项目结构：

配置层

QuartzConfig（Quartz配置类）

AutowiringSpringBeanJobFactory（Spring Bean工厂）

服务层

JobService（任务服务类）

任务层

BaseJob（抽象基础任务类）

LoggingJob（具体日志记录任务实现）

工具层

CronUtils（Cron表达式工具类）

JobUtils（任务工具类）

运行层

JobManagementRunner（任务管理启动器）

模型类

JobInfo

程序的application入口：

SchedulerApplication

QuartzConfig类：

package com.scheduler.config;  
  
import org.quartz.Scheduler;  
import org.quartz.spi.JobFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.scheduling.quartz.SchedulerFactoryBean;  
  
import java.util.Properties;  
  
@Configuration  
public class QuartzConfig {  
  
 @Autowired  
 private ApplicationContext applicationContext;  
  
 @Bean  
 public JobFactory jobFactory() {  
 AutowiringSpringBeanJobFactory jobFactory = new AutowiringSpringBeanJobFactory();  
 jobFactory.setApplicationContext(applicationContext);  
 return jobFactory;  
 }  
  
 @Bean  
 public SchedulerFactoryBean schedulerFactoryBean() {  
 SchedulerFactoryBean schedulerFactoryBean = new SchedulerFactoryBean();  
  
 // 设置JobFactory  
 schedulerFactoryBean.setJobFactory(jobFactory());  
  
 // 配置Quartz属性  
 Properties quartzProperties = new Properties();  
 quartzProperties.put("org.quartz.scheduler.instanceName", "QuartzSchedulerDemo");  
 quartzProperties.put("org.quartz.scheduler.instanceId", "AUTO");  
 quartzProperties.put("org.quartz.threadPool.threadCount", "5");  
  
 schedulerFactoryBean.setQuartzProperties(quartzProperties);  
 schedulerFactoryBean.setAutoStartup(true);  
 schedulerFactoryBean.setStartupDelay(5); // 延迟5秒启动  
  
 return schedulerFactoryBean;  
 }  
  
 @Bean  
 public Scheduler scheduler() {  
 return schedulerFactoryBean().getScheduler();  
 }  
}

AutowiringSpringBeanJobFactory类：

package com.scheduler.config;  
  
import org.quartz.spi.TriggerFiredBundle;  
import org.springframework.beans.factory.config.AutowireCapableBeanFactory;  
import org.springframework.context.ApplicationContext;  
import org.springframework.scheduling.quartz.SpringBeanJobFactory;  
  
public class AutowiringSpringBeanJobFactory extends SpringBeanJobFactory {  
  
 private transient AutowireCapableBeanFactory beanFactory;  
  
 @Override  
 public void setApplicationContext(final ApplicationContext context) {  
 beanFactory = context.getAutowireCapableBeanFactory();  
 }  
  
 @Override  
 protected Object createJobInstance(final TriggerFiredBundle bundle) throws Exception {  
 final Object job = super.createJobInstance(bundle);  
 beanFactory.autowireBean(job);  
 return job;  
 }  
}

JobServiceImpl类：

package com.scheduler.service;  
  
import org.quartz.\*;  
import org.quartz.impl.matchers.GroupMatcher;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import com.scheduler.model.JobInfo;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Set;  
  
@Service  
public class JobServiceImpl implements JobService {  
  
 private static final Logger logger = LoggerFactory.getLogger(JobServiceImpl.class);  
  
 @Autowired  
 private Scheduler scheduler;  
  
 @Override  
 public List<JobDetail> getAllJobs() throws SchedulerException {  
 List<JobDetail> jobList = new ArrayList<>();  
  
 for (String groupName : scheduler.getJobGroupNames()) {  
 Set<JobKey> jobKeys = scheduler.getJobKeys(GroupMatcher.jobGroupEquals(groupName));  
 for (JobKey jobKey : jobKeys) {  
 JobDetail jobDetail = scheduler.getJobDetail(jobKey);  
 if (jobDetail != null) {  
 jobList.add(jobDetail);  
 }  
 }  
 }  
  
 return jobList;  
 }  
  
 @Override  
 public void pauseJob(String jobName, String groupName) throws SchedulerException {  
 JobKey jobKey = JobKey.jobKey(jobName, groupName);  
 if (scheduler.checkExists(jobKey)) {  
 scheduler.pauseJob(jobKey);  
 logger.info("任务已暂停: {}.{}", groupName, jobName);  
 } else {  
 throw new JobExecutionException("任务不存在: " + jobName);  
 }  
 }  
  
 @Override  
 public void resumeJob(String jobName, String groupName) throws SchedulerException {  
 JobKey jobKey = JobKey.jobKey(jobName, groupName);  
 if (scheduler.checkExists(jobKey)) {  
 scheduler.resumeJob(jobKey);  
 logger.info("任务已恢复: {}.{}", groupName, jobName);  
 } else {  
 throw new JobExecutionException("任务不存在: " + jobName);  
 }  
 }  
  
 @Override  
 public void deleteJob(String jobName, String groupName) throws SchedulerException {  
 JobKey jobKey = JobKey.jobKey(jobName, groupName);  
 if (scheduler.checkExists(jobKey)) {  
 scheduler.deleteJob(jobKey);  
 logger.info("任务已删除: {}.{}", groupName, jobName);  
 } else {  
 throw new JobExecutionException("任务不存在: " + jobName);  
 }  
 }  
  
 @Override  
 public void scheduleJob(JobDetail jobDetail, Trigger trigger) throws SchedulerException {  
 if (scheduler.checkExists(jobDetail.getKey())) {  
 throw new JobExecutionException("任务已存在: " + jobDetail.getKey());  
 }  
 scheduler.scheduleJob(jobDetail, trigger);  
 logger.info("任务已调度: {}.{}",  
 jobDetail.getKey().getGroup(), jobDetail.getKey().getName());  
 }  
 // 在JobServiceImpl实现类中添加此方法  
 @Override  
 public void addJob(JobInfo jobInfo) throws SchedulerException {  
 try {  
 // 验证任务信息  
 if (jobInfo == null) {  
 throw new IllegalArgumentException("JobInfo cannot be null");  
 }  
  
 // 获取JobClass  
 Class<?> jobClass;  
 try {  
 jobClass = Class.*forName*(jobInfo.getJobClass());  
 } catch (ClassNotFoundException e) {  
 throw new JobExecutionException("Job class not found: " + jobInfo.getJobClass(), e);  
 }  
  
 // 创建JobDetail  
 JobBuilder jobBuilder = JobBuilder.*newJob*((Class<? extends Job>) jobClass)  
 .withIdentity(jobInfo.getJobName(), jobInfo.getJobGroup())  
 .withDescription(jobInfo.getDescription());  
  
 // 添加JobData  
 if (jobInfo.getJobData() != null && !jobInfo.getJobData().isEmpty()) {  
 JobDataMap jobDataMap = new JobDataMap(jobInfo.getJobData());  
 jobBuilder.usingJobData(jobDataMap);  
 }  
  
 JobDetail jobDetail = jobBuilder.build();  
  
 // 创建触发器  
 Trigger trigger;  
 if ("CRON".equals(jobInfo.getScheduleType())) {  
 // CRON表达式触发器  
 trigger = TriggerBuilder.*newTrigger*()  
 .withIdentity(jobInfo.getJobName() + "Trigger", jobInfo.getJobGroup())  
 .withSchedule(CronScheduleBuilder.*cronSchedule*(jobInfo.getCronExpression()))  
 .build();  
 } else {  
 // 默认简单触发器  
 trigger = TriggerBuilder.newTrigger()  
 .withIdentity(jobInfo.getJobName() + "Trigger", jobInfo.getJobGroup())  
 .startNow()  
 .withSchedule(SimpleScheduleBuilder.simpleSchedule()  
 .withIntervalInSeconds(60) // 默认60秒  
 .repeatForever())  
 .build();  
 }  
  
 // 检查任务是否已存在，如果存在则更新  
 if (scheduler.checkExists(jobDetail.getKey())) {  
 scheduler.deleteJob(jobDetail.getKey());  
 logger.info("Existing job deleted: {}.{}", jobInfo.getJobGroup(), jobInfo.getJobName());  
 }  
  
 // 调度任务  
 scheduler.scheduleJob(jobDetail, trigger);  
 logger.info("Job scheduled: {}.{} with cron: {}",  
 jobInfo.getJobGroup(), jobInfo.getJobName(), jobInfo.getCronExpression());  
 } catch (Exception e) {  
 logger.error("Error scheduling job: ", e);  
 throw new SchedulerException("Failed to schedule job: " + e.getMessage(), e);  
 }  
 }  
}

BaseJob类：

package com.scheduler.job;  
  
import lombok.extern.slf4j.Slf4j;  
import org.quartz.\*;  
import org.springframework.context.ApplicationContext;  
  
@Slf4j  
public abstract class BaseJob implements Job {  
  
 @Override  
 public void execute(JobExecutionContext context) throws JobExecutionException {  
 try {  
 JobKey jobKey = context.getJobDetail().getKey();  
 *log*.info("Job {} in group {} is executing at: {}",  
 jobKey.getName(), jobKey.getGroup(), new java.util.Date());  
  
 // 获取Job数据  
 JobDataMap dataMap = context.getMergedJobDataMap();  
  
 // 执行具体任务逻辑  
 executeJob(context, dataMap);  
  
 *log*.info("Job {} in group {} completed successfully",  
 jobKey.getName(), jobKey.getGroup());  
 } catch (Exception e) {  
 *log*.error("Error executing job: ", e);  
 JobExecutionException jobException = new JobExecutionException(e);  
 // 设置重新执行策略  
 jobException.setRefireImmediately(false);  
 throw jobException;  
 }  
 }  
  
 // 抽象方法，具体的作业逻辑由子类实现  
 protected abstract void executeJob(JobExecutionContext context, JobDataMap dataMap) throws Exception;  
  
 // 从Spring上下文获取Bean的通用方法  
 protected <T> T getBean(JobExecutionContext context, Class<T> beanClass) {  
 try {  
 ApplicationContext appContext = (ApplicationContext) context.getScheduler()  
 .getContext().get("applicationContext");  
 return appContext.getBean(beanClass);  
 } catch (SchedulerException e) {  
 *log*.error("Error getting application context: ", e);  
 throw new RuntimeException("Failed to get bean from context", e);  
 }  
 }  
}

LoggingJob：

package com.scheduler.job;  
  
import org.quartz.JobDataMap;  
import org.quartz.JobExecutionContext;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Component;  
  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
  
@Component  
public class LoggingJob extends BaseJob {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(LoggingJob.class);  
 private static final DateTimeFormatter *formatter* = DateTimeFormatter.*ofPattern*("yyyy-MM-dd HH:mm:ss");  
  
 @Override  
 protected void executeJob(JobExecutionContext context, JobDataMap dataMap) throws Exception {  
 // 实现具体的任务逻辑  
 String jobName = context.getJobDetail().getKey().getName();  
 String jobGroup = context.getJobDetail().getKey().getGroup();  
 LocalDateTime now = LocalDateTime.*now*();  
  
 *logger*.info("=== LoggingJob 执行 ===");  
 *logger*.info("任务名称: {}", jobName);  
 *logger*.info("任务组: {}", jobGroup);  
 *logger*.info("执行时间: {}", now.format(*formatter*));  
 *logger*.info("下次执行时间: {}",  
 context.getNextFireTime() != null  
 ? context.getNextFireTime().toString()  
 : "无");  
  
 // 如果有数据传入，打印数据  
 if (dataMap != null && !dataMap.isEmpty()) {  
 *logger*.info("任务数据:");  
 for (String key : dataMap.getKeys()) {  
 *logger*.info(" {} = {}", key, dataMap.get(key));  
 }  
 }  
  
 *logger*.info("=== LoggingJob 完成 ===");  
  
 // 也打印到控制台，以便更明显地看到任务执行  
 System.*out*.println("\n=== LoggingJob 执行于 " + now.format(*formatter*) + " ===");  
 }  
}

CronUtils类：

package com.scheduler.utils;  
  
import org.quartz.CronExpression;  
import org.springframework.stereotype.Component;  
  
import java.text.ParseException;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.List;  
  
@Component  
public class CronUtils {  
  
 */\*\*  
 \* 验证Cron表达式是否有效  
 \*/* public static boolean isValidCronExpression(String cronExpression) {  
 try {  
 CronExpression.*validateExpression*(cronExpression);  
 return true;  
 } catch (ParseException e) {  
 return false;  
 }  
 }  
  
 */\*\*  
 \* 计算未来N次的触发时间  
 \*/* public static List<Date> getNextFireTimes(String cronExpression, int numTimes) throws ParseException {  
 if (!*isValidCronExpression*(cronExpression)) {  
 throw new ParseException("Invalid cron expression: " + cronExpression, 0);  
 }  
  
 CronExpression cron = new CronExpression(cronExpression);  
 List<Date> dates = new ArrayList<>();  
  
 Date nextDate = new Date();  
 for (int i = 0; i < numTimes; i++) {  
 nextDate = cron.getNextValidTimeAfter(nextDate);  
 if (nextDate != null) {  
 dates.add(nextDate);  
 } else {  
 break;  
 }  
 }  
  
 return dates;  
 }  
  
 */\*\*  
 \* 获取通用的Cron表达式示例  
 \*/* public static String getCommonCronExpressions(String type) {  
 switch (type.toLowerCase()) {  
 case "every\_minute":  
 return "0 \* \* \* \* ?";  
 case "every\_5\_minutes":  
 return "0 \*/5 \* \* \* ?";  
 case "every\_hour":  
 return "0 0 \* \* \* ?";  
 case "every\_day\_midnight":  
 return "0 0 0 \* \* ?";  
 case "every\_day\_noon":  
 return "0 0 12 \* \* ?";  
 case "every\_monday":  
 return "0 0 0 ? \* MON";  
 case "every\_weekday":  
 return "0 0 0 ? \* MON-FRI";  
 case "every\_month\_first\_day":  
 return "0 0 0 1 \* ?";  
 default:  
 return null;  
 }  
 }  
  
 */\*\*  
 \* 生成描述性的Cron表达式说明  
 \*/* public static String describeCronExpression(String cronExpression) {  
 try {  
 if (!*isValidCronExpression*(cronExpression)) {  
 return "Invalid cron expression";  
 }  
  
 // 简化的描述生成逻辑，实际应用中可以更复杂  
 if (cronExpression.equals("0 \* \* \* \* ?")) {  
 return "Triggers every minute";  
 } else if (cronExpression.equals("0 \*/5 \* \* \* ?")) {  
 return "Triggers every 5 minutes";  
 } else if (cronExpression.equals("0 0 \* \* \* ?")) {  
 return "Triggers every hour at the top of the hour";  
 } else if (cronExpression.equals("0 0 0 \* \* ?")) {  
 return "Triggers every day at midnight";  
 } else if (cronExpression.equals("0 0 12 \* \* ?")) {  
 return "Triggers every day at noon";  
 } else if (cronExpression.equals("0 0 0 ? \* MON")) {  
 return "Triggers every Monday at midnight";  
 } else {  
 List<Date> nextDates = *getNextFireTimes*(cronExpression, 3);  
 StringBuilder sb = new StringBuilder("Next 3 executions: ");  
 for (int i = 0; i < nextDates.size(); i++) {  
 if (i > 0) {  
 sb.append(", ");  
 }  
 sb.append(nextDates.get(i));  
 }  
 return sb.toString();  
 }  
 } catch (Exception e) {  
 return "Error describing cron expression: " + e.getMessage();  
 }  
 }  
}

JobUtils类：

package com.scheduler.utils;  
  
import org.quartz.\*;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.List;  
import java.util.Map;  
  
@Component  
public class JobUtils {  
  
 @Autowired  
 private Scheduler scheduler;  
  
 */\*\*  
 \* 检查作业是否存在  
 \*/* public boolean jobExists(String jobName, String jobGroup) throws SchedulerException {  
 return scheduler.checkExists(new JobKey(jobName, jobGroup));  
 }  
  
 */\*\*  
 \* 检查触发器是否存在  
 \*/* public boolean triggerExists(String triggerName, String triggerGroup) throws SchedulerException {  
 return scheduler.checkExists(new TriggerKey(triggerName, triggerGroup));  
 }  
  
 */\*\*  
 \* 获取作业状态  
 \*/* public String getJobStatus(String jobName, String jobGroup) throws SchedulerException {  
 JobKey jobKey = new JobKey(jobName, jobGroup);  
 if (!scheduler.checkExists(jobKey)) {  
 return "NOT\_FOUND";  
 }  
  
 List<? extends Trigger> triggers = scheduler.getTriggersOfJob(jobKey);  
 if (triggers != null && !triggers.isEmpty()) {  
 Trigger.TriggerState state = scheduler.getTriggerState(triggers.get(0).getKey());  
 return state.name();  
 }  
  
 return "UNKNOWN";  
 }  
  
 */\*\*  
 \* 获取作业执行次数  
 \*/* public int getJobExecutionCount(String jobName, String jobGroup) throws SchedulerException {  
 JobKey jobKey = new JobKey(jobName, jobGroup);  
 if (!scheduler.checkExists(jobKey)) {  
 return 0;  
 }  
  
 JobDetail jobDetail = scheduler.getJobDetail(jobKey);  
 JobDataMap dataMap = jobDetail.getJobDataMap();  
  
 if (dataMap.containsKey("totalRuns")) {  
 return dataMap.getInt("totalRuns");  
 }  
  
 return 0;  
 }  
  
 */\*\*  
 \* 获取作业下次执行时间  
 \*/* public Date getNextFireTime(String jobName, String jobGroup) throws SchedulerException {  
 JobKey jobKey = new JobKey(jobName, jobGroup);  
 if (!scheduler.checkExists(jobKey)) {  
 return null;  
 }  
  
 List<? extends Trigger> triggers = scheduler.getTriggersOfJob(jobKey);  
 if (triggers != null && !triggers.isEmpty()) {  
 return triggers.get(0).getNextFireTime();  
 }  
  
 return null;  
 }  
  
 */\*\*  
 \* 获取未来几次执行时间  
 \*/* public List<Date> getNextFireTimes(String jobName, String jobGroup, int count) throws SchedulerException {  
 List<Date> nextFireTimes = new ArrayList<>();  
  
 JobKey jobKey = new JobKey(jobName, jobGroup);  
 if (!scheduler.checkExists(jobKey)) {  
 return nextFireTimes;  
 }  
  
 List<? extends Trigger> triggers = scheduler.getTriggersOfJob(jobKey);  
 if (triggers == null || triggers.isEmpty()) {  
 return nextFireTimes;  
 }  
  
 Trigger trigger = triggers.get(0);  
 Date nextFireTime = trigger.getNextFireTime();  
  
 // 对于CronTrigger，可以预测未来的执行时间  
 if (trigger instanceof CronTrigger) {  
 CronTrigger cronTrigger = (CronTrigger) trigger;  
 Date currentTime = nextFireTime != null ? nextFireTime : new Date();  
  
 for (int i = 0; i < count; i++) {  
 currentTime = cronTrigger.getFireTimeAfter(currentTime);  
 if (currentTime != null) {  
 nextFireTimes.add(currentTime);  
 } else {  
 break;  
 }  
 }  
 }  
 // 对于SimpleTrigger，根据重复次数和间隔计算  
 else if (trigger instanceof SimpleTrigger) {  
 SimpleTrigger simpleTrigger = (SimpleTrigger) trigger;  
 Date currentTime = nextFireTime;  
  
 // 只有当下次执行时间不为空，且当前列表长度小于请求数量时才继续  
 while (currentTime != null && nextFireTimes.size() < count) {  
 nextFireTimes.add(currentTime);  
  
 long interval = simpleTrigger.getRepeatInterval();  
 int remainingCount = simpleTrigger.getTimesTriggered();  
 int maxCount = simpleTrigger.getRepeatCount();  
  
 // 检查是否已达到最大重复次数  
 if (maxCount != SimpleTrigger.REPEAT\_INDEFINITELY && remainingCount >= maxCount) {  
 break;  
 }  
  
 // 计算下一次执行时间  
 currentTime = new Date(currentTime.getTime() + interval);  
 }  
 }  
  
 return nextFireTimes;  
 }  
  
 */\*\*  
 \* 创建作业执行历史记录（可用于监控）  
 \*/* public void recordJobExecution(String jobName, String jobGroup, boolean success, String message) {  
 try {  
 JobKey jobKey = new JobKey(jobName, jobGroup);  
 if (!scheduler.checkExists(jobKey)) {  
 return;  
 }  
  
 JobDetail jobDetail = scheduler.getJobDetail(jobKey);  
 JobDataMap dataMap = jobDetail.getJobDataMap();  
  
 // 更新总执行次数  
 int totalRuns = dataMap.containsKey("totalRuns") ? dataMap.getInt("totalRuns") + 1 : 1;  
 dataMap.put("totalRuns", totalRuns);  
  
 // 更新成功次数  
 if (success) {  
 int successCount = dataMap.containsKey("successCount") ? dataMap.getInt("successCount") + 1 : 1;  
 dataMap.put("successCount", successCount);  
 } else {  
 int failureCount = dataMap.containsKey("failureCount") ? dataMap.getInt("failureCount") + 1 : 1;  
 dataMap.put("failureCount", failureCount);  
 // 记录最后一次失败信息  
 dataMap.put("lastFailureMessage", message);  
 dataMap.put("lastFailureTime", new Date());  
 }  
  
 // 更新最后执行时间  
 dataMap.put("lastExecutionTime", new Date());  
  
 // 更新作业数据  
 JobBuilder jobBuilder = jobDetail.getJobBuilder();  
 JobDetail updatedJobDetail = jobBuilder.usingJobData(dataMap).build();  
 scheduler.addJob(updatedJobDetail, true);  
  
 } catch (SchedulerException e) {  
 // 只记录日志，不抛出异常，以避免影响正常的作业执行  
 System.err.println("Error recording job execution: " + e.getMessage());  
 }  
 }  
  
 */\*\*  
 \* 生成唯一的作业名称  
 \*/* public String generateUniqueJobName(String prefix) {  
 return prefix + "\_" + System.currentTimeMillis();  
 }  
  
 */\*\*  
 \* 从JobDataMap中提取并返回特定数据  
 \*/* public Map<String, Object> extractJobData(String jobName, String jobGroup) throws SchedulerException {  
 JobKey jobKey = new JobKey(jobName, jobGroup);  
 if (!scheduler.checkExists(jobKey)) {  
 return null;  
 }  
  
 JobDetail jobDetail = scheduler.getJobDetail(jobKey);  
 JobDataMap dataMap = jobDetail.getJobDataMap();  
  
 return dataMap.getWrappedMap();  
 }  
  
 */\*\*  
 \* 判断作业是否正在运行  
 \*/* public boolean isJobRunning(String jobName, String jobGroup) throws SchedulerException {  
 List<JobExecutionContext> currentJobs = scheduler.getCurrentlyExecutingJobs();  
 JobKey jobKey = new JobKey(jobName, jobGroup);  
  
 for (JobExecutionContext context : currentJobs) {  
 if (context.getJobDetail().getKey().equals(jobKey)) {  
 return true;  
 }  
 }  
  
 return false;  
 }  
  
 */\*\*  
 \* 获取当前正在运行的所有作业  
 \*/* public List<String> getCurrentlyRunningJobs() throws SchedulerException {  
 List<String> runningJobs = new ArrayList<>();  
 List<JobExecutionContext> currentJobs = scheduler.getCurrentlyExecutingJobs();  
  
 for (JobExecutionContext context : currentJobs) {  
 JobKey jobKey = context.getJobDetail().getKey();  
 runningJobs.add(jobKey.getGroup() + "." + jobKey.getName());  
 }  
  
 return runningJobs;  
 }  
}

JobManagementRunner类

package com.scheduler.runner;  
  
import com.scheduler.job.LoggingJob;  
import com.scheduler.service.JobService;  
import org.quartz.\*;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.ApplicationArguments;  
import org.springframework.boot.ApplicationRunner;  
import org.springframework.stereotype.Component;  
  
import java.util.List;  
import java.util.Scanner;  
import java.util.concurrent.TimeUnit;  
  
@Component  
public class JobManagementRunner implements ApplicationRunner {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(JobManagementRunner.class);  
  
 @Autowired  
 private JobService jobService;  
  
 @Autowired  
 private Scheduler scheduler;  
  
 @Override  
 public void run(ApplicationArguments args) throws Exception {  
 *logger*.info("Quartz Scheduler 状态: Started={}, Standby={}, Shutdown={}",  
 scheduler.isStarted(), scheduler.isInStandbyMode(), scheduler.isShutdown());  
  
 // 启动调度器  
 if (!scheduler.isStarted()) {  
 scheduler.start();  
 *logger*.info("Quartz Scheduler 已启动！");  
 }  
  
 // 添加默认的日志任务 - 每分钟执行一次  
 scheduleLoggingJob();  
  
 // 等待65秒后显示任务管理菜单  
 *logger*.info("Waiting for job to execute at least once...");  
 try {  
 TimeUnit.*SECONDS*.sleep(65);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
  
 // 显示任务管理菜单  
 showJobManagementMenu();  
 }  
  
 private void scheduleLoggingJob() {  
 try {  
 // 检查任务是否已存在  
 JobKey jobKey = new JobKey("loggingJob", "DEFAULT");  
 if (scheduler.checkExists(jobKey)) {  
 *logger*.info("LoggingJob 已存在，跳过创建");  
 return;  
 }  
  
 // 创建JobDetail  
 JobDetail jobDetail = JobBuilder.*newJob*(LoggingJob.class)  
 .withIdentity(jobKey)  
 .withDescription("每分钟执行一次的日志任务")  
 .build();  
  
 // 创建Trigger - 每分钟执行一次  
 Trigger trigger = TriggerBuilder.*newTrigger*()  
 .withIdentity("loggingTrigger", "DEFAULT")  
 .withSchedule(SimpleScheduleBuilder.*simpleSchedule*()  
 .withIntervalInMinutes(1)  
 .repeatForever())  
 .startNow()  
 .build();  
  
 // 调度任务  
 scheduler.scheduleJob(jobDetail, trigger);  
 *logger*.info("成功调度 LoggingJob，将每分钟执行一次");  
 } catch (Exception e) {  
 *logger*.error("调度 LoggingJob 时出错", e);  
 }  
 }  
  
 private void showJobManagementMenu() {  
 Scanner scanner = new Scanner(System.*in*);  
 boolean exit = false;  
  
 // 在新线程中运行菜单，以避免阻塞主线程  
 Thread menuThread = new Thread(() -> {  
 while (!exit) {  
 try {  
 System.*out*.println("\n===== 任务管理菜单 =====");  
 System.*out*.println("1. 查看所有任务");  
 System.*out*.println("2. 暂停任务");  
 System.*out*.println("3. 恢复任务");  
 System.*out*.println("4. 删除任务");  
 System.*out*.println("5. 退出");  
 System.*out*.print("请选择操作 (1-5): ");  
  
 int choice = scanner.nextInt();  
 scanner.nextLine(); // 消费换行符  
  
 switch (choice) {  
 case 1:  
 listAllJobs();  
 break;  
 case 2:  
 pauseJob(scanner);  
 break;  
 case 3:  
 resumeJob(scanner);  
 break;  
 case 4:  
 deleteJob(scanner);  
 break;  
 case 5:  
 return; // 退出线程  
 default:  
 System.*out*.println("无效选择，请重试");  
 }  
 } catch (Exception e) {  
 *logger*.error("处理菜单选择时出错", e);  
 System.*out*.println("发生错误: " + e.getMessage());  
 scanner.nextLine(); // 清除错误输入  
 }  
 }  
 });  
  
 menuThread.setDaemon(true); // 设为守护线程，以便应用关闭时自动终止  
 menuThread.start();  
 }  
  
 private void listAllJobs() {  
 try {  
 System.*out*.println("\n当前所有任务:");  
 List<JobDetail> jobs = jobService.getAllJobs();  
  
 if (jobs.isEmpty()) {  
 System.*out*.println("没有找到任务");  
 return;  
 }  
  
 for (JobDetail job : jobs) {  
 JobKey jobKey = job.getKey();  
 Trigger.TriggerState state = scheduler.getTriggerState(  
 TriggerKey.*triggerKey*(jobKey.getName(), jobKey.getGroup()));  
  
 System.*out*.printf("Job: %s, Group: %s, Description: %s, State: %s%n",  
 jobKey.getName(), jobKey.getGroup(),  
 job.getDescription(), state);  
 }  
 } catch (Exception e) {  
 *logger*.error("获取任务列表时出错", e);  
 System.*out*.println("获取任务列表失败: " + e.getMessage());  
 }  
 }  
  
 private void pauseJob(Scanner scanner) {  
 try {  
 System.*out*.print("输入要暂停的任务名称: ");  
 String jobName = scanner.nextLine();  
  
 System.*out*.print("输入任务组名 (默认为 DEFAULT): ");  
 String groupName = scanner.nextLine();  
 if (groupName.trim().isEmpty()) {  
 groupName = "DEFAULT";  
 }  
  
 jobService.pauseJob(jobName, groupName);  
 System.*out*.println("任务已暂停");  
 } catch (Exception e) {  
 *logger*.error("暂停任务时出错", e);  
 System.*out*.println("暂停任务失败: " + e.getMessage());  
 }  
 }  
  
 private void resumeJob(Scanner scanner) {  
 try {  
 System.*out*.print("输入要恢复的任务名称: ");  
 String jobName = scanner.nextLine();  
  
 System.*out*.print("输入任务组名 (默认为 DEFAULT): ");  
 String groupName = scanner.nextLine();  
 if (groupName.trim().isEmpty()) {  
 groupName = "DEFAULT";  
 }  
  
 jobService.resumeJob(jobName, groupName);  
 System.*out*.println("任务已恢复");  
 } catch (Exception e) {  
 *logger*.error("恢复任务时出错", e);  
 System.*out*.println("恢复任务失败: " + e.getMessage());  
 }  
 }  
  
 private void deleteJob(Scanner scanner) {  
 try {  
 System.*out*.print("输入要删除的任务名称: ");  
 String jobName = scanner.nextLine();  
  
 System.*out*.print("输入任务组名 (默认为 DEFAULT): ");  
 String groupName = scanner.nextLine();  
 if (groupName.trim().isEmpty()) {  
 groupName = "DEFAULT";  
 }  
  
 jobService.deleteJob(jobName, groupName);  
 System.*out*.println("任务已删除");  
 } catch (Exception e) {  
 *logger*.error("删除任务时出错", e);  
 System.*out*.println("删除任务失败: " + e.getMessage());  
 }  
 }  
}

jobinfo：

package com.scheduler.model;  
  
import java.util.Map;  
  
public class JobInfo {  
 private String jobName;  
 private String jobGroup;  
 private String cronExpression;  
 private String description;  
 private String jobClass;  
 private String scheduleType; // CRON, SIMPLE, etc.  
 private Map<String, Object> jobData;  
  
 // Getters and Setters  
 public String getJobName() {  
 return jobName;  
 }  
  
 public void setJobName(String jobName) {  
 this.jobName = jobName;  
 }  
  
 public String getJobGroup() {  
 return jobGroup;  
 }  
  
 public void setJobGroup(String jobGroup) {  
 this.jobGroup = jobGroup;  
 }  
  
 public String getCronExpression() {  
 return cronExpression;  
 }  
  
 public void setCronExpression(String cronExpression) {  
 this.cronExpression = cronExpression;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
 public String getJobClass() {  
 return jobClass;  
 }  
  
 public void setJobClass(String jobClass) {  
 this.jobClass = jobClass;  
 }  
  
 public String getScheduleType() {  
 return scheduleType;  
 }  
  
 public void setScheduleType(String scheduleType) {  
 this.scheduleType = scheduleType;  
 }  
  
 public Map<String, Object> getJobData() {  
 return jobData;  
 }  
  
 public void setJobData(Map<String, Object> jobData) {  
 this.jobData = jobData;  
 }  
  
 @Override  
 public String toString() {  
 return "JobInfo{" +  
 "jobName='" + jobName + '\'' +  
 ", jobGroup='" + jobGroup + '\'' +  
 ", cronExpression='" + cronExpression + '\'' +  
 ", description='" + description + '\'' +  
 ", jobClass='" + jobClass + '\'' +  
 ", scheduleType='" + scheduleType + '\'' +  
 ", jobData=" + jobData +  
 '}';  
 }  
}

SchedulerApplication

package com.scheduler;  
  
import com.scheduler.job.LoggingJob;  
import com.scheduler.model.JobInfo;  
import com.scheduler.service.JobService;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;  
import org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration;  
import org.springframework.context.annotation.Bean;  
  
import java.util.HashMap;  
import java.util.Map;  
  
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class, HibernateJpaAutoConfiguration.class})  
public class SchedulerApplication {  
  
 // 使用构造器注入代替字段注入  
 private final JobService jobService;  
  
 // 构造器注入  
 public SchedulerApplication(JobService jobService) {  
 this.jobService = jobService;  
 }  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(SchedulerApplication.class, args);  
 }  
  
 */\*\*  
 \* 应用启动后执行，创建并调度一个示例任务  
 \*/* @Bean  
 public CommandLineRunner schedulingRunner() {  
 return args -> {  
 System.*out*.println("Creating and scheduling a sample logging job...");  
  
 // 创建任务信息  
 JobInfo jobInfo = new JobInfo();  
 jobInfo.setJobName("sampleLoggingJob");  
 jobInfo.setJobGroup("logGroup");  
 jobInfo.setCronExpression("0 \* \* \* \* ?"); // 每分钟执行一次  
 jobInfo.setDescription("A sample job that logs current time");  
 jobInfo.setJobClass(LoggingJob.class.getName());  
 jobInfo.setScheduleType("CRON"); // 使用CRON触发器  
  
 // 添加任务参数  
 Map<String, Object> jobData = new HashMap<>();  
 jobData.put("taskId", "TASK-" + System.*currentTimeMillis*());  
 jobInfo.setJobData(jobData);  
  
 try {  
 // 添加并调度任务  
 jobService.addJob(jobInfo);  
 System.*out*.println("Job scheduled successfully! It will run every minute.");  
 System.*out*.println("Job details: " + jobInfo);  
 } catch (Exception e) {  
 System.*err*.println("Failed to schedule job: " + e.getMessage());  
 e.printStackTrace();  
 }  
 };  
 }  
}