SpringBoot 结合 Quartz 的数据库定时任务

作者：someone9891
原文链接：https://ld246.com/article/1561629454128
来源网站：链滴

前言

由于工作ETL工具需要一个定时任务调度系统，鉴于本人以前只开发过简单的定时任务，并且都是在置文件配置好的，而本次需要一个可前端维护、需要失败重试等灵活的定时任务调度，于是查询了好资料、求助了社区的各位大佬后关于调度任务的问,基本有了一点思路，按照自己的思路进行了简单的践，现将实践结果分享出来。

技术选型

- Springboot
- Quartz
- spring-data-jpa
- mysql 数据库

本次实践是为了给工作内容先出个demo，所以选取了开发速度较快的Spring Boot框架

开始行动

配置

```
package org.gitors.jobdemo.utils;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.quartz.SchedulerFactoryBean;

import javax.sql.DataSource;
```

```
import java.util.Properties;

/**
 * @author : liuwenlong
 * @desc :
 * @date : 2019-06-26 15:59
 */
@Configuration()
public class QuertzConfig {
    @Bean
    public SchedulerFactoryBean schedulerFactoryBean(DataSource dataSource) {
        SchedulerFactoryBean factory = new SchedulerFactoryBean();
        factory.setDataSource(dataSource);

        //quartz参数
        Properties prop = new Properties();
        prop.put("org.quartz.scheduler.instanceName", "DemoScheduler");
        prop.put("org.quartz.scheduler.instanceId", "AUTO");
        //线程池配置
        prop.put("org.quartz.threadPool.class", "org.quartz.simpl.SimpleThreadPool");
        prop.put("org.quartz.threadPool.threadCount", "20");
        prop.put("org.quartz.threadPool.threadPriority", "5");
        //JobStore配置
        prop.put("org.quartz.jobStore.class", "org.quartz.impl.jdbcjobstore.JobStoreTX");
        //集群配置
        prop.put("org.quartz.jobStore.isClustered", "false");
        prop.put("org.quartz.jobStore.clusterCheckinInterval", "15000");
        prop.put("org.quartz.jobStore.maxMisfiresToHandleAtATime", "1");

        prop.put("org.quartz.jobStore.misfireThreshold", "12000");
        prop.put("org.quartz.jobStore.tablePrefix", "QRTZ_");
        prop.put("org.quartz.jobStore.selectWithLockSQL", "SELECT * FROM {0}LOCKS UPDLOCK
WHERE LOCK_NAME = ?");

        //PostgreSQL数据库，需要打开此注释
        //prop.put("org.quartz.jobStore.driverDelegateClass", "org.quartz.impl.jdbcjobstore.Post
reSQLDelegate");

        factory.setQuartzProperties(prop);

        factory.setSchedulerName("DemoScheduler");
        //延时启动
        factory.setStartupDelay(5);
        factory.setApplicationContextSchedulerContextKey("applicationContextKey");
        //可选，QuartzScheduler 启动时更新己存在的Job，这样就不用每次修改targetObject后删除q
tz_job_details表对应记录了
        factory.setOverwriteExistingJobs(false);
        //设置自动启动，默认为true
        factory.setAutoStartup(true);

        return factory;
    }
}
```

对于Quartz 的配置，这里采用了代码的方式，也可以通过使用 配置文件的方式来进行配置，这里就多说了，如果需要，可自行了解。

初始化任务

初始化任务在在项目启动的时候，通过查询数据库配置好的任务，然后添加到任务队列中的方式来实的，具体代码如下：

```
    @PostConstruct
    public void init(){
        logger.info("--------初始化定时任务----------");
        List<JobEntity> jobEntities = jobDao.findAll();
        logger.info("---------正在初始化"+jobEntities.size()+"个定时任务----------");
        jobEntities.forEach(jobEntity -> {
            //初始化定时任务，如果已经在quartz中，则不重复初始化，如果不在quartz，则进行初始化
            CronTrigger cronTrigger = Jobutils.getCronTrigger(scheduler,jobEntity.getId());
            if (cronTrigger == null){
                logger.info("初始化任务ID：" + jobEntity.getId());
                Jobutils.createScheduleJob(scheduler,jobEntity);
            }
        });
        logger.info("----------定时任务初始化成功----------");
    }
```

失败重试

业务需要在任务中配置失败是否重试、重试次数、重试间隔，然后在任务执行失败时进行重试，知道功或者 重试次数耗尽，则本周期任务结束。

重试是在失败以后进行的，代码如下：在失败以后判断重试次数等，然后重新添加新任务。

```
package org.gitors.jobdemo.utils;


import org.gitors.jobdemo.entity.JobEntity;
import org.gitors.jobdemo.entity.JobLogEntity;
import org.gitors.jobdemo.service.JobLogService;
import org.gitors.jobdemo.service.JobService;
import org.quartz.JobExecutionContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.scheduling.quartz.QuartzJobBean;

import java.util.Date;
import java.util.UUID;


/**
 * 通用任务
 * @author liuwenlong
 */
public class ScheduleJob extends QuartzJobBean {
```

```
    private Logger logger = LoggerFactory.getLogger(getClass());

    private static JobService jobService;

    private static JobLogService jobLogService;

    /**
     * 获取JobService 对象
     * @return JobService 对象
     */
    private JobService getJobService(){
        if (jobService == null){
            jobService = (JobService) SpringContextUtil.getBean("jobServiceImpl");
        }
        return jobService;
    }

    /**
     * 获取日志service bean
     * @return 日志service
     */
    private JobLogService getJobLogService(){
        if (jobLogService == null){
            jobLogService = (JobLogService) SpringContextUtil.getBean("jobLogServiceImpl");
        }
        return jobLogService;
    }


    @Override
    protected void executeInternal(JobExecutionContext context) {
        JobEntity scheduleJob = (JobEntity) context.getMergedJobDataMap()
                .get(JobEntity.JOB_PARAM_KEY);

        //任务开始时间
        long startTime = System.currentTimeMillis();
        JobLogEntity jobLogEntity = new JobLogEntity();
        jobLogEntity.setId(UUID.randomUUID().toString().replace("-",""));

        try {
            //执行任务
            logger.info("任务准备执行，任务ID：" + scheduleJob.getJobName());

            JobEntity jobEntity = this.getJobService().findById(scheduleJob.getId());
            jobLogEntity.setRanTime(new Date(startTime));
            if (jobEntity == null){
                //删除quartz 队列中的 任务
                this.getJobService().deleteJob(scheduleJob.getId());
                throw new RuntimeException("任务已经被删除");
            }
            //任务执行总时长
            long times = System.currentTimeMillis() - startTime;
            logger.info("任务执行完毕，任务ID：" + scheduleJob.getJobName() + "  总共耗时：" + t
mes + "毫秒");
```

```
            jobLogEntity.setRanExpend(times);
            jobLogEntity.setStatus(0);
            jobLogEntity.setJobId(scheduleJob.getId());
            jobLogEntity.setRanNum(scheduleJob.getRetriedNum());
        } catch (Exception e) {
            logger.error("任务执行失败，任务ID：" + scheduleJob.getJobName(), e);
            jobLogEntity.setStatus(1);
            jobLogEntity.setMsg(e.getMessage());
            //需要重试
            if (scheduleJob.getRetry() == 1 && scheduleJob.getRestryNum() > scheduleJob.getRe
riedNum()){
                logger.error("任务准备在" + scheduleJob.getRetryInterval() + "秒后重试。任务名:"+ s
heduleJob.getJobName());
                this.getJobService().reTry(scheduleJob);
            }
        }finally {
            this.getJobLogService().save(jobLogEntity);
        }
    }
}
```

在写本篇文章的时候，发现失败重试的逻辑存在错误

```
    @Override
    @Transactional(rollbackOn = Exception.class)
    public void reTry(JobEntity jobEntity) {
        if (jobEntity.getRetriedNum() < jobEntity.getRestryNum()){
            jobEntity.setRetriedNum(jobEntity.getRetriedNum() + 1);
        }
        jobDao.save(jobEntity);
//        Jobutils.deleteScheduleJob(scheduler,jobEntity.getId());
        Jobutils.retry(scheduler,jobEntity);
    }
```

我在重试的方法里面 删除里Quartz 里的对应任务，那么到下个周期就不会执行了，现在已经修改，管做什么实践，完了还是回顾一下的好啊。

如果已经触发任务、但是数据删除了本条任务

在本demo里面的解决方法是，在执行的时候，会先从数据库查询是否存在这条任务，不存在则不执，并且从quartz中移除该任务。

当然，还有其他的方式，没有一一实现，有兴趣的朋友可以自己改改代码。

写在最后

就目前为止，本demo 还是有很多的问题，由于需要马上开展工作，具体问题我在工作代码中去解决这里以后如果有空，我会再完善~~大概率就这样了~~，

关于管理页面，我写了后台接口，但是页面只写了查询任务列表和 日志列表的，最近比较忙，懒得去了

代码

demo 代码 已经放到Github, 有兴趣的可以去看看。

https://github.com/gitors/jobdemo