quartz是一种调度器，可以让程序在指定时间执行，也可以按照某一个频率来执行，

Quartz 与 Spring 集成

创建一个简单的Job

```
... extends QuartzJobBean{

    @Override
    protected void executeInternal(JobExecutionContext context)
            throws JobExecutionException {
        System.out.println("Myjob 执行了....."+context.getTrigger().getKey().
        ApplicationContext applicationContext = (ApplicationContext)context
                .getJobDetail().getJobDataMap().get("adc");
        System.out.println(applicationContext);

        System.out.println("获取的Spring容器是: " + applicationContext);
        System.out.println("当前时间:"+new Date().toString());
    }
```

配置spring

```
    <bean name="myjobDetail" class="org.springframework.scheduling.quartz.JobDetail
        <!-- 指定具体的job类 -->
        <property name="jobClass" value="com.lich.spring.MyJob" />
        <!-- 指定job的名称 -->
        <property name="name" value="myJob" />
        <!-- 指定job的分组 -->
        <property name="group" value="jobs" />
        <!-- 必须设置为true，如果为false，当没有活动的触发器与之关联时会在调度器中删除该
        <property name="durability" value="true"/>
        <!-- 指定spring容器的key，如果不设定在job中的jobmap中是获取不到spring容器的 -->
        <property name="applicationContextJobDataKey" value="adc"/>
    </bean>

    <!-- 定义触发器 -->
    <bean id="cronTrigger" class="org.springframework.scheduling.quartz.CronTrigger
        <property name="jobDetail" ref="myjobDetail" />
        <!-- 每一分钟执行一次 -->
        <property name="cronExpression" value="*/5 * * * * ?" />
    </bean>

    <!-- 定义调度器 -->
    <bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref bean="cronTrigger" />
                <ref bean="cronTrigger1" />
            </list>
        </property>
    </bean>
```

加载spring容器

```
    public static void main(String[] args) {
        // 写对应的配置文件
        new ClassPathXmlApplicationContext("classpath:Spring.xml");
    }
```

```
public static void main(String[] args) throws SchedulerException {
    //1.创建Scheduler的工厂
    SchedulerFactory sf = new StdSchedulerFactory();
    //2.从工厂中获取调度器实例
    Scheduler scheduler = sf.getScheduler();


    //3.创建JobDetail
    JobDetail jb = JobBuilder.newJob(RAMJob.class)
            .withDescription("this is a ram job") //job的描述
            .withIdentity("ramJob", "ramGroup") //job 的name和group
            .build();

    //任务运行的时间，SimpleSchedle类型触发器有效
    long time=  System.currentTimeMillis() + 3*1000L; //3秒后启动任务
    Date statTime = new Date(time);

    //4.创建Trigger
        //使用SimpleScheduleBuilder或者CronScheduleBuilder
    Trigger t = TriggerBuilder.newTrigger()
                .withDescription("")
                .withIdentity("ramTrigger", "ramTriggerGroup")
                //.withSchedule(SimpleScheduleBuilder.simpleSchedule())
                .startAt(statTime)  //默认当前时间启动
                .withSchedule(CronScheduleBuilder.cronSchedule("0/2 * * * * ?"))
                .build();

    //5.注册任务和定时器
    scheduler.scheduleJob(jb, t);

    //6.启动 调度器
    scheduler.start();
    _log.info("启动时间 :  " + new Date());

}
```

- Scheduler - 与调度器交互的主要API。
- Job - 需要被调度器调度的任务必须实现的接口。
- JobDetail - 用于定义任务的实例。
- Trigger - 用于定义调度器何时调度任务执行的组件。

- JobBuilder - 用于定义或创建JobDetail的实例 。
- TriggerBuilder - 用于定义或创建触发器实例。
(个人总结两条生产线一条整合线:
1. JobBuilder ——JobDetail——Job
2. TriggerBuilder——Trigger
3. Scheduler将Job和Trigger整合在一起)