity.model.TableResponse;
import com.cy.iamlbmp.common.entity.vo.HeraActionVo;
import com.cy.iamlbmp.common.entity.vo.HeraGroupVo;
import com.cy.iamlbmp.common.entity.vo.HeraJobVo;
import com.cy.iamlbmp.common.entity.vo.PageHelperTimeRange;
import com.cy.iamlbmp.common.enums.*;
import com.cy.iamlbmp.common.exception.NoPermissionException;
import com.cy.iamlbmp.common.service.*;
import com.cy.iamlbmp.common.util.ActionUtil;
import com.cy.iamlbmp.common.util.BeanConvertUtils;
import com.cy.iamlbmp.common.util.PasswordUtils;
import com.cy.iamlbmp.common.util.StringUtil;
import com.cy.iamlbmp.common.vo.GroupTaskVo;
import com.cy.iamlbmp.config.HeraGlobalEnv;
import com.cy.iamlbmp.config.RunAuth;
import com.cy.iamlbmp.config.UnCheckLogin;
import com.cy.iamlbmp.core.netty.worker.WorkClient;
import com.cy.iamlbmp.logs.MonitorLog;
import com.cy.iamlbmp.protocol.JobExecuteKind;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeoutException;
import java.util.stream.Collectors;
```

```
/**
 * 调度中心视图管理器
 */
@Controller
@Api(value = "调度中心")
@RequestMapping("/scheduleCenter")
public class ScheduleCenterController extends BaseHeraController {

    private final HeraJobService heraJobService;
    private final HeraJobActionService heraJobActionService;
    private final HeraGroupService heraGroupService;
    private final HeraJobHistoryService heraJobHistoryService;
    private final HeraJobMonitorService heraJobMonitorService;
    private final HeraUserService heraUserService;
    private final HeraPermissionService heraPermissionService;
    private final WorkClient workClient;
    private final HeraHostGroupService heraHostGroupService;
    private final HeraAreaService heraAreaService;
    private final HeraSsoService heraSsoService;

    public ScheduleCenterController(HeraJobMonitorService heraJobMonitorService,
        @Qualifier("heraJobMemoryService") HeraJobService heraJobService, HeraJobActionSe
rvice heraJobActionService,
        @Qualifier("heraGroupMemoryService") HeraGroupService heraGroupService, HeraJobHi
storyService heraJobHistoryService,
        HeraUserService heraUserService, HeraPermissionService heraPermissionService,
        WorkClient workClient, HeraHostGroupService heraHostGroupService, HeraAreaService
heraAreaService, HeraSsoService heraSsoService) {
        this.heraJobMonitorService = heraJobMonitorService;
        this.heraJobService = heraJobService;
        this.heraJobActionService = heraJobActionService;
        this.heraGroupService = heraGroupService;
        this.heraJobHistoryService = heraJobHistoryService;
        this.heraUserService = heraUserService;
        this.heraPermissionService = heraPermissionService;
        this.workClient = workClient;
        this.heraHostGroupService = heraHostGroupService;
        this.heraAreaService = heraAreaService;
        this.heraSsoService = heraSsoService;
    }


    @RequestMapping(method = RequestMethod.GET)
    @ApiOperation(value = "调度中心页面跳转")
    public String login() {
        return "scheduleCenter/scheduleCenter.index";
    }

    @RequestMapping(value = "/init", method = RequestMethod.POST)
    @ResponseBody
    @ApiOperation(value = "获取任务树,包含我的任务和全部任务两个集合")
    public JsonResponse initJobTree() {
        return new JsonResponse(true, Optional.of(heraJobService.buildJobTree(getOwner(),
Integer.parseInt(getSsoId()))).get());
    }

    @RequestMapping(value = "/getJobMessage", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation(value = "获取任务信息")
    public JsonResponse getJobMessage(@ApiParam(value = "任务 ID", required = true) Intege
r jobId) {
        HeraJob job = heraJobService.findById(jobId);
        HeraJobVo heraJobVo = BeanConvertUtils.convert(job);
        heraJobVo.setInheritConfig(getInheritConfig(job.getGroupId()));
        HeraJobMonitor monitor = heraJobMonitorService.findByJobId(jobId);
        StringBuilder focusUsers = new StringBuilder("[ ");
        Optional.ofNullable(monitor).ifPresent(m -> {
            if (StringUtils.isNotBlank(m.getUserIds())) {
                String ssoId = getSsoId();
                Arrays.stream(monitor.getUserIds().split(Constants.COMMA)).filter(StringU
tils::isNotBlank).distinct().forEach(id -> {
                    if (ssoId.equals(id)) {
                        heraJobVo.setFocus(true);
                        focusUsers.append(Constants.BLANK_SPACE).append(getSsoName());
                    } else {
                        Optional.ofNullable(heraSsoService.findSsoById(Integer.parseInt(i
d)))
                                .ifPresent(sso -> focusUsers.append(Constants.BLANK_SPACE).
append(sso.getName()));
                    }
                });
            }
        });
        focusUsers.append(" ]");
        Optional.ofNullable(heraHostGroupService.findById(job.getHostGroupId()))
                .ifPresent(group -> heraJobVo.setHostGroupName(group.getName()));
        heraJobVo.setUIdS(getuIds(jobId, RunAuthType.JOB));
        heraJobVo.setFocusUser(focusUsers.toString());
        heraJobVo.setAlarmLevel(AlarmLevel.getName(job.getOffset()));
        heraJobVo.setCycle(CycleEnum.parse(job.getCycle()).getDesc());
        configDecry(heraJobVo.getConfigs());
        configDecry(heraJobVo.getInheritConfig());
        //如果无权限，进行变量加密
        if (heraJobVo.getConfigs().keySet().stream().anyMatch(key -> key.toLowerCase().co
ntains(Constants.PASSWORD_WORD))
                || heraJobVo.getInheritConfig().keySet().stream().anyMatch(key -> key.toL
owerCase().contains(Constants.PASSWORD_WORD))) {
            try {
                checkPermission(jobId, RunAuthType.JOB);
            } catch (NoPermissionException e) {
                encryption(heraJobVo.getConfigs());
                encryption(heraJobVo.getInheritConfig());
            }
        }
        return new JsonResponse(true, heraJobVo);
    }


    @RunAuth(typeIndex = 1)
    @GetMapping("/checkPermission")
    @ResponseBody
    @ApiOperation(value = "权限检测接口")
    public JsonResponse doAspectAuth(@ApiParam(value = "任务 ID", required = true) Integer
jobId
            , @ApiParam(value = "检测类型", required = true) RunAuthType type) {
        return new JsonResponse(true, true);
    }


    private void configDecry(Map<String, String> config) {
        Optional.ofNullable(config)
                .ifPresent(cxf -> cxf.entrySet()
                        .stream()
                        .filter(pair -> pair.getKey().toLowerCase().contains(Constants.SE
CRET_PREFIX))
                        .forEach(entry -> entry.setValue(PasswordUtils.aesDecrypt(entry.g
etValue()))));
    }


    private void encryption(Map<String, String> config) {
        Optional.ofNullable(config)
                .ifPresent(cxf -> cxf.entrySet()
                        .stream()
                        .filter(pair -> pair.getKey().toLowerCase().contains(Constants.PA
SSWORD_WORD))
                        .forEach(entry -> entry.setValue("******")));
    }

    /**
     * 组下搜索任务
     *
     * @param groupId  groupId
     * @param status   all:全部;
     * @param pageForm layui table 分页参数
     * @return 结果
     */
    @RequestMapping(value = "/getGroupTask", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation(value = "某组下搜索任务")
    public TableResponse getGroupTask(@ApiParam(value = "组 Id", required = true) String g
roupId,
                                      @ApiParam(value = "任务状态") String status,
                                      @ApiParam(value = "日期", required = true) String d
t,
                                      @ApiParam(value = "分页参数", required = true) Tabl
ePageForm pageForm) {

        List<HeraGroup> group = heraGroupService.findDownStreamGroup(StringUtil.getGroupI
d(groupId));

        Set<Integer> groupSet = group.stream().map(HeraGroup::getId).collect(Collectors.t
oSet());
        List<HeraJob> jobList = heraJobService.getAll();
        Set<Integer> jobIdSet = jobList.stream().filter(job -> groupSet.contains(job.getG
roupId())).map(HeraJob::getId).collect(Collectors.toSet());
        SimpleDateFormat format = new SimpleDateFormat("yyMMdd");
        Calendar calendar = Calendar.getInstance();
        String startDate;
        Date start;
        if (StringUtils.isBlank(dt)) {
            start = new Date();
        } else {
            try {
                start = format.parse(dt);
            } catch (ParseException e) {
                start = new Date();
            }
        }

        calendar.setTime(start);
        startDate = ActionUtil.getFormatterDate("yyyyMMdd", calendar.getTime());
        calendar.add(Calendar.DAY_OF_YEAR, +1);
        String endDate = ActionUtil.getFormatterDate("yyyyMMdd", calendar.getTime());
        List<GroupTaskVo> taskVos = heraJobActionService.findByJobIds(new ArrayList<>(job
IdSet), startDate, endDate, pageForm, status);
        return new TableResponse(pageForm.getCount(), 0, taskVos);
    }

    @RequestMapping(value = "/getGroupMessage", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation(value = "获取组信息")
    public JsonResponse getGroupMessage(@ApiParam(value = "组 ID", required = true) String
groupId) {
        Integer id = StringUtil.getGroupId(groupId);
        HeraGroup group = heraGroupService.findById(id);
        HeraGroupVo groupVo = BeanConvertUtils.convert(group);
        groupVo.setInheritConfig(getInheritConfig(groupVo.getParent()));
        groupVo.setUIdS(getuIds(id, RunAuthType.GROUP));
        configDecry(groupVo.getConfigs());
        configDecry(groupVo.getInheritConfig());
        if (groupVo.getConfigs().keySet().stream().anyMatch(key -> key.toLowerCase().cont
ains(Constants.PASSWORD_WORD))
                || groupVo.getInheritConfig().keySet().stream().anyMatch(key -> key.toLow
erCase().contains(Constants.PASSWORD_WORD))) {
            try {
                checkPermission(id, RunAuthType.GROUP);
            } catch (NoPermissionException e) {
                encryption(groupVo.getConfigs());
                encryption(groupVo.getInheritConfig());
            }
        }
        return new JsonResponse(true, groupVo);
    }

    @RequestMapping(value = "/getJobOperator", method = RequestMethod.GET)
    @ResponseBody
    @RunAuth(typeIndex = 1)
    @ApiOperation("获取任务管理员，admin 表示目前有权限操作的用户")
    public JsonResponse getJobOperator(@ApiParam(value = "任务/组 ID", required = true) St
ring jobId,
                                       @ApiParam(value = "任务类型", required = true) Run
AuthType type) {
        Integer groupId = StringUtil.getGroupId(jobId);
        List<HeraPermission> permissions = heraPermissionService.findByTargetId(groupId,
type.getName(), 1);
        List<HeraUser> all = heraUserService.findAllName();
        if (all == null || permissions == null) {
            return new JsonResponse(false, "发生错误，请联系管理员");
        }
        Map<String, Object> res = new HashMap<>(2);
        res.put("allUser", all);
        res.put("admin", permissions);
        return new JsonResponse(true, "查询成功", res);
    }

    @RequestMapping(value = "/getJobVersion", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation("获取任务的所有版本")
    public JsonResponse getJobVersion(@ApiParam(value = "任务 ID", required = true) Long j
obId) {
        return new JsonResponse(true, heraJobActionService.getActionVersionByJobId(jobId)
                .stream()
                .map(id -> HeraActionVo.builder().id(id).build())
                .collect(Collectors.toList()));
    }

    @RequestMapping(value = "/addJob", method = RequestMethod.POST)
    @ResponseBody
    @ApiOperation("新增任务")
    @RunAuth(authType = RunAuthType.GROUP, idIndex = 1)
    public JsonResponse addJob(@ApiParam(value = "任务信息", required = true) HeraJob her
aJob,
                               @ApiParam(value = "所在组目录", required = true) String pa
rentId) {
        heraJob.setGroupId(StringUtil.getGroupId(parentId));
        heraJob.setHostGroupId(HeraGlobalEnv.defaultWorkerGroup);
        heraJob.setOwner(getOwner());
        heraJob.setScheduleType(JobScheduleTypeEnum.Independent.getType());
        int insert = heraJobService.insert(heraJob);
        if (insert > 0) {
            addJobRecord(heraJob);
            return new JsonResponse(true, String.valueOf(heraJob.getId()));
        } else {
            return new JsonResponse(false, "新增失败");
        }
    }

    @PostMapping(value = "/copyJob")
    @ResponseBody
    @RunAuth(authType = RunAuthType.JOB)
    @ApiOperation("复制任务接口")
    public JsonResponse copyJob(
            @ApiParam(value = "复制的任务 ID", required = true) int jobId) {
        HeraJob copyJob = heraJobService.findById(jobId);
        copyJob.setName(copyJob.getName() + "_copy");
        copyJob.setOwner(getOwner());
        copyJob.setScheduleType(JobScheduleTypeEnum.Independent.getType());
        copyJob.setId(0);
        copyJob.setIsValid(0);
        int insert = heraJobService.insert(copyJob);
        if (insert > 0) {
            addJobRecord(copyJob);
            return new JsonResponse(true, String.valueOf(copyJob.getId()));
        } else {
            return new JsonResponse(false, "新增失败");
        }
    }

    private void addJobRecord(HeraJob heraJob) {
        String ssoName = getSsoName();
        String ownerId = getOwnerId();
        MonitorLog.info("{}[{}]【添加】任务{}成功", heraJob.getOwner(), ssoName, heraJob.
getId());
        updateMonitor(heraJob.getId());
        doAsync(() -> addJobRecord(heraJob.getId(), heraJob.getName(), RecordTypeEnum.Add,
ssoName, ownerId));
    }

    @RequestMapping(value = "/addMonitor", method = RequestMethod.POST)
    @ResponseBody
    @ApiOperation("任务添加监控")
    public JsonResponse updateMonitor(@ApiParam(value = "任务 ID", required = true) Intege
r id) {
        String ssoId = getSsoId();
        if (Constants.DEFAULT_ID.equals(ssoId)) {
            return new JsonResponse(false, "组账户不支持监控任务");
        }
        boolean res = heraJobMonitorService.addMonitor(ssoId, id);
        if (res) {
            MonitorLog.info("{}【关注】任务{}成功", getSsoName(), id);
            return new JsonResponse(true, "关注成功");
        } else {
            return new JsonResponse(false, "系统异常，请联系管理员");
        }
    }

    @RequestMapping(value = "/delMonitor", method = RequestMethod.POST)
    @ResponseBody
    @ResponseStatus(HttpStatus.OK)
    @ApiOperation("删除任务关注人")
    public JsonResponse deleteMonitor(@ApiParam(value = "任务 ID", required = true) Intege
r id) {
        String ssoId = getSsoId();
        HeraJobMonitor monitor = heraJobMonitorService.findByJobId(id);
        if (monitor != null) {
            if (monitor.getUserIds().split(Constants.COMMA).length == 1) {
                return new JsonResponse(false, "至少有一个监控人");
            }
        }
        boolean res = heraJobMonitorService.removeMonitor(ssoId, id);
        if (res) {
            MonitorLog.info("{}【取关】任务{}成功", getSsoName(), id);
            return new JsonResponse(true, "取关成功");
        } else {
            return new JsonResponse(false, "系统异常，请联系管理员");
        }
    }

    @RequestMapping(value = "/addGroup", method = RequestMethod.POST)
    @ResponseBody
    @RunAuth(authType = RunAuthType.GROUP, idIndex = 1)
    @ApiOperation("添加组")
    public JsonResponse addJob(@ApiParam(value = "组信息", required = true) HeraGroup her
aGroup,
                               @ApiParam(value = "所在组 id", required = true) String pare
ntId) {
        heraGroup.setParent(StringUtil.getGroupId(parentId));
        heraGroup.setOwner(getOwner());
        heraGroup.setExisted(1);
        int insert = heraGroupService.insert(heraGroup);
        if (insert > 0) {
            MonitorLog.info("{}【添加】组{}成功", getSsoName(), heraGroup.getId());
            return new JsonResponse(true, Constants.GROUP_PREFIX + heraGroup.getId());
        } else {
            return new JsonResponse(false, String.valueOf(-1));

        }
    }

    @RequestMapping(value = "/generateVersion", method = RequestMethod.POST)
    @ResponseBody
    @RunAuth
    @ApiOperation("全量版本生成/单个版本生成")
    public JsonResponse generateVersion(@ApiParam("任务 ID") Long jobId) throws ExecutionE
xception, InterruptedException, TimeoutException {
        return new JsonResponse(true, workClient.generateActionFromWeb(JobExecuteKind.Exe
cuteKind.ManualKind, jobId));
    }

    /**
     * 获取任务历史版本
     *
     * @param pageHelper
     * @return
     */
    @RequestMapping(value = "/getJobHistory", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation("任务历史记录")
    public JsonResponse getJobHistory(@ApiParam(value = "分页", required = true) PageHelp
erTimeRange pageHelper) {
        return new JsonResponse(true, heraJobHistoryService.findLogByPage(pageHelper));
    }

    @RequestMapping(value = "/getHostGroupIds", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation("获取机器组 Id")
    public JsonResponse getHostGroupIds() {
        return new JsonResponse(true, heraHostGroupService.getAll());
    }

    @RequestMapping(value = "getLog", method = RequestMethod.GET)
    @ResponseBody
    @RunAuth()
    @ApiOperation("获取任务日志接口")
    public JsonResponse getJobLog(@ApiParam(value = "任务 ID", required = true) Integer id)
{
        return new JsonResponse(true, heraJobHistoryService.findLogById(id));
    }
    @RequestMapping(value = "jobInstLog", method = RequestMethod.GET)
    @ResponseBody
    @RunAuth()
    @ApiOperation("获取任务日志接口")
    public String jobInstLog(@ApiParam(value = "任务 ID", required = true) Integer id,Inte
ger hisId) {
        return  heraJobHistoryService.findLogById(hisId).getLog();
    }

    @RequestMapping(value = "/status/{jobId}", method = RequestMethod.GET)
    @ResponseBody
    @UnCheckLogin
    @ApiOperation("开放接口,查询任务状态")
    public JsonResponse getStatus(@PathVariable("jobId") @ApiParam(value = "任务 ID", requ
ired = true) Integer jobId
            , @RequestParam("time") @ApiParam(value = "时间戳，只查询该时间戳之后的记录",
required = true) long time) {
        HeraJobHistory history = heraJobHistoryService.findNewest(jobId);
        if (history == null) {
            return new JsonResponse(false, "无执行记录");
        }
        //此时可能正在创建动态集群 或者发送 netty 消息的路上
        if (history.getStartTime() == null && history.getGmtCreate().getTime() >= time) {
            return new JsonResponse(true, StatusEnum.RUNNING.toString());
        }

        if (history.getStartTime() != null && history.getStartTime().getTime() < time) {
            return new JsonResponse(false, "无执行记录");
        }
        return new JsonResponse(true, Optional.ofNullable(history.getStatus()).orElse(Sta
tusEnum.RUNNING.toString()));
    }

    private Map<String, String> getInheritConfig(Integer groupId) {
        HeraGroup group = heraGroupService.findConfigById(groupId);
        Map<String, String> configMap = new TreeMap<>();
        while (group != null && groupId != null && groupId != 0) {
            Map<String, String> map = StringUtil.convertStringToMap(group.getConfigs());
            // 多重继承相同变量，以第一个的为准
            for (Map.Entry<String, String> entry : map.entrySet()) {
                String key = entry.getKey();
                if (!configMap.containsKey(key)) {
                    configMap.put(key, entry.getValue());
                }
            }
            groupId = group.getParent();
            group = heraGroupService.findConfigById(groupId);
        }
        return configMap;
    }
    private String getuIds(Integer id, RunAuthType type) {
        List<HeraPermission> permissions = heraPermissionService.findByTargetId(id, type.
getName(), 1);
        StringBuilder uids = new StringBuilder("[ ");
        if (permissions != null && permissions.size() > 0) {
            permissions.forEach(x -> uids.append(x.getUid()).append(" "));
        }
        uids.append("]");

        return uids.toString();
    }

    @RequestMapping(value = "/getJobImpactOrProgress", method = RequestMethod.POST)
    @ResponseBody
    @ApiOperation("查询任务依赖关系接口")
    public JsonResponse getJobImpactOrProgress(@ApiParam(value = "任务 ID", required = tru
e) Integer jobId
            , @ApiParam(value = "0:上游/1:下游", required = true) Integer type) {
        Map<String, Object> graph = heraJobService.findCurrentJobGraph(jobId, type);
        if (graph == null) {
            return new JsonResponse(false, "当前任务不存在");
        }
        return new JsonResponse(true, "成功", graph);
    }

    @RequestMapping(value = "/getAllArea", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation("获取所有区域")
    public JsonResponse getAllArea() {
        return new JsonResponse(true, "成功", Optional.of(heraAreaService.findAll()).get())
;
    }

    /**
     * @param jobHisId
     * @param status
     * @return
     */
    @RequestMapping(value = "/forceJobHisStatus", method = RequestMethod.GET)
    @ResponseBody
    @ApiOperation("强制设置任务状态")
    public JsonResponse forceJobHisStatus(@ApiParam(value = "执行记录 id", required = true)
Long jobHisId
            , @ApiParam(value = "设置的状态", required = true) String status) {

        String info = "";
        if (status.equals(StatusEnum.WAIT.toString())) {
            info = "手动强制等待状态";
        } else if (status.equals(StatusEnum.FAILED.toString())) {
            info = "手动强制失败状态";
        } else if (status.equals(StatusEnum.SUCCESS.toString())) {
            info = "手动强制成功状态";
        } else if (status.equals(StatusEnum.RUNNING.toString())) {
            info = "手动强制运行中状态";
        }

        String illustrate = heraJobHistoryService.findById(jobHisId).getIllustrate();
        if (StringUtils.isNotBlank(illustrate)) {
            illustrate += ";" + info;
        } else {
            illustrate = info;
        }
        heraJobHistoryService.updateStatusAndIllustrate(jobHisId, status, illustrate, new
Date());
        return new JsonResponse(true, "修改状态成功");
    }
}
```