1-读取配置文件

1.1-定时任务配置

1.1.1 配置方式

从实现技术上可分为三类

java 自带的 java.util.Timer 类，此类允许调度一个 java.util.TimerTask 任务，不建议使用；

使用 Quartz，功能强大，支持分布式，但配置复杂

Spring3.0 以后自带的 task，相当于轻量级的 Quartz,使用较为简单

从作业类继承方式上，可分类两类

继承特定基类，如 Quartz 需要继承 org.springframework.scheduling.quartz.QuartzJobBean；java.util.Timer 中需要继承自 java.util.TimerTask

作业类为普通 java 类，不需要继承任何基类

【corn 表达式】

[秒 分 时 日期 月份 星期 年(可选)]

/ 每隔多久触发

- 区间

* 通配符

? 不设置该字段

1.1.2 Spring Task

注意：配置文件中引入 task 命名空间

1.1.2.1 注解式

作业类

```
@Component(“taskJob”)
public class TaskJob {
	@Scheduled(cron = "0 0 3 * * ?")
	public void job1() {
		System.out.println(“任务进行中。。。”);
	}
}
```

配置文件

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:tx="http://www.springframework.org/schema/tx"
	xmlns:task="http://www.springframework.org/schema/task"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
		http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
		http://www.springframework.org/schema/context
http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd
		http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
		http://www.springframework.org/schema/task
http://www.springframework.org/schema/task/spring-task-3.0.xsd"
	default-lazy-init="false">

	<context:annotation-config />
	<!—spring 扫描注解的配置	-->
	<context:component-scan base-package="com.gy.mytask" />

<!—开启这个配置，spring 才能识别@Scheduled 注解	-->
	<task:annotation-driven scheduler="qbScheduler" mode="proxy"/>
	<task:scheduler id="qbScheduler" pool-size="10"/>
</beans>
```

理论上使用<task:annotation-driven /> 开启@Scheduled 注解即可

1.1.2.2 配置文件方式

job 类

```
@Service
public class TaskJob {

	public void job1() {
		System.out.println(“任务进行中。。。”);
	}
}
```

spring 配置

```
<task:scheduled-tasks>
		<task:scheduled ref="taskJob" method="job1" cron="0 * * * * ?"/>
</task:scheduled-tasks>
```

1.1.3 Quartz

1.1.3.1 作业类继承特定基类

作业类

```
public class Job1 extends QuartzJobBean {
	private int timeout;
	private static int i = 0;
	//调度工厂实例化后，经过 timeout 时间开始执行调度
	public void setTimeout(int timeout) {
		this.timeout = timeout;
	}
	/**
	* 要调度的具体任务
	*/
	@Override
	protected void executeInternal(JobExecutionContext context) throws JobExecutionException {
```

```
            System.out.println("定时任务执行中…");
        }
}
```

配置文件中配置 JobDetailBean

```
<bean name="job1" class="org.springframework.scheduling.quartz.JobDetailBean">
        <property name="jobClass" value="com.gy.Job1" />
        <property name="jobDataAsMap">
            <map>
                <entry key="timeout" value="0" />
            </map>
        </property>
</bean>
```

jobClass 即为 任务类， jobDataAsMap 为任务类中需要注入的属性值

触发器

频率触发

```
<bean id="simpleTrigger" class="org.springframework.scheduling.quartz.SimpleTriggerBean">
        <property name="jobDetail" ref="job1" />
        <property name="startDelay" value="0" /><!-- 调度工厂实例化后，经过 0 秒开始执行调度
-->
        <property name="repeatInterval" value="2000" /><!-- 每 2 秒调度一次 -->
</bean>
```

或

Cron 表达式

```
<bean id="cronTrigger" class="org.springframework.scheduling.quartz.CronTriggerBean">
        <property name="jobDetail" ref="job1" />
        <!—每天 12:00 运行一次 -->
        <property name="cronExpression" value="0 0 12 * * ?" />
</bean>
```

调度工厂

```
<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
        <list>
            <ref bean="cronTrigger" />
        </list>
        </property>
</bean>
```

1.1.3.2 作业类不继承特定基类

Spring 能够支持这种方式，归功于两个类：

org.springframework.scheduling.timer.MethodInvokingTimerTaskFactoryBean

org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean

作业类

```
public class Job2 {
    public void doJob2() {
        System.out.println("不继承 QuartzJobBean 方式-调度进行中...");
    }
}
```

配置作业类

```
<bean id="job2"
    class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
    <property name="targetObject">
        <bean class="com.gy.Job2" />
    </property>
    <property name="targetMethod" value="doJob2" />
    <property name="concurrent" value="false" /><!-- 作业不并发调度 -->
</bean>
```

触发器和调度工厂同【继承特定基类】方式

1.1.3.3 quartz.properties 配置说明

此文件在 quartz 的 jar 包有，可直接拿过来使用不过只有基本的几个配置 自己可根据需要进行扩充；另外如果项目中没有对该配置文件重写，则 Quartz 会加载自己 jar 包中的 quartz.properties 文件。

默认配置，如下：

```
# Default Properties file for use by StdSchedulerFactory
# to create a Quartz Scheduler Instance, if a different
# properties file is not explicitly specified.
#
# ===========================================================================
# Configure Main Scheduler Properties 调度器属性
```

```
# ============================================================================
org.quartz.scheduler.instanceName: DefaultQuartzScheduler
#org.quartz.scheduler.instanceid:AUTO
org.quartz.scheduler.rmi.export: false
org.quartz.scheduler.rmi.proxy: false
org.quartz.scheduler.wrapJobExecutionInUserTransaction: false
# ============================================================================
# Configure ThreadPool 线程池属性
# ============================================================================
#线程池的实现类（一般使用 SimpleThreadPool 即可满足几乎所有用户的需求）
org.quartz.threadPool.class: org.quartz.simpl.SimpleThreadPool
#指定线程数，至少为 1（无默认值）(一般设置为 1-100 直接的整数合适)
org.quartz.threadPool.threadCount: 10
# 设置线程的优先级（最大为 java.lang.Thread.MAX_PRIORITY 10，最小为 Thread.MIN_PRIORITY 1，默认为 5）
org.quartz.threadPool.threadPriority: 5
#设置 SimpleThreadPool 的一些属性
#设置是否为守护线程
#org.quartz.threadpool.makethreadsdaemons = false
#org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread: true
#org.quartz.threadpool.threadsinheritgroupofinitializingthread=false
#线程前缀默认值是：[Scheduler Name]_Worker
#org.quartz.threadpool.threadnameprefix=swhJobThead;
# 配置全局监听(TriggerListener,JobListener) 则应用程序可以接收和执行 预定的事件通知
# ============================================================================
# Configuring a Global TriggerListener 配置全局的 Trigger 监听器
# MyTriggerListenerClass 类必须有一个无参数的构造函数，和 属性的 set 方法，目前 2.2.x 只支持原始数据类型的值（包括字符串）
# ============================================================================
#org.quartz.triggerListener.NAME.class = com.swh.MyTriggerListenerClass
#org.quartz.triggerListener.NAME.propName = propValue
#org.quartz.triggerListener.NAME.prop2Name = prop2Value
# ============================================================================
# Configuring a Global JobListener 配置全局的 Job 监听器
# MyJobListenerClass 类必须有一个无参数的构造函数，和 属性的 set 方法，目前 2.2.x 只支持原始数据类型的值（包括字符串）
# ============================================================================
#org.quartz.jobListener.NAME.class = com.swh.MyJobListenerClass
#org.quartz.jobListener.NAME.propName = propValue
#org.quartz.jobListener.NAME.prop2Name = prop2Value
# ============================================================================
# Configure JobStore 存储调度信息（工作，触发器和日历等）
# ============================================================================
# 信息保存时间 默认值 60 秒
```

org.quartz.jobStore.misfireThreshold: 60000
#保存 job 和 Trigger 的状态信息到内存中的类
org.quartz.jobStore.class: org.quartz.simpl.RAMJobStore
==
Configure SchedulerPlugins 插件属性 配置
==
自定义插件
#org.quartz.plugin.NAME.class = com.swh.MyPluginClass
#org.quartz.plugin.NAME.propName = propValue
#org.quartz.plugin.NAME.prop2Name = prop2Value
#配置 trigger 执行历史日志（可以看到类的文档和参数列表）
org.quartz.plugin.triggHistory.class = org.quartz.plugins.history.LoggingTriggerHistoryPlugin
org.quartz.plugin.triggHistory.triggerFiredMessage = Trigger {1}.{0} fired job {6}.{5} at: {4, date, HH:mm:ss MM/dd/yyyy}
org.quartz.plugin.triggHistory.triggerCompleteMessage = Trigger {1}.{0} completed firing job {6}.{5} at {4, date, HH:mm:ss MM/dd/yyyy} with resulting trigger instruction code: {9}
#配置 job 调度插件 quartz_jobs(jobs and triggers 内容)的 XML 文档
# 加载 Job 和 Trigger 信息的类 （1.8 之前用：org.quartz.plugins.xml.JobInitializationPlugin）
org.quartz.plugin.jobInitializer.class = org.quartz.plugins.xml.XMLSchedulingDataProcessorPlugin
#指定存放调度器(Job 和 Trigger)信息的 xml 文件，默认是 classpath 下 quartz_jobs.xml
org.quartz.plugin.jobInitializer.fileNames = my_quartz_job2.xml
#org.quartz.plugin.jobInitializer.overWriteExistingJobs = false
org.quartz.plugin.jobInitializer.failOnFileNotFound = true
#自动扫描任务单并发现改动的时间间隔,单位为秒
org.quartz.plugin.jobInitializer.scanInterval = 10
#覆盖任务调度器中同名的 jobDetail,避免只修改了 CronExpression 所造成的不能重新生效情况
org.quartz.plugin.jobInitializer.wrapInUserTransaction = false
# ==
# Sample configuration of ShutdownHookPlugin ShutdownHookPlugin 插件的配置样例
# ==
#org.quartz.plugin.shutdownhook.class = \org.quartz.plugins.management.ShutdownHookPlugin
#org.quartz.plugin.shutdownhook.cleanShutdown = true
#
Configure RMI Settings 远程服务调用配置
#
#如果你想 quartz-scheduler 出口本身通过 RMI 作为服务器，然后设置“出口”标志 true(默认值为 false)。
#org.quartz.scheduler.rmi.export = false
#主机上 rmi 注册表(默认值 localhost)
#org.quartz.scheduler.rmi.registryhost = localhost
#注册监听端口号（默认值 1099）
#org.quartz.scheduler.rmi.registryport = 1099

#创建 rmi 注册，false/never：如果你已经有一个在运行或不想进行创建注册
true/as_needed:第一次尝试使用现有的注册，然后再回来进行创建
always:先进行创建一个注册，然后再使用回来使用注册
#org.quartz.scheduler.rmi.createregistry = never
#Quartz Scheduler 服务端端口，默认是随机分配 RMI 注册表
#org.quartz.scheduler.rmi.serverport = 1098
#true:链接远程服务调度(客户端),这个也要指定 registryhost 和 registryport，默认为 false
如果 export 和 proxy 同时指定为 true，则 export 的设置将被忽略
#org.quartz.scheduler.rmi.proxy = false

说明

instanceName ： 调度器示例名，可为任意字符串；多个调度器通常用于集群环境
instanceId ： 这个值必须是在所有调度器中是唯一的，尤其是在一个集群中。需要 Quartz 生成，可设置为 AUTO;若 Quartz 框架运行在非集群环境中，那么自动产生的值将会是 NON_CLUSTERED;集群环境下使用 Quartz,将会是主机名加上当前的日期和时间。一般设为 AUTO。