QuartZ

1. QuartZ概述
   1. QuartZ是java编写的，开源作业调度框架
   2. 支持集群:org.quartz.jobStore.isClustered
   3. 阿里开源项目easySchedule在quartz集群的基础上搭建了一个简单的管理平台。解决了可视化、易配置、统一监控告警功能
2. QuartZ的核心
   1. Job 是一个接口，里面只有一个方法excute()，独立使用QuartZ开发时，需要实现此接口，整合Spring

则无需实现；Job运行时的信息保存在一个HashMap（JobDataMap）中

* 1. JobDetail 接收Job，QuartZ每次执行任务时，创建一个Job，它需要接收Job的类和方法名称，以便通过反射实

例化，并执行

* 1. Trigger 接收JobDetail，描述时间触发规则
  2. Scheduler 代表QuartZ独立运行容器，Trigger和JobDetail可以注册到Scheduler中，两者在Scheduler中拥有各

自的组及名称，组及名称是Scheduler查找定位容器中某一对象的依据，Trigger的组及名称必须唯一，JobDetail的组和名称也必须唯一（但可以和Trigger的组和名称相同，因为它们是不同类型的）。Scheduler定义了多个接口方法，允许外部通过组及名称访问和控制容器中Trigger和JobDetail

* 1. Calendar org.quartz.Calendar和java.util.Calendar不同，它是一个特定时间点的集合，如果我们要求每周一10点

执行，如遇到法定节日，则不执行

* 1. ThreadPool Scheduler使用一个线程池作为任务运行的基础设施，任务通过共享线程池中的线程提高运行效率
  2. 注： 一个Job可以对应多个Trigger, 一个Trigger只能够对应一个Job

1. QuartZ设计分析
   1. quartz.properties文件配置说明
      1. 调度器属性配置
         1. org.quartz.scheduler.instanceName: DefaultQuartzScheduler
         2. #org.quartz.scheduler.instanceid:AUTO
         3. org.quartz.scheduler.rmi.export: false
         4. org.quartz.scheduler.rmi.proxy: false
         5. org.quartz.scheduler.wrapJobExecutionInUserTransaction: false
         6. org.quartz.scheduler.batchTriggerAcquisitionMaxCount=50

一次性获取trigger的数量，可防止在多线程的情况下，同一个trigger被多个线程同时执行，在trigger很少的情况下，意义不大

* + 1. 线程池属性配置
       1. org.quartz.threadPool.class: org.quartz.simpl.SimpleThreadPool

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）

* + - 1. org.quartz.threadPool.threadCount: 10

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)

* + - 1. org.quartz.threadPool.threadPriority: 5

设置线程的优先级（最大为10，最小为1，默认为5）

* + - 1. #org.quartz.threadpool.makethreadsdaemons = false

SimpleThreadPool的一些属性之：设置是否为守护线程

* + 1. 配置全局的Trigger监听器
       1. org.quartz.triggerListener.NAME.class = com.swh.MyTriggerListenerClass

此类为trigger监听类，此类必需含有无参构造及set()

* + - 1. org.quartz.triggerListener.NAME.propName = propValue
      2. org.quartz.triggerListener.NAME.prop2Name = prop2Value
    1. 配置全局的Job监听器
       1. org.quartz.jobListener.NAME.class = com.swh.MyJobListenerClass

此类为job监听类，此类必需含有无参构造及set()

* + - 1. org.quartz.jobListener.NAME.propName = propValue
      2. org.quartz.jobListener.NAME.prop2Name = prop2Value
    1. 存储调度信息（工作，触发器和日历等）
       1. org.quartz.jobStore.misfireThreshold: 60000

超过默认值60秒还未执行的任务，认为是misfire, 将作业重新放到Trigger表中

* + - 1. org.quartz.jobStore.class: org.quartz.simpl.RAMJobStore

保存job和Trigger的状态信息到内存中的类

* + 1. 插件属性 配置

略

* + 1. ShutdownHookPlugin插件的配置 样例

略

* 1. 作业

作业就是一个简单的Java类，出现严重错误时，抛出JobExecutionException

* 1. 作业管理和存储

调度器跟踪作业和它们的执行次数

* + 1. RAMJobStore 由内存实现，重启系统信息丢失
    2. JDBC作业存储
       1. JobStoreTX 自行控制事务或在非应用服务器环境中使用
       2. JobStoreCMT 容器控制事务或在应用服务器环境中使用
  1. 作业和触发器
     1. SimpleTrigger 一定时间内执行多少次
     2. CronTrigger 周期性执行
  2. 错过触发（misfire）
     1. 如果在触发时间点上，没有线程给它调用，那么它将错过触发
     2. Trigger的重要属性
  3. 有状态和无状态作业
     1. 作业无状态时，多次作业不会去维护JobDataMap
     2. 有状态作业实现接口为org.quartz.StatefulJob
     3. 有状态作业和无状态作业最大的区别在于，有状态作业只有一个实例
     4. 有状态作业一般而言不会有太大的问题，但如果频繁执行或长时间执行作业，那么可能会有问题
  4. 监听和插件
     1. 实现监听和插件需要实现JobListner接口

1. 与Spring整合时的配置

<bean id="scheduler" class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="dataSource">

<ref bean="scheduleDataSource" /> //作业任务

</property>

<property name="autoStartup" value="true" />

<property name="schedulerFactoryClass" value="org.quartz.impl.StdSchedulerFactory"></property>

<property name="configLocation" value="classpath:quartz.properties" /> //配置文件

<property name="globalJobListeners">

<list>

<ref bean="jobListener" /> //监听

</list>

</property>

</bean>

1. QuartZ集群的特点
   1. QuartZ集群中的每一个节点都是独立的
   2. 任一一个节点都不和其它节点进行通信
   3. 任一节点通过数据库表来感知其它节点
   4. 只有使用JobStore才能够完成集群
2. 基于Spring的集群配置

<bean id="quartzScheduler" class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="quartzProperties">

<props>

<prop key="org.quartz.scheduler.instanceName">CRMscheduler</prop>

<prop key="org.quartz.scheduler.instanceId">AUTO</prop>

<!-- 线程池配置 -->

<prop key="org.quartz.threadPool.class">org.quartz.simpl.SimpleThreadPool</prop>

<prop key="org.quartz.threadPool.threadCount">20</prop>

<prop key="org.quartz.threadPool.threadPriority">5</prop>

<!-- JobStore 配置 -->

<prop key="org.quartz.jobStore.class">org.quartz.impl.jdbcjobstore.JobStoreTX</prop>

<!-- 集群配置 -->

<prop key="org.quartz.jobStore.isClustered">true</prop>

<prop key="org.quartz.jobStore.clusterCheckinInterval">15000</prop>

<prop key="org.quartz.jobStore.maxMisfiresToHandleAtATime">1</prop>

<prop key="org.quartz.jobStore.misfireThreshold">120000</prop>

<prop key="org.quartz.jobStore.tablePrefix">QRTZ\_</prop>

</props>

</property>

<property name="schedulerName" value="CRMscheduler"/>

<!--必须的，QuartzScheduler 延时启动，应用启动完后 QuartzScheduler 再启动 -->

<property name="startupDelay" value="30"/>

<property name="applicationContextSchedulerContextKey" value="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qrtz\_job\_details表对应记录了 -->

<property name="overwriteExistingJobs" value="true"/>

<!-- 设置自动启动 -->

<property name="autoStartup" value="true"/>

<!-- 注册触发器 -->

<property name="triggers">

<list>

<ref bean="userSyncScannerTrigger"/>

......

</list>

</property>

<!-- 注册jobDetail -->

<property name="jobDetails">

<list>

</list>

</property>

<property name="schedulerListeners">

<list>

<ref bean="quartzExceptionSchedulerListener"/>

</list>

</property>

</bean>

1. QuartZ集群的数据库表
   1. QRTZ\_CALENDARS 存储Quartz的Calendar信息
   2. QRTZCRONTRIGGERS 存储CronTrigger，包括Cron表达式和时区信息
   3. QRTZFIREDTRIGGERS 存储与已触发的Trigger相关的状态信息，以及相联Job的执行信息
   4. QRTZPAUSEDTRIGGER\_GRPS 存储已暂停的Trigger组的信息
   5. QRTZSCHEDULERSTATE 存储少量的有关Scheduler的状态信息，和别的Scheduler实例
   6. QRTZ\_LOCKS 存储程序的悲观锁的信息(实现同步机制的行锁表)
   7. QRTZJOBDETAILS 存储每一个已配置的Job的详细信息
   8. QRTZJOBLISTENERS 存储有关已配置的JobListener的信息
   9. QRTZSIMPLETRIGGERS 存储简单的Trigger，包括重复次数、间隔、以及已触的次数
   10. QRTZBLOGTRIGGERS Trigger作为Blob类型存储
   11. QRTZTRIGGERLISTENERS 存储已配置的TriggerListener的信息
   12. QRTZ\_TRIGGERS 存储已配置的Trigger的信息
2. QRTZ\_LOCKS表数据的说明

共5条记录，代表5把锁

* 1. CALENDAR\_ACCESS
  2. JOB\_ACCESS
  3. MISFIRE\_ACCESS
  4. STATE\_ACCESS
  5. TRIGGER\_ACCESS

1. QuartZ中的2类线程
   1. Scheduler调度线程
   2. 任务执行线程
2. QuartZ流程分析
   1. 当Spring中配置QuartZ时，新建线程，运行run();
   2. Run中死循环不断检测Trigger中维护的开始时间
   3. 调度器在Trigger队列中寻找X秒内一定数量的Trigger，需保证集群节点时间一致
   4. 时间到了，触发Trigger
      1. 时间到了，先通过select \* from 锁名 where 。。。。for update去获取这把锁
      2. 从线程池中拿取线程，执行作业任务
   5. 释放Trigger
      1. 释放锁