# 哑火说明

哑火顾名思义，就是quartz在应该触发（fire）trigger的时候未能及时将其触发，这将导致trigger的下次触发时间落在当前时间之前， 那么按照正常的quartz调度流程，该trigger就再也没有机会被调度了。由于一个调度器实例在每次调度的过程中都会有一定的睡眠时间，存在在一段 时间内所有调度器实例都在睡眠，而无人触发调度的潜在可能。于是调度器需要每隔一段时间（15s~60s）查看一次各trigger的 nextfiretime，检查出否有tirgger的下一次触发落在了当前时间之前足够长的时间，在这里系统设定了一个60s的域 （misfireThreshold），当一个trigger的下一次触发时间早于当前时间60s之外时，调度器判定该触发器misfired，在发现有 触发器哑火之后启动相应的流程回复trigger至正常状态。上述这些过程是在调度器初始化时与主调度线程类quartzSchedulerThread 同时开启的一个线程类MisfireHandler中进行的。



# 参数介绍

## Job类参数

@DisallowConcurrentExecution

不允许并发执行

@PersistJobDataAfterExecution

表示当正常执行完Job后, JobDataMap中的数据应该被改动, 以备下一次调用时用。

## Quartz.properties

|  |
| --- |
| org.quartz.scheduler.instanceName = CommonScheduler  org.quartz.scheduler.instanceId = AUTO  org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool  #同时并发的线程总数  org.quartz.threadPool.threadCount = 50  org.quartz.threadPool.threadPriority = 5  org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread = true  **#超过应该执行的时间多久后，认为是哑火**  **org.quartz.jobStore.misfireThreshold = 60000**  org.quartz.jobStore.class = org.quartz.impl.jdbcjobstore.JobStoreTX  org.quartz.jobStore.driverDelegateClass=org.quartz.impl.jdbcjobstore.StdJDBCDelegate  org.quartz.jobStore.tablePrefix = QRTZ\_  #同时执行的misfired任务值  #20  **org.quartz.jobStore.maxMisfiresToHandleAtATime=10**  org.quartz.jobStore.isClustered = true  #检测集群掉线的频率  #15000  org.quartz.jobStore.clusterCheckinInterval =20000  #-------------new---------------------  org.quartz.plugin.triggHistory.class = org.quartz.plugins.history.LoggingTriggerHistoryPlugin  org.quartz.plugin.triggHistory.triggerFiredMessage = Trigger **{1}**.**{0}** fired job1111111111 **{6}**.**{5}** at\: {4, date, HH\:mm\:ss MM/dd/yyyy}  org.quartz.plugin.triggHistory.triggerMisfiredMessage = Trigger **{1}**.**{0}** misfired33333333 job **{6}**.**{5}** at: {4, date, HH:mm:ss MM/dd/yyyy}. Should have fired at: {3, date, HH:mm:ss MM/dd/yyyy}  org.quartz.plugin.triggHistory.triggerCompleteMessage = Trigger **{1}**.**{0}** completed22222222 firing job **{6}**.**{5}** at {4, date, HH\:mm\:ss MM/dd/yyyy}. |

## applicationContext-quartz-xxxjob.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>    <!-- 启动触发器的配置开始 -->  <!-- 总管理类 如果将lazy-init='false'那么容器启动就会执行调度程序 -->  <bean name=*"startQuertz"* lazy-init=*"false"* autowire=*"no"* class=*"org.springframework.scheduling.quartz.SchedulerFactoryBean"*>  <property name=*"triggers"*>  <!-- 调度清单 -->  <list>  <ref bean=*"simpleTrigger"* />  </list>  </property>  <!-- quartz配置 -->  <property name=*"quartzProperties"*>  <props>  <prop key=*"org.quartz.threadPool.threadCount"*>10</prop>  <prop key=*"org.quartz.jobStore.misfireThreshold"*>1</prop>  </props>  </property>  </bean>    <!-- 启动触发器的配置结束 -->    <!-- 调度的配置开始 -->  <!--  quartz-1.8以前的配置  <bean id="myJobTrigger"  class="org.springframework.scheduling.quartz.CronTriggerBean">  <property name="jobDetail">  <ref bean="myJobDetail" />  </property>  **<property name="cronExpression">**  **<value>0/1 \* \* \* \* ?</value>**  **</property>**  </bean>  -->  <!-- quartz-2.x的配置 -->  <!-- 定义触发时间 -->  <!-- cronTrigger简单触发器配置 -->  <!-- 默认是withMisfireHandlingInstructionFireAndProceed —— 以当前时间为触发频率立刻触发一次执行，然后按照Cron频率依次执行，如没有空闲进程可用，则kill掉前面那个调度，立即触发当前这个-->  <bean id=*"simpleTrigger"* class=*"org.springframework.scheduling.quartz.SimpleTriggerFactoryBean"*>  <property name=*"jobDetail"*>  <ref bean=*"simpleJobDetail"*/>  </property>  <!-- <property name="startDelay" value="1" /> -->  <!-- 每3s执行一次 -->  **<property name=*"repeatInterval"* value=*"3000"* />**  <property name=*"repeatCount"* value=*"5"* />  <!--  MISFIRE\_INSTRUCTION\_FIRE\_NOW 1    Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be fired now by Scheduler.    NOTE: This instruction should typically only be used for 'one-shot' (non-repeating) Triggers. If it is used on a trigger with a repeat count > 0 then it is equivalent to the instruction MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_REMAINING\_REPEAT\_COUNT .  立即执行，会丢失misfire job，等效于MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_REMAINING\_REPEAT\_COUNT      MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_REMAINING\_REPEAT\_COUNT 3    Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be re-scheduled to 'now' (even if the associated Calendar excludes 'now') with the repeat count set to what it would be, if it had not missed any firings. This does obey the Trigger end-time however, so if 'now' is after the end-time the Trigger will not fire again.    NOTE: Use of this instruction causes the trigger to 'forget' the start-time and repeat-count that it was originally setup with. Instead, the repeat count on the trigger will be changed to whatever the remaining repeat count is (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time).    NOTE: This instruction could cause the Trigger to go to the 'COMPLETE' state after firing 'now', if all the repeat-fire-times where missed.  立即执行，继续重复执行的次数等于本次+剩余的次数，即剩余次数+1，misfire job不会再执行.  比如，3s执行一次，重复执行5次，05:08:11开始执行，那么理应重复执行的时间点是05:08:11,05:08:14,05:08:17,05:08:20,05:08:23,05:08:26,  但如果第一次执行的时间是11s，执行完的时间点是：05:08:22,，那么立即执行的时间点是05:08:22,  而且从这点开始，还有05:08:23,05:08:26,这两次没执行，所以立即执行后，还会再执行2次。  所以之后的执行时间点是：05:08:22,05:08:25,05:08:28    MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_EXISTING\_COUNT 5    Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be re-scheduled to the next scheduled time after 'now' - taking into account any associated Calendar, and with the repeat count left unchanged.    NOTE/WARNING: This instruction could cause the Trigger to go directly to the 'COMPLETE' state if the end-time of the trigger has arrived.  不会立即执行，misfire job不会再执行，到下一个触发点再执行，继续重复执行的次数等于剩余次数-1.  比如，3s执行一次，重复执行5次，02:28:38开始执行，那么理应重复执行的时间点是02:28:38，02:28:41，02:28:44，02:28:47，02:28:50，02:28:53  但如果第一次执行时间用了11s，到02:28:49结束，那下次执行时间是02:28:50，02:28:53，只有(2-1)=1次，所以一次执行时间点是：02:28:50        MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_REMAINING\_COUNT 4    Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be re-scheduled to the next scheduled time after 'now' - taking into account any associated Calendar, and with the repeat count set to what it would be, if it had not missed any firings.    NOTE/WARNING: This instruction could cause the Trigger to go directly to the 'COMPLETE' state if all fire-times where missed.  不会立即执行，misfire job不会再执行，到下一个触发点再执行，继续重复执行的次数等于剩余次数-1.  会根据第一次执行的时间，然后从当前时间开始，到终止时间，计算还能执行的剩余次数，重复执行这个剩余次数。相当于重新计算剩余次数，进行调度。  如：3s执行一次，重复执行5次，从 05:18:52开始，理论上依次执行的时间点是 05:18:55,05:18:58,05:19:01,05:19:04,05:19:07  但如果第一次执行了11s，到05:19:03结束，则继续按照本来的调度，下次执行的开始时间是05:19:04,接着执行05:19:07  且错过的05:18:55,05:18:58,05:19:01的触发不会再执行下去      MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT 2    Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be re-scheduled to 'now' (even if the associated Calendar excludes 'now') with the repeat count left as-is. This does obey the Trigger end-time however, so if 'now' is after the end-time the Trigger will not fire again.    NOTE: Use of this instruction causes the trigger to 'forget' the start-time and repeat-count that it was originally setup with (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time).  立即执行，不会丢失misfire job，从当前时刻开始重新计算每次执行时间点，重做misfire的job  如：3s执行一次，重复执行5次，05:24:44开始，每次时间点理应是：05:24:47，05:24:50，05:24:53，05:24:56，05:24:59  但由于第一次执行用了11s，到05:24:55结束，那么从05:24:55开始重新计算剩余次数5次的每次执行的时间点，假设剩余执行每次只需执行2s时间，不会Misfire  那么剩余执行调度的时间点是：05:24:55，05:24:58，05:25:01，05:25:04，05:25:07        MISFIRE\_INSTRUCTION\_SMART\_POLICY 0 default    Instructs the Scheduler that upon a mis-fire situation, the updateAfterMisfire() method will be called on the Trigger to determine the mis-fire instruction, which logic will be trigger-implementation-dependent.    In order to see if this instruction fits your needs, you should look at the documentation for the getSmartMisfirePolicy() method on the particular Trigger implementation you are using.    如果没有自定义的话，当misfire后，立即执行，不会丢失misfire job... 跟MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT差不多策略  -->  <property name=*"**misfireInstruction"*><value>0</value></property>  </bean>  <!-- 调度的配置结束 -->    <!-- job的配置开始 -->  <!-- 定义调用对象和调用对象的方法 -->  <bean id=*"simpleJobDetail"* class=*"org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"*>  <property name=*"targetObject"* ref=*"missfireJob"* />  <property name=*"targetMethod"* value=*"work"* />  <!-- <property name="concurrent" value="false" /> -->  <!-- 同步执行 -->  <property name=*"concurrent"* value=*"false"* />  </bean>  <!-- job的配置结束 -->    <!-- 要调用的工作类 -->  <bean id=*"missfireJob"* class=*"com.liangbinny.quartz.example6.QuartzMissfireJob"* />  </beans> |

# 测试方案

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 策略 | misfireInstruction | fire1（11s） | fire2 | Fire3 | Fire4 | Fire5 | Fire6 |
| 理论值 |  | 0 | 3 | 6 | 9 | 12 | 15 |
| MISFIRE\_INSTRUCTION\_SMART\_POLICY | 0（default）当misfire后，立即执行，不会丢失misfire job | 0 | 11 | 14 | 17 | 20 | 23 |
| MISFIRE\_INSTRUCTION\_FIRE\_NOW | 1立即执行，会丢失misfire job | 0 | 11 | 14 | 17 |  |  |
| MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT | 2（0）立即执行，不会丢失misfire job，从当前时刻开始重新计算每次执行时间点，重做misfire的job | 0 | 11 | 14 | 17 | 20 | 23 |
| MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_REMAINING\_REPEAT\_COUNT | 3（1）立即执行，继续重复执行的次数等于本次+剩余的次数，即剩余次数+1，misfire job不会再执行 | 0 | 11 | 14 | 17 |  |  |
| MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_REMAINING\_COUNT | 4不会立即执行，misfire job不会再执行，到下一个触发点再执行，继续重复执行的次数等于剩余次数. | 0 | 12 | 15 |  |  |  |
| MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_EXISTING\_COUNT | 5不会立即执行，misfire job不会再执行，到下一个触发点再执行，继续重复执行的次数等于剩余次数-1 | 0 | 12 | 15 |  |  |  |
| 增加测试 | misfireThreshold=60000 | 0 | 11 | 11 | 11 | 12 | 15 |

# 改进方案

1.尽可能扩大misfireThreshold值

2.改进EayunTriggerListeners的triggerMisfired（曾波）

3.优化并严格控制quartz的job的执行时间

4.需要走查与钱有关的job代码（目前只有后付费资源计费1h一次，第三方付费情况监测30s一次）（明天走查）

5.增加job上的注解（祝军）

6.目前schedule启动时，是将QRTZ表全删全增的，需要避免正在执行job的点来重启服务，最好不去修改job的信息

7.漏跑的机制

# 其他配置

1.日志配置

2.启动后延迟启动时间